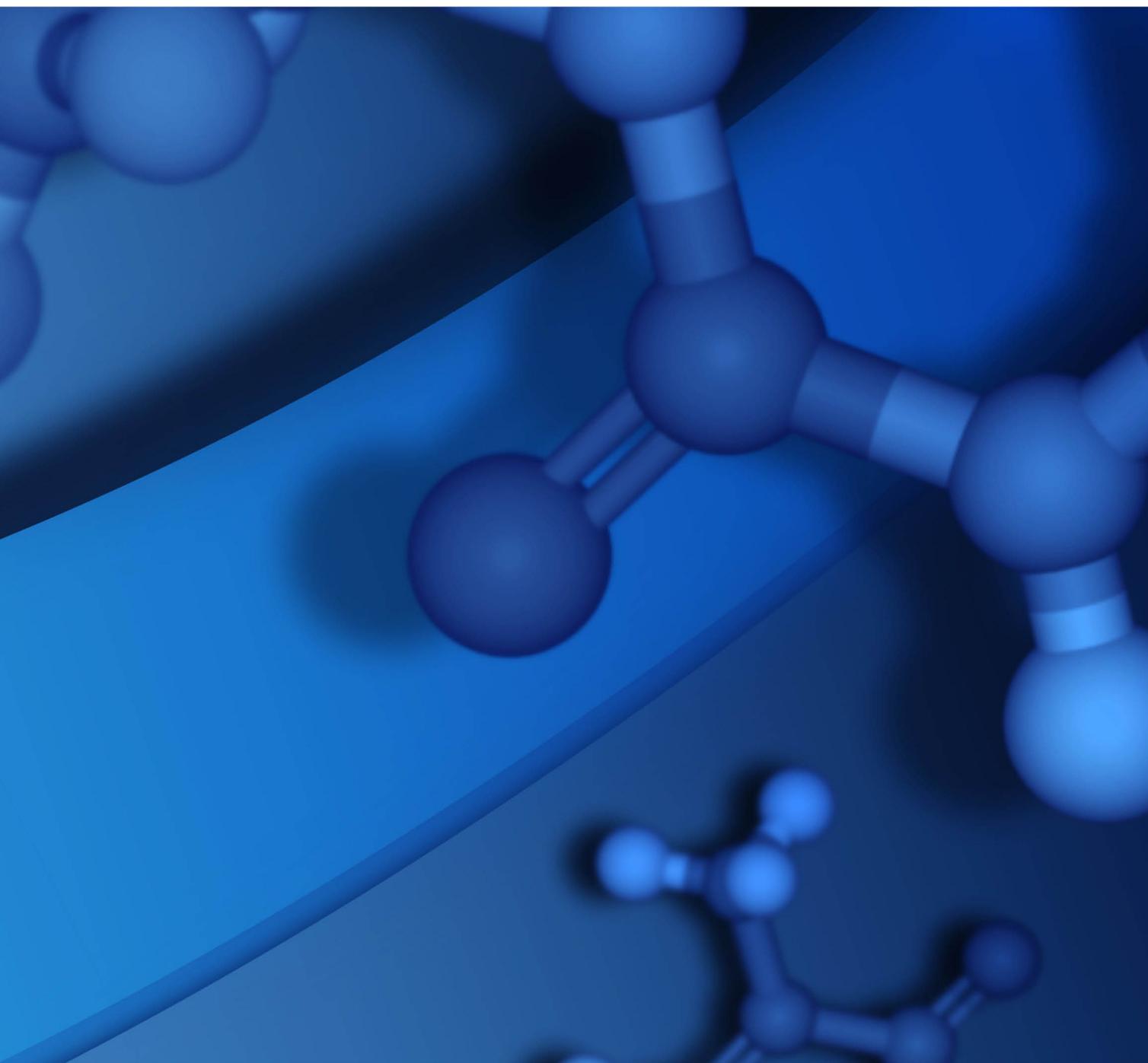


# DEVELOPERS GUIDE

BIOVIA DIRECT 2021



## Copyright Notice

©2020 Dassault Systèmes. All rights reserved. 3DEXPERIENCE, the Compass icon and the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3DVIA, 3DSWYM, BIOVIA, NETVIBES, IFWE and 3DEXCITE, are commercial trademarks or registered trademarks of Dassault Systèmes, a French "société européenne" (Versailles Commercial Register # B 322 306 440), or its subsidiaries in the U.S. and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

## Acknowledgments and References

To print photographs or files of computational results (figures and/or data) obtained by using Dassault Systèmes software, acknowledge the source in an appropriate format. For example:

"Computational results were obtained by using Dassault Systèmes BIOVIA software programs. BIOVIA Direct was used to perform the calculations and to generate the graphical results."

Dassault Systèmes may grant permission to republish or reprint its copyrighted materials. Requests should be submitted to Dassault Systèmes Customer Support, either by visiting <https://www.3ds.com/support/> and clicking **Call us** or **Submit a request**, or by writing to:

Dassault Systèmes Customer Support  
10, Rue Marcel Dassault  
78140 Vélizy-Villacoublay  
FRANCE

# Contents

---

<b>Chapter 1: About Direct</b> .....	<b>1</b>
Direct Overview .....	1
Direct in a SQL Query .....	1
Direct Operators and Functions .....	2
Molecule-Related Tasks .....	3
Reaction-Related Tasks .....	4
Read and Write Files .....	5
Perform Administrative Tasks .....	5
Perform Other Tasks .....	6
Molecules and Reactions as Large Objects .....	6
Molecule and Reaction Objects in SQL Statements .....	6
Temporary LOBs .....	7
Molecule and Reaction Tables .....	9
Direct Domain Indexes .....	9
Direct Domain Index and the Oracle Optimizer .....	10
<b>Chapter 2: How to Use Direct</b> .....	<b>12</b>
Get Information About Molecules .....	12
Fetch Structures .....	12
Retrieve Related Structure Information .....	13
Navigate Structure Search Results .....	17
Save Structure Search Results .....	17
Insert, Update, and Delete Molecules .....	19
Insert and Update Molecule Objects .....	19
Insert Related Structure Information .....	20
Null Structures .....	21
Propagate New Primary Key Value .....	22
Multi-user Registration and Locking .....	23
Biopolymer Search and Registration .....	24
Overview .....	24
Search Monomers in a Biopolymer Sequence .....	24
Substructure Search of Modified Monomers .....	26
Store Biopolymer Sequences .....	26
Get the HELM String .....	27

Convert HELM Strings to molfiles .....	27
Get Information About Reactions .....	28
Search for Reactions .....	28
Fetch Reactions .....	28
Retrieve Related Reaction Information .....	30
Navigate Search Results .....	32
Save the Search Results .....	32
Inserting, Updating, and Deleting Reactions .....	33
Inserting and Updating Reaction Objects .....	34
Inserting Related Reaction Information .....	35
Multi-user Registration and Locking .....	35
Accessing Files .....	36
Checking Errors .....	36
Working with Molecules in a Reaction .....	37
Extracting Molecules from a Reaction .....	37
Inserting Component Molecules into a Molecule Table .....	39
Registration Trigger on a Reaction Table .....	39
Using MakeMolXrefTrigger .....	40
Reaction Tables that Share a Molecule Table .....	41
Maintaining Referential Integrity .....	41
Homology Group Searching and Registration .....	42
Registering Molecules and Homology Group Information .....	43
Searching Structures with Homology Groups .....	43
<b>Chapter 3: Performance Guidelines .....</b>	<b>45</b>
Guidelines Overview .....	45
Indexed Implementation of a Search Operator .....	45
Non-indexed Implementation of a Search Operator .....	45
extproc Memory Usage .....	45
Configuration Issues .....	46
Lack of a Valid Domain Index .....	46
Lack of Valid Statistics .....	46
Schema Mismatch .....	46
Lack of a Valid Fastsearch Index .....	47
Oracle Cache Size .....	47
Optimizing Queries .....	47

Efficient Structure and Reaction Queries .....	48
Performing an Incremental Search .....	48
Avoiding PRODUCT NOT REACTANT and REACTANT NOT PRODUCT Searches .....	48
Usage of the DISTINCT SQL Keyword .....	49
Temporary LOB Usage of Oracle Temporary Tablespace .....	50
Optimizer Hints .....	50
FIRST_ROWS .....	50
INDEX .....	50
FULL .....	51
ORDERED .....	51
SQL Complexity .....	51
WHERE Clause Guidelines .....	51
FROM Clause Guidelines .....	52
SELECT List Guidelines .....	52
Checking the Execution Plan of a SQL Statement .....	52
Generating and Locking Table Statistics .....	53
<b>Chapter 4: Limitations on Resource Conservation Techniques .....</b>	<b>54</b>
Using Oracle Shared Server .....	54
Client Connection and Session Pooling .....	54
Parallel Processes .....	54



# Chapter 1:

## About Direct

---

### Direct Overview

Direct is a read/write data cartridge for searching and registering molecules and reactions in Oracle. Direct extends the features of Oracle to provide direct access to BIOVIA-specific chemical searching and registration capabilities. Direct:

- Stores and manages binary molecules and reactions using the large object data types. See [Molecules and Reactions as Large Objects](#).
- Defines BIOVIA-specific SQL operators to search, retrieve, and register molecules and reactions. See [Direct Operators and Functions](#).
- Defines and uses domain indexes to increase performance of chemical reaction searching. See [Direct Domain Indexes](#).

### Direct in a SQL Query

Direct allows you to use a SQL statement to search and retrieve molecules and reactions from an Oracle table. It also allows you to insert, update, and delete molecules and reactions.

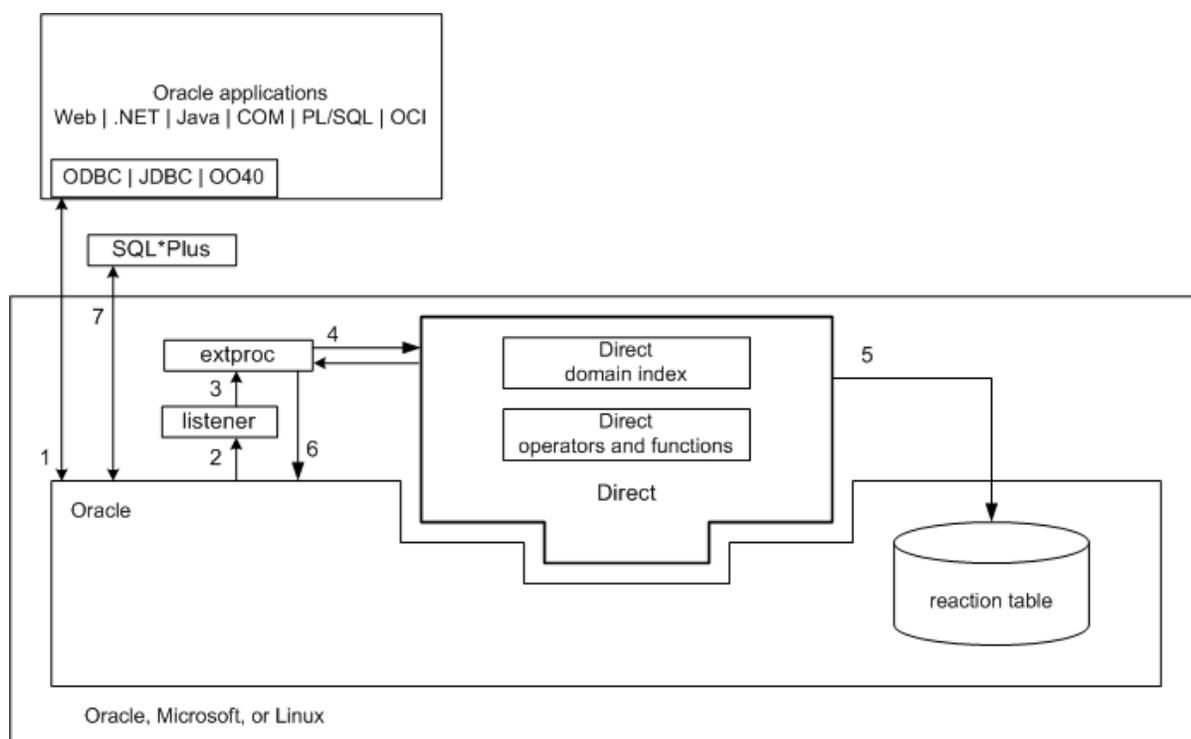
The following is an example of a SQL query that uses Direct:

```
select rxnmdlnumber,  
       rxnfile(rctab)  
  from samplerx_reaction  
 where rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfilesquery.rxn')=1;
```

The SQL query selects the primary key, `rxnmdlnumber`, and the reactions that match a reaction substructure search. `rxnfile` is a Direct operator that returns the `rxnfile` format of a reaction that the search matched in the `rctab` column of `samplerx_reaction`. `rss` is a Direct operator that performs the reaction substructure search. The search parameters are contained in a reaction file (or `rxnfile`), `query.rxn`.

The diagram that follows shows how Direct is invoked when a user submits a SQL query from an Oracle application or from SQL\*Plus.

An example of using `Direct` within Oracle to access a reaction table:



The following list summarizes the flow of control that happens when a user executes a SQL query that uses Direct:

1. The user connects to Oracle from a client application that uses Oracle database drivers. Alternatively, the user connects to Oracle in a SQL\*Plus session. The application or the user submits a SQL statement that queries an Oracle table that contains molecule or reaction objects.
2. Oracle parses the SQL statement. When the SQL parser detects a Direct operator (such as `rss`), Oracle alerts the Oracle listener process.
3. The Oracle listener process receives the request, and spawns a session-specific Oracle process called `extproc`. The listener transfers control to `extproc`.
4. `extproc` loads the library that is registered with the cartridge operator that was detected in Step 2. `extproc` runs the Direct external procedure that is associated with the operator.
5. If there is no domain index on the column, or if Oracle decides not to use the domain index, Oracle provides molecule objects or reaction objects to the `extproc` process. It determines whether each record in the reaction table is a match to the query that was provided to the operator.
6. If a domain index exists and Oracle uses it, the `extproc` process performs a complete reaction substructure search on the reaction table.
7. The `extproc` process returns ROWIDs of the records that match the query to the Oracle server.
8. The Oracle server returns the requested data for the matching records back to the SQL interface.

For general information about Oracle data cartridges, see the Oracle documentation at <https://docs.oracle.com/database/121/ADDCI/index.html>.

## Direct Operators and Functions

Direct provides a set of BIOVIA-specific SQL operators and functions that are analogous to the Oracle SQL operators such as `SORT` and `LENGTH`.

This section lists Direct operators and functions by task. For details about how to use Direct operator and functions, see "*Cartridge Management Functions and Procedures*" in the *Direct Administration Guide*.

## Molecule-Related Tasks

### Search molecules

Use the following operators to search for molecule structures in a molecule table:

- flexmatch
- fmlalike
- fmlamatch
- similar
- sss

For an overview, see [Get Information about Structures](#).

### Search generic and specific structures

- Use the following operators to search for generic (markush) and specific structures in a generic molecule table:
- flexmatch
- sss
- overlap

For an overview, see Searching generic structures.

### Search biopolymers

Use the following operators to search for biopolymer molecule structures:

- flexmatch
- fmlalike
- fmlamatch
- sss
- sequencesearch

For an overview, see [Biopolymer Search and Registration](#).

### Retrieve molecules

Use the following operators to represent molecule structures:

- helm, mdlaux.helm
- molchime
- molfile
- molimage
- smiles, mdlaux.smiles
- iupcacname
- stringsegment
- writetempglob
- tempglob

For an overview, [Fetch Structures](#).

### Register molecules

Use the following operator to cast data types for registering or updating molecules in a molecule table:  
mol

For an overview, see [Insert, Update, and Delete Molecules](#).

### Highlight molecules

Use the following operators to highlight the query structures in the results of a substructure search:

- ssshight
- ssssequenceids

Use the following operator to orient target structures in the results of a Flexmatch search:

- flexmatchhighlight

### Retrieve similarity values

Use the following operator to get the similarity value from a similarity (similar) search:

- similarity

### Retrieve molecule weight and formula

Use the following operators to get the weight or formula of a molecule:

- molwt
- monoisotopicmass
- molfm1a
- isotopicformula

### Retrieve key values

Use the following operators to get the molecule key values:

- molkeys
- molnmakey, mdlaux.molnmakey
- mdlaux.rownmakey
- inchi, mdlaux.inchi
- inchikey, mdlaux.inchikey

## Reaction-Related Tasks

### Search reactions

Use the following operators to search reactions in a table:

- rss
- rxnflexmatch
- rxnsim

For an overview, see [Searching for Reactions](#).

### Retrieve reactions

Use the following operators to retrieve reactions from a table:

- rxnchime
- rxnfile
- rxnimage

- rxnsmiles
- stringssegment
- writetemplob
- templob

For an overview, see [Fetching Reactions](#).

### Register reactions

Use the following operator to cast data types for inserting or updating reactions in a table:

rxn

For an overview, see [Inserting, Updating, and Deleting Reactions](#).

### Highlight reactions

Use the following ancillary operator to fetch and highlight the query in the resulting reactions of a reaction substructure (rss) search:

rsshighlight

### Extract component molecules

Use the following operators to extract the reactant and product molecules from a reaction:

- rncomponents
- rxnmol

For an overview, see [Extracting Molecules from a Reaction](#).

### Automapp reactions

Use the following functions and operators to provide for automatic determination of reacting center bonds and atom-atom mapping in one or more reactions:

- rxnautomap
- rxnautomapchange
- rxnautomapstatus
- mdlaux.regenaamaps

For information about mdlaux.regenaamaps, see "Using the Automapper" in the *BIOVIA Direct Administration Guide*.

### Read and Write Files

Use the following operators to read and write files such as rxnfiles and molfiles:

- readfile
- readbinaryfile
- writefile
- writebinaryfile

For an overview, see [Accessing Files](#).

### Perform Administrative Tasks

Direct provides functions that perform administrative tasks such as setting the global Direct chemical environment, getting the current settings in the Direct chemical environment, and determining the number of molecule or reaction substructure search keys are pending inversion. For information about these functions, see the *Direct Administration Guide*.

### Perform Other Tasks

Direct provides the following functions to perform miscellaneous tasks such as displaying error messages and displaying product information:

- `mdlaux.errors`
- `mdlaux.version`

For overview of error handling, see [Checking Errors](#).

### Molecules and Reactions as Large Objects

Direct uses the following Oracle large object (LOB) data types to store and represent BIOVIA-specific formats of molecules and reactions:

- Binary large object (BLOB) data type - Stores and represents binary data. BIOVIA Direct uses the BLOB data type to:
  - Store the binary, packed representation of a molecule or reaction.
  - Specify a molecule or reaction object in a query.
- Character large object (CLOB) data type - Stores and represents data that is longer than the 4000-character limit on a variable-length string. Direct uses the CLOB data type to allow users to:
  - Retrieve large molecules and reactions as variable-length strings.
  - Specify a molecule or reaction string in a query.

The maximum length for a VARCHAR2 string is 4000 characters. Reaction file (`rxnfile`) representations of reactions typically exceed that length. Representations of molecules and Chime string representations of reactions also sometimes exceed that length.

If your client application does not support Oracle LOBs, Direct provides operators that allow your application to use string segments to access, insert, or update reactions in a database. For more details, see "[Fetching Reactions as String Segments](#)" and "[Copying String Segments into a Temporary CLOB](#)" in [Fetch Reactions](#).

### Molecule and Reaction Objects in SQL Statements

The LOB data types that represent molecules and reactions do not support ordinality and the standard relational operations. This implies that you cannot compare binary reaction and molecule objects, and you cannot use binary reaction and molecule objects in SQL clauses such as `DISTINCT`, `ORDER BY`, and `GROUP BY`. For example, the following SQL statements that use a reaction column are *not* valid:

```
--These SQL statements are invalid!  
select A.rxnmdlnumber  
from my_rxn_table A, samplerx_reaction B  
where A.rxn = B.rctab;  
  
select distinct rctab from samplerx_reaction;  
  
select rxnmdlnumber from samplerx_reaction order by rctab asc;
```

Additionally, reactions and molecules can be drawn in different ways. Because equivalent reactions or molecules can be drawn differently, it is possible for the same reaction or molecule to have many different binary representations. Therefore, you cannot use Oracle binary comparison functions such as `dbms_lob.compare` to compare reactions in a SQL statement.

To compare, search, and retrieve molecules and reactions in SQL statements, use the BIOVIA-specific SQL operators and functions that Direct provides.

## Temporary LOBs

The cartridge uses temporary LOBs in some of its operations. A LOB is an Oracle large object that is of character large object (CLOB) or binary large object (BLOB) data type. The cartridge operators and functions that return BLOB or CLOB data return temporary LOBs.

### Call-duration temporary LOBs

A call-duration temporary LOB is only available within the SQL statement that generated it. Some of the Direct operators and functions use call-duration temporary LOBs. These operators and functions create and return a new temporary LOB with each call. Use the call-duration temporary LOB within the same SQL statement that generated it. If you attempt to use the temporary LOB after the SQL statement has finished executing, the temporary LOB appears empty, or Oracle might return the following Oracle error:

```
ORA-22922: nonexistent LOB value
```

To use a call-duration temporary LOB outside the SQL statement that generated it, use the `templob` operator which returns a session-duration temporary LOB. You can also avoid this error by disabling the prefetch feature in the Oracle database driver that you use with your application. For example, in a Java application that uses JDBC:

```
((OracleConnection)conn).setDefaultRowPrefetch(1);
```

The following Direct operators and functions return call-duration temporary LOBs:

- `helm`
- `inchi`
- `isotopicformula`
- `makeclob`
- `mol`
- `molchime`
- `molfile`
- `molformula`
- `molimage`
- `readmol`
- `readbinaryfile`
- `rsshighlight`
- `rxn`
- `rxnautomap`
- `rxnchime`
- `rxnfile`
- `rxnimage`
- `rxnmol`
- `smiles`
- `ssshighlight`
- `mdlaux.automap`
- `mdlaux.chimeclob`

- `mdlaux.clobchime`
- `mdlaux.helm`
- `mdlaux.helmtomofile`
- `mdlaux.inchi`
- `mdlaux.iupacnametomofile`
- `mdlaux.molimage`
- `mdlaux.rsximage`
- `mdlaux.smiles`
- `mdlaux.smilestomofile`

### Freeing temporary LOBs

The temporary LOBs that are returned by Direct operators are of CALL duration. They occupy space in the temporary tablespace of the Oracle instance until they are freed. Oracle will automatically free temporary LOBs as they are consumed on the server in SQL\*Plus, PL/SQL, or server OCI programs. However, applications using client interfaces such as JDBC create additional temporary LOBs which must be explicitly freed. If the temporary LOBs are not explicitly freed, they will accumulate until the Oracle session is disconnected.

The following Java example frees the temporary LOB associated with the LOB locator object named `clob`:

```
if ( ((oracle.sql.CLOB)clob).isTemporary() ){
    ((oracle.sql.CLOB)clob).freeTemporary();
}
```

### Session-duration temporary LOBs

A session-duration temporary LOB is available during an Oracle session. Within an Oracle session, Direct provides five session-duration temporary LOBs, indexed from 0 to 4.

The following reaction cartridge operators use session-duration temporary LOBs:

- `stringsegment`
- `tempclob`
- `writetempclob`

For more information on these operators, see *BIOVIA Direct Reference Guide*.

If Oracle returns an error when you attempt to access a call-duration temporary CLOB that was returned by one of the Direct functions or operators (see [Call-duration temporary LOBs](#)), you can use `tempclob` or `writetempclob` to write the call-duration temporary CLOB into a session-duration temporary CLOB. For example:

```
select tempclob(rxnchime(rctab))
from samplerx_reaction where rxnmdlnumber='RXCI94058988';
```

In the preceding example, `rxnchime` returns a call-duration temporary CLOB, which `tempclob` writes into a session-duration temporary CLOB. As shown in this example, use `tempclob` only if your query returns a CLOB. If your query uses a CLOB as an input to an operator or function, or if your query returns a VARCHAR2 string instead of a CLOB, you do not need to use `tempclob`. The following example uses `stringsegment`, which returns a VARCHAR2 string:

```
select stringsegment(rxnchime(rctab))
from samplerx_reaction where rxnmdlnumber='RXCI94006733';
```

**IMPORTANT!**

- If you use the `templob` operator in an Oracle OCI or PL/SQL application, do not attempt to free the temporary LOB that `templob` returns. The `templob` operator does not create a new temporary LOB at each call. Instead, it uses or reuses one of the five LOBs that are resident on the cartridge.
- The use of the session-duration LOBs is incompatible with session pooling.

For more information about temporary LOBs, see the *Oracle Application Developer's Guide*.

## Molecule and Reaction Tables

You can use Direct on any table that contains a column of molecule objects. The molecule column must be of type BLOB. Similarly, Direct operates on any table that contains a column of reaction objects, which also must be of type BLOB.

The following example shows how to create a molecule table:

```
--Create molecule table
create table my_mol_table (
  mol_id   varchar2(30),
  mol_name varchar2(80),
  ctab     blob);
```

The following example shows how to create a reaction table:

```
--Create reaction table
create table my_rxn_table (
  rxn_id   varchar2(30),
  rxn_name varchar2(80),
  rctab    blob);
```

To efficiently search a molecule or reaction table using Direct operators, create a domain index on the table. See [Direct Domain Indexes](#).

For more information about creating the reaction tables, see "Creating Reaction Tables" in the *Direct Administration Guide*.

**Notes:**

Oracle provides a mechanism, Fine-Grained Access Control (FGAC), which applies filtering rules that prevent a user from seeing records in a table, based on that user's rights and some value in the row that specifies who can see the row. For more information about FGAC, see the *Oracle Application Developer's Guide - Fundamentals*. FGAC can be applied to tables that have reaction and molecule columns. However:

- Users should not try to apply FGAC to the tables that Direct creates as components of the domain index. (See [Direct Domain Indexes](#)).
- The owner of the molecule or reaction table should always have rights to see all records.
- Maintenance operations done on the domain index must be done by a user that has full rights to see all the records. (See [Direct Domain Indexes](#)).

## Direct Domain Indexes

Direct can use an Oracle domain index to enable efficient indexing methods for searching molecules or reactions. Note that the Direct does not require a domain index. If a domain index does not exist on a molecule or reaction table, Oracle compares each record in the molecule or reaction table with the query in order to find a match.

BIOVIA Direct uses two types of domain indexes:

- `c$direct2021.mxixml` for a molecule column
- `c$direct2021.rxixml` for a reaction column

A Direct domain index provides a search index on a column which contains molecule or reaction objects. This column must be of type BLOB.

The following example shows how to create a Direct domain index on a table that contains reaction objects:

```
--Create domain index on reaction table
create index mytable_idx on mytable(rctab)
  indextype is c$direct2021.rxixml
  parameters ('tablespace=bigspace');
```

For more details about creating the BIOVIA Direct domain indexes, see the *Direct Administration Guide > Creating and Managing Tables and Indexes*.

**Note:** After you create a Direct domain index, call the `mdlAux.errors` function to check if there is an error. Oracle might still create an invalid index even if there was an error. If `mdlAux.errors` returns an error, you must drop and recreate the index, if Oracle created it.

**See also**

[Direct Domain Index and the Oracle Optimizer](#)  
[Molecule and Reaction Tables](#)

### Direct Domain Index and the Oracle Optimizer

The Oracle optimizer might use the Direct domain index if it finds an indexed Direct operator in a SQL statement. An indexed 2021 operator (such as `rss` and `flexmatch`) is used for searching, and typically has an associated domain index. However, if the Oracle optimizer chooses not to use the Direct domain index, your SQL statement might execute more slowly than when the optimizer uses the domain index.

The Oracle command `EXPLAIN PLAN` explains how Oracle chooses to execute a SQL statement. The command also explains how optimal your SQL statement is. For SQL statements that involve more than one expression in the `WHERE` clause, Oracle hints might be needed to optimize the execution of the SQL statement.

If the Direct domain index appears later in the execution plan, you can use Oracle optimizer hints to control the access path for the Oracle optimizer, or to ensure that Oracle optimizer will use the domain index.

For more information about using Oracle hints, see the Oracle documentation for guidelines on tuning and optimization, and for details about using Oracle hints.

**Notes:**

- To improve performance, BIOVIA recommends that you apply statistics to any table that contains a Direct domain index. To apply statistics to a table containing a Direct domain index, use one of the following Oracle commands:

```
ANALYZE TABLE tablename ESTIMATE STATISTICS  
ANALYZE TABLE tablename COMPUTE STATISTICS
```

where *tablename* is the name of the table containing a Direct domain index.

- An invalid domain index can also affect performance.

See also

[Generating and Locking Table Statistics](#)

[Lack of a Valid Domain Index](#)

## Chapter 2:

# How to Use Direct

---

The following are examples of the different types of applications that can use Direct:

- Microsoft .NET
- COM Automation and ActiveX
- Web
- Java
- Oracle Call Interface (OCI)
- PL/SQL

To access Oracle data objects in your application, use the application programming interface of your chosen data access technology. Examples are the API for ActiveX Data Objects (ADO), Java Database Connectivity (JDBC), Oracle Call Interface (OCI), and Oracle Objects for OLE (OO4O).

For the usage and syntax of the Direct operators and functions, see the *BIOVIA Direct Reference Guide*.

To learn about how to execute SQL statements in an application, see the *Application Developer's Guide* from Oracle, or the reference documentation of your chosen data access technology.

## Get Information About Molecules

### Fetch Structures

Use Direct operators to fetch the molecules that match your search, either as CLOBs or as string segments.

#### Fetch structures as images

Use the following Direct operator that returns the image of a structure:

- `molimage` - Returns a BLOB that contains the image of a structure. The image can use either the PNG, BMP, SVG, or EMF format.

#### Fetch structures as CLOBs

If your client application supports CLOBs, use the following Direct operators that return structure CLOBs:

- `molchime` - Returns a CLOB representation of a Chime string.
- `molfile` - Returns a CLOB representation of a molfile.
- `ssshighlight` - Returns a CLOB representation of a Chime structure that matches a substructure query. The structure contains highlight information for the matched substructure.
- `flexmatchhighlight` - Returns a CLOB representation of a Chime structure that matches a Flexmatch query. The structure is oriented to the query.

#### Fetch structures as string segments

If your client application does not support CLOBs, use the following Direct operator that returns string segments of the structure:

- `stringsegment` - Returns a VARCHAR2 string that contains a 4000-character segment of a CLOB structure. For an example, see [Fetch reactions](#) > **Fetching Reactions as String Segments**.

### Copy string segments into a temporary CLOB

If your client application does not support CLOBs, you can write string segments to Oracle temporary CLOBs, and retrieve them later. Direct provides the following operators to write into and read from temporary CLOBs:

- `writetlob` - Writes string segments of the reaction or molecule to be inserted or updated into a temporary CLOB on the server.
- `templob` - Retrieves the temporary CLOB for inserting or updating.

The following ASP.NET example uses the `writetlob` operator to write string segments into a temporary CLOB, and calls the `templob` operator to return CLOB #0:

```
' Function to return a string representing a rxnfile or molfile suitable
'   for use in a SQL statement.
' qrxn is a Chime rxnfile or molfile.
'   If it is less than 4000 characters long, it can be used as-is;
'   in that case return the string enclosed in single quotes.
'   If longer than 4000 characters, use the operator WRITETEMPLOB to
'   construct a package-level temporary CLOB containing the qrxn and
'   return the operator TEMPLOB which when called by Oracle will
'   returnCLOB#0, containing qrxn.
' Uses global variable oCon As OdbcConnection.
Function setChimeOrTemplob(ByVal qrxn As String) As String
    Dim rxlen As Integer = Len(qrxn)
    If (rxlen <= 4000) Then
        setChimeOrTemplob = "'" & qrxn & "'"
    Else
        ' Write the string to temporary CLOB #0
        Dim oCmd As New OdbcCommand("select writetemplob(0,?,?) from
dual",oCon)
        oCmd.Parameters.Add("@STRING",OdbcType.VarChar)
        oCmd.Parameters.Add("@APPEND",OdbcType.Int)
        Dim append As Integer = 0
        Dim sstart As Integer = 1
        Do While (sstart <= rxlen)
            oCmd.Parameters("@STRING").Value = Mid(qrxn,sstart,4000)
            oCmd.Parameters("@APPEND").Value = append
            If oCmd.ExecuteScalar() <> 1 Then
                ' Should handle error if command did not execute properly
                ' and should run this in a Try block
            End If
            sstart += 4000
            append = 1
        Loop
        ' Get the string from the temporary CLOB #0
        setChimeOrTemplob = "templob(0)"
        oCmd.Dispose()
    End If
End Function
```

### Retrieve Related Structure Information

Direct provides operators that return related information about the structures that match your query. You can use the following operators in the SELECT clause of your SQL query:

- `molwt` - Returns the molecular weight.
- `mdl aux . molwt` - Returns the molecular weight.

**Note:** When inserting the molecular weight, you must use the `mdl aux . molwt` function instead of the `molwt` operator. The `mdl aux . molwt` function allows you to specify a molecule table or a molecule domain index that specifies which Ptable to use. The `molwt` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.

- `mol fml a` - Returns the molecular formula.

**Note:** `mol fml a` returns a CLOB; this operator cannot be used in a `DISTINCT`, `ORDER BY`, nor `GROUP BY` clause.

- `mdl aux . mol fml a` - Returns the molecular formula.

**Note:** When inserting the molecular formula, you must use the `mdl aux . mol fml a` function instead of the `mol fml a` operator. The `mdl aux . mol fml a` function allows you to specify a molecule table or a molecule domain index that specifies which Ptable to use. The `mol fml a` operator always uses the global Ptable, which is not appropriate for registration.

- `mono isotop cpmass` - Returns the mono-isotopic mass of a molecule.
- `helm` - Returns the HELM string for a given biopolymer sequence molecule. For details see [Get the HELM String](#).
- `isotopic formula` - Returns the molecular formula including isotope labels.
- `mol keys` - Returns the SSS key string which would be registered for a structure
- `isnostruct` - Indicates whether or not a structure has atoms ("no-structure")
- Operators and functions that generate NEMA keys.
- Operators and functions that generate InChI strings and keys.
- `smiles` - Returns the SMILES string for a given molecule. . For details, see [Getting the SMILES string](#).
- `molwtmin` - Returns the minimum molecular weight for a generic structure.
- `mdl aux . molwtmin` - Returns the minimum molecular weight for a generic structure.

**Note:** When inserting the minimum molecular weight, you must use the `mdl aux . molwtmin` function instead of the `molwtmin` operator. The `mdl aux . molwtmin` function allows you to specify a molecule table or a molecule domain index that specifies which ptable to use. The `molwtmin` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.

- `molwtmax` - Returns the maximum molecular weight for a generic structure.
- `mdl aux . molwtmax` - Returns the maximum molecular weight for a generic structure.

**Note:** when inserting the maximum molecular weight, you must use the `mdl aux . molwtmax` function instead of the `molwtmax` operator. The `mdl aux . molwtmax` function allows you to specify a molecule table or a molecule domain index that specifies which ptable to use. The `molwtmax` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.

- `isgeneric` - Indicates whether or not a structure is generic.

- `issequence` - Indicates whether or not a structure is a biopolymer sequence. (Its function equivalent is `mdlAux.issequence`.)
- `numspecifics` - Returns the number of specific structures which would be enumerated by a generic structure.
- `iupacname` - Returns the IUPAC name for a molecule.

The following SQL example returns the `cdbregno` and the molecular weight of a structure:

```
select cdbregno,  
molwt(ctab)  
from sample2d  
where cdbregno = 337;
```

For the usage and syntax of the different Direct operators and functions, see *BIOVIA Direct Reference*.

### Ancillary Operators for Structure Searches

Direct provides ancillary operators or functions that return related structure information from a structure search. Use an ancillary operator in the `SELECT` clause of the same query that used the Direct search operator. The ancillary functions take a single parameter, which is an arbitrary number. This number must match the appropriate parameter passed to the search operator. In the following example, the parameter for the `ssshighlight` operator matches the last parameter of the `sss` operator.

```
select ssshighlight(2)  
from sample2d  
where sss(ctab, 'c:\BIOVIA\Direct\testmolrxn\struct128.mol', 2)=1;
```

The following table lists the ancillary operators that are available with the different search operators.

Search Operator	Ancillary operators
<code>flexmatch</code>	<code>flexmatchtimeout</code> - Indicates if the flexmatch search timed out <code>flexmatchhighlight</code> - Returns an oriented Chime structure
<code>sss</code>	<code>ssshighlight</code> - Returns a highlighted Chime structure <code>ssstimeout</code> - Indicates if the substructure search timed out
<code>similar</code>	<code>similarity</code> - Returns the percentage of similarity between the query and the matched structure
<code>overlap</code>	<code>overlaptimeout</code> - Returns the timeout status of the overlap search. <code>pctoverlap</code> - Returns the percentage estimated overlap between the query and a hit.

### Get the InChI String and Key

Direct can generate IUPAC *standard* International Chemical Identifier (standard InChI™) strings and keys. The InChI key is a 27-character hashed form of the InChI string. BIOVIA functions generate the key by first generating the InChI string, and then calling an InChI library function to convert the string into the 27-character key. See <http://www.iupac.org/inchi> (or <http://old.iupac.org/inchi>) for more information about the standard InChI string and key.

Direct provides the following operators and functions that return InChI strings and keys for molecules:

- `inchi`
- `inchikey`

- `mdlaux.inchi`
- `mdlaux.inchikey`

### Limitations to the Generation of InChI Strings

Direct does not generate InChI strings for molecules with the following features. The InChI string or key will be returned as NULL, and a suitable error will be placed onto the error stack, if the molecule contains any of the following features:

- Query features
- Enhanced stereochemistry
- Polymer Sgroups
- Rgroups
- Rgroup atoms
- Pseudo-atoms
- Atoms with more than 20 neighbors
- Attachment points
- Multi-endpoint bonds

Molecules with data Sgroups are allowed to generate an InChI string, but the string will not contain any information about the data Sgroups and will thus not be structure differentiating.

No structure molecules generate a NULL InChI string.

The InChI library uses its own list of allowed atom types. Thus molecules which contain pseudo-atoms which are unknown to InChI (even though they exist in the Ptable used by the domain index) will generate a NULL return value.

### Get the SMILES String

SMILES™ (Simplified Molecular Input Line Entry System) is a language that represents molecules by using ASCII character strings that specify atoms and bonds. (For more information about SMILES, see <http://www.daylight.com/smiles/index.html>.) Direct provides the following operators and functions that return SMILES strings for molecules:

- `mdlaux.smiles`
- `smiles`

There are limitations to the generation of SMILES strings. Not all BIOVIA molecule features can be handled. For more information, see the Limitations to the generation of SMILES strings section that follows.

### Conversion of SMILES Strings to Molfile

Direct also provides the following function that converts a SMILES string to a molfile string:

```
mdlaux.smilestomolfile
```

Any function or operator which takes a general molecule argument also accept a SMILES string. When a SMILES string is used as an input parameter, an implicit conversion of the SMILES string to a molfile string occurs. Thus a SMILES string can be inserted into a table and used in a query.

The following example uses a SMILES string as the structure query parameter for the `sss` operator:

```
select cdbregno from sample2d where sss(ctab, 'B1=NB=NB=N1')=1;
```

The following example uses a SMILES string as the structure to be inserted. The `mol` operator converts the SMILES string to a BLOB.

```
SQL> create table moltable (id number, ctab blob);
Table created.
SQL> insert into moltable values (1, mol('Cn1cnc2c1c(=O)n(c(=O)n2C)C'));
1 row created.
SQL> select molfm1a(ctab) from moltable;
MOLFMLA(CTAB)
-----
C8 H10 N4 O2
```

### Limitations to the Generation of SMILES Strings

Direct does not generate SMILES strings for molecules with the following features. The SMILES string will be returned as NULL, and a suitable error will be placed onto the error stack, if the molecule contains any of the following features:

- Query features
- Enhanced stereochemistry
- Polymer Sgroups
- Rgroups
- Rgroup atoms
- Attachment points
- Multi-endpoint bonds

Molecules with data Sgroups are allowed to generate a SMILES string. However the string will not contain any information about the data Sgroups and will thus not be structure differentiating.

No structure molecules (that is, molecules with zero atoms) return a NULL value for the SMILES string.

Molecules with pseudo-atoms will generate a SMILES string where the pseudo-atoms are denoted by "[\*]".

### Navigate Structure Search Results

To navigate the search results of a structure search, use the methods of the application programming interface that you use to access Oracle data objects. Examples are the API for ActiveX Data Objects (ADO), Java Database Connectivity (JDBC), Oracle Call Interface (OCI), and Oracle Objects for OLE (OO4O). Another example is the use of the CURSOR construct in PL/SQL.

For information about how to use the application programming interface to navigate the result set, see the reference documentation of the database connectivity package you use, or the Oracle *Application Developer's Guide*.

### Save Structure Search Results

This section describes how to save the search results of a structure search.

#### Create a Search Results List

To create a list of ROWID or primary key values from the search results, save the results of your query in a temporary Oracle table. For example, in the example table SAMPLE2D, the primary key is CDBREGNO:

1. Create a table whose cdbregno numbers are populated from your query. For example:

```
create table temp_list as
  select cdbregno
  from sample2d
  where sss(ctab, '/home/users/molfiles/query.mol')=1;
```

2. Create an index for the new table, using the `cdbregno` field. For example:

```
create index temp_list_idx
  on temp_list(cdbregno);
```

3. Use the temporary table as a reference list in other searches. For example:

```
select a.cdbregno,
       molwt(a.ctab)
from sample2d a,
     temp_list b
where a.cdbregno = b.cdbregno
     and cdbregno < 100;
```

### Save Reactions to a Searchable Table

You can create a new Oracle table to store the reaction objects from the results of your search on a reaction table:

1. Create a table whose rows are populated from your query. For example:

```
create table temp_result as
  select *
  from samplerx_reaction
  where rss
        (rctab, '/opt/BIOVIA/direct2021/examples/rxnfilesquery.rxn')=1;
```

2. Create an index on the primary key column of the new table. For example:

```
create index temp_result_pkix
  on temp_result(rxnmdlnumber);
```

3. If you plan to search the reactions in the new table, and the table is large, create a reaction domain index for the new reaction column. For example:

```
create index temp_result_rxnidx
  on temp_result(rctab)
  indextype is c$direct2021.rxiidx;
```

**Note:** `c$direct2021.rxiidx` indicates that the index is a reaction domain index.

4. If you performed Step 3, call the `mdlaux.smiles` function to check if there is an error. Oracle might still create an invalid index even if there was an error. If `mdlaux.smiles` returns an error, you must drop and recreate the index, if Oracle created it.
5. If you performed Step 3, BIOVIA highly recommends that you apply statistics on this table for better performance. To apply statistics to any table to which you have applied a domain index, use the Oracle command:

```
ANALYZE TABLE tablename ESTIMATE STATISTICS
```

6. To test the results, select from the newly created table. For example:

```
select count(*)
from temp_result
where rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfilesquery.rxn')=1;
```

This should return the total number of rows in the temporary table.

### See also

[Generating and Locking Table Statistics](#)

## Insert, Update, and Delete Molecules

Using Direct, you can insert into, update, and delete molecule objects directly from an Oracle table that contains them. If the structure table does not have a domain index, Oracle simply performs the insert, update, or delete operation on this table. If the structure table has a domain index on the structure column, Oracle calls the related insert, update, or delete method of the domain index, and performs related operations that support the domain index.

### Notes:

- You must register insert and update structures of BLOB data type. For more details, see [Insert and Update Molecule Objects](#).
- Do not register structures that contain query features. A query feature is a restriction on a structure that specifies that a search will retrieve only certain types of structures from a database. An example of a structure that contains query features is a molfile structure that contains query atoms.
- The user that executes the SQL command must have the correct Oracle INSERT, UPDATE, or DELETE privilege in order to update data in the reaction table.
- Triggers must not reference the table for which the trigger fired. If a trigger or function attempts to access or modify a table that is being modified by the statement which fired the trigger, Oracle returns the following error:  
ORA-04091: table table\_name is mutating, trigger/function may not see it

For information about structural features that can or cannot be registered, and for information about duplicate structures, see "Registration of Molecules" in the *BIOVIA Chemical Representation Guide*.

## Insert and Update Molecule Objects

Direct stores binary, packed structures, or molecule objects, in a BLOB column in an Oracle table. To insert or update a molecule object in a structure table, the molecule that you use for the insert or update operation must be a BLOB. Direct provides the following operator for registration:

- `mol` - Converts a molfile or Chime string representation of a structure to a two-dimensional molecule object

The following example uses the `mol` operator to convert the contents of a molfile into a BLOB, and insert the BLOB into a molecule table:

```
insert into sample2d (
    ctab,
    corp_id,
    f_date
)
values (
    mol('C1CCCC1'),
    'MUSE00100452',
    to_date ('25-jul-2006')
);
```

**Note:** Although it is generally not advisable to register NULL structures, Direct allows the registration of NULL molecule objects. See [Null structures](#).

The following example uses the `mol` operator to convert into a BLOB the CLOB Chime structure selected from a different table,

and use the BLOB to update a specific molecule:

```
update sample2d
set ctab =
    mol(
        (select molchime(ctab)
         from temp_result
         where flexmatch(
             ctab,
             'c:\BIOVIA\direct2021\testmolrxn\struct100.mol', 'match=all')=1
        )
    )
where cdbregno=100;
```

**Note:** The SELECT query must return one structure.

Alternatively, you can also transfer molecule objects from one table to another without using the mol operator. Using the preceding example, you can also select the value of the ctab column directly, and use it in the update:

```
update sample2d
set ctab =
    (select ctab
     from temp_result
     where flexmatch(
         ctab,
         '/home/users/molfiles/struct100.mol',
         'match=all'
     )=1
    )
where cdbregno=100;
```

## Insert Related Structure Information

To insert related structure information, such as molecule weight and formula, into a structure table, insert the values returned by the following Direct functions:

- `mdlAux.molwt` - Returns the molecular weight. Note that when inserting the molecular weight, you must use the `mdlAux.molwt` function instead of the `molwt` operator. The `mdlAux.molwt` function allows you to specify a molecule table or molecule domain index that specifies which ptable to use. The `molwt` operator is not appropriate for registration (because it uses the global Ptable when the specified parameter is a molecule object).
- `mdlAux.molfm1a` - Returns the molecular formula. Note that when inserting the molecular formula, you must use the `mdlAux.molfm1a` function instead of the `molfm1a` operator. The `mdlAux.molfm1a` function allows you to specify a molecule table or molecule domain index that specifies which ptable to use. The `molfm1a` operator always uses the global Ptable which is not appropriate for registration.
- `mdlAux.molkeys` - Returns the SSS key string which would be registered for a structure. Note that when inserting the molecular formula, you must use the `mdlAux.molkeys` function instead of the `molkeys` operator. The `mdlAux.molkeys` function allows you to specify a molecule table or molecule domain index that specifies which key definitions to use. The `molkeys` operator always uses the global key definitions which are not appropriate for registration.
- `mdlAux.monisotopicmass` - Returns the mono-isotopic mass of a molecule.
- `mdlAux.isotopicformula` - Returns the molecular formula including isotope labels.

- `isnostruct` - Indicates whether or not a structure has atoms ("no-structure").
- `mdlAux.molname` - Returns the name of the molecule that is stored in the given molfile.
- `molfile` - Returns a molfile from a binary structure.
- `molchime` - Returns a Chime string from a binary structure.
- `inchi` - Returns the InChI string representation of a molecule.
- `inchikey` - Returns the InChI key for a molecule.
- `smiles` - Returns the SMILES string representation of a molecule.
- `mdlAux.molwtmin` - Returns the minimum molecular weight for a generic structure. Note that when inserting the minimum molecular weight, you must use the `mdlAux.molwtmin` function instead of the `molwtmin` operator. The `mdlAux.molwtmin` function allows you to specify a molecule table or a molecule domain index that specifies which ptable to use. The `molwtmin` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.
- `mdlAux.molwtmax` - Returns the maximum molecular weight for a generic structure. Note that when inserting the maximum molecular weight, you must use the `mdlAux.molwtmax` function instead of the `molwtmax` operator. The `mdlAux.molwtmax` function allows you to specify a molecule table or a molecule domain index that specifies which ptable to use. The `molwtmax` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.
- `mdlAux.isgeneric` - Indicates whether or not a structure is generic.
- `mdlAux.issequence` - Indicates whether or not a structure is a biopolymer sequence.
- `mdlAux.numspecifics` - Returns the number of specific structures which will be enumerated by a generic structure.
- `iupacname` - Returns the IUPAC name for a molecule.

The following example shows an INSERT statement that inserts a molecule along with related structure information into a structure table:

```
insert into
sample2d(ctab, molwt, molformula)
values(
    mol('/home/users/myfiles/struct123.mol'),
    mdlAux.molwt('sample2d', '/home/users/myfiles/struct123.mol'),
    mdlAux.molfm1a('sample2d', '/home/users/myfiles/struct123.mol'));
```

## Null Structures

BIOVIA does not recommend the use of NULL values for structures (NULL CTAB values). NULL is not a valid input for most Direct operators. The presence of NULL values may prevent the completion of searches that involve the CTAB column as a query. The convention for an omitted structure is to store a molecule with zero atoms. Users can prevent the registration of NULL CTAB values by applying a NOT NULL constraint to the CTAB column.

**Note:** Starting with version 6.0, Direct:

- Allows the registration of NULL structures. However, users should avoid registration of NULL structure.
- Does not return NULL structures as hits in any structure search.
- Does not allow a NULL value as the query structure.

For more information about converting a database to Direct, see the *BIOVIA Direct Administration Guide*.

### Using a No-structure

If you need to represent a NULL CTAB value, use a no-structure, or nostruct. A no-structure is a structure with zero atoms. To create a no-structure, select the **Chemistry > No Structure** menu item in BIOVIA Draw and save it as a molfile.

To use the no-structure in a query or for registration, use the readfile operator to read the no-structure molfile. If you use the no-structure frequently, you can create a single-row table that only contains the no-structure, which can either be a BLOB or a CLOB. The following example shows how to store the no-structure molfile into a single-row table with a CLOB molfile:

```
create table nostruct_table(ctab clob);
insert into nostruct_table(ctab)
  values(
    (select readfile('/home/users/nostruct.mol') from dual)
  );
commit;
grant select on nostruct_table to user;
```

You can select from this table to represent the no-structure in your query, for example:

```
select cdbregno
  from sample2d
 where flexmatch(
    ctab,
    (select ctab from nostruct_table),
    'all'
  )=1
    and cdbregno > 300;
```

### Propagate New Primary Key Value

Direct does not automatically generate any primary key or other user-column values in a molecule table. It is up to an application or trigger to do this. If the primary key of a molecule table is linked to other tables in the database, the application or trigger must propagate the primary key values to the related tables during structure registration.

Note that when a relational chemical (RCG) database is converted to Direct, the conversion program, convertrcg, provides the option to add triggers which automatically generate the CDBREGNO values. The rest of this section describes how to obtain the new CDBREGNO values that should be propagated into the related user tables. If your database is not converted from an RCG database and does not use the CDBREGNO field, you can use the information in this section as an example of how to get a newly inserted primary key value.

There are two ways to obtain a newly registered CDBREGNO into the related tables:

- Use :NEW.CDBREGNO in an AFTER INSERT trigger on the main table
- Use the RETURNING clause in an INSERT statement

**Note:** The molecule cartridge function cdcaux.getregnofrominsert is not available in Direct 2021. If your application used this function and a trigger that automatically inserts the CDBREGNO field to related tables, use one of the methods described in this section to get the newly generated CDBREGNO.

### To Use `:NEW.CDBREGNO` in an AFTER INSERT Trigger

To obtain the new CDBREGNO value that Direct generated in the last structure registration, you can use the `:NEW` trigger PL/SQL construct that provides a reference to the new value of a column.

To propagate the new CDBREGNO value from the main table to a related table, create an AFTER INSERT trigger on the main table. The following example is a trigger on the main table `mydb_moltable` that automatically inserts a newly inserted CDBREGNO into another table `mydb_alternate_names`:

```
CREATE OR REPLACE TRIGGER copyregno
AFTER INSERT ON mydb_mol
FOR EACH ROW
BEGIN
  -- Insert a record into mydb_alternate_names
  -- using the cdbregno from the newly inserted row
  -- on mydb_moltable.
  -- mydb_moltable contains the new CTAB structure.
  INSERT INTO mydb_alternate_names(cdbregno,name)
  VALUES (:NEW.cdbregno,'my new structure');
END;
```

### To Use the RETURNING Clause in an INSERT Statement

You can use the RETURNING clause of an INSERT statement to get the value of the newly generated CDBREGNO. Specify RETURNING CDBREGNO INTO :CDBREGNO\_VARIABLE to store the generated CDBREGNO into a bound variable. For example:

```
insert into molTable(molColumn)
values (mol(:molval))
returning cdbregno into :cdbregno
```

## Multi-user Registration and Locking

In versions prior to Direct 6.0, the entire molecule table was locked in exclusive mode during structure registration to prevent duplicate records from being inserted. This technique limited the rate at which records could be inserted, since only one process could insert records at a time. This also resulted in interactive users being able to halt all registration to a database by starting a transaction and then leaving the transaction open for long periods of time without issuing a COMMIT or ROLLBACK.

Starting with version 6.0, Direct locks with a finer degree of granularity, which allows many users to register at the same time with relatively few wait states. The locking mechanism alleviates both the multi-user throughput limitation and makes it much less likely that an interactive user with an open transaction will interfere with any other transaction. The lock identifier that is used to prevent duplicates from being registered is based on the structure's features. It is still possible for structures that are very similar to require the same lock; so very rarely, two processes might need the same lock at the same time, resulting in one process waiting for the other to COMMIT its transaction.

This also means that with Direct, it is possible for deadlock situations to occur. If two processes are each registering more than one structure in a single transaction, it is possible for both processes to need a lock that the other process already holds. This results in `ORA-00060: deadlock detected`, and one of the two processes will have its transaction rolled back. Applications should either COMMIT frequently enough to avoid deadlock conditions, or they should be written to roll back and retry a transaction that encounters an `ORA-00060` condition. Deadlock conditions should be relatively rare, since they require two transactions running at the same time that both want to register two pairs of structures that are similar to each other.

This locking mechanism applies when uniqueness checking has been turned on, and it applies for structures other than no-structures. No-structures are inserted without any duplicate check, since they are considered to be unique by definition.

## Biopolymer Search and Registration

### Overview

Direct provides the ability to store and search biopolymers. Biopolymers are potentially very large molecules with thousands of monomers. However, biopolymers are composed of a relatively small number of distinct monomer types. This allows for significant compression of the structure stored in the database and transmitted in molfile format between different programs, while still retaining the ability to elaborate the atoms and bonds in the full structure.

### Templates for Common Distinct Monomer Types

Direct defines a set of standard templates for common amino acid monomer types. This set of templates contains the twenty common natural amino acids, selenocysteine, and pyrrolysine.

Having templates available allows a peptide of 1000 naturally occurring amino acids to be stored with just 1000 template atoms, the bonds between the amino acids, and the template definitions needed for each of the distinct amino acids present in the peptide. However, BIOVIA chemistry functions can, when necessary, access the full molecular structure of the peptide by expanding template atoms into the full set of atoms and bonds. The expansion is only done as necessary, and is usually only done for one amino acid at a time.

### See also

[Store Monomer Representations](#)

### Template Atoms vs Normal Atoms

Template atoms differ from normal (non-template) atoms in that template atoms include:

- Attachment point information (left, right, first cross-link, second cross-link, and others)
- A sequence ID (residue) number
- A class name (for example, AA)

They are similar to abbreviation Sgroups, but provide for more efficient storage because each template connection table is stored only once in the molecule.

## Search Monomers in a Biopolymer Sequence

### Substructure Searching of Modified and Unmodified Monomers

Direct supports substructure searching (sss) of modified and unmodified monomers in a biopolymer sequence, with limitations described here.

Atom-by-atom substructure mapping of biopolymer, even when template-compressed, is not generally feasible due to the large number of non-template atoms and bonds which might be present. Thus Direct indexes (using Fastsearch and key screens) only the non-template atoms and bonds of monomers which are not present in the set of template definitions associated with the database. Those atoms and bonds may be contained in a template. If the template definition is not included in the database template definitions, the atoms and bonds will be indexed.

In addition to the indexing information for atoms and bonds in modified monomers, Direct also stores the text of the biopolymer sequence. The following types of substructure search are supported:

- Query contains no monomers (that is, template atoms) and only normal atoms and bonds. Examples are benzene, alanine drawn out as the full structure: The Fastsearch or inverted keys index is used to find all structures in the database which contain modified or cross-linked monomers containing the same atoms and bonds in the query, note that alanine drawn as N-C(-C)-COO will not match an unmodified alanine monomer.
- Query contains a mix of unmodified and modified monomers: The Fastsearch or inverted keys index is used to find all structures in the database which contain modified or cross-linked monomers matching those in the query, after those are mapped the unmodified monomers in the query are mapped onto the remainder of the target.
- Query contains only unmodified monomers: The sequence text for the query is generated (for example "LLLL") and the sequence text for each structure in the database is searched for text sequences which contain the query text. A query monomer may map to a modified target monomer if the target monomer has the same text value (letter).

The substructure search ancillary operator `ssssequenceids` returns the template or abbreviation Sgroup sequence ID values for residues which contain the query as a substructure.

### SSS for Molecules Produced by the UniProt Converter

The UniProt converter expands any monomers that are cross-linked, even if one or both are present in the standard set of template definitions. Thus SSS searches can be used to locate cross-linked residues when the molecules are produced by the Direct UniProt converter. For example, using an SSS query of benzene against a database of peptides will hit only those tyrosine-containing peptides where the tyrosine has been modified in some way, for example, the tyrosine was changed to phosphotyrosine. An SSS query of C-S-S-C will find all peptides containing disulfide cross-links. For details about the Direct UniProt converter, see "*UniProt Converter*" in the *Direct Administration Guide*.

### SSS Indexing of Template Atoms

The substructure indexing process includes a small number of the template atoms adjacent to monomers which are indexed due to modifications or cross-links. This means you can construct an SSS query which includes a few template atoms adjacent to the real chemistry being searched. For example, an SSS query of A1a-G1y-N-C(-C-S) would find cross-linked cysteines attached through their N terminus to glycine which is in turn attached to alanine.

#### Notes:

- A query containing only database template atoms will not select molecules which contain modified versions of any of those monomers. Conversely, an SSS query which is the full chemistry for an unmodified monomer will not select molecules containing the template atom version of that monomer. These are limitations of the substructure search algorithm.
- The substructure search ancillary operator `ssssequenceids` returns the template or abbreviation Sgroup sequence ID values for residues which contain the query as a substructure.

### Formula Searching of Biopolymers

Formula searching of biopolymers using the `fm1a1ike` and `fm1amatch` operators is supported. However, note that the search is for non-template atoms such as C, H, N, O, and S, not for residues such as A1a and Cys. Thus, these searches are not generally useful except in cases where the search is for an unusual element such as selenium.

### Similarity Searching of Biopolymers

Similarity searching of biopolymers using the `similar` operator is allowed, but the results are not useful because of the minimal number of substructure keys generated for biopolymers.

## Exact-match Searching of Biopolymers

An exact-match sequence search using the `flexmatch` operator is supported. For exact-match searches BIOVIA recommends using NEMA keys. For more information, see "Using NEMA Key Searching" in [Get Information About Molecules](#).

### Search Biopolymer Sequence Text

Biopolymer sequence text is stored in Direct for use during substructure searching with queries which contain only unmodified residues, as discussed above. The stored sequence text may also be searched with a text query using the `SEQUENCESEARCH` operator in Direct. It searches the text using either the Oracle `LIKE` operator or the Oracle `REGEXP_LIKE` function. For more information see "*sequencesearch*" in the *BIOVIA Direct Reference Guide*.

## Substructure Search of Modified Monomers

Direct supports substructure searching (SSS) of modified monomers in a biopolymer sequence.

Substructure searching of even the template-compressed biopolymer is not generally feasible due to the large number of non-template atoms and bonds which might be present. Thus Direct does not support substructure searching of unmodified monomers, that is, monomers which can be represented using one of the template definitions. Instead, Direct indexes only the non-template atoms and bonds of monomers which are not present in the set of template definitions associated with the database. Those atoms and bonds may be contained in a template. If the template definition is not included in the standard set of template definitions, the atoms and bonds will be indexed.

For example, a 200 residue peptide which contains all 20 natural amino acids as well as the modified amino acid d-alanine will contain:

- 20 template connection tables
- 199 special template atoms
- C,H,N atoms contained in the d-alanine residue

In this example, a search using a substructure query containing a chiral amino-acid fragment with a d-methyl rather than l-methyl would return the record as a hit because the atoms in the non-natural alanine residue are indexed. A benzene substructure query would not hit the record because none of the phenyl substituted amino acids would be indexed. They are all stored as template atoms. If the peptide contained disulfide crosslinks, the two cysteines on either side of each disulfide bond would also be indexed, and could be found with a substructure search.

**Note:** Data Sgroups are used to identify undocumented or ambiguous modifications. It is not recommended to routinely search for residues identified with data Sgroups. This type of search is relatively slow, and some modifications, for example N-glycosylation, are common in proteins. This can result in slow search times.

## Store Biopolymer Sequences

### Store Monomer Representations

Amino acid monomer units that are not modified are represented as single atoms within the molecule. Each of the distinct unmodified monomer types has a corresponding template which defines the chemical structure of the monomer, its attachment points, and its leaving groups. Templates are stored just once in the molfile or binary packed connection table, allowing even very large biopolymers to be represented in a relatively compact form.

Monomer units that have been modified (not in the standard set of template definitions) are stored as their full connection table (but may use abbreviation Sgroups so that a rendering of the molecule will show only a single- or three-letter abbreviation for the monomer).

For example, a 200 residue peptide which contains all 20 natural amino acids as well as the modified amino acid d-alanine will contain 20 template connection tables, 199 special template atoms, and the C,H,N atoms contained in the d-alanine residue.

Molecular weight and molecular formula will be computed taking into account the templates for all of the special template atoms. Note that the formula will contain only non-template atoms such as C, H, N, O, and S, not residue names such as Ala and Cys.

### Compress Sequence Molecules

Oracle provides a mechanism for compressing large objects. Sequence molecule connection tables are very large, and benefit from Oracle LOB compression. Using `SECUREFILE COMPRESS MEDIUM` provides about a two-fold compression of typical sequence molecule connection tables. `HIGH` compression provides very little additional gain, and requires more compute resources. There is no performance penalty to the compression. The performance is slightly better with compressed CTABs.

To create a table with LOB compression, use `SECUREFILE COMPRESS MEDIUM` as shown in the following example:

```
create table peptides
(uniprot_name varchar2(12),
 uniprot_accession_nr varchar2(6),
 ctab blob,
 sequence clob)
lob (ctab) store as securefile (compress medium);
```

### Get the HELM String

HELM, Hierarchical Editing Language for Macromolecules, is a format that can represent natural and modified biological sequences such as peptides, proteins, and nucleic acids, linked to each other and to small molecules to form complex structures.

The format specifications are described in "HELM: A Hierarchical Notation Language for Complex Biomolecule Structure Representation", Tianhong Zhang, Hongli Li, Hualin Xi, Robert V. Stanton, and Sergio H. Rotstein, *J. Chem. Inf. Model.* 2012, 52, 2796–2806.

Direct provides the following operators and functions that return HELM strings for molecules:

- `helm`
- `mdlaux.helm`

There are limitations to the generation of HELM strings. Only biopolymer sequence molecules that do not have modified residues can be converted into HELM strings.

**See also**

[Convert HELM strings to molfiles](#)

### Convert HELM Strings to molfiles

Direct provides the following function that converts a HELM string to a molfile string:

- `mdlaux.helmtomolfile`

Any function or operator which accepts a general molecule argument also accepts a HELM string. When a HELM string is used as an input parameter, an implicit conversion of the HELM string to a molfile string occurs. Thus a HELM string can be inserted into a table and used in a query.

The following example uses a SMILES string as the structure to be inserted. The `mol` operator converts the HELM string to a BLOB.

```
SQL> create table moltable (id number, ctab blob);
Table created.
SQL> insert into moltable values (1, mol(x));
1 row created.
```

## Get Information About Reactions

### Search for Reactions

To search for reactions in an Oracle table that contains reaction objects, you can use any of the following Direct operators in the `WHERE` clause of your SQL query:

- `rss` - Reaction substructure search
- `rxnflexmatch` - Reaction flexible match (`flexmatch`) search
- `rxnsim` - Reaction similarity search

For example, the following SQL query performs a reaction substructure search with a query `rxnfile`:

```
select rxnmdlnumber
from samplerx_reaction
where rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfilesquery.rxn')=1;
```

When you use the search operators `rss`, `rxnflexmatch`, and `rxnsim` in a `WHERE` clause, *always* test for a result of 1.

When you use the search operators `rss`, `rxnflexmatch`, and `rxnsim` in a `SELECT` clause, these operators return 1 for a match, and 0 for an unsuccessful match.

To negate the results of these search operators, use the SQL operator `NOT`. For example, to retrieve all structures that do not contain a certain structure:

```
select rxnmdlnumber
from samplerx_reaction
where not rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfilesquery.rxn')=1;
```

For the usage and syntax of the different Direct operators and functions, see *BIOVIA Direct Reference*.

For detailed information about the reaction `flexmatch` and substructure searches, see "*Exact Search (Flexmatch) and Reaction Substructure Search (RSS)*" in the *BIOVIA Chemical Representation Guide*. To get related information about the results of a reaction search, see "*Ancillary Operators for Reaction Searches*" in [Retrieve Related Reaction Information](#).

For information about improving performance, see [Performance Guidelines](#).

### Fetch Reactions

Use Direct operators to fetch the reactions that match your search, either as CLOBs or as string segments.

#### Fetch Reactions as Images

Use the following Direct operator that returns the image of a structure:

- `rxnimage` - Returns a BLOB that contains the image of a reaction. The image can use either the PNG, BMP, SVG, or EMF format.

### Fetch Reactions as CLOBs

If your client application supports CLOBs, use the following Direct operators that return reaction CLOBs:

- `rxnfile` - Returns a CLOB representation of a rxnfile.
- `rxnchime` - Returns a CLOB representation of a Chime string.
- `rsshighlight` - Returns a CLOB representation of a highlighted Chime string.

### Fetch Reactions as String Segments

If your client application does not support CLOBs, use the following Direct operator that returns string segments of the reaction:

- `stringsegment` - Returns a VARCHAR2 string that contains a 4000-character segment of an rxnfile or Chime string representation of the reaction.

The following Java code fetches a specific reaction in a table as string segments.

**Note:** Exception handling is omitted in this example.

```
//Format the SELECT SQL to fetch the first 4000-character
//segment of Chime string
String SQL =
    "select rxnmdlnumber, stringsegment(0,rxnchime(rctab)) " +
    "from samplerx_reaction where rxnmdlnumber='RXCI94058988'";
//Format another SELECT SQL to fetch the next 4000-character
//segment of Chime string
String fetchSQL = "select stringsegment(0) from dual";
Statement stmt = conn.createStatement();
//Execute the SQL to fetch the first 4000-character segment
ResultSet rset = stmt.executeQuery(SQL);
rset.next();
//Get the string content from the reaction column (column #2)
String RxnStr = rset.getString(2);
String s2;
//Execute the SQL to fetch the next segments,
//until nothing more to fetch
do {
    Statement ftmt = conn.createStatement();
    ResultSet fset = ftmt.executeQuery(fetchSQL);
    fset.next();
    s2 = fset.getString(1);
    if (s2 != null && s2.length() > 0) RxnStr = RxnStr + s2;
} while (s2 != null && s2.length() > 0);
```

If your client application does not support CLOBs, you can also write string segments to Oracle temporary CLOBs, and retrieve them later. See [Fetch structures](#) > **Copying string segments into a temporary CLOB**.

### Set the Default Row Prefetch

If your SQL query returns multiple rows that contain CLOB or BLOB reactions, Oracle might return the following error:

```
ORA-22922: nonexistent LOB value
```

In order to fix this error, your application can disable the prefetch feature in the Oracle database driver that you use with your application. For example, using the Oracle JDBC driver, you can disable prefetch to fetch one row at a time, as follows:

```
// Connect to database
conn = DriverManager.getConnection(
    jdbcUrl, userid, password);
// Prevent ORA-22922 by fetching one row at a time
((OracleConnection)conn).setDefaultRowPrefetch(1);
// Fetch clobs
rset = stmt.executeQuery(
    "select extreg, tempclob(1,rxnfile(rxn)) from rdc1");
while (rset.next()) {
    String extreg = rset.getString(1);
    //JDBC 2 supports Clobs - get the Clob
    CLOB clob = rset.getClob(2);
    String rxnfile = clob.getSubString(1,32000);
    //Free the LOB
    //Client interfaces such as JDBC require that the
    //temporary LOBs returned from BIOVIA Direct
    //must be explicitly freed.
    if ( ((oracle.sql.CLOB)clob).isTemporary() ){
        ((oracle.sql.CLOB)clob).freeTemporary();
    }
}
}
```

Another alternative is to select the ROWID (or other unique row identifier) from the reaction table, instead of selecting a CLOB or BLOB reaction. Then, use the unique identifier in a second SELECT statement to fetch one reaction object at a time.

## Retrieve Related Reaction Information

Direct provides operators that return related information about the reactions that match your query. You can use the following operators in the SELECT clause of your SQL query:

- rxnautomap - Returns a CLOB representation of an automapped rxnfile.
- rxnautomapchange - Returns the number of changes performed in the last automap operation.
- rxnautomapstatus - Returns the status of the last automap operation.
- ncomponents - Returns the number of reactant or product molecules of a reaction structure.
- rxnmol - Returns a CLOB representation of a reactant or product molecule from a reaction structure.
- rxnkeys - Returns the reaction key strings from a reaction object.
- hasnostructs - Indicates whether or not a reaction contains any component that has no atom (no-structure).
- rxnsmiles - Returns the string for a reaction. For more information, see the Getting the reaction SMILES string section that follows.

The following SQL example returns an automapped reaction, and the number of components in the product of a specific reaction:

```
select rxnautomap(rctab, 'default'),
       ncomponents(rctab, 2)
from samplerx_reaction
where rxnmdlnumber='RXCI94058988';
```

### Ancillary operators for reaction searches

Direct provides ancillary operators or functions that return related reaction information from a structure search. Use an ancillary operator in the SELECT clause of the same query that used the Direct search

operator. The ancillary functions take a single parameter, which is an arbitrary number. This number must match the appropriate parameter passed to the search operator. In the following example, the parameter for the `rsshighlight` operator matches the last parameter of the `rss` operator.

```
select rsshhighlight(2)
from samplerx_reaction
where rss(rctab, 'c:\BIOVIA\direct80\testmolrxn\query.rxn', 2)=1;
```

The following table lists the ancillary operators that are available with the different search operators.

Search Operator	Ancillary operators
<code>rxnflexmatch</code>	<code>rxnflexmatchtimeout</code> - Indicates if the reaction flexmatch search timed out.
<code>rss</code>	<code>rsshighlight</code> - Returns a highlighted Chime reaction. <code>rsstimeout</code> - Indicates if the reaction substructure search timed out.
<code>rxnsim</code>	<code>rxnmolsim</code> - Returns the percentage value of the molecule component similarity of a record that matches a reaction similarity search. <code>rxnctrsim</code> - Returns the percentage value of the reacting center similarity of a record that matches a reaction similarity search.

### Get the reaction SMILES string

Reaction SMILES is an extension to SMILES that represents reactions using concatenated molecule SMILES strings. Direct provides the following operators and functions that return SMILES strings for reactions:

- `mdlaux.rxnsmls`
- `rxnsmls`

There are limitations to the generation of the SMILES strings for molecules. The extension of SMILES to support reactions also has limitations, it:

- Cannot store reacting center bond information.
- Cannot distinguish between multiple fragments in a single reaction component, fragments are stored as if they are separate reaction components.

### Conversion of reaction SMILES strings to rxnfile

Direct also provides the following function that converts a reaction SMILES string to a rxnfile string: `mdlaux.smilestorsnfile`

For more information on `mdlrxn.smiles` and `mdlaux.smilestorsnfile`, see the *BIOVIA Direct Administration Guide > Command Reference*.

Any function or operator that takes a general reaction argument also accepts a reaction SMILES string. When a reaction SMILES string is used as an input parameter, an implicit conversion of the reaction SMILES string to a rxnfile string occurs. Thus, a reaction SMILES string can be inserted into a table and used in a query.

The following example uses a reaction SMILES string as the reaction query parameter for the `rss` operator:

```
select rxnmdlnumber from samplerx_reaction where rss(rctab,
'>>C1c1cccc1')=1;
```

**See also**[Work with Molecules in a Reaction](#)[Registration trigger on a reaction table](#)[Retrieve related structure information](#) > **Getting the SMILES string****Navigate Search Results**

To navigate the search results from Direct, use the methods of the application programming interface that you use to access Oracle data objects. Examples are the API for ActiveX Data Objects (ADO), Java Database Connectivity (JDBC), Oracle Call Interface (OCI), and Oracle Objects for OLE (OO4O). Another example is the use of the CURSOR construct in PL/SQL.

For information about how to use the application programming interface to navigate the result set, see the reference documentation of the databases connectivity package you use, or the *Oracle Application Developer's Guide*.

**Save the Search Results**

To create a list of ROWID or primary key values from the search results, save the results of your query in a temporary Oracle table. For example, in the example table `samplerx_reaction`, the primary key is `rxnmdlnumber`:

Create a table of primary key values from the results of your query. For example:

```
create table temp_list as
  select rxnmdlnumber
  from samplerx_reaction
  where rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfiles query.rxn')=1;
```

Create an index for the new table, using the primary key column. For example:

```
create index temp_list_idx
  on temp_list(rxnmdlnumber);
```

Use the temporary table as a reference list in other searches. For example:

```
select a.rxnmdlnumber,
       rxnctrsim(1) "Reacting Center Similarity",
       rxnmolsim(1) "Molecule Similarity"
from samplerx_reaction a,
     temp_list b
where a.rxnmdlnumber = b.rxnmdlnumber
     and
     rxnsim(a.rctab, '/opt/BIOVIA/direct2021/examples/rxnfiles query.rxn', '20
60', 1)=1;
```

**Save Reactions to a Searchable Table**

You can create a new Oracle table to store the reaction objects from the results of your search on a reaction table:

1. Create a table whose rows are populated from your query. For example:

```
create table temp_result as
  select *
  from samplerx_reaction
  where rss(rctab, '/opt/BIOVIA/Direct/examples/rxnfiles/query.rxn')=1;
```

2. Create an index on the primary key column of the new table. For example:

```
create index temp_result_pkix
  on temp_result(rxnmdlnumber);
```

3. If you plan to search the reactions in the new table, and the table is large, create a reaction domain index for the new reaction column. For example:

```
create index temp_result_rxnidx
  on temp_result(rctab)
  indextype is c$Direct.rxixmdl;
```

**Note:** `c$directx.rxixmdl` indicates that the index is a reaction domain index.

4. If you performed Step 3, call the `mdlaux.smiles` function to check if there is an error. Oracle might still create an invalid index even if there was an error. If `mdlaux.smiles` returns an error, you must drop and recreate the index, if Oracle created it.
5. If you performed Step 3, BIOVIA recommends that you apply statistics on this table for better performance. To apply statistics to any table to which you have applied a domain index, use the Oracle command:

```
ANALYZE TABLE tablename ESTIMATE STATISTICS
```

6. To test the results, select from the newly created table. For example:

```
select count(*)
  from temp_result
 where rss(rctab, '/opt/BIOVIA/Direct/examples/rxnfiles/query.rxn')=1;
```

This should return the total number of rows in the temporary table.

### See also

[Generate and Lock Table Statistics](#)

## Inserting, Updating, and Deleting Reactions

You can insert into, update, and delete reaction objects directly from an Oracle table that contains them. If the reaction table does not have a domain index, Oracle performs the insert, update, or delete operation on this table. If the reaction table has a domain index on the reaction column, Oracle calls the related method of the domain index, and performs related Direct operations that support the domain index.

When you use Direct to insert into, update, or delete from an Oracle table that contains reaction objects.

**Notes:**

- You must register and update reactions of BLOB data type. For more details, see the following section, Inserting and updating reaction objects.
- Do not register a reaction that contains query features. A query feature is a restriction on a structure that specifies that a reaction substructure search will retrieve only certain types of structures from a database. For example, a reaction contains query features if it has only a single molecule or product, or if it contains a molecule which contains query atoms.
- Direct does not support registration of reactions which contain only reactants or only products.
- The user that executes the SQL command must have the correct Oracle INSERT, UPDATE, or DELETE privilege in order to update data in the reaction table.
- Triggers must not reference the table for which the trigger fired. If a trigger or function attempts to access or modify a table that is being modified by the statement which fired the trigger, Oracle returns the following error:  
ORA-04091: table table\_name is mutating, trigger/function may not see it

For information about structural features that can or cannot be registered, see "*Registration of Reactions*" in the *BIOVIA Chemical Representation Guide*.

## Inserting and Updating Reaction Objects

Direct stores binary, packed reactions, or reaction objects, in a BLOB column in an Oracle table. To insert or update a reaction object in a reaction table, the reaction that you use for the insert or update operation must be a BLOB. Direct provides the rxn operator that converts a rxnfile or Chime string representation of a reaction to a BLOB reaction object. The rxn operator accepts a filename string, or the rxnfile or Chime string as a CLOB or VARCHAR2 string.

**Note:** Direct allows the registration of NULL reaction objects. See [NULL structures](#).

The following example uses the rxn operator to convert the contents of a rxnfile into a BLOB, and insert the BLOB into a reaction table:

```
insert into samplerx_reaction (
    rxnmdlnumber,
    rctab
)
values (
    'RXCI94058988',
    rxn('/opt/BIOVIA/Direct/examples/rxnfiles/reaction800.rxn')
);
```

The following example uses the rxn operator to convert into a BLOB the CLOB Chime reaction selected from a different table, and use the BLOB to update a specific reaction:

```
update samplerx_reaction
set rctab =
    rxn(
        (select rxnchime(rctab)
         from temp_result
         where rxnflexmatch(
             rctab,
             '/opt/BIOVIA/Direct/examples/rxnfiles/reaction100.rxn',
             'match=all'
```

```

    )=1
  )
)
where rxnmdlnumber='RXCI94058988';

```

**Note:** The SELECT query must return only one reaction.

You can also transfer reaction objects from one table to another without using the rxn operator. Using the preceding example, you can also select the value of the rxn column directly, and use it in the update:

```

update samplerx_reaction
set rctab =
  (select rctab
   from temp_result
   where rxnflexmatch(
    rctab,
    '/opt/BIOVIA/Direct/examples/rxnfiles/reaction100.rxn',
    'match=all'
   )=1
  )
where rxnmdlnumber='RXCI94058988';

```

**Note:** The SELECT query must return only one reaction.

## Inserting Related Reaction Information

To insert related structure information, such as reaction components, into a reaction table, insert the values returned by the following Direct functions:

- `mdlaux.rxnkeys` - Returns the RSS key string which would be registered for a structure.

**Note:** When inserting the RSS keys, you must use the `mdlaux.rxnkeys` function instead of the `rxnkeys` operator. The `mdlaux.rxnkeys` function allows you to specify a reaction table or reaction domain index that specifies which key definitions to use. The `rxnkeys` operator always uses the global key definitions which are not appropriate for registration.

- `rxnfile` - Returns an rxnfile from a binary reaction.
- `rxnchime` - Returns a Chime string from a binary reaction.
- `ncomponents` - Returns the number of reactants or products in a binary reaction.
- `rxnmol` - Returns a molfile CLOB for a molecule in a reaction.
- `mdlaux.hasnostructs` - Indicates whether or not a reaction contains a component with no atoms (no-structure).

## Multi-user Registration and Locking

During registration BIOVIA Direct locks the domain index with a fine degree of granularity that allows many users to register at the same time with fewer wait states. The locking mechanism increases multi-user throughput and makes interference with other transactions less probable. The lock identifier used to prevent the resitration of duplicates is based on the reaction's features. It is still possible for reactions that are very similar to require the same lock; so two processes might need the same lock at the same time, resulting in one process waiting for the other to COMMIT its transaction.

This means that deadlock situations can occur. If two processes are each registering more than one reaction in a single transaction, it is possible for both processes to need a lock that the other process already holds. This results in `ORA-00060: deadlock detected`, and one of the two processes will have its transaction rolled back. Applications should either COMMIT frequently enough to avoid deadlock conditions, or they should be written to roll back and retry a transaction that encounters an `ORA-00060` condition. Deadlock conditions are rare since they require two transactions running at the same time that both want to register two pairs of reactions that are similar.

The locking mechanism applies when uniqueness checking has been turned on, and applies for structures other than no-structures. No-structures are inserted without any duplicate check, because they are considered to be unique by definition.

## Accessing Files

Direct provides operators that allow read and write access to files in the operating system. When you use these operators in a SQL statement, Oracle uses a single operating system account to execute the Direct shared library. This operating system account is the user that owns the `extproc` listener. Because of this:

- Direct cannot use the attributes in the operating system environment of an application user, such as user profile and environment variables.
- The file name that is passed as an operand to the Direct operator, such as `readbinaryfile`, `readfile`, `writebinaryfile`, `writefile`, `sss`, `flexmatch`, `rss`, `rxnflexmatch`, and `rxnsim` must:
  - Include the full path name.
  - Not contain environment variables.
  - Be accessible to the operating system account that executes the Direct shared library.

For example, if your administrator used an operating system account called `MDLIRXN1` to start the Oracle `extproc` listener, Oracle uses the permissions of `MDLIRXN1` on the file that is being read by `readbinaryfile`, or being written to by `writebinaryfile`. On Windows, this operating system account is typically the Oracle account. If Direct is attempting to write a new file, Direct uses the permissions of `MDLIRXN1` in the specified location. On Linux, verify with the system administrator that the profile of the owner of the Oracle listener, for example `MDLIRXN1`, contains the following `umask` command:

```
umask u=rw,g=rw,o=rw
```

Similarly, the log file that is used for debugging Direct must be specified with a full path name, and must be writable by the operating system account that executes the Direct shared library.

For details about how to log debugging information from the Direct, see the *Direct Administration Guide > Logging Information*.

## Checking Errors

Direct creates and maintains a message stack for each Oracle session that uses Direct operators and functions. The first message in the stack is the first Direct informational, warning, or error message that the user encountered in an Oracle session. To display the contents of the message stack, use the Direct function `mdlAux.errors`. If there are no informational, warning, or error messages, the function returns NULL. Direct clears the stack after each invocation of `mdlAux.errors`.

If the SQL statement that uses Direct operators or functions returns an Oracle error, or if you have an Direct query that might take a long time to execute, call `mdlaux.errors` to check for Direct errors. Messages that contain the string `CTLIB(100)` are informational.

Because Direct clears the stack after displaying its contents, frequent calls to `mdlaux.errors` reduce the chances of overflowing the stack. If the stack overflows, you lose the last error message, and the last line in the stack contains the string `...more`.

For details about how to log debugging information from Direct, see the *Direct Administration Guide > Logging Information*.

## Working with Molecules in a Reaction

Direct provides the following operators that return information about the component molecules in a reaction:

- `ncomponents` - Returns the number of reactant or product components in a reaction.
- `rxnmol` - Returns a CLOB representation of a specific reactant or product molecule in a reaction.

### Extracting Molecules from a Reaction

The following PL/SQL example is a function that extracts a specific component molecule in a reaction from a table, and returns the first 4000 characters as a string.

```
CREATE OR REPLACE FUNCTION GetMol(whichmol NUMBER)
RETURN VARCHAR2
IS
  -- Declarations
  rxnval BLOB;
  molval CLOB;
  nreax NUMBER;
  nprod NUMBER;
  comptype NUMBER;
  compidx NUMBER;
  molfile CLOB;
  loblen INTEGER;
  amount BINARY_INTEGER;
  retval VARCHAR2(4000);
BEGIN
  -- Get a specific reaction
  SELECT rxn INTO rxnval FROM isisrx
  WHERE rxnregno = 100;
  retval := NULL;
  comptype := 0;
  compidx := 0;
  molval := NULL;
  -- Count the number of reactant and product molecules
  nreax := mdlaux.ncomponents(rxnval, mdlaux.reactant);
  nprod := mdlaux.ncomponents(rxnval, mdlaux.product);
  -- Determine the index of component molecule to get
  IF (whichmol <= nreax+nprod) THEN
    comptype := mdlaux.reactant; -- 1
    compidx := whichmol;
    IF (whichmol > nreax) THEN
      comptype := mdlaux.product; -- 2
```

```

    compidx := whichmol - nreax;
  END IF;
  --Get the molecule structure into a clob
  molval := mdlaux.rxnmol(rxnval, comptype, compidx);
END IF;
-- Copy the first 4000 chars of the CLOB to a string
molfile := molval;
IF (molfile IS NOT NULL) THEN
  loblen := dbms_lob.getlength(molfile);
  IF (loblen > 4000) THEN
    amount := 4000;
  ELSE
    amount := loblen;
  END IF;
  dbms_lob.read(molfile, amount, 1, retval);
END IF;
RETURN retval;
END;
/
SHOW ERRORS;

```

This is another PL/SQL example that accepts a reaction object, and populates a table with the component molecules:

```

DROP TABLE rxnmolstable;
CREATE TABLE rxnmolstable
  (id NUMBER(12), comptype NUMBER(1),
  compidx NUMBER(3), molval CLOB);
CREATE OR REPLACE PROCEDURE getrxnmols(idval NUMBER, rxnval BLOB)
IS
  comptype NUMBER;
  compidx NUMBER;
  molval CLOB;
BEGIN
  IF (rxnval IS NOT NULL) THEN
    -- Start with first reactant
    comptype := rdcaux.reactant;
    compidx := 1;
    LOOP
      -- Get molecule
      molval := mdlaux.rxnmol(rxnval, comptype, compidx);
      IF (molval IS NULL) THEN
        IF (comptype = mdlaux.reactant) THEN
          -- Done with reactants, switch to products
          comptype := mdlaux.product;
          compidx := 1;
        ELSE
          -- All done (or could be error, can't distinguish)
          EXIT;
        END IF;
      ELSE
        -- Insert comptype, compidx, and molval into table
        EXECUTE IMMEDIATE 'INSERT INTO RXNMOLSTABLE'||
          ' (ID, COMPTYP, COMPIDX, MOLVAL)'||

```

```

                ' VALUES (:1, :2, :3, :4)'
    USING idval, comptype, compidx, molval;
    -- Advance to the next reactant or product component
    compidx := compidx + 1;
  END IF;
END LOOP;
END IF;
END;
/
SHOW ERRORS

```

**Note:** You can execute the sample `getrxnmols` procedure by passing it the BLOB format of either a rxnfile or a Chime string. Use the package function `mdlaux.rxn` to convert the reaction to a BLOB. `mdlaux.rxn` is the package function name of the rxn operator. For example, to execute the sample `getrxnmols` procedure, enter the following command in SQL\*Plus:

```
execute getrxnmols(1, mdlaux.rxn('/tmp/test.rxn'));
```

## Inserting Component Molecules into a Molecule Table

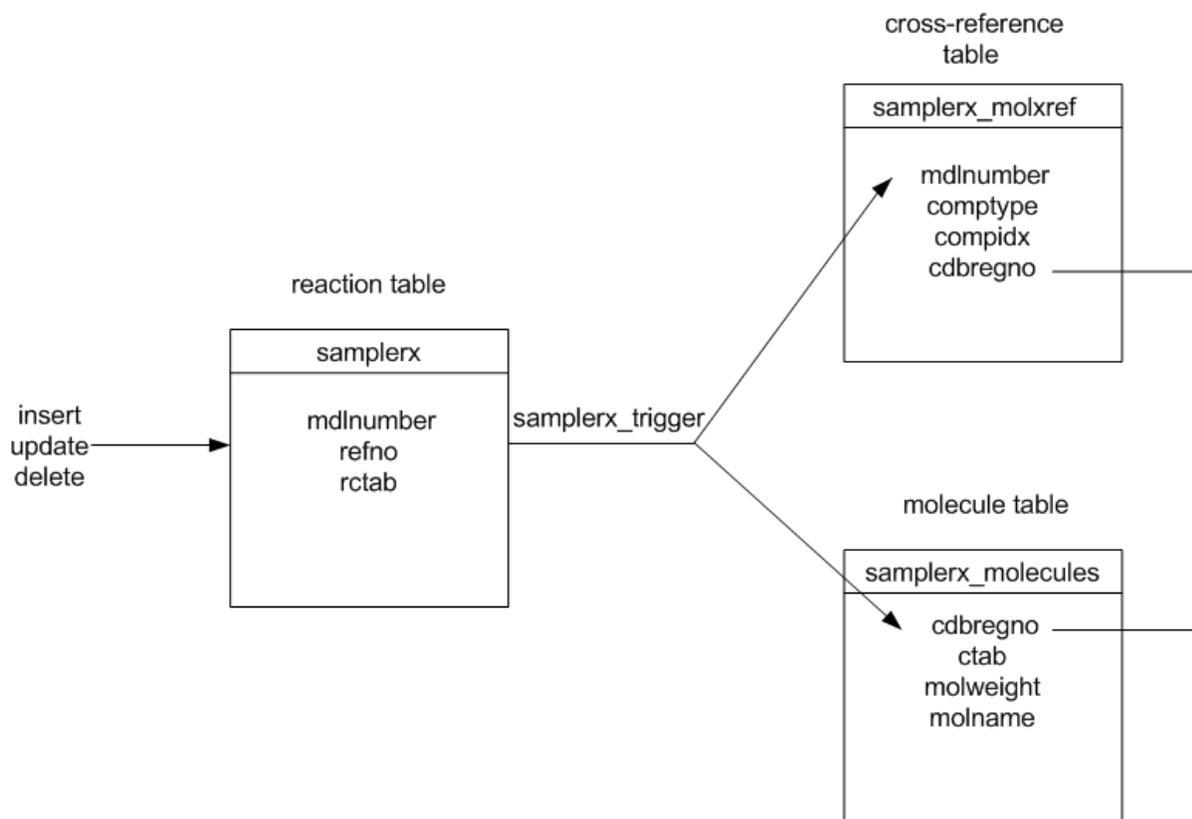
### Registration Trigger on a Reaction Table

The `rxnmol` operator is used in a registration trigger to extract the component molecules from a reaction that was inserted into a reaction table, and inserts the component molecules into a molecule table.

Direct provides a sample procedure, `MakeMolXrefTrigger`, that creates a sample trigger on a reaction table. The sample trigger:

- The sample trigger uses the `rxnmol` operator to extract the component molecules in a reaction.
- After an insert to the reaction table, the sample trigger inserts each component molecule into a molecule table.
- After an insert to the reaction table, the sample trigger inserts rows into a cross-reference table that associates the component molecules in the molecule table with the originating reaction in the reaction table. If the molecule already exists in the molecule table, the trigger uses the CDBREGNO of the existing molecule.
- After an update to the reaction table, the sample trigger deletes the rows from the cross-reference table that are associated with the current reaction, and inserts new rows into the cross-reference table and molecule table. The trigger does not delete the old molecules from the molecule table because they might be used by other reactions.
- After a delete from the reaction table, the sample trigger deletes the rows from the cross-reference table that are associated with the current reaction. The trigger does not delete the molecules from the molecule table because they might be used by other reactions.

The following diagram shows an example of a reaction table (`sample_rx_reaction`) and its trigger (`sample_rx_reaction_trigger`), a cross-reference table (`sample_rx_reaction_molxref`), and a molecule table (`sample_rx_reaction_molecules`), and shows how these are related to each other:



If you insert or update a reaction in a reaction table, and you want to create the sample trigger that automatically inserts or updates the component molecules in a molecule table, use the procedure `MakeMolXrefTrigger` that follows.

## Using MakeMolXrefTrigger

To use `MakeMolXrefTrigger`:

1. In SQL\*Plus, verify that the reaction table trigger you want to create does not already exist. For example:

```
select trigger_name from user_triggers
where trigger_name like 'MY_RXN_TABLE%';
```

where `MY_RXN_TABLE` is the name of your reaction table.

2. If the reaction table trigger already exists, drop it before proceeding. For example:

```
drop trigger my_rxn_table_trigger;
```

where `my_rxn_table_trigger` is the name of the existing trigger on your reaction table.

3. Verify that the molecule table which will contain the reaction components contains a `CDBREGNO` column and that the values in this column are automatically generated. For the `MakeMolXrefTrigger` procedure, see *BIOVIA Direct Reference > RDCAPPS Procedure > Requirements > MakeMolXrefTrigger*.
4. In SQL\*Plus, log in as the user who owns the reaction table. Create the RDCAPPS package that contains the `MakeMolXrefTrigger` procedure:  
`/opt/BIOVIA/direct2021/examples/rdcapps.sql`
5. where `/opt/BIOVIA/direct2021` installation.

6. Execute the `MakeMolXrefTrigger` procedure. For example:

```
SQL> call rdcapps.makemolxreftrigger('my_rxn_table',
                                     'rxnregno',
                                     'my_mol_table');
```

The example shown uses the default values for the names of the reaction table trigger (`MY_RXN_TABLE_TRIGGER`) and the cross-reference table (`MY_RXN_TABLE_MOLXREF`). To override the default values, see the complete syntax for `MakeMolXrefTrigger` in *BIOVIA Direct Reference*.

### Reaction Tables that Share a Molecule Table

If you have multiple reaction tables that share the same molecule table, create a trigger for each reaction table in order to automatically populate component molecules from all reaction tables into one table. If you want to use the sample trigger created by the `MakeMolXrefTrigger` procedure, each reaction column in each table needs its own trigger and cross-reference table. For each reaction table, execute the `MakeMolXrefTrigger` procedure. For example, execute the following commands for two reaction tables `rxn_table1` and `rxn_table2`, which both use the same molecules table `mol_table_mols`:

```
SQL> call rdcapps.makemolxreftrigger('rxn_table1',
                                     'rxnregno',
                                     'mol_table_mols');
SQL> call rdcapps.makemolxreftrigger('rxn_table2',
                                     'rxnregno',
                                     'mol_table_mols');
```

In this example, the two commands create: `RXN_TABLE1_TRIGGER` and `RXN_TABLE1_MOLXREF` for the table `rxn_table1`, and `RXN_TABLE2_TRIGGER` and `RXN_TABLE2_MOLXREF` for the table `rxn_table2`.

### Maintaining Referential Integrity

When a user inserts into, updates, or deletes from a reaction table, you can use a trigger on the reaction table to maintain referential integrity between the reaction and molecule tables.

The sample trigger that is created by the `MakeMolXrefTrigger` procedure maintains referential integrity by maintaining a cross-reference table that links the rows between the reaction table and the molecule table. The cross-reference table allows an application to:

- Display component molecules in a reaction that are also in a molecule table.
- Search for molecules in a molecule table, and find the reactions in the reaction table that contain them.

However, the sample trigger is only defined on the reaction table, and not the molecule table. If a user updates or deletes a molecule from the molecule table, the cross-reference table becomes incorrect. The referential integrity is no longer maintained between the reaction table and the molecule table. This implies that if you use the sample trigger that is created by the `MakeMolXrefTrigger` procedure, the users *must not* update or delete from the molecule table. The users *must only* insert into the molecule table.

To prevent users from updating or deleting molecules from a molecule table that is linked to a reaction table, you can create a trigger on the `CDBREGNO` or other primary key column of the molecule table. The molecule table trigger must:

- Check whether the primary key value exists in the cross-reference table, if a user attempts to update or delete a molecule from the molecule table.
- Return an error, if the primary key value exists in the cross-reference table.

## Homology Group Searching and Registration

### About Homology Groups

A homology group is a possibly ambiguous representation of a structural feature such as *alkyl group* or *acyclic group*. The groups are listed in [The Homology Group Hierarchy and Names](#).

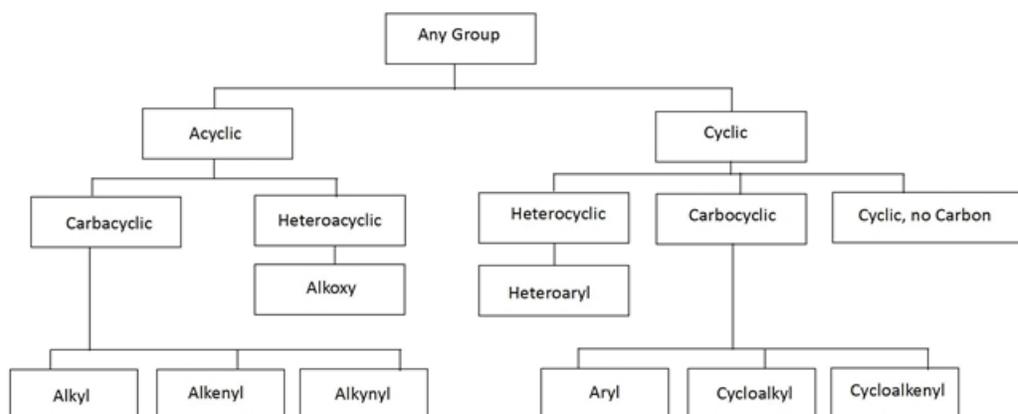
BIOVIA Direct supports representation and searching of homology groups. Direct:

- Allows a structure to contain a homology group with no underlying exact specification, by including a star atom with an abbreviation such as `A1ky1`.
- Searches for homology group features using substructure search (sss) where the query contains a star atom with an abbreviation such as `A1ky1`. The database can contain exactly defined molecules and/or molecules with incompletely specified homology groups as mentioned in the second bullet.

**Note:** Homology groups cannot be used with a database of generic (markush) molecules, they are limited to normal 2D fully specified molecules.

### The Homology Group Hierarchy and Names

Homology groups are hierarchically ordered. It starts with “Any Group” which represents any group, and becomes more specific according to the hierarchy shown in the following figure.



**Note:** The names shown in the previous figure are the exact names of the homology groups. If the group name has special characters such as spaces and commas, include them when specifying the name as an abbreviation label. For example, `Any Group` and `Cyclic, no Carbon`.

There are 16 distinct groups forming the hierarchy, with the first division being between `Cyclic` and `Acyclic` groups. The hierarchy is four levels deep, so substructures can belong to up to four separate groups. For example, a substructure or star atom that belongs to the `Alkyl` group is also a member of `Carbacyclic`, `Acyclic` and `Any Group`.

When created from exactly defined structures, groups consist of connected collections of atoms and bonds. They have one and only one crossing bond per group. The group must be all chains bonds or all ring bonds, combinations are not allowed. Single and double bond types refer to true single and double bonds, not to aromatic bonds (such as the alternating single and double bonds in benzene).

An atom connected to an `Acyclic` group in a query might or might not be part of that group in the target. For example, `C-Alkyl` (carbon bonded to an `Alkyl` group) should map onto chloroethane. This can be rationalized by having chloroethane contain two overlapping `Alkyl` groups, an ethyl and a

methyl group. Because groups are restricted to having only one crossing bond, this overlapping situation does not arise for Cyclic groups.

Homology Group Name	Homology Group Abbreviation	Chain or Ring	Allowed Atom Types	Allowed Bond Types
Alkyl	ALK	Chain	Carbon	Single
Alkenyl	AEL	Chain	Carbon	Single, double
Alkynyl	AYL	Chain	Carbon	Single, triple
Alkoxy	AOX	Chain	Carbon, oxygen	Single to oxygen, single, double, triple between carbon
Carbacyclic	ABC	Chain	Carbon	Single, double, triple
Heteroacyclic	AHC	Chain	Any (at least one non-carbon)	Single, double, triple
Aryl	ARY	Ring	Carbon	Aromatic, single, double, triple (at least one aromatic)
Cycloalkyl	CAL	Ring	Carbon	Single
Cycloalkenyl	CEL	Ring	Carbon	Single, double (at least on double)
Heteroaryl	HAR	Ring	Any (at least on non-carbon)	Aromatic, single, double, triple (at least one aromatic)
Carbocyclic	CBC	Ring	Carbon	Any
Heterocyclic	CHC	Ring	Any (at least one non-carbon)	Any
Cyclic, no carbon	CXX	Ring	Any except carbon	Any
Cyclic	CYC	Ring	Any	Any

## Registering Molecules and Homology Group Information

You can register molecules containing unspecified homology groups. These are indicated in the molecule as a star atom with an abbreviation name that matches one of the homology group names in the diagram shown previously.

**Note:** Although the homology group hierarchy is under Any Group, the Any Group name should **not** be used.

The procedure for adding homology groups to a molecule for registration is the same as when creating a query. See [Searching Structures with Homology Groups](#).

## Searching Structures with Homology Groups

Search for homology groups using a query structure that contains only the homology group name. Create the query structure in BIOVIA Draw by adding a star atom (the atom symbol is “\*”) at the point where you want the homology group, and then connecting to it an abbreviation label using one of the

homology group names shown in [Homology Group Searching](#) > **The Homology Group Hierarchy and Names** Section.

For more details about searching structures with homology groups, see *BIOVIA Chemical Representation Guide* > *Homology Groups*.

# Chapter 3:

## Performance Guidelines

---

### Guidelines Overview

BIOVIA Direct includes a set of functions that are called by Oracle to satisfy SQL queries. Oracle Corporation has been gradually improving the Oracle optimizer to make better choices, and BIOVIA has been gradually improving the capacity of Direct to provide the Oracle optimizer with the input it needs to make better decisions. However, applications developers must still compose SQL statements with considerable care and attention to performance issues in order to get consistent, acceptable performance from SQL queries.

Direct is only one component in the execution of SQL statements. It is essential for the application to write SQL statements that persuade the Oracle optimizer to use the Direct functions efficiently. Direct cannot compensate for performance problems that are outside of its control.

There are two implementations of each Direct search operator: Indexed implementation of a search operator and Non-indexed implementation of a search operator.

As a data cartridge, Direct uses the external procedure (extproc) interface to call Direct routines from within the Oracle database environment. Consequently, applications that use Direct need to manage the memory usage of extproc processes.

### Indexed Implementation of a Search Operator

The indexed implementation of a search operator generates a result set all at once, using the whole structure table as the potential candidates. The indexed implementation takes advantage of the BIOVIA search indexes, such as the fastsearch and flexmatch indexes. The Oracle optimizer chooses the indexed implementation of an operator when it believes that the indexed implementation would be faster than evaluating the candidates row by row, or when an INDEX hint is specified in the SQL statement.

### Non-indexed Implementation of a Search Operator

The non-indexed implementation of a search operator is chosen by the Oracle optimizer when:

- Oracle believes that the non-indexed implementation will be faster than the indexed implementation.
- A FULL table scan hint was supplied in the SQL.
- Using the indexed operator will violate one of the optimizer's rules that prohibit the use of the indexed operator.

### extproc Memory Usage

Each Oracle session that executes Direct has an associated extproc process. An extproc process provides the mechanism for Direct to operate directly within the Oracle environment. Oracle sessions that are created to execute Direct occupy approximately 40MB of virtual memory per user when active, and approximately 30MB per user when inactive. An inactive Oracle session is a session that continues to be connected to Oracle without submitting SQL statements that use Direct.

Applications that need to support large user workloads must restrict the number of Oracle sessions that are kept logged-in to a level that the available system memory and swapfile space can accommodate. These applications might need to implement techniques to conserve system resources. For information about managing Oracle resources, see the *Oracle Database Administrator's Guide*.

## Configuration Issues

The following configuration issues can degrade performance:

### Lack of a Valid Domain Index

The Oracle optimizer will not use indexed operators if the molecule or reaction column specified in the search does not have an appropriate domain index, or the domain index is not valid.

A domain index can become invalid if any of the underlying objects are modified or dropped. This can happen when a schema is restored from a backup during an Oracle upgrade, or when the Direct product is uninstalled and reinstalled. The domain index status can be fetched from the STATUS, DOMIDX\_STATUS, and DOMIDX\_OPSTATUS columns of the USER\_INDEXES view; the values should be 'VALID'.

Molecule and reaction columns must have the domain indexes created or recreated. To create or recreate the domain index, use the SQL commands DROP INDEX and CREATE INDEX.

The following SQL command tests for the existence and validity of the domain index on a molecule or reaction table:

```
SELECT STATUS,ITYP_OWNER FROM USER_INDEXES WHERE TABLE_NAME='tablename'
AND INDEX_TYPE='DOMAIN';
```

For example:

```
SQL> SELECT STATUS,ITYP_OWNER FROM USER_INDEXES WHERE TABLE_NAME='MOLTABLE'
AND INDEX_TYPE='DOMAIN';
STATUS      ITYPE_OWNER
-----
```

```
VALID      c$direct2021
```

If the domain index is *not* valid, it can be recreated using the SQL commands DROP INDEX and CREATE INDEX. For example:

```
SQL> DROP INDEX RXNINDEX;
Index dropped.
SQL> CREATE INDEX RXNINDEX ON RXNTABLE (RCTAB) INDEXTYPE IS
c$direct2021.RXIXMDL;
Index created.
```

### Lack of Valid Statistics

The Oracle optimizer might not use the [Indexed Implementation of a Search Operator](#) if the structure table lacks statistics, or if the statistics reflect a much smaller database. This can be corrected by performing the SQL operation ANALYZE TABLE *tablename* ESTIMATE STATISTICS on the table.

**See also**

[Generating and Locking Table Statistics](#)

### Schema Mismatch

The Oracle optimizer will not use the [Indexed Implementation of a Search Operator](#) if the operator is owned by a different schema than the domain index. This can happen after the installation of an updated version of Direct on a system that already has Direct.

The following SQL command can be used to display the name of the schema that owns the Direct synonyms:

```
SELECT TABLE_OWNER FROM USER_SYNONYMS WHERE SYNONYM_NAME='SSS';  
For example:  
SQL> select table_owner from user_synonyms where synonym_name='SSS';  
TABLE_OWNER  
-----  
c$direct2021
```

The following SQL command shows the owner of the Direct domain index:

```
SQL> SELECT ITYP_OWNER FROM USER_INDEXES WHERE TABLE_NAME='MOLTABLE' AND  
INDEX_TYPE='DOMAIN';  
ITYP_OWNER  
-----  
c$direct2021
```

If the synonyms are not owned by the correct schema, or if the table owner and index owner are different, the synonyms can be redefined by running the MDLAUXOP.SETUP function from the appropriate schema. For example:

```
SQL> EXECUTE c$direct2021.MDLAUXOP.SETUP  
PL/SQL procedure successfully completed.
```

### Lack of a Valid Fastsearch Index

When executing most substructure search queries, Direct uses the fastsearch index. This index significantly improves the performance of many types of queries. Verify that the Direct administrator maintains a valid fastsearch index.

### Oracle Cache Size

Direct stores reaction fastsearch index information in an Oracle table. The name of this table will always be the name of the domain index followed by `_FSIX`. For example, if the domain index name is `CIRX_MDLIX`, the name of the fastsearch table will be `CIRX_MDLIX_FSIX`.

This fastsearch index table can be quite large. On average, the table contains one to three kilobytes of information for each structure in the database. So, in order to obtain the maximum possible benefit from this index, you might want to do some tuning of the Oracle initialization parameters.

BIOVIA recommends that you increase the value of the appropriate Oracle caching parameter (`DB_CACHE_SIZE`, `DB_8K_CACHE_SIZE`, or `DB_KEEP_CACHE_SIZE`) by at least one-fourth of the size of the reaction fastsearch index table.

For example, if the current value of the `DB_CACHE_SIZE` parameter is 500 megabytes and the size of the reaction fast-search index table is 1000 megabytes, consider increasing the value of `DB_CACHE_SIZE` to 750 megabytes. System resource limitations might affect your ability to increase the cache size.

If the fastsearch table already exists, you can use a SQL statement to calculate the approximate size of that table. For example, the following statement computes the size of a reaction fastsearch table named `CIRX_MDLIX_FSIX`:

```
SELECT SUM(DBMS_LOB.GET_LENGTH(FSINDEX)) FROM CIRX_MDLIX_FSIX;
```

For additional information on Oracle tuning procedures, consult the Oracle documentation. For information about the reaction fastsearch index, see the *Direct Administration Guide > Maintaining Reaction Tables and Indexes*.

## Optimizing Queries

The following are guidelines for executing queries more efficiently and receiving results more quickly:

- [Writing Efficient Structure and Reaction Queries](#)

Writing efficient SQL statements can also result in a significant improvement in query performance. Oracle and Direct configuration can also affect query performance.

**See also**

[SQL Complexity](#)

[Checking the Execution Plan of a SQL Statement.](#)

[Performing an Incremental Search](#)

[Avoiding PRODUCT NOT REACTANT and REACTANT NOT PRODUCT Searches](#)

[Usage of the DISTINCT SQL Keyword](#)

[Temporary LOB Usage of Oracle temporary tablespace](#)

[Configuration Issues](#)

[Lack of a Valid Fastsearch Index](#)

[Oracle Cache Size](#)

## Efficient Structure and Reaction Queries

When writing chemical structure and reaction queries, BIOVIA strongly recommends that application users follow these guidelines, in order to ensure that their queries execute quickly:

- Always map reactants to products in your reaction query. A mapped query often executes much faster than an unmapped query. Because a mapped query can still return unmapped reactions, there is no disadvantage to submitting a mapped query.
- Make the query as specific as possible. If possible, add atom and bond query features to the query.
- Avoid writing queries that contain multiple fragments.

## Performing an Incremental Search

An application can perform a substructure (sss) or reaction substructure (rss) search in an *incremental* manner. When an application performs an incremental search, Direct runs the RSS or SSS portion of the query for approximately two seconds. It then stops executing the query and returns the results that it has obtained thus far. If and when the application requests additional records, Direct continues to execute the search (in increments of approximately two seconds).

**Note:** An incremental search is slightly slower than a comparable search that is not incremental. However, for many applications, the advantage of being able to display initial results to the application user quickly far outweighs the slight performance degradation.

To perform an incremental search, use the FIRST\_ROWS Oracle hint.

Here is an example of a SQL statement for an RSS search that uses the FIRST\_ROWS hint for this purpose:

```
SELECT /*+ FIRST_ROWS */ EXTREG
```

## Avoiding PRODUCT NOT REACTANT and REACTANT NOT PRODUCT Searches

The rss search operator provides the option to search a molecule as a reactant or product substructure. You can specify how the molecule should be searched in the context of reactions by specifying one of the following values in the third rss parameter:

- **REACTANT** - Finds reactions that contain the specified molecule as a substructure in one of the reactants.
- **PRODUCT** - Finds reactions that contain the specified molecule as a substructure in one of the products.
- **REACTANT NOT PRODUCT** - Finds reactions that contain the specified molecule as a substructure in one of the reactants, and do *not* contain this substructure in any of the products.
- **PRODUCT NOT REACTANT** - Finds reactions that contain the specified molecule as a substructure in one of the products, and do *not* contain this substructure in any of the reactants.

If performance is critical, it is generally not advisable to perform **PRODUCT NOT REACTANT** and **REACTANT NOT PRODUCT** reaction searches. An example of this type of search is `RSS(RCTAB, 'molfile', 'PRODUCT NOT REACTANT')`.

If a database has a separate table (or separate tables) that contains reactant and product molecule information, performing two SSS searches on these tables is almost always preferable. Two exceptions (cases in which performing a **PRODUCT NOT REACTANT** search or a **REACTANT NOT PRODUCT** search might be preferable) are:

- Databases in which the reactant and product molecule tables are unindexed.
- Applications for which converting the SSS results into reactions requires a huge amount of work.

However, if the database lacks a separate table for reactant and product molecule information, there is no practical alternative to performing the **PRODUCT NOT REACTANT** or **REACTANT NOT PRODUCT** search.

Here is an example of a SQL statement for a **PRODUCT NOT REACTANT** RSS search:

```
SELECT RXNMDLNUMBER
FROM SAMPLERX_REACTION
WHERE RSS(RCTAB, 'molfile', 'PRODUCT NOT REACTANT') = 1;
```

Here is an example of a SQL statement for an alternative to that search:

```
SELECT RXNMDLNUMBER
FROM SAMPLERX_REACTION
WHERE RXNMDLNUMBER IN
  (SELECT P.RXNMDLNUMBER
   FROM PRODUCT P,
        MOLECULE M
   WHERE P.MOLREGNO = M.MOLREGNO
        AND SSS(M.CTAB, 'molfile') = 1)
AND NOT RXNMDLNUMBER IN
  (SELECT R.RXNMDLNUMBER
   FROM REACTANT R,
        MOLECULE M
   WHERE R.MOLREGNO = M.MOLREGNO
        AND SSS(M.CTAB, 'molfile') = 1);
```

### Usage of the DISTINCT SQL Keyword

Oracle does not support the usage of the **DISTINCT** SQL keyword if a query retrieves any LOB values. For example, a query cannot include the **DISTINCT** keyword if it selects `MOLCHIME(CTAB)` or `RXNCHIME(CTAB)`. The return values from those operators are both CLOB values.

#### See also

[Molecule and Reaction Objects in SQL Statements](#)

## Temporary LOB Usage of Oracle Temporary Tablespace

The temporary LOBs that are returned by Direct operators are of CALL duration. Temporary LOBs occupy space in the Oracle instance's temporary tablespace until they are freed. If the temporary LOBs are not explicitly freed, they accumulate until the Oracle session is disconnected. Observe the following guidelines:

- The Oracle temporary tablespace must have enough storage to accommodate the temporary LOB activities of all users.
- Oracle will automatically free temporary LOBs as they are consumed on the server (in SQL\*Plus, PL/SQL, or server OCI programs). However, applications using client interfaces such as JDBC create additional temporary LOBs which must be explicitly freed.

The following Java example frees the temporary LOB associated with the LOB locator object named "clob":

```
if ( ((oracle.sql.CLOB)clob).isTemporary() ){
    ((oracle.sql.CLOB)clob).freeTemporary();
}
```

- Where practical, separate the generation of the result set from the selection of the data that the end-user is to see. Generate and fetch only those result rows that the end user really wants to see.

## Optimizer Hints

It might be necessary to supply Oracle optimizer hints to obtain good performance with SQL queries that contain Direct operators. See the Oracle documentation to determine which hints apply to domain index searches and which hints can be used in combination in the same SELECT clause.

**Note:** Always check the execution plan when you use Oracle hints in your query. Note that the optimizer can choose different execution paths according to the size of data and the speed of the machine. See [Checking the Execution Plan of a SQL Statement](#).

## FIRST\_ROWS

When used in conjunction with an SSS or RSS search, the FIRST\_ROWS Oracle hint directs Oracle to perform an *incremental* search. When an application requests an incremental search, Direct runs the SSS or RSS portion of the query for approximately two seconds. It then stops executing the query and returns the results that it has obtained so far. If and when the application requests additional records, Direct continues to execute the search in increments of approximately two seconds.

An incremental search is slightly slower than a comparable non-incremental search. The ability to quickly display initial results to the user outweighs the slight performance degradation. For example:

```
SELECT /*+ FIRST_ROWS */ EXTREG
FROM RXNTABLE
WHERE SSS(MOLCOL, '/home/user/query.mol') = 1;
```

## INDEX

The Oracle optimizer often chooses to execute non-indexed operators in situations where the indexed operator would be more efficient. The choices made by the Oracle optimizer change with each release of Oracle. These choices also depend on the `init.ora` settings and the quality of the statistics for the table.

You can force an indexed operation by specifying an INDEX hint that specifies both the table name and the domain index name. For example:

```
SELECT /*+INDEX(rxntable rxnindex)*/ EXTREG
FROM RXNTABLE
WHERE RSS(RCTAB, (SELECT QRY FROM QUERIES WHERE QID=1))=1;
```

### FULL

In cases where it is more efficient to process each candidate row, you can specify the FULL table scan hint. For example:

```
SELECT /*+FULL (RXNTABLE)*/ A.CDBREGNO
FROM SOURCE_DATA S,
     RXNTABLE A
WHERE S.IDENTIFIER='XXX'
     AND S.EXTREG=A.EXTREG
     AND RSS(RCTAB, (SELECT QRY FROM QUERIES WHERE QID=2))=1;
```

### ORDERED

In cases where you want to specifically control the order of execution of the WHERE clause, you can use the ORDERED hint. For example:

```
SELECT /*+ORDERED*/ A.EXTREG
FROM SOURCE_DATA S,
     RXNTABLE A
WHERE S.IDENTIFIER='XXX'
     AND S.EXTREG=A.EXTREG
     AND RSS(RCTAB, (SELECT QRY FROM QUERIES WHERE QID=2))=1;
```

## SQL Complexity

Keep individual SQL statements as simple as possible. Investigate alternatives to complex SQL statements, such as creating temporary tables of intermediate results. A long SQL statement that tries to perform all of the processing at once might not be better than several simpler SQL statements that accomplish portions of the task in ways that the Oracle optimizer can work well with.

### WHERE Clause Guidelines

- Reduce the result set as quickly as possible. That is, run the most specific portion of the query first.
- Touch the minimum number of rows possible.
- Use Direct indexed operators where practical. Additionally, use hints if necessary (see [Optimizer Hints](#)).
- Exploit any indexes on available table columns.
- Avoid the use of outer joins. Outer joins greatly multiply the number of rows that have to be processed, and might preclude the use of indexed operators.
- Where joins are necessary, join over stored columns, not expressions, and make sure that the columns are indexed. Check the output of EXPLAIN PLAN to ensure that there are no full table scans. Full table scans might indicate that an index is missing or invalid.

The Oracle data cartridge interface specifies a method for returning result rows that involves additional overhead. With result sets of up to a few thousand rows, this overhead is small. However, when processing hundreds of thousands of rows, the overhead of returning the result set to the Oracle data cartridge interface can become very significant. Search queries will perform poorly if they are so general that they return a large percentage of the database as hits.

## FROM Clause Guidelines

- Keep the FROM clause as simple as possible.
- Use table names and not expressions.
- Avoid using VIEWS in the FROM clause. Views might perform CPU-intensive formatting operations on many rows that will not be needed in the final result. Oracle cannot take advantage of the index on a column and so the performance advantage of having the index is lost when the index is ignored. View results are cached in an intermediate form, which can result in greater memory and I/O load on your system. Consider separating the generation of the result set of rows from the selection (and formatting) of the data values to be presented to the end-user.
- Never use an Direct indexed search operator on a view. Use the underlying table instead. The intermediate data caching of the view can greatly degrade the speed of the returning of the result rows from the indexed search operator to Oracle.
- Avoid nested SELECTs.

## SELECT List Guidelines

- Separate the generated results from the data selection. Generate and fetch the result rows that the end user requests.
- Separating lists is important when using client interfaces like JDBC and ODBC that cause LOB data such as rxnfiles and Chime strings to accumulate in the Oracle temporary tablespace until the cursor is closed. This behavior can cause the temporary tablespace to fill up. See [Temporary LOB Usage of Oracle Temporary Tablespace](#).
- Use the SELECT list to fetch data items, rather than to format them. Let the application format rows the end-user wants to view.

## Checking the Execution Plan of a SQL Statement

Oracle provides an EXPLAIN PLAN command used to display the execution plan of a SQL statement.

The EXPLAIN PLAN command requires a table named PLAN\_TABLE. Create this table in the schema before executing EXPLAIN PLAN. You can create the table by executing the `ut\lplan.sql` script, located in the `$ORACLE_HOME/rdbms/admin` directory.

You must embed the command to be explained in a script, as shown in the following example. Insert the command to be explained after the words EXPLAIN PLAN FOR.

```
set echo off
set feedback off
delete from PLAN_TABLE;
EXPLAIN PLAN FOR
select r.rxnregno, p.yield_min from cirx98 r,
cirx98_product p, cirx98_variation v where p.yield_min > 80 and
rss(r.rxn, '/work/rssqrys/gq5.rxn')=1 and r.rxnregno = v.rxnno and
v.rxnno = p.rxnno and v.varno = p.varno;
col operation format a30
col Options   format a20
col Object    format a20
select lpad(' ', 2*LEVEL) || OPERATION ||
       decode(ID, 0, ' Cost = '||POSITION) "Operation",
       OPTIONS "Options",
       OBJECT_NAME "Object"
```

```

from PLAN_TABLE
  connect by prior ID = PARENT_ID start with ID = 0
         order by ID
/

set feedback on
set echo on

```

The resulting output looks like this:

Operation	Options	Object
SELECT STATEMENT	Cost = 6	
TABLE ACCESS	BY INDEX ROWID	CIRX98_PRODUCT
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCESS	BY INDEX ROWID	CIRX98
DOMAIN INDEX		CIRX98RXNIX
TABLE ACCESS	BY INDEX ROWID	CIRX98_VARIATION
INDEX	RANGE SCAN	CIRX98_VARIATION_IX1
INDEX	RANGE SCAN	CIRX98_PRODUCT_IX1

The entry that says that the reaction table is being accessed via the DOMAIN INDEX indicates that an indexed search is being done. If the access had said anything else, then it would have been a non-indexed search. It is usually undesirable to perform non-indexed searches.

**Note:** The EXPLAIN PLAN command can give varying results on the same query, based on the size of data and the speed of the machine. For example, if you run the same query on a test versus a production database, the optimizer will choose different execution path if it thinks it is faster to perform a full table scan on a small table instead of using indexes on a large table. Also, if you run the same query on a test versus a production machine, the optimizer can choose different execution paths depending on the I/O bus speed and CPU speed of the machine.

## Generating and Locking Table Statistics

Oracle uses table statistics for query optimization. To improve structure query performance, generate statistics on the BIOVIA Direct molecule and reaction tables (and in general, any table your SQL statements will access). In addition, lock the table statistics because Oracle periodically updates the statistics unless the table is locked. If the table statistics are not locked and Oracle updates them, Oracle can set incorrect values for the LOB columns, which affects query performance.

For versions prior to Oracle 10, use the ANALYZE TABLE command. For Oracle 10 and later versions, use the DBMS\_STATS package. For example:

```

EXECUTE DBMS_STATS.UNLOCK_TABLE_STATS('schema_name', 'moltable_name');
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('schema_name', 'moltable_name');
EXECUTE DBMS_STATS.LOCK_TABLE_STATS('schema_name', 'moltable_name');

```

**Note:** For Oracle 10 and later versions, if you do not gather *and* lock table statistics, the AVG\_COL\_LEN value reported in the Oracle USER\_TAB\_COLUMNS view for the CTAB column of the molecule table *and/or* the AVG\_ROW\_LEN value reported in the Oracle USER\_TABLES view for the molecule table might become inaccurate.

For more information about using optimizer statistics, see the Oracle documentation.

# Chapter 4:

## Limitations on Resource Conservation Techniques

---

Different resource conservation techniques have been attempted with Direct to try to reduce the workload on the server. However there are limitations which cause some of these techniques to fail with Direct.

### Using Oracle Shared Server

Oracle's *Shared Server* feature improves the scalability of large workloads by reducing the number of processes that run on the server. This is accomplished by processing all user requests through a fixed set of Oracle shadow tasks. The traditional, dedicated server approach causes a new shadow task to be created for each user session. Since these processes occupy approximately 4MB to 10MB of virtual memory, the memory usage to support hundreds of connected users can be reduced by hundreds of megabytes.

According to Oracle Corporation, *Oracle Shared Server* is designed to be transparent to applications like Direct.

Oracle Shared Server uses a small number of shadow processes to service a queue of database access requests. With the traditional, dedicated server model, Oracle created one shadow process for each connected user, and that user had exclusive use of the shadow process. The shared server model reduces the memory load on the database server by using fewer shadow processes, while still maintaining each user's logical session (logon) to the database. With both approaches, the same number of extproc processes are created to service Direct user sessions.

#### **IMPORTANT!**

- On Windows systems, if you intend to use the POOL=ON setting, verify that you are running a version of Oracle that contains the fix for Oracle bug 2994216.
- Oracle Shared Server shadow processes may hang with undetected deadlocks when waiting to acquire locks using the SYS.DBMS\_LOCK package. The lock wait used by the DBMS\_LOCK package does not release the shadow process for use by other sessions until the lock is acquired. Direct administration functions use the DBMS\_LOCK package to prevent collisions.

### Client Connection and Session Pooling

Typical client (JDBC or OCI) session and connection pooling schemes require that all sessions created for a particular Oracle user be equivalent and that every SQL transaction be independent of the session history. These are required so that sessions can be redeployed at will and connections redeployed without consequence to the user activity. While this might generally be compatible with Direct, the user error stack is maintained at the session level. Errors from a previous use of the session could show up in a subsequent use, causing confusion.

### Parallel Processes

Oracle provides the capability to distribute the SQL processing among multiple parallel processes. The PARALLEL degree option for creating a table and the PARALLEL hint are intended to encourage the Oracle optimizer to break queries into pieces that can be run simultaneously in different threads.

**Note:** The PARALLEL feature does not apply to domain index operations. Oracle will serialize them; therefore the PARALLEL feature is of limited value when Direct operators are included. Oracle will not multithread the extproc, nor will it create multiple extproc processes to run the Direct portions of the query.

For more information about parallel processes, see the Oracle *Database Administrator's Guide* > *Managing Processes for Parallel Execution*.