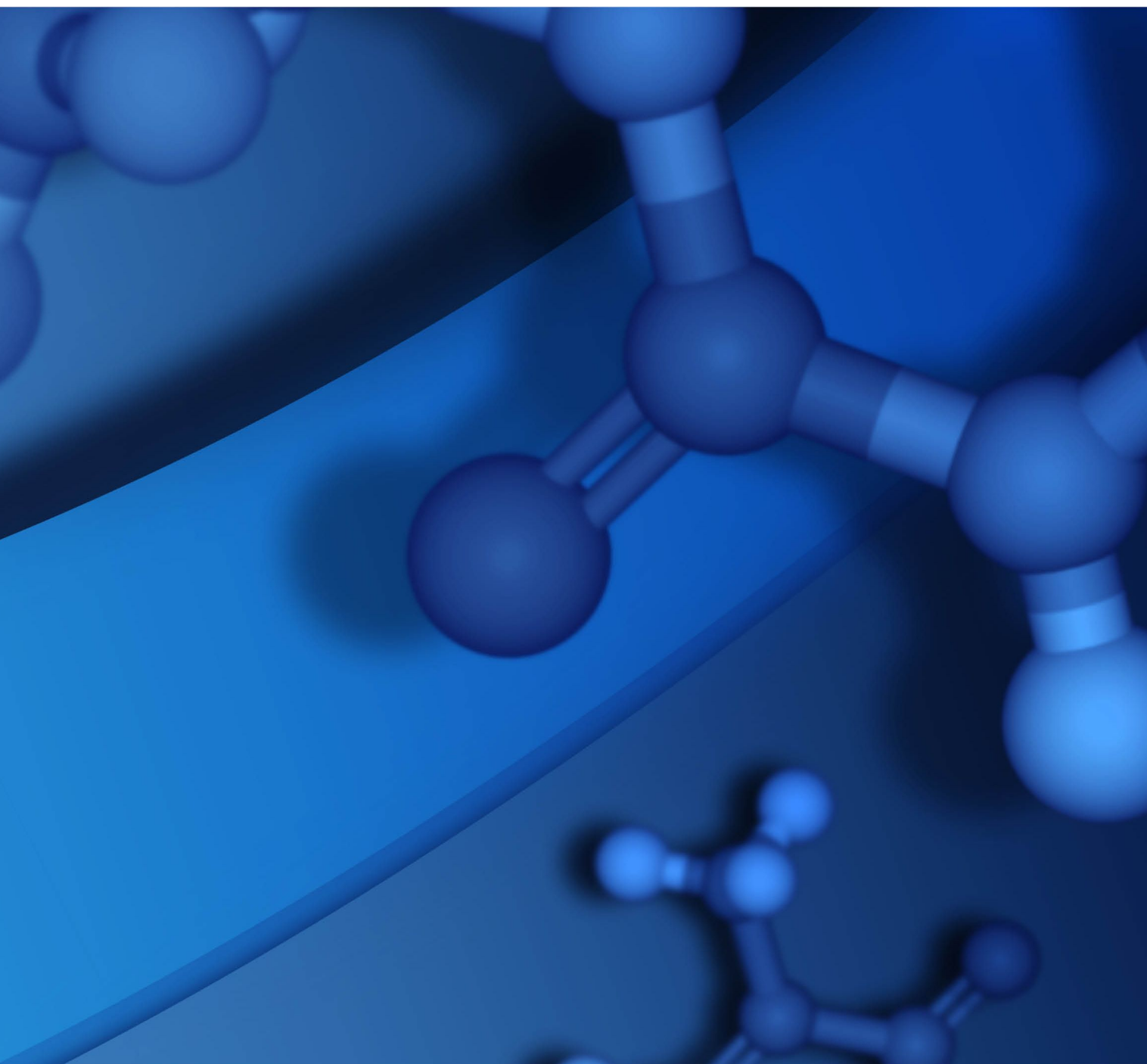


# REFERENCE GUIDE

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: Reference List, By Name</b>	<b>1</b>
<b>Chapter 2: General Operators and Functions</b>	<b>3</b>
General Operators	3
readbinaryfile	3
readfile	4
stringsegment	6
tempclob	9
writebinaryfile	10
writefile	12
writetempclob	13
General Functions	14
mdlaux.chimetoclob	15
mdlaux.clobtochime	16
mdlaux.clobtozip64	17
mdlaux.errors	17
mdlaux.externalcommand	18
mdlaux.gzip64toclob	24
mdlaux.readbinaryfile	24
mdlaux.readfile	24
mdlaux.stringsegment	24
mdlaux.tempclob	25
mdlaux.version	25
mdlaux.writebinaryfile	25
mdlaux.writefile	25
mdlaux.writetempclob	25
<b>Chapter 3: Molecule-Specific Operators and Functions</b>	<b>26</b>
Molecule-Specific Operators	26
chime	27
chime_string	28
chime_string_seg	28
flexmatch	29
flexmatchhighlight	33
flexmatchtimeout	34

fmla_eq .....	35
fmla_like .....	35
fmlalike .....	35
fmlamatch .....	37
helm .....	38
helm2 .....	40
inchi .....	41
Syntax .....	41
Return value .....	42
Usage .....	42
Example .....	42
Comments .....	42
inchiauxinfo .....	42
Syntax .....	43
Return value .....	43
Usage .....	43
Example .....	43
Comments .....	44
inchikey .....	44
isgeneric .....	45
isnostruct .....	46
isotopicformula .....	47
isrna .....	48
issequence .....	49
iupacname .....	50
makeclob .....	52
mol .....	53
molchime .....	54
molfile .....	55
molfile_string .....	56
molfile_string_seg .....	57
molfmla .....	58
molgzip64 .....	60
molimage .....	61
molkeys .....	63

molnemakey .....	64
molsim .....	67
molwt .....	69
molwtmax .....	70
molwtmin .....	71
molwtrange .....	72
monoisotopicmass .....	74
numspecifics .....	75
overlap .....	76
overlaptimeout .....	79
pctoverlap .....	80
readmol .....	81
sequencesearch .....	81
sequencetext .....	83
similar .....	83
similarity .....	86
smiles .....	86
sss .....	88
sss_highlight_chime .....	93
sss_highlight_molfile .....	94
ssshighlight .....	95
ssssequenceids .....	97
sssttimeout .....	98
writemol .....	99
xhelm .....	99
Molecule-Specific Functions .....	100
mdlaux.getsavedmolname .....	101
mdlaux.helm .....	102
mdlaux.helm2 .....	104
mdlaux.helmtomolfile .....	105
mdlaux.inchi .....	107
mdlaux.inchiauxinfo .....	109
mdlaux.inchikey .....	110
mdlaux.inchitomolfile .....	112
mdlaux.isgeneric .....	114

mdlaux.isnostruct .....	115
mdlaux.isotopicformula .....	115
mdlaux.isrna .....	117
mdlaux.issequence .....	118
mdlaux.iupacname .....	119
mdlaux.iupacnametomolfile .....	121
mdlaux.mol .....	122
mdlaux.molchime .....	122
mdlaux.molfile .....	122
mdlaux.molfmla .....	122
mdlaux.molimage .....	123
mdlaux.molkeys .....	125
mdlaux.molname .....	128
mdlaux.molnmakey .....	128
mdlaux.molwt .....	132
mdlaux.molwtmax .....	133
mdlaux.molwtmin .....	134
mdlaux.monoisotopicmass .....	135
mdlaux.numspecifics .....	137
mdlaux.rownmakey .....	138
mdlaux.sequencetext .....	139
mdlaux.setmolname .....	140
mdlaux.sgroupfields .....	141
mdlaux.smiles .....	142
mdlaux.smilestomolfile .....	143
mdlaux.xhelm .....	144
<b>Chapter 4: Reaction-Specific Operators and Functions .....</b>	<b>147</b>
Reaction-Specific Operators .....	147
hasnostructs .....	147
ncomponents .....	148
rinchi .....	149
rinchiauxinfo .....	151
rinchikey .....	153
rss .....	154
rsshighlight .....	160

rssttimeout .....	161
rxn .....	162
rxnautomap .....	165
rxnautomapchange .....	168
rxnautomapstatus .....	169
rxnchime .....	170
rxnctrsim .....	171
rxnfile .....	172
rxnflexmatch .....	173
rxnflexmatchtimeout .....	176
rxngzip64 .....	177
rxnimage .....	178
rxnkeys .....	180
rxnmol .....	182
rxnmolsim .....	183
rxnsim .....	184
rxnsmiles .....	189
rxnstringsegment .....	190
Reaction-Specific Functions .....	193
mdlaux.automap .....	193
mdlaux.hasnostructs .....	196
mdlaux.rinchi .....	196
mdlaux.rinchiauxinfo .....	198
mdlaux.rinchikey .....	200
mdlaux.rinchorxnfile .....	201
mdlaux.rxnimage .....	204
mdlaux.rxnkeys .....	206
mdlaux.rxnsmiles .....	209
mdlaux.smilestorxnfile .....	210
<b>Chapter 5: Examples .....</b>	<b>213</b>
Flexmatch Search .....	213
Substructure Search .....	215
Molecule Formula Search .....	217
Molecule Similarity Search .....	218
Reading a Molfile .....	219

Retrieving Molfile Structures .....	219
Retrieving Chime Structures .....	220
Structure Registration .....	220
Reaction Flexmatch Search .....	222
Reaction Substructure Search .....	223
Reaction Similarity Search .....	226
Writing a File .....	228
Fetching Reactions Using the Rxnfile Format .....	228
Reading a Rxnfile .....	230
Fetching Reactions Using the Chime Format .....	231
Reaction Registration .....	234
<b>Chapter 6: Molecule Searches .....</b>	<b>236</b>
Flexmatch Search .....	236
Flexmatch Search of Generic Structures .....	237
Flexmatch Search of Biopolymer Sequence Structures .....	237
Substructure Search .....	237
Substructure Search of Generic Structures .....	238
Substructure Search of Biopolymer Structures .....	238
Similarity Search .....	238
Types of Similarity .....	238
Molecule Formula Search .....	239
<b>Chapter 7: Reaction Searches .....</b>	<b>241</b>
Reaction Flexmatch Search .....	241
Reaction Substructure Search .....	242
Reaction Similarity Search .....	242
Types of Similarity .....	242
Degrees of Similarity .....	243
<b>Chapter 8: Specifying the Query Structure .....</b>	<b>244</b>
Molfile .....	244
Chimestring .....	245
BLOB (Binary Large Object) .....	246
CLOB (Character Large Object) .....	247
HELM String .....	247
SMILES String .....	247
<b>Appendix A: RDCAPPS Procedures .....</b>	<b>248</b>



Using the RDCAPPS Procedures .....	248
ReadRxnRDF .....	248
ReadMolRDF .....	250
MakeMolXrefTrigger .....	252



# Chapter 1:

## Reference List, By Name

---

The following is an alphabetic reference listing of all Direct functions and operators:

<a href="#">chime</a>	<a href="#">mdlaux.isotopicformula</a>	<a href="#">mdlaux.writefile</a>	<a href="#">rxnchime</a>
<a href="#">chime_string</a>	<a href="#">mdlaux.isrna</a>	<a href="#">mdlaux.writetemp</a>	<a href="#">rxnctrsim</a>
<a href="#">chime_string_seg</a>	<a href="#">mdlaux.issequence</a>	<a href="#">mdlaux.xhelm</a>	<a href="#">rxnfile</a>
<a href="#">flexmatch</a>	<a href="#">mdlaux.iupacname</a>	<a href="#">mol</a>	<a href="#">rxnflexmatch</a>
<a href="#">flexmatchhighlight</a>	<a href="#">mdlaux.iupacnametomolfile</a>	<a href="#">molchime</a>	<a href="#">rxnflexmatchtimeout</a>
<a href="#">flexmatchtimeout</a>	<a href="#">mdlaux.mol</a>	<a href="#">molfile</a>	<a href="#">rxngzip64</a>
<a href="#">fmla_eq</a>	<a href="#">mdlaux.molchime</a>	<a href="#">molfile_string</a>	<a href="#">rxnimage</a>
<a href="#">fmla_like</a>	<a href="#">mdlaux.molfile</a>	<a href="#">molfile_string_seg</a>	<a href="#">rxnkeys</a>
<a href="#">fmlalike</a>	<a href="#">mdlaux.molfmla</a>	<a href="#">molfmla</a>	<a href="#">rxnmol</a>
<a href="#">fmlamatch</a>	<a href="#">mdlaux.molimage</a>	<a href="#">molgzip64</a>	<a href="#">rxnmolsim</a>
<a href="#">hasnostructs</a>	<a href="#">mdlaux.molkeys</a>	<a href="#">molimage</a>	<a href="#">rxnsim</a>
<a href="#">helm</a>	<a href="#">mdlaux.molname</a>	<a href="#">molkeys</a>	<a href="#">rxnsmiles</a>
<a href="#">inchi</a>	<a href="#">mdlaux.molnemaakey</a>	<a href="#">molnemaakey</a>	<a href="#">rxnstringsegment</a>
<a href="#">inchikey</a>	<a href="#">mdlaux.molwt</a>	<a href="#">molsim</a>	<a href="#">sequencesearch</a>
<a href="#">isgeneric</a>	<a href="#">mdlaux.molwtmax</a>	<a href="#">molwt</a>	<a href="#">sequencetext</a>
<a href="#">isnostruct</a>	<a href="#">mdlaux.molwtmin</a>	<a href="#">molwtmax</a>	<a href="#">similar</a>
<a href="#">isotopicformula</a>	<a href="#">mdlaux.monoisotopicmass</a>	<a href="#">molwtmin</a>	<a href="#">similarity</a>
<a href="#">isrna</a>	<a href="#">mdlaux.numspecifics</a>	<a href="#">molwtrange</a>	<a href="#">smiles</a>
<a href="#">issequence</a>	<a href="#">mdlaux.readbinaryfile</a>	<a href="#">monoisotopicmass</a>	<a href="#">sss</a>
<a href="#">iupacname</a>	<a href="#">mdlaux.readfile</a>	<a href="#">ncomponents</a>	<a href="#">sss_highlight_chime</a>
<a href="#">makeclob</a>	<a href="#">mdlaux.rownemaakey</a>	<a href="#">numspecifics</a>	<a href="#">sss_highlight_molfile</a>
<a href="#">mdlaux.automap</a>	<a href="#">mdlaux.rxnimage</a>	<a href="#">overlap</a>	<a href="#">ssshighlight</a>

## Chapter 1: Reference List, By Name

<a href="#">mdlaux.chimetoclob</a>	<a href="#">mdlaux.rxnkeys</a>	<a href="#">overlaptimeout</a>	<a href="#">ssssequenceids</a>
<a href="#">mdlaux.clobtochime</a>	<a href="#">mdlaux.rxnsmiles</a>	<a href="#">pctoverlap</a>	<a href="#">sssttimeout</a>
<a href="#">mdlaux.clobtogzip64</a>	<a href="#">mdlaux.sequencetext</a>	<a href="#">readbinaryfile</a>	<a href="#">stringsegment</a>
<a href="#">mdlaux.errors</a>	<a href="#">mdlaux.setmolname</a>	<a href="#">readfile</a>	<a href="#">tempclob</a>
<a href="#">mdlaux.externalcommand</a>	<a href="#">mdlaux.sgroupfields</a>	<a href="#">readmol</a>	<a href="#">writebinaryfile</a>
<a href="#">mdlaux.getsavedmolname</a>	<a href="#">mdlaux.smiles</a>	<a href="#">rss</a>	<a href="#">writefile</a>
<a href="#">mdlaux.gzip64toclob</a>	<a href="#">mdlaux.smilestomolfi le</a>	<a href="#">rsshighlight</a>	<a href="#">writemol</a>
<a href="#">mdlaux.hasnostructs</a>	<a href="#">mdlaux.smilestorxnfi le</a>	<a href="#">rssttimeout</a>	<a href="#">writetempclob</a>
<a href="#">mdlaux.helm</a>	<a href="#">mdlaux.stringsegment</a>	<a href="#">rxn</a>	<a href="#">xhelm</a>
<a href="#">mdlaux.helmtomolfi le</a>	<a href="#">mdlaux.tempclob</a>	<a href="#">rxnautomap</a>	
<a href="#">mdlaux.inchi</a>	<a href="#">mdlaux.version</a>	<a href="#">rxnautomapchange</a>	
<a href="#">mdlaux.inchikey</a>	<a href="#">mdlaux.writebinaryfi le</a>	<a href="#">rxnautomapstatus</a>	
<a href="#">mdlaux.isgeneric</a>			
<a href="#">mdlaux.isnostruct</a>			

## Chapter 2:

# General Operators and Functions

---

This chapter contains the reference listings for the functions and operators that are not specific to molecules or reactions. Use the functions and operators in this chapter when working with molecules, reactions, or both.

## General Operators

All operators described in this chapter have corresponding package function names. Use the package function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators. The package function uses the same syntax as the operator.

For example, in a PL/SQL assignment, use the package function name for `readbinaryfile`, which is `mdlaux.readfile`:

```
query := mdlaux.readfile('/home/user/rxnfiles/rss1.rxn');
```

For all operators in this chapter, the corresponding package function name is **mdlaux**, followed by the function name.

### readbinaryfile

Copies the contents of a binary file on disk to a temporary BLOB.

#### Syntax

```
readbinaryfile(file)
```

Parameter	Description
<i>file</i>	A VARCHAR2 string that contains the full path and the name of the file. This file must: <ul style="list-style-type: none"><li>- Be readable by the extproc process started by Oracle</li><li>- Be located on the Oracle server. Note that this computer is not necessarily the same computer as the client.</li></ul>

#### Return value

A temporary BLOB that contains the contents of the specified file.

#### Usage

```
select dbms_lob.getlength(readbinaryfile('/disk-location/file-  
location/filename'))  
from dual;
```

Alternatively, in PL/SQL:

```
binaryvalue := mdlaux.readbinaryfile('/disk-location/file-  
location/filename');
```

#### Example

The following example stores a binary image file in a table:

```
insert into image_table(image_blob)
values(readbinaryfile('/home/user/myimage.png'));
```

The following example updates a binary image file in a table:

```
update moltable set imagefile = readbinaryfile
('/home/user/mol100.png')
where cdbregno = 100;
```

### Comments

- Specify the full path of the file to be read. Because Oracle uses a single operating system account to execute the Direct cartridge, it cannot use the attributes in the operating system environment of an application user, such as user profile, working directory, and environment variables. This means that you cannot use environment variables to specify the location of the file. The specified file must include the full path (and not a relative path).
- The file to be read must have correct permissions. Because Oracle uses a single operating system account to execute the Direct cartridge, this account must have correct permissions to read the specified file.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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 "blob":

```
if ( ((oracle.sql.BLOB)blob).isTemporary() ){
    ((oracle.sql.BLOB)blob).freeTemporary();
}
```

### See also

[mdlaux.readbinaryfile](#)  
[writebinaryfile](#)

*BIOVIA Direct Developers Guide > Using Direct > Accessing Files*

## readfile

Copies the contents of a file on disk to a temporary CLOB.

### Syntax

`readfile(file)`

Parameter	Description
<i>file</i>	A VARCHAR2 string that contains the full path and the name of the file. This file must: <ul style="list-style-type: none"><li>- Be readable by the extproc process started by Oracle</li><li>- Be located on the Oracle server. Note that this computer is not necessarily the same computer as the client.</li></ul>

### Return value

A CLOB that contains the contents of the specified file. The CLOB includes line-feed characters (0x0a) that separate the lines in the specified file. readfile stores the return value in a temporary CLOB.

**Synonyms**

readmol

**Usage**

```
select readfile('/disk-location/file-location/filename')
from dual;
```

Alternatively, in PL/SQL:

```
rxnvalue = mdlaux.readfile('/disk-location/file-
location/filename');
```

**Example**

The following example returns the contents of a file:

```
select readfile('/opt/BIOVIA/Direct/examples/rxnfiles/test.txt')
from dual;
```

The following PL/SQL example uses the package function name `mdlaux.readfile` to read the contents of a reaction file, and use it to perform a reaction substructure search:

```
DECLARE
  candidate BLOB;
  query CLOB;
  match NUMBER;
BEGIN
  candidate := mdlaux.rxn(
    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn');
  query := mdlaux.readfile(
    '/opt/BIOVIA/direct2021/examples/rxnfiles/query3.rxn');
  select rss(candidate,query) into match from dual;
END;
```

**Comments**

- Specify the full path of the file to be read. Because Oracle uses a single operating system account to execute the Direct cartridge, it cannot use the attributes in the operating system environment of an application user, such as user profile, working directory, and environment variables. This means that you cannot use environment variables to specify the location of the file. The specified file *must* include the full path (and not a relative path).
- The file to be read must have correct permissions. Because Oracle uses a single operating system account to execute the Direct cartridge, this account must have correct permissions to read the specified file.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**[mdlaux.readfile](#)[writefile](#)[writebinaryfile](#)[readbinaryfile](#)

BIOVIA Direct Developers Guide &gt; Using Direct &gt; Accessing Files

**stringsegment**

Returns a VARCHAR2 string that contains up to 4000 characters of a CLOB.

**Syntax**

```
stringsegment([tempclob-number,] clob)
stringsegment(tempclob-number)
```

Parameter	Description
<i>tempclob-number</i>	A NUMBER from 0 to 4 that specifies the temporary CLOB that stores the input clob. If you do not specify tempclob-number in your initial call to stringsegment, the default is 0. If the reaction contains more than 4000 characters, use stringsegment(0) to get the next portion of the CLOB .
<i>clob</i>	A CLOB that contains the string to be copied to the temporary CLOB. The stringsegment(clob) syntax is equivalent to stringsegment(0, clob). Be sure to use zero, stringsegment(0), when fetching subsequent segments.

**Return value**

The stringsegment([tempclob-number,] clob) syntax returns VARCHAR2 data that contains the first 4000 characters of a CLOB.

The stringsegment(tempclob-number) syntax returns VARCHAR2 data that contains the next 4000 characters. This operator will return NULL if there are no more characters left in the CLOB.

**Synonyms**

rxnstringsegment

**Usage**

```
select stringsegment(tempclob-number, clob)
      [, other-column-data]
from   tablename
where  condition;
select stringsegment(clob)
      [, other-column-data]
from   tablename
where  condition;
select stringsegment(tempclob-number)
from   dual;
```

**Example**

The following example uses stringsegment to get the first 4000 characters of the Chime representation of a reaction:



```
select stringsegment(1, rxnchime(rctab))
from samplerx_reaction
where rxnmdlnumber = 'RXCI94070168';
```

The next example uses `stringsegment` to return the next portion of the same Chime string from the preceding example. The Chime string is less than 4000 characters, so this statement returns a NULL:

```
select stringsegment(1)
from dual;
```

The following is another example that uses `stringsegment` to get the first 4000 characters of the rxnfile representation of a reaction. The default number for the temporary CLOB is 0 (zero):

```
select stringsegment(rxnfile(rctab))
from samplerx_reaction
where rxnmdlnumber = 'RXCI94070168';
```

The next example uses `stringsegment` to return the next portion of the same rxnfile. Note that the number for the temporary CLOB is 0 (zero):

```
select stringsegment(0)
from dual;
```

### Comments

- Use `stringsegment` if your application does not support CLOBs. If your application supports CLOBs, use an operator that returns CLOBs instead.
- The temporary CLOB is freed when the Oracle session is disconnected. If an application disconnects from Oracle before retrieving all data, the remaining data is lost. The application cannot retrieve the data in a subsequent session.
- Repeatedly call `stringsegment` to retrieve reactions or molecules that exceed 4000 characters.

The initial call must use the syntax:

```
stringsegment([tempclob-number], rxnclob)
```

The subsequent calls must use the following syntax. Repeatedly call `stringsegment` using this syntax until `stringsegment` returns NULL or a string whose length is less than 4000. Use the same `tempclob-number` that you used in the initial call.

```
stringsegment(tempclob-number)
```

- For the initial call to `stringsegment`, the *tempclob-number* parameter can be an arbitrary number from 0 to 4. For subsequent calls to `stringsegment`, the *tempclob-number* parameter must be the same as what you used in the initial call. If you did not specify a *tempclob-number* parameter in the initial call, the *tempclob-number* parameter in the subsequent call must be 0 (zero).
- Although it is possible to write a query that contains multiple calls to `stringsegment`, **BIOVIA does not recommend it**. Oracle does not necessarily call the operators in the order they appear in the SELECT statement. For example, do not call `stringsegment(tempclob-number, clob)` and `stringsegment(tempclob-number)` in a single SELECT statement, where *tempclob-number* is the same for both calls. The syntax `stringsegment(tempclob-number, clob)` saves the data in a temporary CLOB, and the syntax `stringsegment(tempclob-number)` reads the contents of the temporary CLOB. If Oracle calls `stringsegment(tempclob-number)` first, you will get the contents of an old temporary CLOB that might have been created in a separate operation.

It is more reliable to issue separate SELECT statements for each segment, as shown in the following example. Note that this example uses the default number 0 (zero) for the temporary CLOB:

```
DECLARE
  strseg VARCHAR2(4000);
  bigstr VARCHAR2(32000);
  numval NUMBER;
BEGIN
  SELECT STRINGSEGMENT(RXNFILE(rctab)) INTO bigstr
  FROM samplerx_reaction
  WHERE rxnmdlnumber = 'RXCI94070168';
  LOOP
    SELECT STRINGSEGMENT(0) INTO strseg FROM dual;
    IF strseg IS NULL THEN EXIT; END IF;
    bigstr := bigstr || strseg;
  END LOOP;
END;
```

- Use the same number to identify the same temporary CLOB in one Oracle session. The stringsegment operator shares the package-level, session-duration temporary CLOBs with the tempclob and writetempclob operators. Make sure that you use the correct, corresponding numbers in order to reference multiple temporary CLOBs within one Oracle session. The following example shows how corresponding numbers are used to reference multiple temporary CLOBs:

```
DECLARE
  str1 varchar2(4000);
  str2 varchar2(4000);
  bigstr1 varchar2(32000);
  bigstr2 varchar2(32000);
BEGIN
  --Fetch the first segments of the:
  --unhighlighted Chime string (temp CLOB #1)
  --highlighted Chime string (temp CLOB #2)
  select stringsegment(1, rxnchime(rctab)),
         stringsegment(2, rsshighlight(99))
  into bigstr1, bigstr2
  from samplerx_reaction
  where rss(rctab,
    '/opt/BIOVIA/direct/examples/rxnfiles/rssq1.rxn',
    99
  )=1;
  --Fetch the rest of the Chime strings:
  --unhighlighted Chime string (temp CLOB #1)
  --highlighted Chime string (temp CLOB #2)
  loop
    select stringsegment(1)
           stringsegment(2)
    into str1, str2 from dual;
    if str1 is NULL and str2 is NULL then exit; end if;
    bigstr1 := bigstr1 || str1;
    bigstr2 := bigstr2 || str2;
  end loop;
END;
```

**See also**[writetempclob](#)Examples of [Fetching Reactions Using the Rxnfile Format](#)Examples of [Fetching Reactions Using the Chime Format](#)**tempclob**

Creates a session-duration temporary CLOB. Alternatively, tempclob returns a session-duration temporary CLOB that was constructed by a previous call to [writetempclob](#) or tempclob operator.

**Syntax**

```
tempclob([tempclob-number,] clob)
```

```
tempclob(tempclob-number)
```

Parameter	Description
<i>tempclob-number</i>	A NUMBER from 0 to 4 that specifies the temporary CLOB that stores the input <i>clob</i> , or the temporary CLOB that was constructed by an earlier call to the <a href="#">writetempclob</a> or <a href="#">tempclob</a> operators. Specify only the <i>tempclob-number</i> to get the contents of a temporary CLOB that was constructed by an earlier call to the <a href="#">writetempclob</a> or <a href="#">tempclob</a> operators. If you do not specify <i>tempclob-number</i> , the default <i>tempclob-number</i> is 0.
<i>rxnclob</i>	A CLOB that contains the data to be copied to the temporary CLOB.

**Return value**

A session-duration temporary CLOB which is a copy of an input CLOB. The return value is one of the five session-duration temporary CLOBs, identified by a number from 0 to 4. The [tempclob](#) operator returns NULL if the *tempclob-number* is out of range.

**Usage**

```
select tempclob(tempclob-number, rxnclob)
from tablename
where condition;
```

```
select tempclob(rxnclob)
[, other-column-data]
from tablename
where condition;
```

```
select tempclob(tempclob-number)
from dual;
```

**Example**

The following example uses [tempclob](#) to create a session-duration temporary CLOB that contains a Chime representation of a reaction. The default number for the temporary CLOB is 0 (zero):

```
select tempclob(rxnchime(rctab))
from samplerx_reaction
where rxnmdlnumber = 'RXCI94070168';
```

The next example uses `tempclob` to return the content of the temporary CLOB that was created in the previous example. Note that the number for the temporary CLOB is 0 (zero):

```
select tempclob(0)
from dual;
```

The following is another example that uses `tempclob` to get the reaction that the `writetempclob` operator stored in a temporary CLOB, and uses this reaction in a reaction substructure ([rss](#)) query. The number for the temporary CLOB is 1:

```
select writetempclob(1, query-string, 0) from dual;
select writetempclob(1, next-query-string, 1) from dual;
select writetempclob(1, last-query-string, 1) from dual;
select rxnmdlnumber
from samplerx_reaction
where rss(rctab, tempclob(1))=1;
```

### Comments

- To store a CLOB reaction or molecule into a session-duration temporary CLOB, use the syntax:

```
tempclob(tempclob-number, rxnclob).
```

To get the contents of a session-duration temporary CLOB, use the syntax: `tempclob(tempclob-number)`. This temporary CLOB was constructed by an earlier call either to the `writetempclob` operator, or `tempclob(tempclob-number, rxnclob)`.

- Use `writetempclob` to write the temporary CLOB, then use `tempclob` to get the temporary CLOB. You can use the `tempclob` operator in conjunction with the `writetempclob` operator to specify a query in an application that cannot handle CLOBs. Use the `writetempclob` operator to construct the temporary CLOB, then use the `tempclob` operator to reference the temporary CLOB in the SELECT statement. For example:

```
select writetempclob(query-string, 0) from dual;
select writetempclob(next-query-string, 1) from dual;
select writetempclob(last-query-string, 1) from dual;
select rxnmdlnumber
from samplerx_reaction
where rss(rctab, tempclob(0))=1;
```

Use the same number to identify the same temporary CLOB in one Oracle session. Make sure that you reference the correct temporary CLOB numbers between these operators.

### See also

[writetempclob](#)

Examples of [Fetching Reactions Using the Rxnfile Format](#)

Examples of [Fetching Reactions Using the Chime Format](#)

*BIOVIA Direct Developers Guide > Using Direct > Copying String Segments into a Temporary CLOB*

## writebinaryfile

Copies BLOBdata to a file in a disk location.

### Syntax

```
writebinaryfile(blobdata, file)
```

Parameter	Description
<i>blobdata</i>	A BLOB to be written to <i>file</i> .
<i>file</i>	A VARCHAR2 string that contains the full path and the name of the output binary file. Use the appropriate file extension for the type of binary data to be written. Note that this file will be written on the Oracle server, which is not necessarily the same computer as the client.

**Return value**

The NUMBER 1 indicates that Direct wrote the file successfully

**Usage**

```
select writebinaryfile(
    blobdata,
    '/disk-location/file-location/filename'
)
from tablename
where condition;

select writebinaryfile(
    blobdata, '/disk-location/file-location/filename'
)
from dual;
```

**Example**

The following example uses `wri te binary fi le` to write a molecule image as a .PNG file. Note that it uses the `mol image` operator to get the BLOB image of a structure field.

```
select writebinaryfile(
    molimage(ctab),
    '/home/user/mol100.png')
from sample2d
where cdbregno = 100;
```

**Comments**

Because Oracle uses a single operating system account to execute the Direct cartridge:

- Specify the full path of the file to be written. Direct cannot use the attributes in the operating system environment of an application user, such as user profile, working directory, and environment variables. This means that you cannot use environment variables to specify the location of the file. The specified file *must* include the full path.
- The Direct account and the file location must have correct permissions. The operating system account that executes Direct must have correct permissions to write the file in the specified location. Additionally, the file or the specified location of the file must have the correct write permission (must allow anyone to write in it).

The file to be written will have read permissions. When the `wri te binary fi le` operator creates the file, it assigns permissions to the file so that it is readable by everyone.

If the specified file already exists, the `wri te binary fi le` operator replaces the existing file.

**See also**

[mdlaux.writebinaryfile](#)

[readfile](#)[readbinaryfile](#)

BIOVIA Direct Developers Guide &gt; Using Direct &gt; Accessing Files

## writefile

Copies CLOB or BLOB data to a file in a disk location.

### Syntax

```
writefile(data, file)
```

Parameter	Description
<i>data</i>	A CLOB or BLOB to be written to <i>file</i> . If data is a CLOB, the output file will be a text file. If data is a BLOB, the output file will be a binary file (equivalent to <a href="#">writebinaryfile</a> ).
<i>file</i>	A VARCHAR2 string that contains the full path and the name of the output file. Note that this file will be written on the Oracle server, which is not necessarily the same computer as the client.

### Return value

The NUMBER 1 indicates that Direct wrote the file successfully

### Usage

```
select writefile(  
    data, '/disk-location/file-location/filename.rxn'  
)  
from tablename  
where condition;
```

```
select writefile(  
    data, '/disk-location/file-location/filename'  
)  
from dual;
```

### Example

The following example uses `writefile` to write a structure into

```
/opt/BIOVIA/direct/examples/rxnfiles/regno100.rxn.  
select writefile(  
    rxnfile(rctab),  
    '/opt/BIOVIA/direct/examples/rxnfiles/regno100.rxn')  
from samplerx_reaction  
where rxnmdlnumber = 'RXCI94070168';
```

The following example uses `writefile` to write the contents of a temporary CLOB to a file.

```
select writefile(  
    tempclob(1),  
    '/opt/BIOVIA/direct/examples/rxnfiles/output.txt')  
from dual;
```

The next example writes a group of molfiles:

```
select cdbregno,
writefile(molfile(ctab),
'/home/user/mol' || TO_CHAR(cdbregno) || '.mol')
from sample2d;
```

### Comments

Because Oracle uses a single operating system account to execute the Direct cartridge:

- Specify the full path of the file to be written. Direct cannot use the attributes in the operating system environment of an application user, such as user profile, working directory, and environment variables. This means that you cannot use environment variables to specify the location of the file. The specified file must include the full path.
- The Direct account and the file location must have correct permissions. The operating system account that executes BIOVIA Direct must have correct permissions to write the file in the specified location. Additionally, the file or the specified location of the file must have the correct write permission.

The file to be written will have read permissions. When the `writefile` operator creates the file, it assigns permissions to the file so that it is readable by everyone.

If the specified file already exists, the `writefile` operator replaces the existing file.

### See also

[mdlaux.writefile](#)

[writebinaryfile](#)

[readfile](#)

[readbinaryfile](#)

*BIOVIA Direct Developers Guide > Using Direct > Accessing Files*

## writetempclob

Initializes with a string, or appends a string to, a session-duration temporary CLOB.

### Syntax

```
writetempclob([tempclob-number,] string, init-append)
```

Parameter	Description
<i>tempclob-number</i>	A NUMBER from 0 to 4 that specifies the temporary CLOB that stores the input <i>string</i> . If you do not specify <i>tempclob-number</i> , the default <i>tempclob-number</i> is 0.
<i>string</i>	A VARCHAR2 string that contains the string to be copied to the temporary CLOB.
<i>init-append</i>	A NUMBER that indicates whether to initialize with or append string into the temporary CLOB. Possible values are: 0 - Initialize the temporary CLOB with string 1 (or any non-zero value) - Append string into the temporary CLOB

### Return value

The NUMBER 1 indicates that the temporary CLOB was created. The number 0 indicates that the number specified to reference the temporary CLOB is out of range.

### Usage

```
select writetempclob(tempclob-number, string, init-append)  
from dual;  
select writetempclob(string, init-append)  
from dual;
```

### Example

The following example uses `writetempclob` to copy segments of a query reaction string into a temporary CLOB, and use this reaction in a reaction substructure ([rss](#)) query. The number for the temporary CLOB is 1:

```
select writetempclob(1, query-string, 0) from dual;  
select writetempclob(1, next-query-string, 1) from dual;  
select writetempclob(1, last-query-string, 1) from dual;
```

### Comments

If a client application cannot create LOBs, use `writetempclob` to create a temporary CLOB on the server. You can use the `writetempclob` operator in conjunction with the `tempclob` operator in an application that cannot create or send CLOBs. Use the `writetempclob` operator to construct the temporary CLOB on the server, then use the `tempclob` operator to get the contents of the temporary CLOB. For example:

```
select writetempclob(query-string, 0) from dual;  
select writetempclob(next-query-string, 1) from dual;  
select writetempclob(last-query-string, 1) from dual;  
select rxnmdlnumber  
from samplerx_reaction  
where rss(rctab, tempclob(0))=1;
```

The temporary CLOB is freed when the Oracle session is disconnected. If an application disconnects from Oracle before retrieving all data, the remaining data is lost. The application cannot retrieve the data in a subsequent session.

Use the same number to identify the same temporary CLOB in one Oracle session. The `writetempclob` operator shares the package-level, session-duration temporary CLOBs with the `stringsegment` and `tempclob` operators. Make sure that you reference the correct temporary CLOB numbers between these operators.

### See also

[tempclob](#)

*BIOVIA Direct Developers Guide > About Direct*

## General Functions

Some functions described in this chapter have corresponding operators. The function and its corresponding operator behave identically to each other. For example, the `readfile` operator and the `mdlaux.readfile` function do the same thing. In these cases, the description of the operator appears under [General Operators](#). The General Functions section lists the name of the function and then references the description in General Operators.

Use the package function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators. The package function uses the same syntax as the operator.



For example, in a PL/SQL assignment, use the package function name for `readbinaryfile`, which is `mdlaux.readfile`:

```
query := mdlaux.readfile('/home/user/rxnfiles/rss1.rxn');
```

<a href="#">mdlaux.chimetoclob</a>	15
<a href="#">mdlaux.clobtochime</a>	16
<a href="#">mdlaux.clobtozip64</a>	17
<a href="#">mdlaux.errors</a>	17
<a href="#">mdlaux.externalcommand</a>	18
<a href="#">mdlaux.zip64toclob</a>	24
<a href="#">mdlaux.readbinaryfile</a>	24
<a href="#">mdlaux.readfile</a>	24
<a href="#">mdlaux.stringsegment</a>	24
<a href="#">mdlaux.tempclob</a>	25
<a href="#">mdlaux.version</a>	25
<a href="#">mdlaux.writebinaryfile</a>	25
<a href="#">mdlaux.writefile</a>	25
<a href="#">mdlaux.writetempclob</a>	25

## mdlaux.chimetoclob

Converts a Chime string to the `molfile` or `rxnfile` string representation of the structure.

### Syntax

```
mdlaux.chimetoclob(chimeclob)
```

Parameter	Description
<i>chimeclob</i>	A CLOB that contains the Chime string representation of a molecule or reaction.

### Usage

```
select mdlaux.chimetoclob(chimeclob) from dual;
```

### Return value

A CLOB that contains the molfile or rxnfile string representation of the input Chime structure. The CLOB includes line-feed characters (0x0a) that separate the lines within the molfile or rxnfile. If `mdlaux.chimetoclob` fails, it returns NULL.

### Example

The following example converts a Chime string that is stored in a file into a rxnfile string:

```
select mdlaux.chimetoclob(
  readfile('/opt/BIOVIA/direct/examples/rxnfiles/rxnchime.txt')
)
from dual;
```

Or, in PL/SQL:

```
molfile_clob := MDLAUX.CHIMETOCLOB(chime_clob);
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## mdlaux.clobtochime

Converts a molfile or rxnfile string into the Chime string representation of the structure.

### Syntax

```
mdlaux.clobtochime(molrxnclob)
```

Parameter	Description
<i>molrxnclob</i>	A CLOB that contains the molfile or rxnfile string representation of a molecule or reaction. Each line in <i>molrxnclob</i> must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).

### Usage

```
select mdlaux.clobtochime(molrxnclob) from dual;
```

### Return value

A CLOB that contains the Chime string representation of the input structure. If `mdlaux.clobtochime` fails, it returns NULL.

### Example

The following example converts the contents of a rxnfile to a Chime string:

```
select mdlaux.clobtochime(
    readfile('/opt/BIOVIA/c$direct2021/examples/rxnfiles/query1.rxn')
)
from dual;
```

The following PL/SQL example uses the [rxnmol](#) operator to get the molfile string representation of the first molecule in a specific reaction, and uses `mdlaux.clobtochime` to convert it to a Chime string:

```
DECLARE
    chimeclob CLOB;
    molclobCLOB;
BEGIN
    select rxnmol(rctab,1,1) into molclob
    from samplerx_reaction where rxnmdlnumber = 'RXCI94070168';
    chimeclob := mdlaux.clobtochime(molclob);
END;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## mdlaux.clobtozip64

Converts a molfile or rxnfile string into a gzip compressed and base-64 encoded string.

### Syntax

```
mdlaux.clobtozip64(molrxnclob)
```

Parameter	Description
<i>molrxnclob</i>	A CLOB that contains the molfile or rxnfile string representation of the molecule or reaction.

### Usage

```
select mdlaux.clobtozip64(text-clob) from dual;
```

### Return value

A compressed base-64 format file that represents the input structure. If `mdlaux.clobtozip64` fails, it returns NULL.

### Example

The following example reads a molfile and returns a CLOB containing the gzip compressed and base-64 encoded molfile:

```
select mdlaux.clobtozip64(readfile('c:\benzene.mol')) from dual;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## mdlaux.errors

Returns informational, warning, or error messages that Direct maintains in a message stack for each Oracle session.

### Usage

```
select mdlaux.errors from dual;
```

### Return value

A VARCHAR2 that contains the first 4000 characters of the accumulated informational, warning, and error messages from Direct. If there are no messages, the return value is NULL. If the message stack overflowed, "...more" appears at the end of the string. The following is an example of the return value:

### ERRORS

-----  
RDC-0039: Unable to open file "/home/user/rxn.": Permission denied  
RDC-0012: Unable to copy rxnfile from file "/home/user/rxn.rxn" to CLOB

### Comments

- Use `mdlaux.errors` to check if an error occurred after an operation. This is especially important if you are performing an administrative task, such as altering a reaction domain index.
- `mdlaux.errors` uses a message stack. Calling `mdlaux.errors` clears the message stack. The first message in the stack is the first informational, warning, or error message that you encountered in the Oracle session. The message stack includes newline characters to separate the different messages.
- `mdlaux.errors` reports errors or warnings from Direct. If you received an Oracle error after you executed a SQL statement that uses the Direct operators, or if you have a query that might take a long time to execute, call `mdlaux.errors` to check for errors. The stack accumulates messages that are related to the execution of Direct operators, not the execution of Oracle SQL operators.
- Frequently call `mdlaux.errors`. Direct clears the message stack after each invocation of the function `mdlaux.errors`. If you frequently call `mdlaux.errors` within an Oracle session, you reduce the chance of overflowing the stack, and the chance of losing messages.
- Direct provides a debug logging mechanism. If the Oracle external process that runs Direct terminates for any reason, the current message stack is lost. Direct creates a new message stack for the next Oracle session. If you are debugging a problem and want to see the errors, enable the logging mechanism for the reaction. For more information about debugging and logging, see *Logging Information* in *BIOVIA Direct Administration Guide*.
- If you use session pooling, the error stack might contain information about error conditions that were caused by other users of your session.

### See also

*BIOVIA Direct Developers Guide* > *Using Direct* > *Checking Errors*

## mdlaux.externalcommand

Executes an operating system command with three arguments. Returns the contents of a file as a CLOB or BLOB, the output from the command (`stdout`) as a `VARCHAR2` and errors from the command (`stderr`) as a `VARCHAR2`.

The first and second arguments are file names of temporary files on the Oracle server. The third argument is a text string. The temporary files are created by `mdlaux.externalcommand`; the first contains the contents of an input CLOB and the second is initially empty. The external command can write data to the second file, upon completion of the external command this will be read by `mdlaux.externalcommand` and its contents returned to the caller in a CLOB or BLOB.

### Syntax

```
mdlaux.externalcommand(  
  command in varchar2,  
  inputclob in clob,  
  args in varchar2,  
  binaryoutput in boolean,  
  outputclob out clob,  
  outputblob out blob,  
  stdout out varchar2,
```

```
stderr out varchar2)
```

Parameter	Description
<i>command</i>	The command to execute, for example <code>c:\work\normalizeMolecule.bat</code> .
<i>inputclob</i>	A CLOB containing text to copy to a temporary file, the temporary file's name is passed to the command as its first argument. For example this could be a CLOB containing a molfile.
<i>args</i>	A character string containing any arguments to be passed to the command as its third argument, for example <code>removeHydrogens fixNitroGroups</code> .
<i>binaryoutput</i>	Specify the value as TRUE if the output file from the command should be copied to a BLOB, or specify FALSE if the output file from the command should be copied to a CLOB. If returning a molfile from the command specify FALSE.
<i>outputclob</i>	If the binaryoutput argument is TRUE this value is NULL. If the binaryoutput argument is FALSE this is a temporary CLOB containing the contents of the output file from the command. The output file's name is passed to the command as its second argument.
<i>outputblob</i>	If the binaryoutput argument is FALSE this value is NULL. If the binaryoutput argument is TRUE this is a temporary BLOB containing the contents of the output file from the command. The output file's name is passed to the command as its second argument.
<i>stdout</i>	The first 4000 characters of output from the command are returned in this argument.
<i>stderr</i>	The first 4000 characters of errors from the command are returned in this argument.

### Return value

A BOOLEAN, value is TRUE if the function successfully calls the operating system command or FALSE if there is an error trying to execute the command string.

### Usage

Create an operating system executable or script file which takes three arguments.

- The first argument is the name of a temporary file on the Oracle server, this file should only be read by the command. The contents of this file are a copy of the contents of the `inputclob` CLOB.
- The second argument is the name of an empty temporary file on the Oracle server, this file should only be written to by the command. The contents of this file are copied to a CLOB or a BLOB and returned to the caller.
- The third argument may contain any additional arguments to the command.

For an operating system command which should return a CLOB to the user:

```
declare
    boolean status;
    input_file clob;
    output_file clob;
    unused_output blob;
    input_arguments varchar2(4000);
```

```
        stdout varchar2(4000);
        stderr varchar2(4000);
begin
    input_file := input-clob;
    input_arguments := 'input-arguments';
    status := mdlaux.externalcommand('command-to-execute',
                                     input_file,
                                     input_arguments,
                                     FALSE,
                                     output_file,
                                     unused_output,
                                     stdout,
                                     stderr);
end;
```

For an operating system command which should return a BLOB to the user the only changes are the value of the Boolean flag, the output\_file type and the unused\_output type:

```
declare
    boolean status;
    input_file clob;
    output_file blob;
    unused_output clob;
    input_arguments varchar2(4000);
    stdout varchar2(4000);
    stderr varchar2(4000);
begin
    input_file := input-clob;
    input_arguments := 'input-arguments';
    status := mdlaux.externalcommand('command-to-execute',
                                     input_file,
                                     input_arguments,
                                     TRUE,
                                     unused_output,
                                     output_file,
                                     stdout,
                                     stderr);
end;
```

### Example

The following example creates a Pipeline Pilot Client Chemistry Java SDK program which converts all lead atoms in an input molecule into gold atoms. This will be called from Oracle using Direct's ExternalCommand function to process molecules in a table. This example is not complete and is not suitable for use in a production environment.

First, create and compile the Java program which will perform the desired element transmutation operation. The program reads a molecule in molfile format from the first file name argument, operates on the molecule, and writes the modified molecule, again in molfile format, to the file specified by the second file name argument. The third argument is not used in this example.

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.BufferedWriter;
```

```

import java.io.FileNotFoundException;
// ppchemapi Java sdk
import com.accelrys.chem.*;

public class Transmute {

    private MolIO molIO = null;
    private Molecule inputMol = null;

    public Transmute(String inputFileName, String outputFileName,
                     String options)
    {
        try {
            molIO = PPChemSDKFactory.createMolIO();
            inputMol = PPChemSDKFactory.createMolecule();

            if (!readFile(inputFileName)) {
                return;
            }

            // Convert all atoms of lead to gold
            int na = inputMol.getNumAtoms();
            for (int i=0; i<na; i++) {
                Atom atom = inputMol.getAtom(i);
                if (atom.getType() == Atom.AtomType.Lead)
                    atom.setType(Atom.AtomType.Gold);
            }

            // write output molfile
            String outputMolString =
                molIO.writeMoleculeString(inputMol, MolIO.FormatType.SDFile_
String);

            FileWriter f = new FileWriter(outputFileName);
            BufferedWriter bw = new BufferedWriter(f);
            bw.write(outputMolString);
            bw.close();
        }
        catch (Throwable e) {
            System.err.println("Unable to write file '" + outputFileName +
"":\n" + e);
        }
    }

    private boolean readFile(String filename)
    {
        boolean ret = false;
        if (filename == null || filename.length() == 0) {
            System.err.println("No input filename provided");
        }
        else {
            try {
                FileReader f = new FileReader(filename);

```

```
        BufferedReader br = new BufferedReader(f);
        String line;
        String file = "";
        while ((line = br.readLine()) != null) {
            file += line + "\n";
        }
        br.close();

        molIO.readMolecule(file, MolIO.FormatType.SDFile_String,
inputMol);

        ret = true;
    }
    catch (FileNotFoundException x) {
        System.err.println("Input file '" + filename + "' was not
found");
    }
    catch (Exception x) {
        System.err.println("Input file '" + filename + "' could not be
read:\n" + x);
    }
    }
    return ret;
}

public static void main(String[] args)
{
    String options = null;
    String inputFileName = null;
    String outputFileName = null;

    if (args.length > 0 && args[0].length() > 0) inputFileName = args[0];
    if (args.length > 1 && args[1].length() > 0) outputFileName = args[1];
    if (args.length > 2 && args[2].length() > 0) options = args[2];

    try {
        Transmute pm =
            new Transmute(inputFileName, outputFileName, options);
    }
    catch (Exception e) {
        System.err.println("Caught exception: " + e);
    }
}
}
```

Next, create the program that will be executed by the ExternalCommand function. For this example the program is a Windows batch file that executes the Java program created in the first step.

```
@setlocal

@REM Arguments are:
@REM     Full path and name of input molfile.
@REM     Full path and name of output molfile.
@REM     Options.
```



```

@REM Command invocation for 64-bit java executable
@set JAVA=d:\programs\Java64\jdk1.8.0_31\bin\java

@REM Location of PPChem Java chemistry SDK
@set CHEMISTRYSDK=d:\programs\chemistrysdk

@REM Location of this lib directory (includes trailing backslash)
@set THISDIR=%~dp0

@set CLASSPATH=%THISDIR%.;%CHEMISTRYSDK%\bin\lang\java\jars\ppchemsdk.jar

@%JAVA% -classpath "%CLASSPATH%" ^
        -Djna.library.path="%CHEMISTRYSDK%\bin" ^
        Transmute ^
        %1 %2 %3

@endlocal

```

Finally, create a function in Oracle which accepts an input molecule, processes it using the batch file above, and returns the possibly modified molecule.

```

create or replace function Transmute(inputmol in clob)
return clob
is
    outputmol clob;
    status Boolean;
    unusedblob blob;
    stdout varchar2(4000);
    stderr varchar2(4000);
begin
    status := mdlaux.externalcommand('d:\programs\Transmute.bat',
                                     inputmol,
                                     null,
                                     false,
                                     outputmol,
                                     unusedblob,
                                     stdout,
                                     stderr);
    -- Transmute returns errors in stderr
    if (length(stderr) > 0 or (not status)) then
        dbms_output.put_line('Transmute failed: '||stderr);
        outputmol := null;
    end if;
    return outputmol;
end;
/

```

Run the function to process molecules:

```

create table transmuted_molecules (id number, ctab blob);
insert into transmuted_molecules
    select id, mol(transmute(molfile(ctab))) from input_molecules;

```

## mdlaux.zip64toclob

Converts a gzip compressed and base-64 encoded string to the molfile or rxnfile string representation of the structure.

### Syntax

```
mdlaux.zip64toclob(gzip64-clob)
```

Parameter	Description
<i>gzip64-clob</i>	ACLOB that contains the gzip compressed and base-64 encoded representation of a molecule or reaction.

### Usage

```
select mdlaux.zip64toclob(gzip64-clob) from dual;
```

### Return value

A CLOB that contains the molfile or rxnfile string representation of the input compressed format structure. The CLOB includes line-feed characters (0x0a) that separate the lines within the molfile or rxnfile. If `mdlaux.zip64toclob` fails, it returns NULL.

### Example

The following example reads a file containing a compressed molfile and returns a CLOB containing the molfile:

In PL/SQL:

```
select mdlaux.zip64toclob(readfile('c:\benzene.zip64')) from dual;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}

```

## mdlaux.readbinaryfile

This package function is equivalent to [readbinaryfile](#). See the documentation for `readbinaryfile` for more information.

## mdlaux.readfile

This package function is equivalent to [readfile](#). See the documentation for `readfile` for more information.

## mdlaux.stringsegment

This package function is equivalent to `stringsegment`. See the documentation for [stringsegment](#) for more information.

## mdlaux.tempclob

This package function is equivalent to `tempclob`. See the documentation for [tempclob](#) for more information.

## mdlaux.version

Returns the version of Direct.

### Syntax

```
mdlaux.version[(mode)]
```

Parameter	Description
<i>mode</i>	Optional. Specify the value 'subsys' to return the internal version numbers for subsystems within Direct.

### Usage

```
select mdlaux.version from dual;
select mdlaux.version('subsys') from dual;
```

### Return value

A VARCHAR2 that contains the version number and other information about the Direct product that is currently installed in

Oracle. For example (for Microsoft Windows), `mdlaux.version` returns the following:

```
SQL> SELECT MDLAUX.VERSION FROM DUAL;
```

VERSION
-----
BIOVIA Direct Revision2021 (Microsoft Windows Oracle11) (9.1.0.10) (c) Copyright BIOVIA, Inc. 1999-2014

## mdlaux.writebinaryfile

This package function is equivalent to `writebinaryfile`. See the documentation for [writebinaryfile](#) for more information.

## mdlaux.writefile

This package function is equivalent to `writefile`. See the documentation for [writefile](#) for more information.

## mdlaux.writetempclob

This package function is equivalent to `writetempclob`. For more information, see the documentation for [writetempclob](#).

# Chapter 3:

## Molecule-Specific Operators and Functions

---

This chapter contains the reference listings for the functions and operators that are useful when working with molecules.

### Molecule-Specific Operators

In some cases, Direct offers both a function and an operator with the same name that behave identically to each other. For example, the `readfile` operator and the `mdlaux.readfile` function have the same functionality. In these cases, the description of the operator appears under this section. The [Molecule-Specific Functions](#) section lists the name of the function and then references the description in this section.

If both a function and an operator are available, use the function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators.

<a href="#">chime</a>	27
<a href="#">chime_string</a>	28
<a href="#">chime_string_seg</a>	28
<a href="#">flexmatch</a>	29
<a href="#">flexmatchhighlight</a>	33
<a href="#">flexmatchtimeout</a>	34
<a href="#">fmla_eq</a>	35
<a href="#">fmla_like</a>	35
<a href="#">fmlalike</a>	35
<a href="#">fmlamatch</a>	37
<a href="#">helm</a>	38
<a href="#">helm2</a>	40
<a href="#">inchi</a>	41
<a href="#">inchiauxinfo</a>	42
<a href="#">inchikey</a>	44
<a href="#">isgeneric</a>	45
<a href="#">isnostruct</a>	46
<a href="#">isotopicformula</a>	47
<a href="#">isrna</a>	48
<a href="#">issequence</a>	49
<a href="#">iupacname</a>	50
<a href="#">makeclob</a>	52
<a href="#">mol</a>	53
<a href="#">molchime</a>	54
<a href="#">molfile</a>	55
<a href="#">molfile_string</a>	56
<a href="#">molfile_string_seg</a>	57
<a href="#">molfmla</a>	58
<a href="#">molgzip64</a>	60
<a href="#">molimage</a>	61
<a href="#">molkeys</a>	63
<a href="#">molnmakekey</a>	64
<a href="#">molsim</a>	67

<a href="#">molwt</a>	69
<a href="#">molwtmax</a>	70
<a href="#">molwtmin</a>	71
<a href="#">molwtrange</a>	72
<a href="#">monoisotopicmass</a>	74
<a href="#">numspecifics</a>	75
<a href="#">overlap</a>	76
<a href="#">overlaptimeout</a>	79
<a href="#">pctoverlap</a>	80
<a href="#">readmol</a>	81
<a href="#">sequencesearch</a>	81
<a href="#">sequencetext</a>	83
<a href="#">similar</a>	83
<a href="#">similarity</a>	86
<a href="#">smiles</a>	86
<a href="#">sss</a>	88
<a href="#">sss_highlight_chime</a>	93
<a href="#">sss_highlight_molfile</a>	94
<a href="#">ssshighlight</a>	95
<a href="#">ssssequenceids</a>	97
<a href="#">sssttimeout</a>	98
<a href="#">writemol</a>	99
<a href="#">xhelm</a>	99

## chime

Returns a CLOB, an Oracle character large object, that contains the Chime string representation of a molecule structure.

**Note:** Direct provides this operator to emulate the CHIME operator in the molecule cartridge prior to version 6.0. CHIME is a synonym for the operator MOLCHIME. Because it is a synonym, it cannot be used within a PL/SQL procedure or function. Dassault Systèmes recommends that applications using CHIME with a BLOB argument use [molchime](#) instead, and applications using CHIME with a CLOB argument use [mdlaux.clobtochime](#).

### Syntax

chime(ctab)

Parameter	Description
ctab	Possible values: - The name of the BLOB field that contains the binary chemical structures. - A CLOB value that contains a molfile string to be converted to a Chime string <b>Note:</b> The ctab parameter cannot be a filename.

### Return value

ACLOBthat contains the molecule Chime string

### Usage

```
select chime(ctab)
[, other-column-data]
```

```
from tablename  
where condition;
```

### Example

The following example uses the `chime` operator to return the Chime string representation of a molecule:

```
select chime(ctab)  
from sample2d  
where cdbregno=364;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

## chime\_string

Returns a VARCHAR2 string that contains the first 4000 characters of a molecule Chime string.

**Note:** Direct provides this operator to emulate the `chime_string` operator in the molecule cartridge prior to version 6.0. `chime_string` is equivalent to `stringsegment(chime(arg))`.

### Syntax

```
chime_string(ctab)
```

Parameter	Description
ctab	Possible values: - The name of the BLOB field that contains the binary chemical structures. - A CLOB value that contains a molfile string to be converted to a Chime string. The CLOB contains the first 4000 characters of the Chime string. <b>Note:</b> The ctab parameter cannot be a filename.

### Return value

A VARCHAR2 that contains the Chime string representation of a molecule structure. If the structure exceeds 4000 characters, `chime_string` returns NULL.

### Usage

```
select chime_string(ctab)  
      [, other-column-data]  
from tablename  
where condition;
```

## chime\_string\_seg

Returns a string that contains a segment of a molecule Chime string, up to 4000 characters at a time.

**Note:** Direct provides this operator to emulate the `chime_string_seg` operator in the molecule cartridge prior to version 6.0. After error and limit checking, `chime_string_seg` is equivalent to `dbms_lob.read(molchime(arg), start, stop-start+1)`.

### Syntax

`chime_string_seg(ctab, start, stop)`

Parameter	Description
<code>ctab</code>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The field value cannot be NULL.
<code>start</code>	A number that indicates the starting position in the Chime string. 1 is the first position.
<code>stop</code>	A number that indicates the ending position in the Chime string. <code>start + 3999</code> is the maximum position. The difference between <code>start</code> and <code>stop</code> ( <code>stop - start</code> ) must not exceed 4000.

### Return value

A VARCHAR2 that contains a segment, up to 4000 characters long, of the Chime string representation of a molecule structure.

### Usage

```
select chime_string_seg(ctab, start, stop)
      [, other-column-data]
from tablename
where condition;
```

## flexmatch

Finds records that are an exact match of the structure that you specify in the query. `flexmatch` accepts `flexmatch`-parameters that allow you to restrict or relax the definition of an exact match.

### Syntax

`flexmatch(molecule, query, flexmatch-parameters|'GENERIC' [, flexmatch-number])`

Parameter	Description
<code>molecule</code>	The name of the BLOB field that contains the molecule structures. <code>molecule</code> can also be a molecule object. If a molecule object is specified, the global Ptable, salts file and key definition files will be used during searching.
<code>query</code>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ A Direct molecule object (BLOB)</li> <li>■ A SMILES string</li> <li>■ An InChI string</li> <li>■ An IUPAC name</li> <li>■ A HELM string</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul> <p><b>Note:</b> query cannot be NULL.</p>
<i>flexmatch-parameters</i>	<p>Is a string containing the flexmatch switches. For details about the flexmatch switches, see the "Exact Search (Flexmatch)" chapter in the <i>BIOVIA Chemical Representation Guide</i>.</p> <p><b>Note:</b> For flexmatch searching of generic structures, specify 'GENERIC' instead of the flexmatch parameters. See the following description of 'GENERIC'.</p> <ul style="list-style-type: none"> <li>■ ORIENTONLY - Add the keyword ORIENTONLY to the flexmatch-parameters string to cause flexmatchhighlight to return a structure that is oriented to the query.</li> <li>■ ORIENT - Add the keyword ORIENT to the flexmatch-parameters string to cause flexmatchhighlight to return a structure that is both oriented and highlighted.</li> <li>■ ALLOW_TIMEOUT - Add the keyword ALLOW_TIMEOUT to the flexmatch-parameters string to never return search timeouts as hits. When the keyword is not present and the query times out during a match against a target in the database, the record is returned as a hit and the flexmatchtimeout ancillary operator will return a value of "1". When the keyword is present the record is not returned as a hit. Use this option with care, as a timeout may or may not be identical to the query.</li> </ul> <p><b>Note:</b> Separate the keyword from the other flexmatch-parameters with a space.</p>
'GENERIC' (or 'GENERIC')	<p>Causes flexmatch to accept the query as a specific or generic query molecule, and finds all generics or specifics in the table which enumerate to exactly the same set of specifics. The 'GENERIC' or 'GENERIC' keyword changes the normal flexmatch search to a generic exact-match search. Fastsearch is not used; instead the pre-screen consists of the minimum and maximum molecular weights and the number of enumerated specifics.</p> <p><b>Note:</b> If you specify the 'GENERIC' or 'GENERIC' keyword, do not specify flexmatch-parameters. The generic search always performs an exact match that ignores all data Sgroups.</p>
<i>flexmatch-number</i>	<p>A number that is equal to the <i>flexmatch-number</i> parameter used with the flexmatchhighlight or flexmatchtimeout operator. This parameter only applies if you use flexmatchhighlight or flexmatchtimeout.</p>

### Return value

The NUMBER 1 indicates that the query matched one or more records. When you use flexmatch in a WHERE clause, always test the return value for a result of 1.



**Usage**

```
select column-data
from tablename
where flexmatch(ctab, query, flexmatch-parameters)=1
[operator other-conditions];
```

The `flexmatch` operator can also be used in the `SELECT` clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a structure is really a flexmatch search hit from a complex `WHERE` clause.

```
select flexmatch(ctab, query, flexmatch-parameters)
[, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses the `flexmatch` parameter `tau` to perform a tautomer search on a specific structure:

```
select cdbregno
from sample2d
where flexmatch(
  ctab,
  (select ctab from sample2d where cdbregno=364),
  'match=tau'
)=1;
```

**Comments**

- To negate the results of `flexmatch`, use the SQL operator `NOT`. For example:

```
select count(*) from sample2d
where not flexmatch(ctab, 'c=c-c=c-c=c-@1', 'all')=1;
```

- The query structure for the `flexmatch` search must not contain query features. The query must be a structure that would be suitable for registration. For information on query features and other restrictions on `flexmatch` query structures, see the "Exact Search (Flexmatch)" chapter in *BIOVIA Chemical Representation*.

If you use a structure with query features, the [mdlaux.errors](#) function returns the following error:

```
Error CHEMICALDB-1110 (1): <prep_specific_flex:Invalid
FLEXMATCH search query: No query features allowed>
```

- `flexmatch` is an indexed operator. If Direct cannot locate the domain index for the operator `flexmatch`, the search executes more slowly than an indexed `flexmatch`. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`. For more information, see *Direct Domain Index and the Oracle Optimizer* in the *About Direct* chapter of *BIOVIA Direct Developers Guide*.
- Clearly specify the `flexmatch` switches. If the specified `flexmatch-parameters` is `NULL`, or an empty string, a string of blank characters, `"MATCH=NONE"` is assumed. This is also equivalent to `"IGNORE=ALL"`. For details about the `flexmatch` switches, see the "Exact Search (Flexmatch)" chapter in *BIOVIA Chemical Representation*.

- The flexmatch operator can be used to compare two tables of generic molecules. The flexmatch operator can be used to perform library comparison. Two molecule tables can be compared to determine which molecules in the source table also exist in the target table. The following SQL statement compares a list of source structures (as a table) with a target database:

```
CREATE TABLE HITTABLE (source_idcolumn type, target_idcolumn type);

INSERT INTO HITTABLE
  SELECT /*+ ORDERED INDEX(TRG targettable_domainindex) USE_NL(TRG) */
    SRC.idcolumn, TRG.idcolumn
  FROM sourcetable SRC, targettable TRG
  WHERE FLEXMATCH(TRG.CTAB, SRC.querycolumn, 'GENERIC')=1;
```

- Use hints to force Oracle to use the domain index in a generic flexmatch search. When using the flexmatch search to compare tables, use hints to force Oracle to use the domain index in the search. The flexmatch search will be very slow when the domain index is not used. The ORDERED, INDEX and USE\_NL hints in the following example cause the domain index to be used in most cases. However, use Oracle's EXPLAIN PLAN to verify that the query uses the domain index in your specific case.
- If one or both of the tables contain generic molecules, use the 'GENERIC' flag with the flexmatch operator. For example:

```
INSERT INTO RESULTTABLE
SELECT
  /*+ ORDERED INDEX(TRG targettable_domainindex) USE_NL(TRG) */
  SRC.idcolumn,
  TRG.idcolumn
FROM sourcetable SRC, targettable TRG
WHERE FLEXMATCH(TRG.CTAB, SRC.CTAB, 'GENERIC')=1;
```

### Tautomer Matching Parameters

It is not possible to control generation of tautomeric regions in the flexmatch operator itself. However the tautomer generation parameters can be set on a per-session basis by using the PL/SQL procedure MDLAUX.SETFLAGS. Once a parameter is set to a value it will retain that setting either until the Oracle session terminates or until MDLAUX.SETFLAGS is executed again with another value for the parameter. Possible parameter settings are shown below, these match the settings which are possible with the *Identify Tautomeric Fragments* component except that all spaces have been removed. Spaces are not allowed in names or values used with MDLAUX.SETFLAGS.

```
execute mdlaux.setflags('ConsiderCarbonAsDonor=Never');
execute mdlaux.setflags
('ConsiderCarbonAsDonor=BondedToTwoOrMoreTautomericAtoms');
execute mdlaux.setflags
('ConsiderCarbonAsDonor=BondedToOneOrMoreTautomericAtom');
execute mdlaux.setflags('ConsiderCarbonAsDonor=BondedToAcceptor(1_3)');
execute mdlaux.setflags('MakeAllSp2AtomsAcceptors=True');
execute mdlaux.setflags('MakeAllSp2AtomsAcceptors=False');
execute mdlaux.setflags('AmidesTautomerization=SkipAllAmides');
execute mdlaux.setflags('AmidesTautomerization=TautomerizeOnlyDiamides');
execute mdlaux.setflags('AmidesTautomerization=TautomerizeAllAmides');
```

```
execute mdlaux.setflags('PerceiveChargeTautomerization=True');
execute mdlaux.setflags('PerceiveChargeTautomerization=False');
execute mdlaux.setflags('TautomerizeHydrogenIsotopes=True');
execute mdlaux.setflags('TautomerizeHydrogenIsotopes=False');
```

**See also**[flexmatchhighlight](#)[flexmatchtimeout](#)[BIOVIA Chemical Representation > Exact Search \(Flexmatch\)](#)[BIOVIA Chemical Representation > Molecule Representation > Tautomers](#)[Flexmatch Search](#)[Examples of Flexmatch Search](#)[Specifying the Query Structure](#)**flexmatchhighlight**

Returns the oriented or oriented and highlighted molecule from a flexmatch search.

**Syntax**

```
flexmatchhighlight(flexmatch-number)
```

Parameter	Description
<i>flexmatch-number</i>	A NUMBER equal to the <i>flexmatch-number</i> parameter that is used with the <a href="#">flexmatch</a> operator.

**Return value**

A CLOB that contains the Chime representation of a candidate molecule oriented to match query orientation, and optionally contain highlight information. `flexmatchhighlight` stores the return value in a temporary CLOB. The Chime string uses the V3000 format, and contains highlight information as CTlib collection objects.

**Usage**

```
select flexmatchhighlight(flexmatch-number)
from tablename
where flexmatch(ctab, query, 'flexmatch-parameters orientonly', flexmatch-number)=1;
```

**Parameter description**

*flexmatch-number* = A NUMBER equal to the *flexmatch-number* parameter that is used with the `flexmatch` operator.

**Example**

The following example returns highlighted molecule Chime strings after a flexmatch search.

```
select flexmatchhighlight(3),
from moltable
where flexmatch(ctab,query,('match=all orientonly,3'))=1;
```

**Notes**

- The number 3 is used to correlate the `flexmatch` operator in the `WHERE` clause with the `flexmatchhighlight` operator in the `SELECT` clause. This could be any number as long as the values in the two operators match.
- Use `ORIENTONLY` in the `flexmatch` parameters to orient the candidate structures based on the query, but not add any highlight information. Use `ORIENT` in the `flexmatch` parameters to both orient the candidate structures and add highlight information. For more information on `ORIENT` and `ORIENTONLY`, see [flexmatch](#).

**See also**

[flexmatch](#)

**flexmatchtimeout**

Returns a the timeout status value from an `flexmatch` search.

`flexmatch` will return as matches those candidates which for which the matching algorithm times out. Such candidates may or may not be actual matches. This ancillary operator gives the user information about that timeout status.

**Syntax**

`flexmatchtimeout(flexmatch-number)`

Parameter	Description
<i>flexmatch-number</i>	A NUMBER equal to the <i>flexmatch-number</i> parameter that is used with the <a href="#">flexmatch</a> operator.

**Return value**

A NUMBER that indicates the status of the `flexmatch` search. The possible values are:

Value	Description
0	The <code>flexmatch</code> search did not time out.
1	The <code>flexmatch</code> search timed out.
NULL	The target was not a match to the query.

**Usage**

```
select flexmatchtimeout(flexmatch-number)
      [,other-column-data]
from tablename
where flexmatch(mol, query, v1-v2-options, flexmatch-number)=1
      [operator other-conditions];
```

**Example**

The following example returns the timeout status while searching for a duplicate.

```
select extreg,
       flexmatchtimeout(3) "Timeout"
from moltable
where flexmatch(
```

```
molcol,
'/opt/BIOVIA/direct/query1.mol',
'match=all',
3
)=1;
```

**Note:** The number 3 is used to correlate the `flexmatch` operator in the WHERE clause with the `flexmatchtimeout` operator in the SELECT clause. This could be any number as long as the values in the two operators match.

### Comments

The `flexmatch`-number parameters for `flexmatchtimeout` and `flexmatch` operators must match. If the `flexmatch`-number parameters do not match, or if you use `flexmatchtimeout` without using `flexmatch`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

The `flexmatchtimeout` operator can be used with a generic `flexmatch`. `flexmatchtimeout` returns the timeout status of the search.

### See also

[flexmatch](#)

### fmla\_eq

This `fmla_eq` operator is a synonym of `fmlamatch`. See the documentation for [fmlamatch](#) for more information.

**Note:** Direct provides this operator to emulate the `fmla_eq` operator in the molecule cartridge prior to version 6.0.

### fmla\_like

This `fmla_like` operator is a synonym of `fmlalike`. See the documentation for `fmlalike` for more information.

**Note:** Direct provides this operator to emulate the `fmla_like` operator in the molecule cartridge prior to version 6.0.

### fmlalike

Finds records that contain the molecule formula that you specify in the query. The specified formula can be a subformula. Matching records might contain atomic symbols that do not occur in your query.

### Syntax

`fmlalike(ctab, formula-query)`

Parameter	Description
<code>ctab</code>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>fm1alike</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>fm1alike</code> uses the global Ptable. The <code>fm1alike</code> operator also uses the global Ptable if the value of the parameter is a molecule object. The field value cannot be NULL.
<i>formula-query</i>	A string that contains the formula that lists the atomic symbols and the corresponding number of atoms. The string cannot be NULL. For details about the possible formats of a query, see <a href="#">Molecule Formula Search</a> .

**Return value**

The NUMBER 1 indicates that the formula matched one or more records. When you use `fm1alike` in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where fm1alike(ctab,query)=1
    [operator other-conditions];
```

The `fm1alike` operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a structure is really a `fm1alike` search hit from a complex WHERE clause.

```
select fm1alike(ctab,query)
    [, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses `fm1alike` to find the molecules that contain 6 carbon atoms and 6 hydrogen atoms. This example uses the operator `molformula` to display the formula of the molecules that matched the query.

```
select cdbregno,
    molformula(ctab)
from sample2d
where fm1alike(ctab,'C6 H6')=1;
```

**Comments**

- To negate the results of `fm1alike`, use the SQL operator NOT. For example:

```
select count(*)
from sample2d
where not fm1alike(ctab,'C6 H6')=1;
```

- `fmlalike` is an indexed operator. If Direct cannot locate the domain index for the operator `fmlalike`, the search executes more slowly than an indexed `fmlalike`. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`.

- Formula searching of biopolymers using the `fmlalike` operator is supported. However, when you search for biopolymers, search for non-template atoms (C, H, N, O, S, etc.) and not for residues (names of amino acids such as Ala, Cys, etc.). For example, the following query works:

```
select uniprot_name from human where fmlalike(ctab, 'c(1000-1100)')=1;
```

But the following query will not work because template atoms with the symbol “Cys” are not indexed:

```
select uniprot_name from human where fmlalike(ctab, 'Cys')=1;
```

#### See also

[fmlamatch](#)

[molfm1a](#)

[Molecule Formula Search](#)

## fmlamatch

Finds records that exactly match the molecule formula that you specify in the query. The specified formula can be a subformula. (Matching records may contain atomic symbols that do not occur in your query.)

#### Syntax

`fmlamatch(ctab, formula-query)`

Parameter	Description
<code>ctab</code>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>fmlamatch</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>fmlamatch</code> uses the global Ptable. The <code>fmlamatch</code> operator also uses the global Ptable if the value of the parameter is a molecule object. The field value cannot be NULL.
<code><i>formula-query</i></code>	A string that contains the formula that lists the atomic symbols and the corresponding number of atoms. The string cannot be NULL. For details about the possible formats of a query, see <a href="#">Molecule Formula Search</a> .

#### Return value

The NUMBER 1 indicates that the formula matched one or more records. When you use `fmlamatch` in a WHERE clause, always test the return value for a result of 1.

#### Usage

```
select column-data
from tablename
where fmlamatch(ctab, query)=1
[operator other-conditions];
```

The `fmlamatch` operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can

be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a structure is really a `fm1amatch` search hit from a complex `WHERE` clause.

```
select fm1amatch(ctab,query)
[, other-column-data]
from tablename
where condition;
```

### Example

The following example uses `fm1amatch` to find molecules whose molecule formulas contain exactly two carbon atoms and four hydrogen atoms.

```
select cdbregno
from sample2d
where fm1amatch(ctab,'c2 h4')=1;
```

### Comments

- `fm1amatch` behaves like `fm1alike`, except that `fm1amatch` performs an equality search.
- `fm1amatch` is an indexed operator. If Direct cannot locate the domain index for the operator `fm1amatch`, the search executes more slowly than an indexed `fm1amatch`. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`.
- Formula searching of biopolymers using the `fm1amatch` operator is supported. However, note that the search is for non-template atoms (C, H, N, O, S, etc.) not for residues (Ala, Cys, etc.). Thus, these searches are not useful except in cases where the search is for an unusual element such as selenium.

### See also

[fm1alike](#)

[molfm1a](#)

[Molecule Formula Search](#)

[Examples of Molecule Formula Search](#)

## helm

Returns a HELM string representation of a biopolymer sequence molecule.

### Syntax

```
helm(molecule)
```

*molecule* is the name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

### Return value

A temporary CLOB that contains the HELM string. The `helm` operator returns NULL if the HELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdlaux.errors` to see the related error message.

### Usage

```
select helm(molecule)
[, other-column-data]
from tablename
where condition;
```



**Example**

The following example shows the HELM strings for two molecules from a table of human proteins:

```
select name, helm(ctab) from human_subset where rownum <= 2;
```

NAME	HELM(CTAB)
A20AL_HUMAN	PEPTIDE1{M.K.L.F.G.F.R.S.R.R.G.Q.T.V.L.G.S.I.D.H.L.Y.T.G.S.G .Y.R.I.R.Y.S.E.L.Q.K.I.H.K.A.A.V.K.G.D.A.A.E.M.E.R.C.L.A.R.R .S.G.D.L.D.A.L.D.K.Q.H.R.T.A.L.H.L.A.C.A.S.G.H.V.K.V.V.T.L.L .V.N.R.K.C.Q.I.D.I.Y.D.K.E.N.R.T.P.L.I.Q.A.V.H.C.Q.E.E.A.C.A .V.I.L.L.E.H.G.A.N.P.N.L.K.D.I.Y.G.N.T.A.L.H.Y.A.V.Y.S.E.S.T .S.L.A.E.K.L.L.F.H.G.E.N.I.E.A.L.D.K.V}\$\$\$PEPTIDE1{ChainName :Putative ankyrin repeat domain-containing protein 20A-like protein MGC26718} PEPTIDE1{ChainDescription:chain}\$
APPT_HUMAN	PEPTIDE1{M.S.S.G.N.Y.Q.Q.S.E.A.L.S.K.P.T.F.S.E.E.Q.A.S.A.L.V .E.S.V.F.G.L.K.V.S.K.V.R.P.L.P.S.Y.D.D.Q.N.F.H.V.Y.V.S.K.T.K .D.G.P.T.E.Y.V.L.K.I.S.N.T.K.A.S.K.N.P.D.L.I.E.V.Q.N.H.I.I.M .F.L.K.A.A.G.F.P.T.A.S.V.C.H.T.K.G.D.N.T.A.S.L.V.S.V.D.S.G.S .E.I.K.S.Y.L.V.R.L.L.T.Y.L.P.G.R.P.I.A.E.L.P.V.S.P.Q.L.L.Y.E .I.G.K.L.A.A.K.L.D.K.T.L.Q.R.F.H.H.P.K.L.S.S.L.H.R.E.N.F.I.W .N.L.K.N.V.P.L.L.E.K.Y.L.Y.A.L.G.Q.N.R.N.R.E.I.V.E.H.V.I.H.L .F.K.E.E.V.M.T.K.L.S.H.F.R.E.C.I.N.H.G.D.L.N.D.H.N.I.L.I.E.S .S.K.S.A.S.G.N.A.E.Y.Q.V.S.G.I.L.D.F.G.D.M.S.Y.G.Y.Y.V.F.E.V .A.I.T.I.M.Y.M.M.I.E.S.K.S.P.I.Q.V.G.G.H.V.L.A.G.F.E.S.I.T.P .L.T.A.V.E.K.G.A.L.F.L.L.V.C.S.R.F.C.Q.S.L.V.M.A.A.Y.S.C.Q.L .Y.P.E.N.K.D.Y.L.M.V.T.A.K.T.G.W.K.H.L.Q.Q.M.F.D.M.G.Q.K.A.V .E.E.I.W.F.E.T.A.K.S.Y.E.S.G.I.S.M}\$\$\$PEPTIDE1{ChainName:Pro bable phosphotransferase LOC123688} PEPTIDE1{ChainDescriptio n:chain}\$

**Comments**

- HELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

BIOVIA Direct Developers Guide > Using Direct > Getting the HELM string

**helm2**

Returns a HELM version 2 string representation of a biopolymer sequence molecule.

**Syntax**

```
helm2(molecule)
```

*molecule* is the name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

**Return value**

A temporary CLOB that contains the HELM string. The helm operator returns NULL if the HELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdl_aux.errors` to see the related error message.

**Usage**

```
select helm2(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows the HELM strings for two molecules from a table of human proteins:

```
SQL> select name, helm2(ctab) from human_subset order by 1;
```

NAME	HELM2(CTAB)
A20AL_HUMAN	PEPTIDE1{M.K.L.F.G.F.R.S.R.R.G.Q.T.V.L.G.S.I.D.H.L.Y.T.G.S.G.Y.R.I.R.Y.S.E.L.Q.K.I.H.K.A.A.V.K.G.D.A.A.E.M.E.R.C.L.A.R.R.S.G.D.L.D.A.L.D.K.Q.H.R.T.A.L.H.L.A.C.A.S.G.H.V.K.V.V.T.L.L.V.N.R.K.C.Q.I.D.I.Y.D.K.E.N.R.T.P.L.I.Q.A.V.H.C.Q.E.E.A.C.A.V.I.L.L.E.H.G.A.N.P.N.L.K.D.I.Y.G.N.T.A.L.H.Y.A.V.Y.S.E.S.T.S.L.A.E.K.L.L.F.H.G.E.N.I.E.A.L.D.K.V}"ChainDescription:chain,ChainName:Putative ankyrin repeat domain-containing protein 20A-like protein MGC26718"\$\$\$\$V2.0
APPT_HUMAN	PEPTIDE1{M.S.S.G.N.Y.Q.Q.S.E.A.L.S.K.P.T.F.S.E.E.Q.A.S.A.L.V.E.S.V.F.G.L.K.V.S.K.V.R.P.L.P.S.Y.D.D.Q.N.F.H.V.Y.V.S.K.T.K.D.G.P.T.E.Y.V.L.K.I.S.N.T.K.A.S.K.N.P.D.L.I.E.V.Q.N.H.I.I.M.F.L.K.A.A.G.F.P.T.A.S.V.C.H.T.K.G.D.N.T.A.S.L.V.S.V.D.S.G.S.E.I.K.S.Y.L.V.R.L.L.T.Y.L.P.G.R.P.I.A.E.L.P.V.S.P.Q.L.L.Y.E.I.G.K.L.A.A.K.L.D.K.T.L.Q.R.F.H.H.P.K.L.S.S.L.H.R.E.N.F.I.W.N.L.K.N.V.P.L.L.E.K.Y.L.Y.A.L.G.Q.N.R.N.R.E.I.V.E.H.V.I.H.L.F.K.E.E.V.M.T.K.L.S.H.F.R.E.C.I.N.H.G.D.L.N.D.H.N.I.L.I.E.S.S.K.S.A.S.G.N.A.E.Y.Q.V.S.G.I.L.D.F.G.D.M.S.Y.G.Y.Y.V.F.E.V.A.I.T.I.M.Y.M.M.I.E.S.K.S.P.I.Q.V.G.G.H.V.L.A.G.F.E.S.I.T.P.L.T.A.V.E.K.G.A.L.F.L.L.V.C.S.R.F.C.Q.S.L.V.M.A.A.Y.S.C.Q.L.Y.P.E.N.K.D.Y.L.M.V.T.A.K.T.G.W.K.H.L.Q.Q.M.F.D.M.G.Q.K.A.V

```
.E.E.I.W.F.E.T.A.K.S.Y.E.S.G.I.S.M}"ChainDescription:chain,C  
hainName:Probable phosphotransferase LOC123688"$$$V2.0
```

### Comments

- HELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

### See also

*BIOVIA Direct Developers Guide > Using Direct > Getting the HELM string*

## inchi

Returns an IUPAC standard International Chemical Identifier, InChI string, or a non-standard InChI string for the specified molecule. For more information about InChI, see *BIOVIA Direct Developer's Guide > Using Direct > Getting the InChI String and Key*.

### Syntax

`inchi(molecule, options)`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options. If no additional options are provided in the call to INCHI, the standard InChI string is generated. If any of the following options are specified, a non-standard InChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the InChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

## Return value

A temporary CLOB that contains the InChI string. The output string length will exceed 4000 characters for very large molecules. The `inchi` operator returns NULL if the InChI string cannot be generated. Use `mdlaux.errors` to see the related error message.

## Usage

```
select inchi(molecule,options)
      [, other-column-data]
from tablename
where condition;
```

## Example

The following example shows the InChI string for the molecules in a table, using the default option:

```
select inchi(ctab) from moltable;

INCHI(CTAB)
-----
InChI=1S/C6H5Cl/c7-6-4-2-1-3-5-6/h1-5H
```

## Comments

- There are limitations to the generation of InChI strings. Not all BIOVIA molecule features can be handled. The `inchi` operator returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message. For details, see *Limitations to the Generation of InChI Strings* in the *Using Direct* chapter of *BIOVIA Direct Developers Guide*.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## See also

[mdlaux.inchi](#)  
[inchikey](#)

## inchiauxinfo

Returns the auxilliary information (AuxInfo) that is computed along with the IUPAC International Chemical Identifier (InChI) string for a molecule.

## Syntax

`inchiauxinfo(ctab [, options])`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options. If no additional options are provided in the call to INCHIAUXINFO, the standard InChI AuxInfo string is generated. If any of the following options are specified, a non-standard InChI AuxInfo string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the InChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

## Return value

A temporary CLOB that contains the InChI AuxInfo string. The output string length will exceed 4000 characters for very large molecules. The `inchiauxinfo` operator returns NULL if the InChI AuxInfo string cannot be generated. Use `mdl aux.errors` to see the related error message.

## Usage

```
select inchiauxinfo(molecule,options)
  [, other-column-data]
from tablename
where condition;
```

## Example

The following example shows the InChI AuxInfo string for the molecules in a table, using the default option:

```
select inchiauxinfo(ctab) from moltable;
```

INCHI(CTAB)

```
-----
AuxInfo=1/0/N:5,1,6,3,4,2,7/E:(2,3)(4,5)/rA:7CCCCCCC1/rB::d-1s2;d-
2;s1;s4d-5;s2;/rC:11.6039,-8.8012,0;13.0335,-8.8008,0;12.3201,-
8.388,0;13.0335,-9.6278,0;11.6039,-9.6315,0;12.3219,-
10.0405,0;13.7497,-8.3873,0;
```

## Comments

- There are limitations to the generation of InChI AuxInfo strings. Not all BIOVIA molecule features can be handled. The `inchiauxinfo` operator returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message. For details, see *Limitations to the Generation of InChI Strings* in the *Using Direct* chapter of *BIOVIA Direct Developers Guide*.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## See also

[mdlaux.inchiauxinfo](#)

[inchi](#)

## inchikey

Returns an IUPAC standard International Chemical Identifier, InChI string, or a non-standard InChI string for the specified molecule. The `inchikey` operator generates the key by first generating the InChI string, and then calling an InChI library function to convert the string into the 27-character key. For more information about InChI, see *BIOVIA Direct Developers Guide > Using Direct > Getting the InChI String and Key*.

## Syntax

`inchikey(molecule, options)`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options. If no additional options are provided in the call to <code>INCHI</code>, the standard InChI string is generated. If any of the following options are specified, a non-standard InChI string is generated.</p> <ul style="list-style-type: none"> <li>■ <b>NEWPSOFF</b> - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ <b>FixedH</b> - Include Fixed H layer (default is 'not')</li> <li>■ <b>RecMet</b> - Include reconnected metals results (default is 'not')</li> <li>■ <b>SABs</b> - Absolute stereo (default)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the InChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A VARCHAR2 that contains the 27-character InChI key. The `inchikey` operator returns NULL if the InChI string cannot be generated. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select inchikey(molecule,options)
       [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows the InChI key the molecules in a table, using the default option:

```
select inchikey(ctab) from moltable;
INCHIKEY(CTAB)
-----
MVPPADPHJFYWMZ-UHFFFAOYSA-N
```

**Comments**

There are limitations to the generation of InChI strings. Not all BIOVIA molecule features can be handled. The `inchikey` operator returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message. For details, see *BIOVIA Direct Developers Guide > Using Direct > Limitations to the generation of InChI strings*.

**See also**

[mdlaux.inchikey](#)  
[inchi](#)

**isgeneric**

Returns 1 if the specified molecule is a generic structure, 0 if not. A generic structure is a Markush structure that represents actual structures.

**Syntax**

```
isgeneric(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

**Return value**

A NUMBER that indicates whether the specified molecule is a generic structure (1) or not (0).

**Usage**

```
select column-data
from tablename
where isgeneric(molecule)=1
[operator other-conditions];
```

**Example**

The following example uses `isgeneric` to find generic structures in the `samplegen` table.

```
select parent_sampleid
from samplegen
where isgeneric(ctab)=1;
```

**Comments**

This operator is not indexed. When possible, avoid using it when running queries on large tables.

**See also**

[mdlaux.isgeneric](#)

**isnostruct**

Returns 1 if the specified molecule is a nostruct ("no-structure"), 0 if not. A no-structure is a chemical structure that consists of zero fragments, that is, zero atoms and zero bonds.

**Syntax**

```
isnostruct(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

**Return value**

A NUMBER that indicates whether the specified molecule is a no-structure (1) or not (0).

**Usage**

```
select column-data
from tablename
where isnostruct(ctab)=1
[operator other-conditions];
```

**Example**

The following example uses `isnostruct` to find "no-structures" in the `sample2d` table.



```
select cdbregno
from sample2d
where isnostruct(ctab)=1;
```

### Comments

This operator is not indexed. When possible, avoid using it when running queries on large tables.

### See also

[mdlaux.isnostruct](#)

## isotopicformula

Displays the molecule formula with isotope labels. The `isotopicformula` operator uses BIOVIA Pipeline Pilot Client to get the formula.

### Syntax

```
isotopicformula(molecule, 'format-options')
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>isotopicformula</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>isotopicformula</code> uses the global Ptable. The <code>isotopicformula</code> operator also uses the global Ptable if the value of the parameter is a molecule object.
<i>format-options</i>	Optional string to control formatting of the output molecular formula. If no options are present the formula will include space between elements, and the formula for each fragment in a multi-fragment structure will be separated by a dot. Use <code>NOSPACE</code> to remove the space between elements, <code>NOFRAGMENT</code> to not separate fragment formulas, and <code>NOSPACE NOFRAGMENT</code> for both changes. To match the formula string which is output by Insight use <code>NOSPACE NOFRAGMENT</code> . A third optional argument, <code>USEDANDT</code> , switches the default display of hydrogen isotopes, 2H and 3H, to the older one-letter designations D and T.

### Return value

A temporary CLOB that contains the molecular formula string, including isotope labels. The domain index Ptable is automatically used to resolve atom symbols.

### Usage

```
select isotopicformula(molecule)
[, other-column-data]
from tablename
where condition;
```

### Example

The following example uses `isotopicformula` to return the isotopic formula of the molecules in a table:

```
select cdbregno,
```

```
molformula(ctab),
isotopicformula(ctab)
from acd2d_moltable;
```

### Comments

- The `isotopicformula` operator cannot be used for registration. Use the `mdlaux.isotopicformula` function when registering mono-isotopic mass into a table.
- To match the formula output by Direct using the Molecular Formula component in Pipeline Pilot, set the component options as follows:
  - Ignore Isotopes = (True for MOLFMLA, False for ISOTOPICFORMULA)
  - Use 2H and 3H for Hydrogen Isotopes = True
  - Include Space Between Elements = (True for default, False if using NOSPACE)
  - Separate Fragments = (True for default, False if using NOGRAGMENT)
  - Add HTML Tags = False
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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":
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[mdlaux.isotopicformula](#)

[molformula](#)

## isrna

Returns 1 if the molecule argument is an RNA or DNA biopolymer sequence, 0 if it is not.

A molecule is considered an RNA or DNA biopolymer sequence if it contains one or more nucleotide template atoms, one or more nucleotide template definitions, or one or more Sgroup abbreviations (superatoms) that have nucleotide sequence information associated with them.

### Syntax

```
isrna(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

### Return value

A NUMBER that is 1 if the molecule is an RNA or DNA biopolymer sequence, or 0 if it is not.

**Usage**

```
select isrna(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows whether or not each molecule in a table is a nucleotide sequence:

```
select id, isrna(ctab) from moltable;
      ISRNA(CTAB),
-----
```

adenine	1
PhCl	0

**See also**

[mdlaux.isrna](#)

**issequence**

Returns 1 if the molecule argument is a biopolymer sequence, 0 if it is not.

A molecule is considered a biopolymer sequence if it contains one or more template atoms, one or more template definitions, or one or more Sgroup abbreviations (superatoms) that have sequence information associated with them. For more information about biopolymer sequences, see *BIOVIA Direct Developers Guide > Using Direct > Biopolymer Searching and Registration*.

**Syntax**

```
issequence(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

**Return value**

A NUMBER that is 1 if the molecule is a biopolymer sequence, or 0 if it is not.

**Usage**

```
select issequence(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows whether or not each molecule in a table is a sequence:

```
select id, issequence(ctab) from moltable;
      IDISSEQUENCE(CTAB)
-----
```

alanine	1
AATC2	1
PhCl	0

**Comments**

The `issequence` operator is not an indexed operator and cannot be used in a WHERE clause.

**See also**

[mdlaux.issequence](#)

**iupacname**

Returns the IUPAC (International Union of Pure and Applied Chemistry) name of a molecule.

**Syntax**

`iupacname(molecule, options)`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	(Optional) An optional VARCHAR2 argument to specify the language, name style, and character set. The argument can specify one or more of the LANGUAGE, NAMESTYLE, and CHARSET. For example: 'LANGUAGE=language NAMESTYLE=name-style CHARSET=character-set' If no options are specified, the defaults are: 'LANGUAGE=ENGLISH NAMESTYLE=DEFAULT CHARSET=HTML ' For a list of valid options, see the list that follows.

**List of valid language, name-style, and character-set options**

■ Valid language options are:

- ENGLISH (default)
- BRITISH
- INTERNATIONAL
- CHINESE
- DANISH
- DUTCH
- FRENCH
- GERMAN
- GREEK
- HUNGARIAN
- IRISH
- ITALIAN
- JAPANESE

- POLISH
- PORTUGUESE
- ROMANIAN
- RUSSIAN
- SLOVAK
- SPANISH
- SWEDISH
- Valid name-style options are:
  - DEFAULT (default)
  - OPENEYE
  - TRADITIONAL
  - SYSTEMATIC
  - IUPAC
  - CAS
  - CASINDEX
  - AUTONOM
  - IUPAC79
  - IUPAC93
  - ACDNAME
- Valid character-set options are:
  - DEFAULT
  - ASCII
  - HTML (default)
  - UTF8

**Return value**

A temporary CLOB that contains the IUPAC name of the molecule. The `iupacname` operator returns NULL if a name cannot be generated for the molecule. Use `mdl aux.errors` to see related error message.

**Usage**

```
select iupacname(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows the IUPAC name of a molecule in the `sample2d` table.

```
select iupacname(ctab)
from sample2d
where cdbregno=17;
IUPACNAME(CTAB)
-----
```

diphenyl benzene-1,2-dicarboxylate

### Comments

The `iupacname` operator uses functionality provided by OpenEye Scientific Software.

### See also

[mdlaux.iupacname](#)

## makeclob

Converts a string or VARCHAR data to a CLOB, an Oracle character large object. Use `makeclob` to register or update a CLOB field in a molecule table with a string or VARCHAR value.

Direct provides this operator to emulate the `makeclob` operator of the molecule cartridge prior to version 6.0. `makeclob` is equivalent to `tempclob(writetempclob(str, 0))`.

### Syntax

`makeclob(string)`

Parameter	Description
<i>string</i>	A string or VARCHAR value

### Return value

A CLOB that contains the given string.

### Usage

```
insert into tablename(  
    clob-field  
    [, other-column-data]  
)  
values (  
    makeclob(string-value)  
    [, other-value-data]  
);  
  
update tablename  
set clob-field = makeclob(string-value)  
    [, other-column=other-value]  
where condition;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

### See also

[tempclob](#)

[writetempclob](#)**mol**

Converts a molfile filename or string to a 2D molecule object. Use the `mol` operator to:

- Register or update a structure in a table
- Produce a molecule object to use as a candidate for a search operator

**Syntax**

`mol(structure)`

Parameter	Description
<i>structure</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A 2D molecule object contains the "packed" binary molecule. Z coordinates are set to zero, with any 3D stereochemistry converted to up and down bonds.

**Usage**

```
insert into tablename(
  ctab
  [,other-column-data]
)
values (
  mol(molfile-structure)
  [,other-value-data]
);
update tablename
set ctab = mol(chime-structure)
  [,other-column=other-value]
where condition;
insert into tablename(
  ctab
  [,other-column-data]
)
values (
```

```
mol('/pathname/filename.mol')
[,other-value-data]
);
```

### Example

The following registration example uses mol to generate a molecule object for registration:

```
insert into moltable(extreg, molcol)
values ('DLG-123', MOL('/home/user/dlg123.mol'));
```

### Comments

**Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function.** 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();
}
```

Warnings appear when a molecule is modified during registration. The following warnings occur when the stereochemistry perception determines that an atom marked as a stereocenter is not actually a valid stereocenter. This might occur because a trivalent nitrogen stereocenter was removed, which caused a remaining stereocenter to be invalid because of symmetry.

```
MDL-2010: Warning: Removed invalid Chiral flag
MDL-2011: Warning: Removed invalid non-tetrahedral stereo center(s)
MDL-2012: Warning: Removed invalid atom stereo center(s)
MDL-2129: Warning: Flattening 3D molecule to 2D
MDL-2134: Warning: Removed wedge from atom that is not a stereo center
MDL-2141: Warning: Removed invalid double-either bond(s)
MDL-5092: Warning: Added implicit hydrogens to metal atom
```

**Tip:** To preserve the original molecule or reaction, add a new column to the table and store the original molfile, rxnfile, or Chimestring in it.

## molchime

Converts a BLOB molecule object into a CLOB that contains the Chime string representation of a molecule.

### Syntax

```
molchime(mol)
```

Parameter	Description
<i>mol</i>	The name of the BLOB column that contains the molecules. mol can also be a BLOB that contains a molecule object, such as a molecule selected from some other table by a PL/SQL routine.

The name of the BLOB column that contains the molecules. mol can also be a BLOB that contains a molecule object, such as a molecule selected from some other table by a PL/SQL routine.



**Return value**

A CLOB that contains the Chime string representation of a molecule. `molchime` stores the return value in a temporary CLOB.

**Usage**

```
select molchime(mol)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example fetches a single molecule and writes it to a chime CLOB:

```
select molchime(molcolumn)
from moltable
where extreg='DLG-123';
```

**Comments**

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[Fetching Reactions Using the Chime Format](#)

BIOVIA Direct Developers Guide > Using Direct > Fetching Reactions

**molfile**

Returns a CLOB, an Oracle character large object, that contains the molfile representation of a molecule structure.

**Syntax**

```
molfile(mol)
```

Parameter	Description
<i>mol</i>	<p>Possible values:</p> <ul style="list-style-type: none"> <li>- The name of the BLOB column that contains the molecules. <code>mol</code> can also be a BLOB that contains a molecule object, such as a molecule selected from some other table by a PL/SQL routine.</li> <li>- A CLOB value that contains a Chime string to be converted to a molfile.</li> </ul> <p>Note: The <code>ctab</code> parameter cannot be a filename.</p>

**Return value**

A CLOB that contains the `molfile` representation of a molecule structure. The CLOB includes newline (\n) characters that separate the lines within the molfile.

**Usage**

```
select molfile(ctab)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses `molfile` to return the CLOB that contains the `molfile` representation of a molecule:

```
select molfile(ctab)
from sample2d
where cdbregno=20;
```

**Comments**

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

`molfile_string`  
`molfile_string_seg`

**molfile\_string**

Returns a string that contains the first 4000 characters of the `molfile` representation of a molecule structure.

**Note:** Direct provides this operator to emulate the `molfile_string` operator in the molecule cartridge prior to version 6.0.

**Syntax**

```
molfile_string(ctab)
```

Parameter	Description
ctab	Possible values: <ul style="list-style-type: none"> <li>■ The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The field value cannot be NULL.</li> <li>■ A molecule object</li> <li>■ A CLOB value that contains a Chime string to be converted to a molfile</li> </ul>

**Return value**

A `VARCHAR2` that contains the `molfile` representation of a molecule structure. The string includes newline (`\n`) characters that separate the lines within the `molfile`. If the structure exceeds 4000

characters, `molfile_string` returns NULL.

#### Usage

```
select molfile_string(ctab)
  [, other-column-data]
from tablename where condition;
```

#### Example

The following example uses `molfile_string` to return the variable-length string that contains the molfile representation of a molecule:

```
select molfile_string(ctab)
from sample2d
where cdbregno=20;
```

#### Comments

- If the content of the molfile exceeds 4000 characters, `molfile_string` returns an empty string.
- To retrieve larger structures, use `molfile`. If the application does not support CLOB data, iteratively use `molfile_string_seg` instead.

#### See also

[molfile](#)

[molfile\\_string\\_seg](#)

[Retrieving Molfile Structures](#)

### molfile\_string\_seg

Returns a string that contains a segment of the `molfile` representation of a molecule structure, up to 4000 characters at a time.

**Note:** Direct provides this operator to emulate the `molfile_string_seg` operator in the molecule cartridge prior to version 6.0.

#### Syntax

```
molfile_string_seg(ctab, start, stop)
```

Parameter	Description
<i>ctab</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>molfile_string_seg</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>molfile_string_seg</code> uses the global Ptable. The <code>molfile_string_seg</code> operator also uses the global Ptable if the value of the parameter is a molecule object. The field value cannot be NULL.
<i>start</i>	A number that indicates the starting position in the <code>molfile</code> . 1 is the first position.
<i>stop</i>	A number that indicates the ending position in the <code>molfile</code> . <code>start + 3999</code> is the maximum position. The difference between start and stop ( <code>stop - start</code> ) must not exceed 4000.

### Return value

A VARCHAR2 that contains a segment, up to 4000 characters long, of the molfile representation of a molecule structure. The string includes newline (\n) characters that separate the lines within the molfile.

### Usage

```
select molfile_string_seg(ctab,start,stop)
      [, other-column-data]
from   tablename
where  condition;
```

### Example

The following example uses molfile\_string\_seg to return the first 3000 characters of the molfile representation of a molecule:

```
select molfile_string_seg(ctab,1,3000)
from   sample2d
where  cdbregno=150;
```

The next example uses molfile\_string\_seg to return the next 3000 characters of the molfile representation of the same molecule:

```
select molfile_string_seg(ctab,3001,6000)
from   sample2d
where  cdbregno=150;
```

### Comments

- Use molfile\_string\_seg to retrieve structures whose molfile string exceeds 4000 characters. Starting at position 1, iteratively use molfile\_string\_seg to retrieve the segments of the molfile until the return value is empty or is shorter than the requested length.
- If the application that uses Direct supports string buffers that can contain more than 4000 characters, use the operator molfile instead of molfile\_string\_seg.

### See also

[molfile](#)

[molfile\\_string](#)

[Examples of Retrieving Molfile Structures](#)

## molformula

Returns the molecular formula for the given molecule.

### Syntax

```
molformula(molecule, 'format-options')
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>mol_fm1a</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>mol_fm1a</code> uses the global Ptable. The <code>mol_fm1a</code> operator also uses the global Ptable if the value of the parameter is a molecule object. The field value cannot be NULL.
<i>format-options</i>	Optional string to control formatting of the output molecular formula. If no options are present the formula will include space between elements, and the formula for each fragment in a multi-fragment structure will be separated by a dot. Use <code>NOSPACE</code> to remove the space between elements, <code>NOFRAGMENT</code> to not separate fragment formulas, and <code>NOSPACE NOFRAGMENT</code> for both changes. To match the formula string which is output by Insight use <code>NOSPACE NOFRAGMENT</code> .

**Return value**

A CLOB that contains the molecular formula for the given molecule.

**Usage**

```
select mol_fm1a(ctab)
      [, other-column-data]
from tablename
where condition;
```

**Example**

```
select mol_fm1a(ctab)
      from sample2d
      where cdbregno = 150;
```

**Comments**

- Do not use `mol_fm1a` in the WHERE clause of a SQL statement. To search for molecular formulas that meet specific criteria, use the `fm1a1ike` or `fmlamatch` operators instead.
- Do not use `mol_fm1a` in a DISTINCT, ORDER BY or GROUP BY clause. Because `mol_fm1a` returns a CLOB, you cannot use `mol_fm1a` in a DISTINCT, ORDER BY, or GROUP BY clause. For example, the query "SELECT DISTINCT MOLFMLA(CTAB) FROM MOLTABLE WHERE CDBREGNO=1" will return an error.
- It is more efficient to store the molecular formula in the molecule table and to retrieve the stored values than to recompute them at search time. Consider selecting the columns directly instead of specifying `mol_fm1a(ctab)`.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

- 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 is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.
- To match the formula output by Direct using the Molecular Formula component in Pipeline Pilot, set the component options as follows:
  - Ignore Isotopes = (True for MOLFMLA, False for ISOTOPICFORMULA)
  - Use 2H and 3H for Hydrogen Isotopes = True
  - Include Space Between Elements = (True for default, False if using NOSPACE)
  - Separate Fragments = (True for default, False if using NOGRAGMENT)
  - Add HTML Tags = False

#### See also

[isotopicformula](#)

## molgzip64

Converts a BLOB molecule object into a CLOB that contains the gzip compressed and base-64 encoded representation of a molecule.

#### Syntax

```
molgzip64(mol)
```

Parameter	Description
<i>mol</i>	The name of the BLOB column that contains the molecules. <i>mol</i> can also be a BLOB that contains a molecule object, such as a molecule selected from some other table by a PL/SQL routine.

#### Return value

A CLOB that contains the gzip compressed and base-64 encoded string representation of a molecule. `molgzip64` stores the return value in a temporary CLOB.

#### Usage

```
select molgzip64(mol)
    [, other-column-data]
from tablename
where condition;
```

#### Example

The following example fetches a single molecule and writes it to a compressed CLOB:

```
select molgzip64(molcolumn)
from moltable
where extreg='DLG-123';
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### molimage

Returns a temporary BLOB containing a PNG, BMP, SVG, or EMF image of the molecule rendered.

#### Syntax

`molimage(molecule [, options])`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	<p>Optional. A VARCHAR2 argument to control the type of image created, its size, and other preferences. Specify this argument as a string of comma separated options, each option takes the form keyword=value.</p> <p>Possible options are:</p> <ul style="list-style-type: none"> <li>■ <code>imagetype=png</code> - Creates a PNG image (default)</li> <li>■ <code>imagetype=bmp</code> - Creates a BMP</li> <li>■ <code>imagetype=svg</code> - Creates a SVG</li> <li>■ <code>imagetype=emf</code> - Creates a EMF image</li> <li>■ <code>width=number</code> - Specifies a width number, typically 100 to 1000 (default is 500)</li> <li>■ <code>height=number</code> - Specifies a height number, typically 100 to 1000 (default is 500)</li> <li>■ <code>ColorAtomsByType=TRUE   FALSE</code> - Specifies whether to color the atom labels by their type. The default is TRUE.</li> <li>■ <code>HydrogenDisplayMode=mode</code> - Specifies how to display implicit hydrogen atoms. The default is HYDROGEN_HETERO. Valid mode values are: <ul style="list-style-type: none"> <li>■ HYDROGEN_OFF - Does not display implicit hydrogen atoms</li> <li>■ HYDROGEN_HETERO - Displays implicit hydrogens on heteroatoms.</li> <li>■ HYDROGEN_TERMINAL - Displays implicit hydrogens on terminal atoms.</li> <li>■ HYDROGEN_TERMINAL_AND_HETERO - Displays implicit hydrogens on terminal atoms and heteroatoms.</li> <li>■ HYDROGEN_ALL - Displays implicit hydrogens on all atoms.</li> </ul> </li> <li>■ <code>BackgroundColor=color</code> - Specifies the background color. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is white.</li> <li>■ <code>ForegroundColor=color</code> - Specifies the color of the shape or text. Use either the</li> </ul>

Parameter	Description
	<p>name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is black.</p> <ul style="list-style-type: none"> <li>■ <code>ChiralityLabels=ANDtext,ABStext,ORtext,MIXEDtext</code> – Specifies the chirality label text to display for structures with stereocenters. The default is “AND Enantiomer,,OR Enantiomer,Mixed”. You must include the double quote characters and four comma-separated fields within the quotes. An empty field will not display anything for that type of chirality, for example the default text does not display anything for a structure that has only absolute stereocenters. ■</li> <li>■ <code>DisplayRS=TRUE   FALSE</code> – Specifies whether to display R and S stereocenter labels on the structure. The default is FALSE. ■</li> <li>■ <code>DisplayEZ=TRUE   FALSE</code> – Specifies whether to display E and Z double bond labels on the structure. The default is FALSE.</li> <li>■ <code>PolAtomDisplayMode=POL_STYLE_BEAD   POL_STYLE_TEXT</code> - Specifies how to indicate the atom(s) that bind the structure to a polymer (atoms of type ‘Pol’). Default is POL_STYLE_BEAD. Valid values are: <ul style="list-style-type: none"> <li>■ <code>POL_STYLE_BEAD</code> - Displays polymer atoms as shaded circles that resemble beads</li> <li>■ <code>POL_STYLE_TEXT</code> - Displays polymer atoms with the label text Pol.</li> </ul> </li> </ul> <p>The following shows an example that specify image options:  <code>molimage(mol, 'imagetype=png,width=100,height=100')</code></p>

### Return value

A BLOB that contains the binary image data. The `molimage` operator returns NULL if an image cannot be generated for the molecule. Use `mdl_aux.errors` to see related error message.

### Usage

```
select molimage(molecule[,options])
      [, other-column-data]
from tablename
where condition;
```

### Example

The following example inserts images into a new column for all molecules in a table:

```
alter table moltable add (imagefile blob);
update moltable set imagefile=molimage(ctab);
```

### Comments

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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 "blob":

```
if ( ((oracle.sql.BLOB)blob).isTemporary() ){
    ((oracle.sql.BLOB)blob).freeTemporary();
}
```



```
}
```

- Applications that use this function in a SQL SELECT statement must be aware that the temporary LOBs are only freed when the statement ends. If the statement selects many rows Oracle may run out of temporary space needed to store the LOBs. To work around this you can increase the temporary tablespace size, or you can convert the SELECT into a PL/SQL function which computes the image and then frees the BLOB immediately.

#### See also

[mdlaux.molimage](#)

## molkeys

Returns the SSS keys which would be registered for a molecule as a printable string.

#### Syntax

```
molkeys(molecule, print)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the molecule structures. <i>molecule</i> can also be a molecule object. If <i>molecule</i> is a molecule object, <i>molkeys</i> will use the global key definitions.
<i>print</i>	<p>Specifies what is to be printed, and how. The specified print string should follow the format "<i>outputFormat delimiterType</i>". It can contain the following keywords and options, separated by whitespace. Extra characters are ignored, thus 'DEC' or 'DECIMAL' would be allowed.</p> <p>Normally, the full set of 960 SSS keys are used. The following option can be added to restrict output to the subset of 166 "user" keys:</p> <p>SUB Subset of 166 "user" keys</p> <p>Output format:</p> <p>BIN - Binary, i.e. '1' and '0'. Delimiter is applied between bits.</p> <p>HEX - Hexadecimal, i.e. 'fa03'. Lowest key (key 1) is highest bit in first word, thus 'a000' would set keys 1 and 3. Delimiter is applied between 32-bit words.</p> <p>DEC - Decimal key numbers.</p> <p>WTS or WEI Decimal key weights. All key positions are output, keys which are not set have a weight of zero.</p> <p>SET - Returns only the number of keys set, not the key values.</p> <p>TOT - Returns only the total number of keys, not the key values. This is independent of the mol. Default: DEC</p> <p>Type and placement of delimiter character:</p> <ul style="list-style-type: none"> <li>■ DELIM=c - Output key values separated by 'c'.</li> <li>■ LEAD - Include a leading delimiter.</li> <li>■ TRAIL - Include a trailing delimiter.</li> </ul> <p>Default: None, unless option string is all blank.</p> <p>If the option string is " or 'SUB', the default is to add 'DEC DELIM=, LEAD TRAIL'.</p> <p>Examples:</p>

Parameter	Description	
	""	Same as "DEC DELIM=, LEAD TRAIL"
	"DEC DELIM=,	"23,47,230"
	"DEC SUBSET DELIM=, LEAD TRAIL	",2,6"
	"BIN"	,"00000101011101..."
	"TOT"	960" [number of SSS keys]

**Return value**

A VARCHAR2 that contains the SSS keys which would be registered for a molecule as a printable string

**Usage**

```
select molkeys(ctab, print)
      [, other-column-data]
from tablename
where condition;
```

**Comments**

- There is no package function identical to the molkeys operator, as it is indexed. However, there is a package function which accepts the index or table name as an argument: `mdlaux.molkeys`.
- Although molkeys is formally an indexed operator, there is no indexed implementation of it. Thus, the following search to find all molecules which have no keys will fail: `select extreg from moltable where MOLKEYS(molcol,'SET') = '0'`;
- If you want to use molkeys in a WHERE clause, you must force Oracle to use the non-indexed implementation. For example:
 

```
SELECT extreg FROM moltable WHERE MOLKEYS(molcol,'SET') || 'X' = '0X';
or
SELECT extreg FROM moltable WHERE TO_NUMBER(MOLKEYS(molcol,'SET')) = 0;
```
- 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.

**molnemaakey**

Returns a string that contains the NEMA key for the specified molecule structure.

**Syntax**

```
molnemaakey(molecule, option)
```

Parameter	Description
<i>molecule</i>	<p>The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.</p> <p>If the value of this parameter is a field name and the specified field has a domain index, <code>molnema</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>molnema</code> uses the global Ptable. The <code>molnema</code> operator also uses the global Ptable if the value of the parameter is a molecule object.</p>
<i>option</i>	<p>Specifies what to generate, and consists of one or more of the following three-letter keywords. The keywords are:</p> <ul style="list-style-type: none"> <li>■ CON - Returns constitutional NEMAKEY (30 characters). The constitutional key does not include any stereochemical information. It should be used to compare two molecules without regard to stereochemistry.</li> <li>■ STE - Returns stereochemical NEMAKEY (30 characters). This key includes stereochemical information for most cases, but will not differentiate mixtures of different types of enhanced stereochemical collections.</li> <li>■ EXA - Returns stereochemical NEMAKEY (30 characters). Returns no key if a FLEXMATCH verification is required. This occurs if the molecule contains mixtures of different types of enhanced stereochemical collections.</li> <li>■ FLG - Returns three (zero-padded) digits of NEMAKEY and cartridge flag information, as described below.</li> <li>■ REV - Returns the NEMAKEY revision number, for example "1001". This number will change if the NEMAKEY computed for a molecule might be different than for a previous release.</li> </ul> <p>The specified values are appended to the output string in the order in which the keywords appear in the <code>option</code> parameter. Any characters which are not keywords are appended to the output string as-is.</p> <p>Thus, to generate the stereo NEMAKEY followed by a dash followed by the flags, use 'STE-FLG' for the option parameter. If the <code>option</code> parameter is not present, is NULL, or is blank, then the default is to return 'EXA'. The <code>option</code> parameter is optional; if it is not specified, it is the same as if NULL were specified.</p> <div> <p><b>Note:</b> Do not include text which contains the letters in the option flags, the text will not be printed verbatim but will print the option with the remaining text. For example 'Stereo:ste' will print the stereo NEMAKEY twice separated by "reo:". Use Oracle text concatenation to add additional text to the NEMAKEY output.</p> </div> <p>In addition to the keywords above which specify what to include in the output string, there is another option which controls whether the generated NEMAKEY will be the same as the one created by Pipeline Pilot Client and Draw. This option is /NODAT and specifies that the NEMAKEY will not include information about data Sgroups. By default, Direct includes information about data Sgroups in its NEMAKEYs, which allows the keys to be used for exact-match comparisons. Pipeline Pilot Client and Draw do not include information about data Sgroups, so to create a NEMAKEY that matches the one in Pipeline Pilot Client or Draw use the options 'EXA/NODAT'.</p>

### Return value

A VARCHAR2 that contains the information generated in response to the keywords used in the `option` parameter.

### Usage

```
select molnmakekey(molecule, option)
      [, other-column-data]
from tablename
where condition;
```

### Example

The following example uses `molnmakekey` to return the NEMAKEKEY string for the structures in a substructure search:

```
select cdbregno,
       molnmakekey(ctab)
from acd2d_moltable
where sss(ctab, 'c:\query.mol')=1;
```

The following example is the same as the previous one but returns the same NEMAKEKEY that Pipeline Pilot Client or Draw would return for the structures:

```
select cdbregno,
       opicmolnmakekey(ctab, 'exa/nodat')
from acd2d_moltable
where sss(ctab, 'c:\query.mol')=1;
```

### Comments

- `molnmakekey` is an indexed operator equivalent of `mdlaux.molnmakekey`. `molnmakekey` will fetch the domain index schema and name from the information passed to it by Oracle, and then call `mdlaux.molnmakekey`.
- When the input molecule is a biopolymer sequence molecule, `molnmakekey` generates a special sequence NEMA key. Currently, only an exact NEMA key (considering stereochemistry) can be generated for sequences. Thus, if the input molecule is a sequence molecule:
  - The 'CON' option will return a blank string
  - Only the 'STE' option will return a NEMA key .

In addition, a flag value of 128 denotes a sequence NEMA key. This example shows the flag value of 128:

```
select cdbregno, molnmakekey(ctab, 'ste-flg') from moltable where
flexmatch(ctab, '/home/user/alanine.mol', 'all')=1;
```

```
CDBREGNO  MOLNMAKEKEY(CTAB, 'STE-FLG') -
```

```
-----
73  2GSUQPXFN9FEBYJHJJFQ65CXS58RF4-128
```

- The NEMAKEKEY will be empty if it cannot be generated because the molecule is a generic (contains Rgroups), or contains polymer Sgroups, or it contains numeric data Sgroups. It may also be empty if NEMA times out. Unless the FLG keyword is present, the return value will be NULL in these cases. If for example the `option` parameter was 'STE-FLG', and the molecule is a polymer, the output will be '-258', that is, the 'STE' portion is not present.

**Note:** Text data sgroups do not prevent generation of the NEMA key, and will contribute to the definition of the key.

- Flags are normally a bitwise OR of zero or more of the following:
  - 4: Molecule has mixed stereogroups and requires FLEXMATCH to resolve equality if stereochemistry is NOT ignored
  - 128: Molecule generated a (biopolymer) sequence NEMA key
- Flags can also be set to exactly one of the following, in this case the key is empty (no characters):
  - 257: NEMA generation failed due to timeout
  - 258: Molecule has polymer Sgroups or numeric data Sgroups
  - 259: Molecule is a generic (contains Rgroups)

#### See also

[mdl aux.molnema key](#)

BIOVIA Direct Developers Guide > Using Direct > NEMAKEY searching and key generation

## mol sim

Finds records that are structurally similar to the structure in your query, and returns the degree of similarity between the query and retrieved structures.

**Note:** Direct provides this operator to emulate the `mol sim` operator in the molecule cartridge prior to version 6.0. `mol sim` is equivalent to `similar`. See [similar](#) for more details.

#### Syntax

`mol sim(ctab, query, simtype)`

Parameter	Description
<code>ctab</code>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. The field value cannot be NULL.
<code>query</code>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

Parameter	Description
	<b>Notes:</b> <ul style="list-style-type: none"> <li>■ Molecules used in a similarity query must not include substructure query features.</li> <li>■ <i>query</i> cannot be NULL.</li> </ul>
<i>simtype</i>	<p>A string that may contain the keyword FINGERPRINT along with one of the following three values.</p> <ul style="list-style-type: none"> <li>■ NORMAL - Normal similarity. The retrieved molecules contain the same structural complexity as the query.</li> <li>■ SUB - Subsimilarity. The retrieved molecules are typically larger than the query structure, which is a substructure.</li> <li>■ SUPER - Supersimilarity. The retrieved molecules are typically smaller than the query structure, which is a superstructure.</li> </ul> <p>When FINGERPRINT is present, the user-defined fingerprints are used for the similarity search. If the keyword FINGERPRINT is not present, a traditional substructure key similarity search is executed.</p>

**Return value**

A number between 0 and 100 that indicates the degree of similarity between the query structure and the retrieved structure. The higher the value, the more similarity.

**Usage**

```
select column-data
from tablename
where molsim(ctab,query,simtype) between minimum and maximum;
select column-data
from tablename
where molsim(ctab,query,simtype)
      comparison-operator comparison-value;
```

The `mol`sim operator can also be used in the SELECT clause. Generally, this type of operation can be expected to be as slow as a non-indexed search. This operator can be used to determine the degree of similarity between two structures.

```
select molsim(ctab,query,simtype)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses `mol`sim to find the molecules within a range of degrees of similarity with the query structure:

```
select cdbregno from sample2d where molsim(
      ctab,
      (select ctab from sample2d where cdbregno=364),
      'normal'
      ) between 50 and 60;
```

The following example is the same but with user-defined fingerprint similarity:

```
select cdbregno from sample2d where molsim(
  ctab,
  (select ctab from sample2d where cdbregno=364),
  'fingerprint normal'
) between 50 and 60;
```

### Comments

- When you use `mol``sim` in a comparison expression, the `>` and `<` comparison operators are always inclusive. The SQL operator `>` is equivalent to `>=` (greater than or equal to), and `<` is equivalent to `<=` (less than or equal to).
- `mol``sim` is an indexed operator. If Direct cannot locate the domain index for the operator `mol``sim`, the search executes more slowly than an indexed `mol``sim`. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`.
- Avoid using this operator in both the `SELECT` clause and the `WHERE` clause of the same SQL statement. If you do so, Direct will compute the similarity value twice. Instead, use the `similar` and `similarity` operators.
- If the domain index is not used, no-structures are always treated as if the similarity is zero. Structures that set no keys at all, such as no-structures, are effectively skipped during the similarity search when the index is used; they never show up as hits even if the query is asking for zero percent similarity. In effect, a difference in search results can be seen between indexed and non-indexed (full table scan) searches. For example, the following indexed search returns 361 hits:

```
select count(*) from isismx2d_mol where
not (molsim(ctab, 'C1CCCCC1', 'normal') >= 80);
```

But the following non-indexed, full table scan search returns 364 hits:

```
select count(*) from
(select cdbregno from isismx2d_mol where
not (molsim(ctab, 'C1CCCCC1', 'normal') >= 80));
```

### See also

[similar](#)

[Similarity Search](#)

[Examples of Molecule Similarity Search](#)

[Specifying the Query Structure](#)

### molwt

Returns the molecular weight of a molecule.

### Syntax

```
molwt(molecule)
```

Parameter	Description
<i>molecule</i>	<p>The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.</p> <p>If the value of this parameter is a field name and the specified field has a domain index, <code>molwt</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>molwt</code> uses the global Ptable. The <code>molwt</code> operator also uses the global Ptable if the value of the parameter is a molecule object.</p> <p>The field value cannot be NULL.</p>

**Return value**

A NUMBER that contains molecule weight of the specified molecule.

**Usage**

```
select molwt(ctab)
      [, other-column-data]
from tablename
where condition;
```

**Example**

```
select molwt(ctab)
      from sample2d
      where cdbregno = 150;
```

**Comments**

- Do not use `molwt` in the WHERE clause of a SQL statement. Although `molwt` is formally an indexed operator, there is no indexed implementation of it. `molwt` will return an error if a search such as this is attempted:  

```
SELECT extreg FROM moltable WHERE MOLWT(molcol) < 100.0;
```
- It is more efficient to store the molecular weight in the molecule table and to retrieve the stored values than to recompute them at search time. Consider selecting the columns directly instead of specifying `molwt(ctab)`.
- There is no package function identical to the `molwt` operator, as it is indexed.
- 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.

**See also**

[monoisotopicmass](#)

**molwtmax**

Returns the maximum molecular weight for a generic structure, that is, the molecular weight of the heaviest enumerated specific structure of the generic structure. If the given molecule is a specific structure, `molwtmax` is the same as `molwt`.

**Syntax**

```
molwtmax(molecule)
```



Parameter	Description
<i>molecule</i>	<p>The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.</p> <p>If the value of this parameter is a field name and the specified field has a domain index, <code>molwt</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>molwt</code> uses the global Ptable. The <code>molwt</code> operator also uses the global Ptable if the value of the parameter is a molecule object.</p> <p>The field value cannot be NULL.</p>

**Return value**

A NUMBER that contains maximum molecular weight of a generic structure

**Usage**

```
select molwtmax(molecule)
      [ ,other-column-data]
from tablename
where condition;
```

**Example**

```
select molwtmax(ctab)
      from samplegen
      where parent_sampleid = 'BENZ';
```

**Comments**

Do not use `molwtmax` in the WHERE clause of a SQL statement. Although `molwtmax` is formally an indexed operator, there is no indexed implementation of it. `molwtmax` will return an error if a search such as this is attempted:

```
select parent_sampleid from samplegen where molwtmax(ctab) < 430.0;
```

**See also**

[mdlaux.molwtmax](#)

[molwtmin](#)

**molwtmin**

Returns the minimum molecular weight for a generic structure, that is, the molecular weight of the lightest enumerated specific structure of the generic structure. If the given molecule is a specific structure, `molwtmin` is the same as `molwt`.

**Syntax**

```
molwtmin(molecule)
```

Parameter	Description
<i>molecule</i>	<p>The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.</p> <p>If the value of this parameter is a field name and the specified field has a domain index, <code>molwt</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>molwt</code> uses the global Ptable. The <code>molwt</code> operator also uses the global Ptable if the value of the parameter is a molecule object.</p> <p>The field value cannot be NULL.</p>

**Return value**

A NUMBER that contains minimum molecular weight of a generic structure

**Usage**

```
select molwtmin(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

```
select molwtmin(ctab)
      from samplegen
      where parent_sampleid = 'Peptoid';
```

**Comments**

Do not use `molwtmin` in the WHERE clause of a SQL statement. Although `molwtmin` is formally an indexed operator, there is no indexed implementation of it. `molwtmin` will return an error if a search such as this is attempted:

```
select parent_sampleid from samplegen where molwtmin(ctab) < 360.0;
```

**See also**

[mdlaux.molwtmin](#)

[molwtmax.htm](#)

**molwtrange**

Returns the molecular weight as a NUMRANGE object with two members. If the molecule argument is a generic structure, the two members contain the minimum and maximum molecular weight. Otherwise, if the molecule argument is a specific structure, the two members are identical.

**Syntax**

```
molwtrange(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <i>molwtrange</i> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <i>molwtrange</i> uses the global Ptable. The <i>molwtrange</i> operator also uses the global Ptable if the value of the parameter is a molecule object. The field value cannot be NULL.

**Return value**

A NUMRANGE object that contains the molecular weight with two members. If the molecule argument is a generic structure, the two members contain the minimum and maximum molecular weight. Otherwise, if the molecule argument is a specific structure, the two members are identical.

**Usage**

```
select molwtrange(ctab)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example gets the *molwtrange* of a structure.

```
select molwtrange(ctab)
      from sample2d
      where cdbregno = 150;
```

The following example queries the *molwtrange* of generic and non-generic (specific) structures in the *samplegen* table:

```
select parent_sampleid,
       sampleid,
       isgeneric(ctab) isgeneric,
       molwtrange(ctab) molweight
from samplegen
where parent_sampleid = 'Lib1';
```

The following shows the result of the preceding example. Note that for generic structures (*isgeneric* (ctab) = 1), the *molwtrange* displays minimum and maximum values; for specific structures (*isgeneric* (ctab) = 0), the *molwtrange* displays identical values:

PARENT\_SAMPLEID SAMPLEIDISGENERIC MOLWEIGHT(MIN, MAX)

Li b1	CH3-Ph- C1	0 NUMRANGE(248.70812, 248.70812)
Li b1	CH3-R2- R3	1 NUMRANGE(248.70812, 265.30826)

Li b1	H-Chx-R3	1 NUMRANGE(240.72918, 251.28168)
Li b1	H-Ph-R3	1 NUMRANGE(234.68154, 245.23404)
Li b1	H-R2-R3	1 NUMRANGE(234.68154, 251.28168)
Li b1	Lib1-1	1 NUMRANGE(234.68154, 341.40422)
Li b1	Ph-Ph- NO2	0 NUMRANGE(335.35658, 335.35658)
Li b1	Ph-R2-R3	1 NUMRANGE(324.80408, 341.40422)

8 rows selected.

#### Comments

- There is no package function identical to the `molwtrange` operator, as it is indexed.
- It is more efficient to store the molecular weight in the molecule table and to retrieve the stored values than to recompute them at search time. Consider selecting the columns directly instead of specifying `molwtrange(ctab)`.
- When inserting the molecular weight for a generic structure, you must use the `mdlaux.molwtmin` and `mdlaux.molwtmax` functions instead of the `molwtrange` operator. The `mdlaux.molwtmin` and `mdlaux.molwtmax` functions allow you to specify a molecule table or molecule domain index that specifies which Ptable to use. The `molwtrange` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.

### monoisotopicmass

Computes the mono-isotopic mass of a molecule.

#### Syntax

`monoisotopicmass(molecule)`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. If the value of this parameter is a field name and the specified field has a domain index, <code>monoisotopicmass</code> uses the local Ptable associated with that domain index. If the specified field does not have a domain index, <code>monoisotopicmass</code> uses the global Ptable. The <code>monoisotopicmass</code> operator also uses the global Ptable if the value of the parameter is a molecule object.

#### Return value

A NUMBER that contains the mono-isotopic mass of the molecule, or NULL if the mass cannot be computed. The domain index Ptable is automatically used to resolve atom symbols.

**Usage**

```
select monoisotopicmass(molecule)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses `monoisotopicmass` to return the mono-isotopic mass of the molecules in a table:

```
select cdbregno, molwt(ctab),
      monoisotopicmass(ctab)
from acd2d_moltable;
```

**Comments**

- The `monoisotopicmass` operator cannot be used for registration. Use the `mdlaux.monoisotopicmass` function when registering mono-isotopic mass into a table.
- The `monoisotopicmass` operator only provides a result when the molecule contains atoms found in nature. If an input molecule contains pseudoatoms such as Pol, Mod, or X, the `monoisotopicmass` operator returns NULL and the following error:  
MDL-1590: MonoisotopicMass failed: Not available

**See also**

[mdlaux.monoisotopicmass](#)

[molwt](#)

**numspecifics**

Returns the number of specific structures from enumerating the specified generic structure. A specific structure is a fully defined chemical structure (with no generic features). The `numspecifics` operator returns 1 if the specified argument is a specific structure.

**Syntax**

```
numspecifics(molecule)
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

**Return value**

A NUMBER (1 or higher) that indicates the number of specific structures from enumerating the generic structure.

**Usage**

```
select column-data,
      numspecifics(molecule)
from tablename
where [conditions];
```

**Example**

The following example returns the number of specific structures for the generic structures in the samplegen table.

```
select parent_sampleid,
       numspecifics(ctab)
from samplegen
where isgeneric(ctab)=1;
```

**Comments**

This operator is not indexed. When possible, avoid using it when running queries on large tables.

**See also**

[mdlaux.numspecifics](#)

**overlap**

Finds all generics or specifics in the table which contain at least one enumerated specific in common with the query. The overlap operator uses Fastsearch.

**Syntax**

```
overlap(molecule, query [,overlap-number])
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the molecule structures. <i>molecule</i> can also be a molecule object. If a molecule object is specified, the global Ptable, salts file and key definition files will be used during searching.
<i>query</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> <li>■ Note: <i>query</i> cannot be NULL.</li> </ul>
<i>overlap-number</i>	A number that is equal to the overlap-number parameter used with the overlaptimeout and pctoverlap operators.

**Return value**

The NUMBER 1 indicates that the query matched one or more records. When you use overlap in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where overlap(ctab, query)=1
      [operator other-conditions];
```

**Example**

The following example uses a generic database, a sample database named SAMPLEGEN, database libraries of benzodiazepines, as well as some specific structures from those libraries. In the following example, the query molfile is one of the enumerated specifics.

```
select parent_sampleid,
       sampleid,
       numspecifics(ctab),
       pctoverlap(1)
from samplegen
where overlap(ctab, '/test/benzo1.mol',1)=1
order by pctoverlap(1) desc;
```

The overlap results show that the query molfile hits itself with 100% overlap, and hits various generic structures that contain it:

PARENT_SAMPLEID	SAMPLEID	NUMSPECIFICS(CTAB)	PCTOVERLAP(1)
-----	-----	-----	-----
BENZ	Ph-Chx-NO2-Me	1	100
BENZ	Ph-R2-R3-R4	18	5.55556
BENZ	R1-R2-R3-Me	45	2.22222
BENZ	BENZ - 1	90	1.11111

**Comments**

- To negate the results of overlap, use the SQL operator NOT. For example:  

```
select count(*) from sample2d
where not overlap(ctab, 'c=c-c=c-c-c-@1')=1;
```
- overlap is an indexed operator. If Direct cannot locate the domain index for the operator overlap, the search executes more slowly than an indexed overlap. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command EXPLAIN PLAN.
- A generic overlap search requires a generic structure domain index. This index allows both generic and specific structures to be registered. When the generic and specific structures are registered, they can be searched using the overlap operator. If you use the overlap operator without a generic structure domain index on the table, the overlap operator returns the following error:  
ORA-20100: MDL-0885: Cannot perform a OVERLAP search against a non-generic molecule index

- The overlap operator can be used to compare two tables of generic molecules. The overlap operator can be used to perform library comparison. The following SQL statement compares a list of source structures (as a table) with a target database:

```
CREATE TABLE HITTABLE (source_idcolumn type, target_idcolumn type);

INSERT INTO HITTABLE
  SELECT /*+ ORDERED INDEX(TRG targettable_domainindex) USE_NL(TRG) */
    SRC.idcolumn, TRG.idcolumn
  FROM sourcetable SRC, targettable TRG
  WHERE OVERLAP(TRG.CTAB, SRC.querycolumn)=1;
```

- Use hints to force Oracle to use the domain index in an overlap search. When using the overlap search to compare tables, use hints to force Oracle to use the domain index in the search. The overlap search will be very slow when the domain index is not used. The ORDERED, INDEX, and USE\_NL hints in the following example cause the domain index to be used in most cases. (Use Oracle's EXPLAIN PLAN to verify that the query uses the domain index in your specific case.)

```
INSERT INTO RESULTTABLE
  SELECT
    /*+ ORDERED INDEX(TRG targettable_domainindex) USE_NL(TRG) */
    SRC.idcolumn,
    TRG.idcolumn,
    PCTOVERLAP(1) AS PERCENT_OVERLAP FROM
  sourcetable SRC, targettable TRG WHERE
  OVERLAP(TRG.CTAB, SRC.CTAB, 1)=1;
```

- Do not combine an overlap search with another query in the same WHERE clause. Do not combine use the overlap operator with another WHERE clause condition to refine the overlap search within the same query. For example, you want to determine the overlap between a generic query and all of the libraries in a database which have a parent\_sampleid that starts with 'BENZ'. You might be very tempted to use this query:

```
select parent_sampleid, pctoverlap(1) from mylibdb
where parent_sampleid like 'BENZ%'
and overlap(ctab, '/myquery.mol', 1)=1
```

In the previous example query, Oracle might not use the domain index to run the overlap search in this case. If it does not, and there are quite a few libraries that start with 'BENZ', the search could take a long time to run. The one-on-one overlap matching can be VERY slow, much slower than say a one-on-one sss or flexmatch of two non-generics!

Instead of using a combined query in a single SELECT statement, you should further refine your search (if needed) by running a second search. For example:

-- This search uses ONLY an overlap

```
insert into libhits
  select parent_sampleid, pctoverlap(1) as pctoverlap from mylibdb
  where overlap(ctab, '/myquery.mol', 1)=1;
```

-- Further refinement is done here

```
select parent_sampleid, pctoverlap from libhits
where parent_sampleid like 'BENZ%';
```

In this example, Oracle will use the domain index in the search. If Oracle performs a full table scan (instead of using the domain index), the search might take much longer than expected.



**See also**

*BIOVIA Direct Developers Guide > Using Direct > Searching generic structures*

*BIOVIA Direct Developers Guide > Using Direct > Biopolymer Searching and Registration*

*BIOVIA Direct Developers Guide > About Direct > Direct domain index and the Oracle optimizer*

**overlaptimeout**

Returns the timeout status value from an `overlap` search.

`overlap` will return as matches those candidates for which the matching algorithm times out. Such candidates might or might not be actual matches. This ancillary operator gives information about that timeout status.

**Syntax**

`overlaptimeout(overlap-number)`

Parameter	Description
<i>overlap-number</i>	A NUMBER equal to the <code>overlap-number</code> parameter that is used with the <code>overlap</code> operator.

**Return value**

A NUMBER that indicates the status of the substructure search. The possible values are:

Value	Description
0	The <code>overlap</code> search did not time out.
1	The <code>overlap</code> search timed out.
NULL	The target was not a match to the query.

**Usage**

```
select overlaptimeout(overlap-number)
  [, other-column-data]
from tablename
where overlap(mol, query, overlap-number)=1
  [operator other-conditions];
```

**Example**

The following example returns the timeout status while performing an overlap search.

```
select extreg,
  overlaptimeout(3) "Timeout"
from moltable
where overlap(
  molcol,
  '/opt/BIOVIA/direct/examples/rxnfiles/query1.mol',
  3
)=1;
```

**Note:** The number 3 is used to correlate the `overlap` operator in the WHERE clause with the `overlaptimeout` operator in the SELECT clause. This could be any number as long as the values in the two operators match.

### Comments

The `overlap-number` parameters for `overlaptimeout` and `overlap` operators must match. If the `overlap-number` parameters do not match, or if you use `overlaptimeout` without using `overlap`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

### See also

[overlap](#)

## pctoverlap

An ancillary operator that returns the percentage estimated overlap between the query and a hit.

### Syntax

`pctoverlap(overlap-number)`

Parameter	Description
<i>overlap-number</i>	A NUMBER equal to the <code>overlap-number</code> parameter that is used with the <code>overlap</code> operator.

### Return value

A NUMBER between 0 and 100 that contains an estimated percentage overlap between the query and a hit. It is only an estimate of the number of enumerated specific structures in common between the query and hit. Because enumeration is not actually performed the estimated value may be incorrect especially when the query or target has hydrogen Rgroup members.

### Usage

```
select pctoverlap(overlap-number)
      [, other-column-data]
from tablename
where overlap(mol, query, overlap-number)=1
      [operator other-conditions];
```

### Example

The following example returns the timeout status while searching the table.

```
select extreg,
       pctoverlap(3) "% Overlap "
from moltable
where overlap(
  molcol,
  '/opt/BIOVIA/direct/examples/rxnfiles/query1.mol',
  3
)=1;
```

**Note:** The number 3 is used to correlate the `overlap` operator in the `WHERE` clause with the `sssttimeout` operator in the `SELECT` clause. This could be any number as long as the values in the two operators match.

### Comments

The `overlap-number` parameters for `pctoverlap` and `overlap` operators must match. If the `overlap-number` parameters do not match, or if you use `pctoverlap` without using `overlap`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

### See also

[overlap](#)

## readmol

Reads a `molfile` from disk, and returns a CLOB representation of a molfile that can be used as a query structure.

Direct provides this operator to emulate the `readmol` operator of the molecule cartridge prior to version 6.0. `readmol` is equivalent to `readfile`. See [readfile](#) for details.

### Syntax

`readmol(molfile)`

Parameter	Description
<i>molfile</i>	A string that contains the full path and the name of the <code>molfile</code>

### Return value

A CLOB data that represents the contents of the `molfile`

### Usage

```
select readmol('/disk-location/file-location/filename.mol')
from dual;
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## sequencesearch

Finds records with biopolymer sequence text that match your query.

### Syntax

`sequencesearch(ctab, oracle-operator, query [, flags])`

Parameter	Description
<i>ctab</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. The field value cannot be NULL.
<i>oracle-operator</i>	A string that specifies the type of matching, either LIKE or REGEXP_LIKE. The string cannot be NULL.
<i>query</i>	A string that contains query text. If using 'like', the query may contain the wildcard characters '%' and '_'. If using 'regexp_like' the string is an Oracle regular expression, use flags for additional control over matching. The string cannot be NULL.
<i>flags</i>	An optional string that contains additional flags for the REGEXP_LIKE function. The string may be NULL. If the value of oracle-operator is 'like', Direct executes a SQL query similar to: <ul style="list-style-type: none"> <li>■ select regno from indexname_bsq where sequence like 'query';</li> </ul> If the value of oracle-operator is 'regexp_like', Direct executes a SQL query similar to: <ul style="list-style-type: none"> <li>■ select regno from indexname_bsq where regexp_like(sequence, 'query');</li> <li>■ select regno from indexname_bsq where regexp_like(sequence, 'query', 'flags');</li> </ul>

**Return value**

The NUMBER 1 indicates that the query matched one or more records. When you use sequencesearch in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where sequencesearch(ctab, op query[flags])=1
  [operator other-conditions];
```

The sequencesearch operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search.

**Example**

The following example uses sequencesearch with the REGEXP\_LIKE operator to find biopolymer sequences that contain unmodified amino acids isoleucine (I), cysteine (C), and either glutamate (E) or glutamine (Q) anywhere in the text string.

```
select cdbregno,
       sequencetext(ctab)
from samplebio
where sequencesearch(ctab, 'regexp_like', '.*IC[EQ].*')=1;
```

**Comments**

Oracle's LIKE and REGEXP\_LIKE searches typically cannot use a native Oracle index and may be slow if the table contains many rows of sequence text data. If the sequencesearch search is too slow, consider using an external BLAST search engine for sequence text searching.

To negate the results of sequencesearch, use the SQL operator NOT.sequencesearch is an indexed

operator. If Direct cannot locate the domain index for the operator `sequencesearch`, the search executes more slowly than an indexed `sequencesearch`. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`.

## sequencetext

Generates the biopolymer sequence text for a molecule.

### Syntax

`sequencetext(ctab)`

Parameter	Description
<code>ctab</code>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object. The field value cannot be NULL.

### Return value

A CLOB containing the biopolymer sequence text, a string of single letters representing the individual monomers, for example amino acids.

The operator returns NULL if the structure is not an SCSR sequence molecule.

### Usage

```
select sequencetext(ctab) from tablename where condition;
```

### Example

```
select sequencetext(ctab) from human_sequence where name = 'CT18_HUMAN';
```

```
SEQUENCETEXT(CTAB)
```

```
-----
```

```
-----
```

```
MSPPSSMCSPVPLLAASGQNRMTQGQHFLQKV
```

## similar

Molecule similarity search

### Syntax

`similar(molecule, query, similarity-type-values, [similar-number])`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the molecule structures. <i>molecule</i> can also be a molecule object. If a molecule object is specified, the GLOBAL key definition files will be used to generate the keys used during searching.
<i>query</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> </ul>

Parameter	Decription
	<ul style="list-style-type: none"> <li>■ Direct molecule object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ InChI string</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul> <div> <b>Notes:</b> <ul style="list-style-type: none"> <li>■ Molecules used in a similarity query must not include substructure query features.</li> <li>■ <i>query</i> cannot be NULL.</li> </ul> </div>
<i>similarity-type-values</i>	<p>A VARCHAR2 containing a flag specifying a fingerprint search instead of a traditional similarity search, the similarity threshold values (as percentages), and an optional 'SUB' or 'SUPER' flag.</p> <p>For a traditional similarity search the substructure keys define the features that are compared. Only include the threshold values and 'SUB' or 'SUPER' flag:</p> <p>'min [SUB SUPER] '</p> <p>or</p> <p>'min max [SUB SUPER] '</p> <p>For example, '60 SUB'.</p> <p>For a fingerprint similarity search the features are defined by the user as either Accord fingerprints or Scitegic fingerprints. Add the 'FINGERPRINT' flag to the value:</p> <p>'fingerprint min [SUB SUPER] '</p> <p>or</p> <p>'fingerprint min max [SUB SUPER] '</p> <p>For example, 'FINGERPRINT 0 20'.</p> <p>If the third argument is blank or NULL, a traditional similarity search with a threshold value of '80' is used.</p> <p>The min and max values must range from 0 to 100, with min &lt;= max. The first form is equivalent to specifying max as 100 in the second form. For example, to return all hits that are at least 80% similar, use '80'. To return all hits that are very dissimilar, for example 5% or less similarity, use '0 5'.</p> <p>The normal similarity calculation uses the Tanimoto coefficient. (For a traditional similarity search set intersection/union set counts are replaced with the sum of the key weights for the bits in the set.) The 'SUB' calculation uses the features of the query in the denominator, thus corresponds to a "substructure similarity"—you will get close to 100% similarity if the query exists as a substructure within the candidate. The 'SUPER' calculation is the reverse—it uses the features of the candidate in the denominator, thus corresponds to a "superstructuresimilarity"—you will get close to 100% if the candidate exists as a substructure within the query.</p>
<i>similar-number</i>	<p>A number that is equal to the similar-number parameter used with the similarity operator. This parameter only applies if you use the similarity operator.</p>

**Return value**

The NUMBER 1 indicates that the query matched one or more records. When you use similar in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where similar(molecule,query,similarity-type-values)=1
[operator other-conditions];
select similarity(similar-number)
[, other-column-data]
from tablename
where similar(molecule,query,similarity-type-values)=1
[operator other-conditions];
```

**Example**

The following example searches for similar molecules.

```
select extreg
from moltable
where
similar(molcol, '/home/user/query.mol', '80') = 1;
```

This next example uses the user-defined fingerprint values to find similar molecules.

```
select extreg
from moltable
where
similar(molcol, '/home/user/query.mol', 'fingerprint 80') = 1;
```

**Comments**

If the domain index is not used, no-structures are always treated as if the similarity is zero. Structures that set no keys at all, such as no-structures, are effectively skipped during the similarity search when the index is used. They never show up as hits even if the query is asking for zero percent similarity. In effect, a difference in search results can be seen between indexed and non-indexed (full table scan) searches.

For example, the following indexed search returns 361 hits:

```
select /*+ index(isismx2d_mol ISISMX2D_MOL_DMNIDX) */ count(*)
from isismx2d_mol where
(similar(ctab, 'C1CCCCC1', '0 80') = 1);
```

But the following non-indexed, full table scan search returns 364 hits:

```
select /*+ full(isismx2d_mol) */ count(*)
from isismx2d_mol where
(similar(ctab, 'C1CCCCC1', '0 80') = 1);
```

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.

## similarity

Return similarity value from a similarity (similar) search. The similar operator always returns 1 (to indicate that the structure matched the query criteria) or 0 (to indicate that the structure did not match the query criteria). The similarity ancillary operator returns information about the actual degree of similarity.

### Syntax

`similarity(similar-number)`

Parameter	Description
<i>similar-number</i>	A NUMBER equal to the similar-number parameter that is used with the similar operator.

### Return value

A NUMBER that indicates the percentage similarity value ranging from 0.0 to 100.0.

### Usage

```
select similarity(similar-number)
      [, other-column-data]
from tablename
where similar(mol, query, similarity-values, similar-number)=1
      [operator other-conditions];
```

### Example

The following example returns the similarity values after a search.

```
select extreg,
       similarity(3) "Similarity"
from moltable
where similar(molcol,
             '/home/user/query.mol',
             '80 sub',
             3) = 1;
```

**Note:** The number 3 is used to correlate the similar operator in the WHERE clause with the similarity operator in the SELECT clause. This could be any number as long as the values in the two operators match.

### Comments

The similar-number parameters for similarity and similar operators must match. If the similar-number parameters do not match, or if you use similarity without using similar, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

### See also

[similar](#)

## smiles

Returns a SMILES string representation of a molecule.



**Syntax**

```
smiles(molecule, 'noncanonical')
```

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>noncanonical</i>	(Optional) To generate noncanonical SMILES strings, use the argument 'noncanonical'. The default SMILES string is canonical.

**Return value**

A temporary CLOB that contains the SMILES string. The smiles operator returns NULL if the SMILES string cannot be generated. Use `mdaux.errors` to see the related error message.

**Usage**

```
select smiles(molecule)
      [, other-column-data]
from tablename
where condition;
```

```
select smiles(molecule, 'noncanonical')
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example shows the SMILES string for the molecules in a table:

```
select cdbregno, smiles(ctab) smiles from acd2d_moltable
where cdbregno between 1001 and 1010 order by cdbregno;
CDBREGNO SMILES
```

```
-----
1001 C\CCCCOc1ccccc1
1002 CC(=O)OCCCCC\
1003 C\CCCC#C
1004 CCCCCC\
1005 C\CCCCCBr
1006 C\CCCCC\
1007 CCCCCC\
1008 C\CCCCCBr
1009 C\CCCCC\
1010 CCCCCC\
```

10 rows selected.

The following example shows the noncanonical SMILES string for the molecules in a table:

```
select cdbregno, smiles(ctab, 'noncanonical') noncanonical_smiles
from acd2d_moltable
where cdbregno between 1001 and 1010 order by cdbregno;
CDBREGNO NONCANONICAL_SMILES
```

```

-----
1001 c1(ccccc1)OCCCCC1
1002 C(CCCC1)OC(=O)C
1003 C(C#C)CCC1
1004 C(CC)CCC1
1005 C(CCB r)CCC1
1006 C(CCC1)CCC1
1007 C(CCC1)CCC
1008 C(CCC1)CCCB r
1009 C(CCC1)CCCC1
1010 C(CCC)CCCC1

```

10 rows selected.

### Comments

- There are limitations to the generation of SMILES strings. Not all BIOVIA molecule features can be handled. The smiles operator returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}

```

### See also:

[mdlaux.smiles](#)

*BIOVIA Direct Developers Guide > Using Direct > Getting the SMILES String*

*BIOVIA Direct Developers Guide > Using Direct > Limitations to the Generation of SMILES Strings*

### SSS

Finds structures that contain the substructure that you specify in the query. A substructure is a portion of a larger molecule structure.

### Syntax

`sss(molecule, query [, option] [, sss-number])`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the molecule structures. molecule can also be a molecule object.
<i>query</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"><li>■ Direct molecule object (BLOB)</li><li>■ IUPAC name (VARCHAR2 or CLOB)</li><li>■ HELM string (VARCHAR2 or CLOB)</li><li>■ XHELM string (VARCHAR2 or CLOB)</li><li>■ Chime string (VARCHAR2 or CLOB)</li><li>■ SMILES string (VARCHAR2 or CLOB)</li><li>■ InChI string (VARCHAR2 or CLOB)</li><li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li></ul>

Parameter	Description
option	<p>Any or a combination of the following values, separate multiple options with a space:</p> <ul style="list-style-type: none"> <li>■ <b>NOFS</b> - The presence of 'NOFS' causes sss to refrain from using the fastsearch index when performing the search. If 'GENERICS' is also specified, the 'NOFS' option is ignored.</li> <li>■ <b>ORIEN</b> - The presence of 'ORIEN' causes ssshight to return the highlighted structure oriented to match the query.</li> <li>■ <b>GENERICS (or GENERIC)</b> - Causes sss to accept the query as a specific query molecule, and use Fastsearch to perform a substructure search of both generic and specific structures.</li> </ul> <p>For specific target structures, a normal substructure search is performed. For generic target structures, a substructure search is performed on a table of all enumerated specifics for that generic. However, the substructure search does not perform enumeration and is much faster than this equivalent set of sss searches.</p> <p>The query parameter <i>cannot</i> contain any of the following features (all of these are usable in a normal, specific structure, substructure search):</p> <ul style="list-style-type: none"> <li>■ Rgroups (that is, Markush features)</li> <li>■ Polymer Sgroups</li> <li>■ Data Sgroups</li> <li>■ Link nodes</li> <li>■ Atom lists containing hydrogen</li> </ul> <p>If 'GENERICS' is specified with 'NOFS', 'NOFS' is ignored.</p> <ul style="list-style-type: none"> <li>■ <b>IgnoreChargesInPiSystems</b> – When absent, substructure mapping of pi systems, i.e. of haptic bonds, takes total charge into account and does not allow a query that has a charged pi system or metal connected to the pi system to map to a target which is uncharged. When the option is present, total charge in the pi system and metal attached to the pi system is ignored in both query and target. This allows the radical representation of ferrocene to map to the charged representation of ferrocene; without the option these two do not match.</li> <li>■ <b>RingHomologyGroupsOnlyMapTerminalRings</b> – When this option is present, ring homology query atoms maps only to terminal ring assemblies in the target. The atoms does not map to ring assemblies that have any non-ring attachments.</li> <li>■ <b>InterpretRAtomsLiterally</b> - When this option is present an R atom in the query matches only an R atom in the target. It does not have its normal meaning of matching any atom including hydrogen.</li> <li>■ <b>InterpretXAtomsLiterally</b> - When this option is present an X atom in the query matches only an X atom in the target. It does not have its normal meaning of matching any atom including hydrogen.</li> <li>■ <b>InterpretQueryAtomsLiterally</b> - When this option is present an A, Q, X or M atom in the query matches only a corresponding A, Q, X or M atom in the target. It does not have its normal meaning as an atom query feature.</li> <li>■ <b>MatchTautomers</b> - Adding this option expands the number of structures which match the query to include structures which would match a tautomer of the query.</li> </ul>

Parameter	Description
	<p>When this option is present tautomers are generated for those parts of the query which do not contain traditional substructure query features or aromatic atoms and bonds. If one or more tautomers can be computed the query molecule has substructure query features added to it so that it matches all of the possible tautomers. Post-processing ensures that matches that are not tautomers are rejected.</p> <ul style="list-style-type: none"> <li>■ <b>IgnoreStereo</b> - When this option is present all atom stereochemistry in the query are ignored during matching. This includes enhanced stereochemistry and higher-order stereochemistry. Double bond stereochemistry is still matched when marked in the query.</li> <li>■ <b>IgnoreTerminalPhosphates</b> - Adding this option allows different representations of RNA or DNA sequences to match each other. When this option is present, any 3' or 5' terminal phosphates are removed from the query nucleic acid sequence, allowing it to match a target irrespective of whether the target contains 3' or 5' terminal phosphates. Without this option, the HELM query 'RNA1{R(A)P}\$\$\$\$' does not match an adenine nucleotide created with BIOVIA Draw because the Draw structure is missing the 3' phosphate present in the query.</li> </ul>
<i>sss-number</i>	A number that is equal to the sss-number parameter used with the ssshhighlight, ssstimeout, sss_highlight_molfile, and sss_highlight_chime operators. This parameter only applies if you use these other operators.

### Return value

The NUMBER 1 indicates that the query matched one or more records. When you use sss in a WHERE clause, always test the return value for a result of 1.

### Usage

```
select column-data
from tablename
where sss(ctab,query)=1
  [operator other-conditions];
```

```
select ssshhighlight(sss-number)
  [, other-column-data]
from tablename
where sss(ctab,query,sss-number)=1
  [operator other-conditions]:
```

```
select ssstimeout(sss-number)
  [, other-column-data]
from tablename
where sss(ctab,query,'ORIEN',sss-number)=1
  [operator other-conditions];
```

The sss operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a structure is really an sss search hit from a complex WHERE clause.

```
select sss(ctab,query)
      [, other-column-data]
from tablename
where condition;
```

### Example

The following example uses sss to find and count the molecules that contain a substructure.

```
select count(*)
from sample2d
where sss(ctab, '/home/user/substruct.mol')=1;
```

### Comments

- To negate the results of sss, use the SQL operator NOT. For example, to find all molecules that do not contain a benzene ring:

```
select count(*) from isis.isisrc2d_mol
where not sss(ctab, 'c=c-c=c-c-c-@1')=1;
```

- If you want to highlight the substructure in the resulting structures, use the ssshighlight ancillary operator. You can use any number as the sss-number parameter, but it must match the sss-number parameter for the ancillary operators.
- sss is an indexed operator. If Direct cannot locate the domain index for the operator sss, the search executes more slowly than an indexed sss. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command EXPLAIN PLAN.
- A generic substructure search requires a generic structure domain index. This index allows both generic and specific structures to be registered. When the generic and specific structures are registered, they can be searched using the sss operator.
- When using the 'GENERICs' option, do not combine an sss search with another query in the same WHERE clause. Do not use the sss operator with another WHERE clause condition to refine the generic sss search within the same query. For example, you want to perform a generic substructure search on all of the libraries in a database which have a parent\_sampleid that starts with 'BENZ'. You might be very tempted to use this query:

```
select parent_sampleid from mylibdb
where parent_sampleid like 'BENZ%'
and sss(ctab, '/myquery.mol', 'GENERICs')=1
```

In the previous example query, Oracle might not use the domain index to run the sss search in this case. If it does not, and there are quite a few libraries that start with 'BENZ', the search could take a long time to run. The one-on-one sss matching can be VERY slow, much slower than say a FLEXMATCH of two non-generic!

Instead of using a combined query in a single SELECT statement, you should further refine your search (if needed) by running a second search. For example:

-- This search uses ONLY an sss

```
insert into libhits
select parent_sampleid from mylibdb
where sss(ctab, '/myquery.mol', 'GENERICs')=1;
```

-- Further refinement is done here

```
select parent_sampleid from libhits
where parent_sampleid like 'BENZ%';
```

In this example, Oracle uses the domain index in the search. If Oracle performs a full table scan (instead of using the domain index), the search might take much longer than expected.

- Substructure searching of biopolymers using the `sss` operator is supported. Because of the potentially for a very large number of atoms in a biopolymer, only modified monomer units and cross-linked monomer units are indexed for structure searching. If an `sss` query contains only unmodified and un-cross-linked monomers, Direct uses a text search to find all registered biopolymers which have sequence text which contains the query sequence text.

For example, the query `leu-leu-leu-leu` would generate the sequence text “LLLL” which would find proteins such as human protein CM036 which has the sequence text “MSEPDTSSGFGSGSVENGTFLELFPTSLSTSVDPSSGHLNVIYVSIFLSLLAFLLLLIIALQRLKNISSSSSYPEYPSD AGSSFTNLEVCSISSQRSTFSNLSS”. For more information, see *Substructure searching of modified and unmodified monomers* in *BIOVIA Direct Developers Guide*.

- Multi-threaded substructure search can be enabled. By default, substructure search is not multi-threaded. To set the number of threads used during substructure searching, use the administrative function `mdl aux.setproperty('NTHREADS', numberOfThreads)`. For details, see *Command Reference* in the *BIOVIA Direct Administration Guide*.

## sss\_highlight\_chime

Returns the Chime string representation of a structure that:

- Matches a substructure query
- `sss_highlight_chime` is an ancillary operator of the `sss` operator. To use `sss_highlight_chime`, you must also use `sss` within the same SQL statement.
- Contains highlight information for the matched substructure. The highlighted structure can be rendered by BIOVIA Draw.

**Note:** Direct provides this operator to emulate the `sss_highlight_chime` operator in the molecule cartridge prior to version 6.0. `sss_highlight_chime` is equivalent to `ssshighlight`. See [ssshighlight](#) for more details.

### Syntax

```
sss_highlight_chime(sss-number)
```

Parameter	Description
<i>sss-number</i>	A number that is equal to the <i>sss-number</i> parameter that is used with the <code>sss</code> operator.

### Return value

A CLOB that contains the Chime string representation of a structure that contains highlight information for the matched substructure.

### Usage

```
select sss_highlight_chime(sss-number)
      [, other-column-data]
from   tablename
where  sss(ctab, query, sss-number)=1
      [operator other-conditions];
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### sss\_highlight\_molfile

Returns the molfile representation of a structure that:

- Matches a substructure query  
`sss_highlight_molfile` is an ancillary operator of the `sss` operator. To use `sss_highlight_molfile`, you must also use `sss` within the same SQL statement.
- Contains highlight information for the matched substructure. The highlighted molecule can be rendered by BIOVIA Draw.

**Note:** Direct provides this operator to emulate the `sss_highlight_molfile` operator in the molecule cartridge prior to version 6.0. `sss_highlight_chime` is equivalent to `mdlaux.chimetoclob(ssshighlight(n))`.

### Syntax

`sss_highlight_molfile(sss-number)`

Parameter	Description
<i>sss-number</i>	An Oracle search operator identifier. It is used to specify with which search operator an ancillary function is associated. For example, if the SQL statement has <code>WHERE SSS (... ,1)=1 OR SSS (... ,2)=1</code> , the <code>SELECT</code> list will retrieve the highlighted structure for each <code>SSS</code> clause separately.

### Return value

A CLOB that contains the molfile representation of a structure that contains highlight information for the matched substructure. The CLOB includes newline (\n) characters that separate the lines within the molfile.

### Usage

```
select sss_highlight_molfile(sss-number)
      [, other-column-data]
from   tablename
where  sss(ctab, query, sss-number)=1
      [operator other-conditions];
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

## ssshighlight

Returns a Chime representation of a molecule that:

- Matches a substructure query  
ssshighlight is an ancillary operator of the sss operator. To use ssshight, you must also use sss within the same SQL statement.
- Contains highlight information for the matched substructure. The highlighted molecule can be rendered by BIOVIA Draw.

### Syntax

ssshighlight(*sss-number*)

Parameter	Description
<i>sss-number</i>	A NUMBER equal to the sss-number parameter that is used with the sss operator.

### Return value

A CLOB that contains the Chime representation of a candidate molecule, and contains highlight information for the matched substructure. ssshight stores the return value in a temporary CLOB. The Chime string uses the V3000 format, and contains highlight information as CTlib collection objects.

### Usage

```
select ssshight(sss-number)
    [, other-column-data]
from tablename
where sss(mol, query, sss-number)=1
    [operator other-conditions];
```

### Example

The following example returns highlighted molecule Chime strings after a substructure search.

```
select extreg,
    ssshight(3)
from moltable
where sss(
    molcol,
    '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.mol',
    3
)=1;
```

**Note:** The number 3 is used to correlate the sss operator in the WHERE clause with the ssshight operator in the SELECT clause. This could be any number as long as the values in the two operators match.

The following example returns highlighted molfile strings after a substructure search:

```
select extreg,  
       mdlaux.chimetoclob(ssshighlight(3))  
from moltable  
where sss(  
       molcol,  
       '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.mol',  
       3  
       )=1;
```

### Comments

- The sss-number parameters for ssshhighlight and sss operators must match. If the sss-number parameters do not match, or if you use ssshhighlight without using sss, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

- Because the typical usage is to display highlighted molecules using Chime, this function returns a Chime string. If a molfile is desired the function mdlaux.chimetoclob can be used:

```
SELECT extreg, MDLAUX.CHIMETOCLOB(SSSHIGHLIGHT(3)) FROM moltable WHERE  
SSS(molcol,  
    '/home/user/query.mol', 3) = 1;
```

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

- If the corresponding SSS query included the "ORIEN" argument, the highlighted structure will be oriented in the same way as the query structure.
- Because the ssshhighlight operator is always used with the sss operator, highlighting a structure always involves a substructure search. However, if you simply want to highlight a structure without searching a table, you can directly map a query to the target, such as the following example:

```
select ssshhighlight(1)  
from dual  
where sss(mol('/home/users/mols/target.mol'),  
          '/home/users/mols/query.mol',1)=1;
```

The following example shows that you can also put sss with the ssshhighlight operator in the SELECT clause. In this example, if the query is not a substructure of the specified target, ssshhighlight returns the unhighlighted target structure.

```
select sss(mol('/home/users/mols/target.mol') "SSSResult",  
          '/home/users/mols/query.mol',1),  
       ssshhighlight(1) "SSSHighlight"  
from dual;
```

### See also

[SSS](#)

[Examples of Reaction Substructure Search](#)

## ssssequenceids

Returns a comma-separated list of the template or abbreviation Sgroup sequence ID values for residues which contain the query as a substructure. For example, the peptide ACH1\_ACHF2 contains a non-natural D-amino acid at residue 2.

### Syntax

`ssssequenceids(sss-number)`

Parameter	Description
<i>sss-number</i>	A NUMBER equal to the sss-number parameter that is used with the sss operator.

### Return value

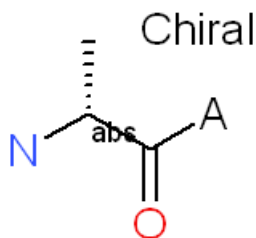
A VARCHAR2 string containing a comma-separated list of the template or abbreviation Sgroup sequence ID values for residues which contain the query as a substructure. For example, the peptide ACH1\_ACHF2 contains a non-natural D-amino acid at residue 2.

### Usage

```
select ssssequenceids(sss-number)
      [, other-column-data]
from tablename
where sss(mol, query, sss-number)=1
      [operator other-conditions];
```

### Example

Consider the following structure for this example:



When the structure is used in an SSS search in a table containing ACH1\_ACHF2 the results are:

```
select uniprot_name, ssssequenceids(1) from test where sss(ctab,
'query', 1)=1;
```

```
UNIPROT_NAME  SSSSEQUENCEIDS(1)
-----
ACH1_ACHF2, 3
```

Residue 2 contains the D chiral center, while the “A” (any) query atom matches residue 3 which in this case is not a modified amino acid but is a defined template atom. Thus SSSSEQUENCEIDS returns the string “2,3”.

**Note:** The number 1 is used to correlate the sss operator in the WHERE clause with the ssssequenceids operator in the SELECT clause. This could be any number as long as the values in the two operators match.

**Comments**

The `sss-number` parameters for `ssssequenceids` and `sss` operators must match. If the `sss-number` parameters do not match, or if you use `ssssequenceids` without using `sss`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

**See also**

[SSS](#)

**ssstimeout**

Returns the timeout status value from an `sss` search.

`sss` will return as matches those candidates for which the matching algorithm times out. Such candidates may or may not be actual matches. This ancillary operator gives information about that timeout status.

**Syntax**

`ssstimeout(sss-number)`

Parameter	Description
<i>sss-number</i>	A NUMBER equal to the <code>sss-number</code> parameter that is used with the <code>sss</code> operator.

**Return value**

A NUMBER that indicates the status of the substructure search. The possible values are:

Value	Description
0	The <code>sss</code> search did not time out.
1	The <code>sss</code> search timed out.
NULL	The target was not a match to the query.

**Usage**

```
select ssstimeout(sss-number)
      [,other-column-data]
from   tablename
where  sss(mol, query, sss-number)=1
      [operator other-conditions];
```

**Example**

The following example returns the timeout status while searching the table.

```
select extreg,
       ssstimeout(3) "Timeout"
from   moltable where sss(
      molcol,
      '/opt/BIOVIA/direct/examples/rxnfiles/query1.mol',
      3
      )=1;
```

**Note:** The number 3 is used to correlate the `sss` operator in the `WHERE` clause with the `sssttimeout` operator in the `SELECT` clause. This could be any number as long as the values in the two operators match.

### Comments

- The `sss-number` parameters for `sssttimeout` and `sss` operators must match. If the `sss-number` parameters do not match, or if you use `sssttimeout` without using `sss`, you get the following error:  
ORA-29908: missing primary invocation for ancillary operator
- The `sssttimeout` operator can be used with a generic `sss`. `sssttimeout` returns the timeout status of the search.

### See also

[SSS](#)

## writemol

Saves a BLOB or a CLOB representation of a single structure to a molfile in a disk location.

Direct provides this operator to emulate the `writemol` operator of the molecule cartridge prior to version 6.0. The following are equivalent functions:

`writemol(filename, CLOB)` is equivalent to `writefile(CLOB, filename)`.

`writemol(filename, BLOB)` is equivalent to `writefile(molfile(obj), filename)`.

See [writefile](#) for details.

### Syntax

`writemol(molfile, fieldname)`

Parameter	Description
<i>molfile</i>	A string that contains the full path and the name of the molfile
<i>fieldname</i>	The name of the CLOB or BLOB field that stores the structure.

### Return value

The NUMBER 1 indicates that Direct wrote the molfile successfully

### Usage

```
select writemol(
    '/disk-location/file-location/filename.mol',
    fieldname
)
from tablename
where condition;
```

## xhelm

Returns a XHELM string representation of a biopolymer sequence molecule.

### Syntax

`xhelm(molecule)`

*molecule* is the name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.

### Return value

A CLOB containing the XHELM format text including detailed information about each custom amino acid structure. The operator returns NULL if the XHELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdlaux.errors` to see the related error message.

### Usage

```
select xhelm(molecule)
      [, other-column-data]
from tablename
where condition;
```

### Example

Direct allows an XHELM input format wherever it expects a molecule. For example in registration:

```
INSERT INTO MOLTABLE (ID, CTAB) VALUES (:1, MOL(:2))
```

The Direct application supplies an XHELM CLOB as argument #2.

### Comments

- XHELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

*Getting the HELM String in the BIOVIA Direct Developers Guide*

## Molecule-Specific Functions

In some cases, Direct offers both a function and an operator with the same name. The function and operator behave identically to each other. For example, the `readfile` operator and the `mdlaux.readfile` function have the same functionality. In these cases, the description of the operator appears under [Molecule-Specific Operators](#).

If both a function and an operator are available, use the function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators.

<a href="#">mdlaux.getsavedmolname</a>	101
<a href="#">mdlaux.helm</a>	102
<a href="#">mdlaux.helm2</a>	104
<a href="#">mdlaux.helmtomolfile</a>	105
<a href="#">mdlaux.inchi</a>	107
<a href="#">mdlaux.inchiauxinfo</a>	109
<a href="#">mdlaux.inchikey</a>	110

<a href="#">mdlaux.inchitomolfile</a>	112
<a href="#">mdlaux.isgeneric</a>	114
<a href="#">mdlaux.isnostruct</a>	115
<a href="#">mdlaux.isotopicformula</a>	115
<a href="#">mdlaux.isrna</a>	117
<a href="#">mdlaux.issequence</a>	118
<a href="#">mdlaux.iupacname</a>	119
<a href="#">mdlaux.iupacnametomolfile</a>	121
<a href="#">mdlaux.mol</a>	122
<a href="#">mdlaux.molchime</a>	122
<a href="#">mdlaux.molfile</a>	122
<a href="#">mdlaux.molfmla</a>	122
<a href="#">mdlaux.molimage</a>	123
<a href="#">mdlaux.molkeys</a>	125
<a href="#">mdlaux.molname</a>	128
<a href="#">mdlaux.molnmakekey</a>	128
<a href="#">mdlaux.molwt</a>	132
<a href="#">mdlaux.molwtmax</a>	133
<a href="#">mdlaux.molwtmin</a>	134
<a href="#">mdlaux.monoisotopicmass</a>	135
<a href="#">mdlaux.numspecifics</a>	137
<a href="#">mdlaux.rownmakekey</a>	138
<a href="#">mdlaux.sequencetext</a>	139
<a href="#">mdlaux.setmolname</a>	140
<a href="#">mdlaux.sgroupfields</a>	141
<a href="#">mdlaux.smiles</a>	142
<a href="#">mdlaux.smilestomolfile</a>	143
<a href="#">mdlaux.xhelm</a>	144

## mdlaux.getsavedmolname

Returns the saved molecule name from the last call to the mol. When the mol operator executes, it extracts the molecule name from the specified molfile or Chime string. It then stores this value temporarily, such that the `mdlaux.getsavedmolname` function can retrieve it. The `getsavedmolname` function also clears the stored value. Because of this, after calling `mdlaux.getsavedmolname`, an immediately subsequent call will return NULL.

### Syntax

```
mdlaux.getsavedmolname >
```

### Usage

```
select mdlaux.getsavedmolname from dual;
```

### Return value

A VARCHAR2 that contains the stored molname. If there is no stored molname, this function returns NULL.

### Comments

This function will not return the correct molecule name if it is called in the same SQL statement as the mol operator. It must be called in a subsequent statement or in a trigger (fired by the statement that contains the mol operator). For example:

```

DECLARE
REGNO NUMBER;
BEGIN
  INSERT INTO MYDB_MOL(CTAB)
  VALUES(MOL('/home/smith/benzene.mol'))
  RETURNING CDBREGNO INTO REGNO;
  UPDATE MYDB_MOL
  SET MOLNAME = MDLAUX.GETSAVEDMOLNAME WHERE CDBREGNO=REGNO;
COMMIT;
END;

```

The `mdlaux.getsavedmolname` returns a molecule name only if an `INSERT` or `UPDATE` statement using `mol` was called prior to calling `mdlaux.getsavedmolname`. There will not be any name saved if the structure being inserted is already a packed `ctab`. For example, these will both work correctly:

```

insert into moltable
  values ('mol-1', mol('/benz.mol'));
update moltable
  set molname=mdlaux.getsavedmolname from dual;

insert into moltable
  values ('mol-2', mol('some-long-chime-string-abcdef123'));
update moltable
  set molname=mdlaux.getsavedmolname from dual;

```

But this will not; the "saved" name will either be empty or wrong:

```

insert into moltable
  select key, ctab from moltable2;
update moltable
  set molname=mdlaux.getsavedmolname from dual;

```

An alternative way to setting the molecule name during registration is by using the `mdlaux.molname` function.

**See also**

[mdlaux.molname](#)

[mdlaux.setmolname](#)

## mdlaux.helm

Returns a HELM string representation of a biopolymer sequence molecule.

### Syntax

`mdlaux.helm(molIndexOrTable, molecule)`

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule index, or the name of a table which contains exactly one molecule index. The schema may be included, e.g. 'schema.table'. If the value of this parameter is NULL the global environment is used.
<i>molecule</i>	A molecule that uses one of the following formats:



Parameter	Description
	<ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A temporary CLOB that contains the HELM string. The helm operator returns NULL if the HELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdlAux.errors` to see the related error message.

**Usage**

```
select mdlAux.helm(molecule) from dual;
select mdlAux.helm(molIndexOrTable, molecule) from dual;
```

**Example**

The following example shows the HELM strings for a molecule:

```
select mdlAux.helm('f:/work/mols/a20a1_human.mol') from dual;
MDLAUX.SMILES('F:/WORK/MOLS/A20AL_HUMAN.MOL')
```

```
-----
PEPTIDE1{M.K.L.F.G.F.R.S.R.R.G.Q.T.V.L.G.S.I.D.H.L.Y.T.G.S.G
.Y.R.I.R.Y.S.E.L.Q.K.I.H.K.A.A.V.K.G.D.A.A.E.M.E.R.C.L.A.R.R
.S.G.D.L.D.A.L.D.K.Q.H.R.T.A.L.H.L.A.C.A.S.G.H.V.K.V.V.T.L.L
.V.N.R.K.C.Q.I.D.I.Y.D.K.E.N.R.T.P.L.I.Q.A.V.H.C.Q.E.E.A.C.A
.V.I.L.L.E.H.G.A.N.P.N.L.K.D.I.Y.G.N.T.A.L.H.Y.A.V.Y.S.E.S.T
.S.L.A.E.K.L.L.F.H.G.E.N.I.E.A.L.D.K.V}$$$PEPTIDE1{ChainName
:Putative ankyrin repeat domain-containing protein 20A-like
protein MGC26718}|PEPTIDE1{ChainDescription:chain}$
```

**Comments**

- HELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.
- The environment is used to specify biopolymer template definitions and HELM monomer definitions. The HELM monomer definitions are always taken from the global environment even if a local environment is specified with the `molIndexOrTable` parameter.

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

#### See also

*BIOVIA Direct Developers Guide > Using Direct > Getting the HELM string*

## mdlaux.helm2

Returns a HELM version 2 string representation of a biopolymer sequence molecule.

#### Syntax

`mdlaux.helm2(molIndexOrTable, molecule)`

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule index, or the name of a table which contains exactly one molecule index. The schema may be included, e.g. 'schema.table'. If the value of this parameter is NULL the global environment is used.
<i>molecule</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

#### Return value

A temporary CLOB that contains the HELM string. The helm operator returns NULL if the HELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdlaux.errors` to see the related error message.

#### Usage

```
select mdlaux.helm2(molecule) from dual;
select mdlaux.helm2(molIndexOrTable, molecule) from dual;
```

#### Example

The following example shows the HELM strings for a molecule:

```
SQL> select mdlaux.helm2(null, 'c:/temp/a20a1_human.mol') from dual;
```

```
MDLAUX.HELM2(NULL, 'C:/TEMP/A20AL_HUMAN.MOL')
```

```
-----
PEPTIDE1{M.K.L.F.G.F.R.S.R.R.G.Q.T.V.L.G.S.I.D.H.L.Y.T.G.S.G
.Y.R.I.R.Y.S.E.L.Q.K.I.H.K.A.A.V.K.G.D.A.A.E.M.E.R.C.L.A.R.R
.S.G.D.L.D.A.L.D.K.Q.H.R.T.A.L.H.L.A.C.A.S.G.H.V.K.V.V.T.L.L
.V.N.R.K.C.Q.I.D.I.Y.D.K.E.N.R.T.P.L.I.Q.A.V.H.C.Q.E.E.A.C.A
.V.I.L.L.E.H.G.A.N.P.N.L.K.D.I.Y.G.N.T.A.L.H.Y.A.V.Y.S.E.S.T
.S.L.A.E.K.L.L.F.H.G.E.N.I.E.A.L.D.K.V}"ChainDescription:cha
in,ChainName:Putative ankyrin repeat domain-containing prote
n 20A-like protein MGC26718"$$$V2.0
```

### Comments

- HELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.
- The environment is used to specify biopolymer template definitions and HELM monomer definitions. The HELM monomer definitions are always taken from the global environment even if a local environment is specified with the `molIndexOrTable` parameter.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

*BIOVIA Direct Developers Guide > Using Direct > Getting the HELM string*

## mdlaux.helmtomolfile

Returns a molfile string (CLOB) from a HELM string (CLOB).

### Syntax

```
mdlaux.helmtomolfile(molIndexOrTable, helm)
```

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule index, or the name of a table which contains exactly one molecule index. The schema may be included, e.g. 'schema.table'. If the value of this parameter is NULL or if this parameter is omitted the global environment is used.
<i>molecule</i>	A CLOB containing the HELM string.

### Return value

A temporary CLOB that contains the converted molfile string. If the HELM string cannot be converted, the `mdlaux.helmtomolfile` function returns NULL. Use `mdlaux.errors` to see related error messages.

### Usage

```
select mdlaux.helmtomolfile(molIndexOrTable, helm) from dual;
```

**Example**

The following example shows the converted molfile from a simple HELM string:

```
select mdlaux.helmtomolfile('PEPTIDE1{A}') from dual;
```

```
MDLAUX.HELMTOMOLFILE('PEPTIDE1{A}')
```

```
-----
SciTegic11171415502D
```

```
0 0 0 0 0 0          999 V3000
M V30 BEGIN CTAB
M V30 COUNTS 1 0 0 0 1
M V30 BEGIN ATOM
M V30 1 A 1.5 2.5 0 0 CLASS=AA SEQID=1
M V30 END ATOM
M V30 END CTAB
M V30 BEGIN TEMPLATE
M V30 TEMPLATE 1 AA/a1a/A
M V30 BEGIN CTAB
M V30 COUNTS 7 6 3 0 1
M V30 BEGIN ATOM
M V30 1 O 6.6266 -2.0662 0 0
M V30 2 H 5.0016 -2.0876 0 0
M V30 3 N 5.1358 -2.0784 0 0
M V30 4 C 5.7844 -1.5983 0 0 CFG=2
M V30 5 C 6.4753 -2.0653 0 0
M V30 6 O 6.4753 -2.8977 0 0
M V30 7 C 5.7844 -0.7662 0 0
M V30 END ATOM
M V30 BEGIN BOND
M V30 1 1 3 4
M V30 2 1 4 5
M V30 3 2 5 6
M V30 4 1 4 7 CFG=1
M V30 5 1 3 2
M V30 6 1 5 1
M V30 END BOND
M V30 BEGIN SGROUP
M V30 1 SUP 1 ATOMS=(1 1) XBONDS=(1 6) LABEL=OH BRKXYZ=(9 7.02 -2.26 0 7.02
-
M V30 -1.85 0 0 0 0) CSTATE=(4 6 -0.15 0 0) CLASS=LGRP
M V30 2 SUP 2 ATOMS=(1 2) XBONDS=(1 5) LABEL=H BRKXYZ=(9 4.58 -1.87 0 4.6 -
M V30 -2.28 0 0 0 0) CSTATE=(4 5 0.13 0.01 0) CLASS=LGRP
M V30 3 SUP 3 ATOMS=(5 3 4 5 6 7) XBONDS=(2 5 6) LABEL=A1a BRKXYZ=(9 3.95 -
M V30 -3.33 0 3.95 -0.38 0 0 0 0) CSTATE=(4 5 -0.13 -0.01 0) CSTATE=(4 6
0.15 -
M V30 0 0) CLASS=AA SAP=(3 3 2 A1) SAP=(3 5 1 Br)
M V30 END SGROUP
M V30 END CTAB
M V30 END TEMPLATEM END
```

**Comments**

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[helm](#)

BIOVIA Direct Developers Guide > Using Direct > Conversion of SMILES strings to molfile

**mdlaux.inchi**

Returns an IUPAC standard International Chemical Identifier (standard "InChI") string for the specified molecule.

**Syntax**

`mdlaux.inchi(molecule, options)`

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ A SMILES string</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options.</p> <p>If no additional options are provided in the call to INCHI, the standard InChI string is generated.</p> <p>If any of the following options are specified, a non-standard InChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ Sabs - Absolute stereo (default)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ <b>SRe1</b> - Relative stereo</li> <li>■ <b>SRac</b> - Racemic stereo</li> <li>■ <b>SUCF</b> - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ <b>SNon</b> - Exclude stereo</li> <li>■ <b>SUU</b> - Include omitted unknown/undefined stereo</li> <li>■ <b>SLUUD</b> - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ <b>KET</b> - Account for keto/enol tautomerization (default is off)</li> <li>■ <b>15T</b> - Account for 1-5 tautomerization (default is off)</li> <li>■ <b>SaveOpt</b> - Save non-default options in the InChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A temporary CLOB that contains the InChI string. The output string length will exceed 4000 characters for very large molecules. If the InChI string cannot be generated, the `mdlaux.inchi` functions returns NULL. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select mdlaux.inchi(molecule, options) from dual;
```

**Example**

The following example shows the InChI string for a molecule, using the default option:

```
select mdlaux.inchi('/work/mols/muse1.mol') from dual;
```

```
MDLAUX.INCHI('/WORK/MOLS/MUSE1.MOL')
```

```
-----
InChI=1S/C8H10N4O2/c1-10-4-9-6-5(10)7(13)12(3)8(14)11(6)2/h4H,1-3H3
```

**Comments**

- There are limitations to the generation of InChI strings. Not all BIOVIA molecule features can be handled. The `mdlaux.inchi` function returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[mdlaux.inchikey](#)

[inchi](#)

*BIOVIA Direct Developers Guide > Using Direct> Getting the InChI string and key*

*BIOVIA Direct Developer's Guide > Using Direct> Limitations to the generation of InChI strings*

## mdlaux.inchiauxinfo

Returns the auxiliary information (AuxInfo) that is computed along with the IUPAC International Chemical Identifier (InChI) string for a molecule.

### Syntax

`mdlaux.inchiauxinfo(any-molecule [, options])`

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ A SMILES string</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options. If no additional options are provided in the call to INCHIAUXINFO, the standard InChI AuxInfo string is generated. If any of the following options are specified, a non-standard InChI AuxInfo string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbS - Absolute stereo (default)</li> <li>■ SRe1 - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the InChI AuxInfo string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

### Return value

A temporary CLOB that contains the InChI AuxInfo string. The output string length will exceed 4000 characters for very large molecules. If the InChI AuxInfo string cannot be generated, the `mdlaux.inchiauxinfo` function returns NULL. Use `mdlaux.errors` to see the related error message.

### Usage

```
select mdlaux.inchiauxinfo(molecule, options) from dual;
```

### Example

The following example shows the InChI AuxInfo string for a molecule, using the default option:

```
select mdlaux.inchiauxinfo('/work/mols/muse1.mol') from dual;
```

```
MDLAUX.INCHIAUXINFO('/WORK/MOLS/MUSE1.MOL')
-----
AuxInfo=1/0/N:10,12,13,9,1,2,3,11,6,4,5,7,8,14/rA:14CCCNNOCCCCCO/rB:d-
1;s1;s1;s2;s2;s3;d3;s4d-
6;s4;s5s7;s5;s7;d11;/rC:.9129,.6671,0;.9129,-.8497,0;-.4214,1.4361,0;2.3701,
1.1447,0;-.4108,-1.6082,0;2.3701,-1.3237,0;-
1.7346,.6671,0;-.4249,2.974,0;3.269,-.0772,0;2.8476,2.6089,0;-
1.7346,-.8462,0;-.4003,-3.1531,0;-3.0618,1.4431,0;-3.0723,-1.6152,0;
```

### Comments

- There are limitations to the generation of InChI AuxInfo strings. Not all BIOVIA molecule features can be handled. The `mdlaux.inchiauxinfo` function returns NULL if the specified molecule cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[mdlaux.inchi](#)  
[inchiauxinfo](#)

*BIOVIA Direct Developers Guide > Using Direct> Getting the InChI string and key*

*BIOVIA Direct Developer's Guide > Using Direct> Limitations to the generation of InChI strings*

### mdlaux.inchikey

Returns an IUPAC International standard Chemical Identifier (standard "InChI") key for the specified molecule. The InChI key is a 27-character hashed form of the InChI string. The `mdlaux.inchikey` function generates the key by first generating the InChI string, and then calling an InChI library function to convert the string into the 27-character key.

### Syntax

```
mdlaux.inchikey(molecule, options)
```



Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a) , or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct Molecule object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ SMILES string</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of InChI library options.            If no additional options are provided in the call to INCHI, the standard InChI string is generated.            If any of the following options are specified, a non-standard InChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the InChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A VARCHAR2 that contains the 27-character InChI key. The `mdlaux.inchikey` function returns NULL if the InChI string cannot be generated. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select mdlaux.inchikey(molecule, options) from dual;
```

### Example

The following example shows the InChI key for a molecule, using the default option:

```
select mdlaux.inchikey('/work/mols/muse2.mol') from dual;
```

```
MDLAUX.INCHIKEY('/WORK/MOLS/MUSE2.MOL')
```

```
-----  
UHOVQNZJYSORNB-UHFFFAOYSA-N
```

### Comments

There are limitations to the generation of InChI strings. Not all BIOVIA molecule features can be handled. If the specified molecule cannot be handled, the `mdlaux.inchikey` functions returns NULL. Use `mdlaux.errors` to see the related error message.

### See also

[mdlaux.inchi](#)  
[inchikey](#)

*BIOVIA Direct Developers Guide > Using Direct > Limitations to the Generation of InChI Strings*

### mdlaux.inchitomolfile

Returns the molfile string representation of an InChI (IUPAC International Chemical Identifier) string or InChI AuxInfo string.

### Syntax

```
mdlaux.inchitomolfile(inchi-string-or-auxinfo)
```

Parameter	Description
<i>inchi-string-or-auxinfo</i>	A VARCHAR2 or CLOB containing an InChI string or an InChI AuxInfo string. Using an InChI AuxInfo string will provide more information to the conversion including the original atom coordinates, the output molecule will generally be more similar to the molecule from which InChI was calculated.

### Return value

A temporary CLOB that contains the converted molfile string. If the InChI input cannot be converted, the `mdlaux.inchitomolfile` function returns NULL. Use `mdlaux.errors` to see related error message.

### Usage

```
select mdlaux.inchitomolfile(inchi_string) from dual;
```

### Examples

The following example shows the converted molfile from an InChI string calculated for chlorobenzene:

```
select mdlaux.inchitomolfile('InChI=1S/C6H5Cl/c7-6-4-2-1-3-5-6/h1-5H') from dual;
```

```
MDLAUX.INCHITOMOLFILE('INCHI=1S/C6H5CL/C7-6-4-2-1-3-5-6/H1-5H')
```

```
-----  
-
```

```
SciTegic06291614572D
```

```

7 7 0 0 0 0          999 v2000
-3.0000 -1.0000 0.0000 C 0 0
-2.5000 -1.8660 0.0000 C 0 0
-2.5000 -0.1340 0.0000 C 0 0
-1.5000 -1.8660 0.0000 C 0 0
-1.5000 -0.1340 0.0000 C 0 0
-1.0000 -1.0000 0.0000 C 0 0
0.0000 -1.0000 0.0000 C 0 0
1 2 2 0
1 3 1 0
2 4 1 0
3 5 2 0
4 6 2 0
5 6 1 0
6 7 1 0
M END

```

The next example shows the molfile converted from the InChI AuxInfo value for ethane:

```

select mdlaux.inchitomolfile('AuxInfo=1/0/N:1,2/E:
(1,2)/rA:2CC/rB:s1;/rC:1.3343,.7672,0;0,1.5418,0;') from dual;

```

```

MDLAUX.INCHITOMOLFILE('AUXINFO=1/0/N:1,2/E:
(1,2)/rA:2CC/rB:S1;/rC:1.3343,.767

```

```

-----
-

```

SciTegic06222015142D

```

2 1 0 0 0 0          999 v2000
1.3343 0.7672 0.0000 C 0 0
0.0000 1.5418 0.0000 C 0 0
1 2 1 0
M END

```

### Comments

InChI has limitations, for example it does not encode information about whether stereochemistry is relative or absolute nor does it always differentiate between two tautomeric forms of a molecule. Nitro groups in an InChI string are always converted to the uncharged hypervalent nitrogen form in the molfile. You will not always get the same molecule that you started with when converting from molfile to InChI and then from that InChI back to molfile.

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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(); }

```

### See also

[inchi](#)

[inchiauxinfo](#)

## mdlaux.isgeneric

Returns 1 if the specified molecule is a generic structure, 0 if not. A generic structure is a Markush structure that represents actual structures.

### Syntax

`mdlaux.isgeneric(molecule)`

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct Molecule object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

A NUMBER that indicates whether the specified molecule is a generic structure (1) or not (0).

### Usage

```
select column-data
from tablename
where mdlaux.isgeneric(ctab)=1
      [operator other-conditions];
```

### Example

The following example uses isgeneric to find generic structures in the samplegen table.

```
select parent_sampleid
from samplegen
where mdlaux.isgeneric(ctab)=1;
```

### Comments

This operator is not indexed. When possible, avoid using it when running queries on large tables.

### See also

[isgeneric](#)

## mdlaux.isnostruct

Returns 1 if the specified molecule is a nostruct ("no-structure"), 0 if not. A no-structure is a chemical structure that consists of zero fragments, that is, zero atoms and zero bonds.

### Syntax

`mdlaux.isnostruct(molecule)`

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Direct molecule object (BLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).

A NUMBER that indicates whether the specified molecule is a no-structure (1) or not (0).

### Usage

```
select column-data
from tablename
where mdlaux.isnostruct(molecule)=1
[operator other-conditions];
```

### Example

The following example uses `isnostruct` to find "no-structures" in the `sample2d` table.

```
select cdbregno
from sample2d
where mdlaux.isnostruct(ctab)=1;
```

### Comments

This function is not indexed. When possible, avoid using it when running queries on large tables.

### See also

[isnostruct](#)

## mdlaux.isotopicformula

Computes the formula for the specified molecule, including isotope labels.

### Syntax

`mdlaux.isotopicformula(molIndexOrTable, molecule, 'format-options')`

Parameter	Description
<code>molIndexOrTable</code>	The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index. The implicit Ptable controls the atom symbols used. If <code>molIndexOrTable</code> is NULL, the global cartridge Ptable is used for symbol resolution.
<code>molecule</code>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<code>format-options</code>	Optional string to control formatting of the output molecular formula. If no options are present the formula will include space between elements, and the formula for each fragment in a multi-fragment structure will be separated by a dot. Use NOSPACE to remove the space between elements, NOFRAGMENT to not separate fragment formulas, and NOSPACE NOFRAGMENT for both changes. To match the formula string which is output by Insight use NOSPACE NOFRAGMENT. A third optional argument, USEDANDT, switches the default display of hydrogen isotopes, 2H and 3H, to the older one-letter designations D and T.

**Return value**

A temporary CLOB that contains the formula string including isotope labels.

**Usage**

```
insert into tablename(
  isotopicformula-field
  [,other-column-data]
)
values (
  mdlaux.isotopicformula(molIndexOrTable, molecule)
  [,other-value-data]
);
```

**Example**

The following example registers isotopic formula into a table:

```

insert into moltable
    (id, ctab,
     molformula,
     isotopicmolformula)
values ('Mol1', mol('/home/joe/mol1.mol'),
       mdlaux.molfmla('moltable', '/home/joe/mol1.mol'),
       mdlaux.isotopicformula('moltable', '/home/joe/mol1.mol'));

```

The following is an example showing the traditional and isotopic formulae for C13 labeled chlorobenzene:

```

select mdlaux.molfmla(null, '/work/mols/phclc13.mol')
from dual;

```

```

MDLAUX.MOLFMLA(NULL, '/WORK/MOLS/PHCLC13.MOL')
-----

```

```

C6 H5 Cl

```

```

select mdlaux.isotopicformula(null, '/work/mols/phclc13.mol')
from dual

```

```

MDLAUX.ISOTOPICFORMULA(NULL, '/WORK/MOLS/PHCLC13.MOL')
-----

```

```

13C C5 H5 Cl

```

#### Comments

- Use the mdlaux.isotopicformula function when registering mono-isotopic mass into a table. The isotopicformula operator cannot be used for registration.
- To match the formula output by Direct using the Molecular Formula component in Pipeline Pilot, set the component options as follows:
  - Ignore Isotopes = (True for MOLFMLA, False for ISOTOPICFORMULA)
  - Use 2H and 3H for Hydrogen Isotopes = True
  - Include Space Between Elements = (True for default, False if using NOSPACE)
  - Separate Fragments = (True for default, False if using NOGRAGMENT)
  - Add HTML Tags = False
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}

```

#### See also

[isotopicformula](#)

#### mdlaux.isrna

Returns 1 if the molecule argument is an RNA or DNA sequence, 0 if it is not.

A molecule is considered an RNA or DNA biopolymer sequence if it contains one or more nucleotide template atoms, one or more nucleotide template definitions, or one or more Sgroup abbreviations (superatoms) that have associated nucleotide sequence information.

### Syntax

```
mdlaux.isrna(molIndexOrTable, molecule)
```

Parameter	Description
<code>molIndexOrTable</code>	The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index. The implicit Ptable controls the atom symbols used. If <code>molIndexOrTable</code> is NULL, the global cartridge Ptable is used for symbol resolution.
<code>molecule</code>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

A NUMBER that is 1 if the molecule is an RNA or DNE biopolymer sequence, or 0 if it is not.

### Usage

```
select mdlaux.isrna(index-or-tablename, molecule) from dual;
```

### See also

[isrna](#)

## mdlaux.issequence

Returns 1 if the molecule argument is a biopolymer sequence, 0 if it is not.

A molecule is considered a biopolymer sequence if it contains one or more template atoms, one or more template definitions, or one or more Sgroup abbreviations (superatoms) that have associated sequence information.

### Syntax

```
mdlaux.issequence(molIndexOrTable, molecule)
```



Parameter	Description
<code>molIndexOrTable</code>	The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index. The implicit Ptable controls the atom symbols used. If <code>molIndexOrTable</code> is NULL, the global cartridge Ptable is used for symbol resolution.
<code>molecule</code>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A NUMBER that is 1 if the molecule is a biopolymer sequence, or 0 if it is not.

**Usage**

```
select mdlaux.issequence(index-or-tablename, molecule) from dual;
```

**See also**

[issequence](#)

*BIOVIA Direct Developers Guide > Using Direct > Biopolymer Searching and Registration*

**mdlaux.iupacname**

Returns the IUPAC (International Union of Pure and Applied Chemistry) name of a molecule.

**Syntax**

```
mdlaux.iupacname(molecule, options)
```

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) An optional VARCHAR2 argument to specify the language, name style, and character set. The argument can specify one or more of the LANGUAGE, NAMESTYLE, and CHARSET.</p> <p>For example:  'LANGUAGE=language NAMESTYLE=name-style CHARSET=character-set'</p> <p>If no options are specified, the defaults are:  'LANGUAGE=ENGLISH NAMESTYLE=DEFAULT CHARSET=HTML '</p> <p>For a list of valid options, see the <a href="#">List of valid language, name-style, and character-set options</a>.</p>

### Return value

A temporary CLOB that contains the IUPAC name of the molecule. If a name cannot be generated for the molecule, the `iupacname` operator returns NULL. Use `mdlaux.errors` to see related error message.

### Usage

```
select mdlaux.iupacname(molecule)
      [, other-column-data]
from tablename
where condition;
```

### Example

The following example shows the IUPAC name of a molecule in the `sample2d` table.

```
select mdlaux.iupacname(ctab)
from sample2d
where cdbregno=18;

MDLAUX.IUPACNAME(CTAB)
-----
3,6-dichlorophthalic acid
```

### Comments

The `mdlaux.iupacname` function uses functionality provided by OpenEye Scientific Software.

**See also**[iupacname](#)**mdlaux.iupacnametomolfile**

Returns the molfile string representation of an IUPAC (International Union of Pure and Applied Chemistry) name.

**Syntax**

```
mdlaux.iupacnametomolfile(iupacname)
```

Parameter	Description
<i>iupacname</i>	A VARCHAR2 or CLOB containing an IUPAC molecule name

**Return value**

A temporary CLOB that contains the converted molfile string. If the IUPAC molecule name cannot be converted, the `mdlaux.iupacnametomolfile` function returns NULL. Use `mdlaux.errors` to see related error message.

**Usage**

```
select mdlaux.iupacnametomolfile(iupacname) from dual;
```

**Examples**

The following example shows the converted molfile from an IUPAC name:

```
select mdlaux.iupacnametomolfile('ethylene') from dual;
MDLAUX.IUPACNOMETOMOLFILE('ETHYLENE')
```

```
-----
-----
-OEChem-01071013082D

  2  1  00  0  0  0  0  0999 v2000
    2.0000-2.00000.0000 c0 0 0 0 0 0 0 0 0 0 0 0
    3.0000-2.00000.0000 c0 0 0 0 0 0 0 0 0 0 0 0
  1  2  2  0  0  0  0
M  END
```

The following example shows a flexmatch search for chlorobenzene using the converted IUPAC name as the query structure:

```
select cdbregno from sample2d where
flexmatch(ctab, mdlaux.iupacnametomolfile('chlorobenzene'),
'match=all')=1;
```

```
  CDBREGNO
-----
         4
```

**Comments**

The `mdlaux.iupacnametomolfile` function uses functionality provided by OpenEye Scientific Software, and is subject to all of the limitations of OpenEye's structure conversion library.

See also

[mdlaux.iupacname](#)

## mdlaux.mol

This package function is equivalent to `mol`. See the documentation for [mol](#) for more information.

## mdlaux.molchime

This package function is equivalent to `molchime`. See the documentation for [molchime](#) for more information.

## mdlaux.molfile

This package function is equivalent to `molfile`. See the documentation for [molfile](#) for more information.

## mdlaux.molfmla

Returns the molecular formula for a molecule.

### Syntax

```
mdlaux.molfmla(molIndexOrTable, molecule, 'format-options')
```

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index. The implicit Ptable controls the atom symbols used. If <i>molIndexOrTable</i> is NULL, the global cartridge Ptable is used for symbol resolution.
<i>molecule</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

Parameter	Description
<i>format-options</i>	Optional string to control formatting of the output molecular formula. If no options are present the formula will include space between elements, and the formula for each fragment in a multi-fragment structure will be separated by a dot. Use NOSPACE to remove the space between elements, NOFRAGMENT to not separate fragment formulas, and NOSPACE NOFRAGMENT for both changes. To match the formula string which is output by Insight use NOSPACE NOFRAGMENT.

**Return value**

A CLOB that contains the molecule formula for the specified molecule.

**Usage**

The typical usage for the `mdlaux.molfmla` function is to get the molecule formula for registration using the same ptable which is associated with the molecule domain index on the table into which the molecule is being registered. For example:

```
INSERT INTO moltable (extreg, ctab, molformula)
VALUES ('ABC-123',
MOL('/home/user/mol123.mol'),
MDLAUX.MOLFMLA('moltable', '/home/user/mol123.mol'));
```

**Comments**

- When inserting the molecular formula, you must use the `mdlaux.molfmla` function instead of the `molfmla` operator. The `mdlaux.molfmla` function allows you to specify a molecule table or molecule domain index that specifies which ptable to use. The `molfmla` operator is not appropriate for registration because it uses the global Ptable when the specified parameter is a molecule object.
- To match the formula output by Direct using the Molecular Formula component in Pipeline Pilot, set the component options as follows:
  - Ignore Isotopes = (True for MOLFMLA, False for ISOTOPICFORMULA)
  - Use 2H and 3H for Hydrogen Isotopes = True
  - Include Space Between Elements = (True for default, False if using NOSPACE)
  - Separate Fragments = (True for default, False if using NOFRAGMENT)
  - Add HTML Tags = False
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**mdlaux.molimage**

Returns a temporary BLOB containing a PNG, BMP, SVG, or EMF image of the molecule.

**Syntax**

`mdlaux.molimage(molecule [, options])`

Parameter	Description
<i>molecule</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a molecule object.
<i>options</i>	<p>Optional. A VARCHAR2 argument to control the type of image created, its size, and other preferences. Specify this argument as a string of comma separated options, each option takes the form keyword=value.</p> <p>Possible options are:</p> <ul style="list-style-type: none"> <li>■ <code>imagetype=png</code> - Creates a PNG image (default)</li> <li>■ <code>imagetype=bmp</code> - Creates a BMP</li> <li>■ <code>imagetype=svg</code> - Creates a SVG</li> <li>■ <code>imagetype=emf</code> - Creates a EMF image</li> <li>■ <code>width=number</code> - Specifies a width number, typically 100 to 1000 (default is 500)</li> <li>■ <code>height=number</code> - Specifies a height number, typically 100 to 1000 (default is 500)</li> <li>■ <code>ColorAtomsByType=TRUE   FALSE</code> - Specifies whether to color the atom labels by their type. The default is TRUE.</li> <li>■ <code>HydrogenDisplayMode=mode</code> - Specifies how to display implicit hydrogen atoms. The default is HYDROGEN_HETERO. Valid mode values are: <ul style="list-style-type: none"> <li>■ <code>HYDROGEN_OFF</code> - Does not display implicit hydrogen atoms</li> <li>■ <code>HYDROGEN_HETERO</code> - Displays implicit hydrogens on heteroatoms.</li> <li>■ <code>HYDROGEN_TERMINAL</code> - Displays implicit hydrogens on terminal atoms.</li> <li>■ <code>HYDROGEN_TERMINAL_AND_HETERO</code> - Displays implicit hydrogens on terminal atoms and heteroatoms.</li> <li>■ <code>HYDROGEN_ALL</code> - Displays implicit hydrogens on all atoms.</li> </ul> </li> <li>■ <code>BackgroundColor=color</code> - Specifies the background color. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is white.</li> <li>■ <code>ForegroundColor=color</code> - Specifies the color of the shape or text. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is black. ■</li> <li>■ <code>ChiralityLabels=ANDtext,ABStext,ORtext,MIXEDtext</code> – Specifies the chirality label text to display for structures with stereocenters. The default is “AND Enantiomer,,OR Enantiomer,Mixed”. You must include the double quote characters and four comma-separated fields within the quotes. An empty field will not display anything for that type of chirality, for example the default text does not display anything for a structure that has only absolute stereocenters. ■</li> <li>■ <code>DisplayRS=TRUE   FALSE</code> – Specifies whether to display R and S stereocenter labels on the structure. The default is FALSE. ■</li> <li>■ <code>DisplayEZ=TRUE   FALSE</code> – Specifies whether to display E and Z double bond labels on the structure. The default is FALSE.</li> <li>■ <code>PolAtomDisplayMode=POL_STYLE_BEAD   POL_STYLE_TEXT</code> - Specifies how to indicate the atom(s) that bind the structure to a polymer (atoms of type ‘Pol’). Default is POL_STYLE_BEAD. Valid values are:</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ POL_STYLE_BEAD - Displays polymer atoms as shaded circles that resemble beads</li> <li>■ POL_STYLE_TEXT - Displays polymer atoms with the label text Pol.</li> </ul> <p>The following shows an example that specify image options:  <code>molimage(mol, 'imagetype=png,width=100,height=100')</code></p>

**Return value**

A BLOB that contains the binary image data. The `mdlaux.molimage` function returns NULL if an image cannot be generated for the molecule. Use `mdlaux.errors` to see related error message.

**Usage**

```
select mdlaux.molimage(molecule [, options]) from dual;
```

**Comments**

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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 "blob":

```
if ( ((oracle.sql.BLOB)blob).isTemporary() ){
    ((oracle.sql.BLOB)blob).freeTemporary();
}
```

- Applications that use this function in a SQL SELECT statement must be aware that the temporary LOBs are only freed when the statement ends. If the statement selects many rows Oracle may run out of temporary space needed to store the LOBs. To work around this you can increase the temporary tablespace size, or you can convert the SELECT into a PL/SQL function which computes the image and then frees the BLOB immediately.

**See also**

[molimage](#)

**mdlaux.molkeys**

Returns the SSS keys that would be registered for a molecule as a printable string.

**Syntax**

```
mdlaux.molkeys(molIndexOrTable, molecule, print)
```

Parameter	Description
<code>molIndexOrTable</code>	<p>The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index.</p> <p>The implicit Ptable controls the atom symbols used. If <code>molIndexOrTable</code> is NULL, the global cartridge Ptable is used for symbol resolution.</p>
<code>molecule</code>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>



Parameter	Description										
<i>print</i>	<p>Specifies what is to be printed, and how. The specified print string should follow the format <i>"outputFormat, delimiterType"</i>. It can contain the following keywords and options, separated by whitespace. Extra characters are ignored; thus, 'DEC' or 'DECIMAL' would be allowed.</p> <p>Normally, the full set of 960 SSS keys are used. The following option can be added to restrict output to the subset of 166 "user" keys:</p> <ul style="list-style-type: none"> <li>■ SUB - Subset of 166 "user" keys</li> </ul> <p>Output format:</p> <ul style="list-style-type: none"> <li>■ BIN - Binary, i.e. '1' and '0'. Delimiter is applied between bits.</li> <li>■ HEX - Hexadecimal, i.e. 'fa03'. Lowest key (key 1) is highest bit in first word, thus 'a000' would set keys 1 and 3. Delimiter is applied between 32-bit words.</li> <li>■ DEC - Decimal key numbers.</li> <li>■ WTS or WEI -Decimal key weights. All key positions are output, keys which are not set have a weight of zero.</li> <li>■ SET - Returns only the number of keys set, not the key values.</li> <li>■ TOT - Returns only the total number of keys, not the key values. This is independent of the mol.</li> </ul> <p>Default: DEC</p> <p>Type and placement of delimiter character:</p> <ul style="list-style-type: none"> <li>■ DELIM=c Output key values separated by 'c'.</li> <li>■ LEAD Include a leading delimiter.</li> <li>■ TRAIL Include a trailing delimiter.</li> </ul> <p>Default: None, unless option string is all blank.</p> <p>If the option string is " or 'SUB', the default is to add 'DEC DELIM=, LEAD TRAIL'.</p> <p>Examples:</p> <table border="1"> <tr> <td></td><td>Same as "DEC DELIM=, LEAD TRAIL"</td></tr> <tr> <td>"DEC DELIM=,"</td><td>"23,47,230"</td></tr> <tr> <td>"DEC SUBSET DELIM=, LEAD TRAIL"</td><td>",2,6"</td></tr> <tr> <td>"BIN"</td><td>"000000101011101..."</td></tr> <tr> <td>"TOT"</td><td>"960" [number of SSS keys]</td></tr> </table>		Same as "DEC DELIM=, LEAD TRAIL"	"DEC DELIM=,"	"23,47,230"	"DEC SUBSET DELIM=, LEAD TRAIL"	",2,6"	"BIN"	"000000101011101..."	"TOT"	"960" [number of SSS keys]
	Same as "DEC DELIM=, LEAD TRAIL"										
"DEC DELIM=,"	"23,47,230"										
"DEC SUBSET DELIM=, LEAD TRAIL"	",2,6"										
"BIN"	"000000101011101..."										
"TOT"	"960" [number of SSS keys]										

**Return value**

A VARCHAR2 that contains the SSS keys which would be registered for a molecule as a printable string

**Usage**

The typical usage for the `mdl aux.molkeys` function is to get the printable keys for registration (using the same key definition file which is associated with the molecule domain index on the table into which the molecule is being registered). For example:

```
INSERT INTO moltable (extreg, ctab, molkeys) VALUES
('ABC-123', MOL('/home/user/mol123.mol'),
MDLAUX.MOLKEYS('moltable', '/home/user/mol123.mol',
'dec delim=', lead trail'));;
```

### Comments

When inserting the molecular formula, you must use the `mdl_aux.molkeys` function instead of the `molkeys` operator. The `mdl_aux.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.

### See also

[molkeys](#)

## mdl\_aux.molname

Returns the molecule name stored within a molfile.

### Syntax

```
mdl_aux.molname(molecule)
```

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

A VARCHAR2 that contains the name of the specified molecule

### Usage

The typical usage for the `mdl_aux.molname` function is to get the molecule name from a molfile for registration. For example:

```
INSERT INTO moltable (extreg, ctab, molname)
VALUES('ABC-123',
      MOL('/home/user/mol123.mol'),
      MDLAUX.MOLNAME('/home/user/mol123.mol'));
```

### See also

[mdl\\_aux.setmolname](#)

[mdl\\_aux.getsavedmolname](#)

## mdl\_aux.molnemakey

Returns a string that contains the NEMA key for the specified molecule structure.

**Syntax**

```
mdlaux.molnmakekey(molIndexOrTable, molecule, option)
```

Parameter	Description
<code>molIndexOrTable</code>	The name of a molecule index, or the name of a table which contains exactly one molecule index. The schema might be included, e.g., 'schema.table'. If non-NULL, the Ptable defined by the domain index is used to generate the NEMAKEY. If <code>molIndexOrTable</code> is NULL, the global Ptable is used. You should generally use the table or index name if you are generating values for use with a specific molecule table. If you wish to compare values between two tables, and the tables use Ptables with different atom symbols, you can specify NULL for this argument if the global ptable contains all of the atom symbols required.
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

Parameter	Description
<i>option</i>	<p>Specifies what to generate, and consists of one or more of the following three-letter keywords.</p> <p>The keywords are:</p> <ul style="list-style-type: none"> <li>■ CON - Returns constitutional NEMAKEY (30 characters). The constitutional key does not include any stereochemical information. It should be used to compare two molecules without regard to stereochemistry.</li> <li>■ STE - Returns stereochemical NEMAKEY (30 characters). This key includes stereochemical information for most cases, but will not differentiate mixtures of different types of enhanced stereochemical collections.</li> <li>■ EXA - Returns stereochemical NEMAKEY (30 characters). Returns no key if a FLEXMATCH verification is required, this occurs if the molecule contains mixtures of different types of enhanced stereochemical collections.</li> <li>■ FLG - Returns three (zero-padded) digits of NEMAKEY and cartridge flag information, as described below.</li> <li>■ REV - Returns the NEMAKEY revision number, for example, "1001". This number will change if the NEMAKEY computed for a molecule might be different than for a previous release.</li> </ul> <p>The specified values are appended to the output string in the order in which the keywords appear in the option parameter. Any characters which are not keywords are appended to the output string as-is.</p> <p>Thus, to generate the stereo NEMAKEY followed by a dash followed by the flags, use 'STE-FLG' for the option parameter. If the option parameter is not present, is NULL, or is blank, then the default is to return 'EXA'. The option parameter is optional; if it is not specified, it is the same as if NULL were specified.</p> <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"> <p><b>Note:</b> Do not include text which contains the letters in the option flags, the text will not be printed verbatim but will print the option with the remaining text. For example 'Stereo:ste' will print the stereo NEMAKEY twice separated by "reo:". Use Oracle text concatenation to add additional text to the NEMAKEY output.</p> </div> <p>In addition to the keywords above which specify what to include in the output string, there is another option which controls whether the generated NEMAKEY will be the same as the one created by Pipeline Pilot Client and Draw. This option is /NODAT and specifies that the NEMAKEY will not include information about data Sgroups. By default, Direct includes information about data Sgroups in its NEMAKEYS, which allows the keys to be used for exact-match comparisons. Pipeline Pilot Client and Draw do not include information about data Sgroups, so to create a NEMAKEY that matches the one in Pipeline Pilot Client or Draw use the options 'EXA/NODAT'.</p>

**Return value**

A VARCHAR2 data that contains the information generated in response to the keywords used in the option parameter.

**Example**

The following example uses `mdlaux.molnmakekey` to return the NEMAKEY string for the structures in a substructure search:

```
select cdbregno,
       mdlaux.molnmakekey('acd2d_mdlix', ctab, 'exa')
from   acd2d_moltable
where  sss(ctab, 'c:\query.mol')=1;
```

The following example uses `mdlaux.molnmakekey` to return the NEMAKEY string for a specified molfile:

```
select mdlaux.molnmakekey(null, 'c:\molecule.mol', 'ste-flg') as "NemaKey"
from   dual;
```

The following example is the same as the previous one but returns the same NEMAKEY that Pipeline Pilot or Draw would return for the structures:

```
select cdbregno,
       mdlaux.molnmakekey('acd2d_mdlix', ctab, 'exa/nodat')
from   acd2d_moltable
where  sss(ctab, 'c:\query.mol')=1;
```

**Comments**

- `mdlaux.molnmakekey` is the function equivalent of `molnmakekey`. `molnmakekey` will fetch the domain index schema and name from the information passed to it by Oracle, and then call `mdlaux.molnmakekey`.
- When the input molecule is a biopolymer sequence molecule, `molnmakekey` generates a special sequence NEMA key. Currently, only an exact NEMA key (considering stereochemistry) can be generated for sequences. Thus, if the input molecule is a sequence molecule: The 'CON' option will return a blank string Only the 'STE' option will return a NEMA key. In addition, a flag value of 128 denotes a sequence NEMA key. This example shows the flag value of 128:

```
select mdlaux.molnmakekey(null, '/home/user/alanine.mol', 'STE-FLG') from
dual;
```

```
MDLAUX.MOLNMAKEKEY(NULL, '/HOME/USER/ALANINE.MOL', 'STE-FLG')
```

```
-----
2GSUQPXFN9FEBYJHJJFQ65CXS58RF4-128
```

- The NEMAKEY will be empty if it cannot be generated because the molecule is a generic (contains rgroups), contains polymer sgroups, or it contains numeric data sgroups. It may also be empty if NEMA times out. Unless the FLG keyword is present, the return value will be NULL in these cases. If for example the option parameter was 'STE-FLG', and the molecule is a polymer, the output will be '-258', that is, the 'STE' portion is not present.

**Note:** Text data sgroups do not prevent generation of the NEMA key, and will contribute to the definition of the key.

- Flags are normally a bitwise OR of zero or more of the following:
  - 4 = Molecule has mixed stereogroups and requires FLEXMATCH to resolve equality if stereochemistry is NOT ignored
  - 128 = Molecule generated a (biopolymer) sequence NEMA key

- Flags can also be set to exactly one of the following, in this case the key is empty (no characters):

257 = NEMA generation failed due to timeout

258 = Molecule has polymer Sgroups and numeric data Sgroups

259 = Molecule is a generic (contains Rgroups)

See also

[molnemaakey](#)

BIOVIA Direct Developers Guide > Using Direct > NEMAKEY searching and key generation

## mdlaux.molwt

Returns the molecular weight of a molecule.

### Syntax

`mdlaux.molwt(molIndexOrTable, molecule)`

Parameter	Description
<i>molIndexOrTable</i>	<p>The name of a molecule index, or the name of a table which contains exactly one molecule index. (The schema may be included, e.g. 'schema.table'.)</p> <p>The value may be NULL, and can be used as one of the following:</p> <pre>mdlaux.molwt(NULL, ctab)</pre> <pre>mdlaux.molwt(' ', ctab)</pre> <p>in which case the global environment is used.</p> <p>Use the first argument to control what ptable is used to supply atomic weights for calculation of the molecular weight. a NULL value will cause the global Ptable to be used.</p>
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>Direct molecule object (BLOB)</li> <li>IUPAC name (VARCHAR2 or CLOB)</li> <li>HELM string (VARCHAR2 or CLOB)</li> <li>XHELM string (VARCHAR2 or CLOB)</li> <li>Chime string (VARCHAR2 or CLOB)</li> <li>SMILES string (VARCHAR2 or CLOB)</li> <li>InChI string (VARCHAR2 or CLOB)</li> <li>Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

A NUMBER that contains molecule weight of the specified molecule.

## Usage

The typical usage for the `mdlaux.molwt` function is to get the molecular weight for registration using the same ptable which is associated with the molecule domain index on the table into which the molecule is being registered. For example:

```
INSERT INTO moltable (extreg, ctab, molweight) VALUES
('ABC-123', MOL('/home/user/mol123.mol'),
MDLAUX.MOLWT('moltable', '/home/user/mol123.mol'));
```

## Examples

The `mdlaux.molwt` function can be used in a SELECT statement, such as to compute the molecular weight for a molecule in the local table `MOLTABLE` using the atomic weights associated with the corporate database. For example:

```
SELECT mdlaux.molwt('corpdb_mol_mdlix', ctab) FROM
moltable WHERE cdbregno = 1;
```

Another example of the `mdlaux.molwt` function highlights registration:

```
INSERT INTO moltable (corpid, ctab, molecularweight, formula) VALUES
('12345', mol('chimestring'),
mdlaux.molwt('moltable', 'chimestring'), mdlaux.molfmla
('moltable', 'chimestring'));
```

## Comments

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.molwtmax

Returns the maximum molecular weight for a generic structure, that is, the molecular weight of the heaviest enumerated specific structure of the generic structure. If the given molecule is a specific structure, `molwtmax` is the same as `mdlaux.molwt`.

## Syntax

`mdlaux.molwtmax(molIndexOrTable, molecule)`

Parameter	Description
<i>molIndexOrTable</i>	<p>The name of a molecule index, or the name of a table which contains exactly one molecule index. (The schema may be included, e.g. '<i>schema.table</i>'.)</p> <p>The value may be NULL, and can be used as one of the following:</p> <pre>mdlaux.molwtmax(NULL, ctab) mdlaux.molwtmax(' ', ctab)</pre> <p>in which case the global environment is used.</p> <p>Use the first argument to control what Ptable is used to supply atomic weights for calculation of the molecular weight. a NULL value will cause the global Ptable to be used.</p>
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server</li> </ul>

Parameter	Description
	<p>where Direct is installed.</p> <ul style="list-style-type: none"> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A NUMBER that contains maximum molecular weight of a generic structure

**Usage**

```
select mdlaux.molwtmax(molIndexOrTable, molecule)
   [, other-column-data]
from tablename
where condition;
```

**Example**

```
select mdlaux.molwtmax(' ', ctab)
   from samplegen
   where parent_sampleid = 'BENZ';
```

**See also**

[molwtmax](#)

[mdlaux.molwtmin](#)

**mdlaux.molwtmin**

Returns the minimum molecular weight for a generic structure, that is, the molecular weight of the lightest enumerated specific structure of the generic structure. If the given molecule is a specific structure, molwtmin is the same as mdlaux.molwt.

**Syntax**

```
mdlaux.molwtmin(molIndexOrTable, molecule)
```



Parameter	Description
<i>molIndexOrTable</i>	<p>The name of a molecule index, or the name of a table which contains exactly one molecule index. (The schema may be included, e.g. '<i>schema.table</i>'.)</p> <p>The value may be NULL, and can be used as one of the following:</p> <pre>mdlaux.molwtmin(NULL, ctab)</pre> <pre>mdlaux.molwtmin('', ctab)</pre> <p>in which case the global environment is used.</p> <p>Use the first argument to control what Ptable is used to supply atomic weights for calculation of the molecular weight. a NULL value will cause the global Ptable to be used.</p>
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A NUMBER that contains minimum molecular weight of a generic structure

**Usage**

```
select mdlaux.molwtmin(molIndexOrTable, molecule)
   [, other-column-data]
from tablename
where condition;
```

**Example**

```
select mdlaux.molwtmin('', ctab)
   from samplegen
   where parent_sampleid = 'Peptoid';
```

**See also**

[molwtmin](#)

[mdlaux.molwtmax](#)

**mdlaux.monoisotopicmass**

Computes the mono-isotopic mass of a molecule.

## Syntax

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule table which contains a single molecule domain index, or the name of a molecule domain index. The implicit Ptable controls the atom symbols used. If molIndexOrTable is NULL, the global cartridge Ptable is used for symbol resolution.
<i>molecule</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

## Return value

A NUMBER that contains the mono-isotopic mass of the molecule, or NULL if the mass cannot be computed. The domain index Ptable is automatically used to resolve atom symbols.

## Usage

```
insert into tablename(
    monoisotopicmass-field
    [,other-column-data]
)
values (
    mdlaux.monoisotopicmass(molIndexOrTable, molecule)
    [,other-value-data]
);
```

## Example

The following example registers monoisotopic mass into a table:

```
insert into moltable
(id, ctab,
 molweight,
 monoisotopicmolweight)
values ('mol1', mol('/home/joe/mol1.mol'),
 mdlaux.molwt('moltable', '/home/joe/mol1.mol'),
 mdlaux.monoisotopicmass('moltable', '/home/joe/mol1.mol'));
```

The following example shows the difference in mass for methane:

```
select mdlaux.molwt(null,'c'),
mdlaux.monoisotopicmass(null,'c')
from dual;

MDLAUX.MOLWT(NULL,'C') MDLAUX.MONOISOTOPICMASS(NULL,'C')
-----
16.0424616.                0313001
```

**Comments**

- Use the mdlaux.monoisotopicmass function when registering mono-isotopic mass into a table. The monoisotopicmass operator cannot be used for registration.
- The mdlaux.monoisotopicmass function only provides a result when the molecule contains atoms found in nature. If an input molecule contains pseudoatoms such as Pol, Mod, or X, the mdlaux.monoisotopicmass function returns NULL and the following error:  
MDL-1590: MonoisotopicMass failed: Not available

**See also**

[monoisotopicmass](#)

**mdlaux.numspecifics**

Returns the number of specific structures from enumerating the specified generic structure. A specific structure is a fully defined chemical structure (with no generic features). The mdlaux.numspecifics function returns 1 if the specified argument is a specific structure.

**Syntax**

mdlaux.numspecifics(*molecule*)

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A NUMBER (1 or higher) that indicates the number of specific structures from enumerating the generic structure.

### Usage

```
select column-data,
       mdlaux.numspecifics(molecule)
from tablename
where [conditions];
```

### Example

The following example returns the number of specific structures for the generic structures in the samplegen table.

```
select parent_sampleid,
       mdlaux.numspecifics(ctab)
from samplegen
where mdlaux.isgeneric(ctab)=1;
```

### Comments

This function is not indexed. When possible, avoid using it when running queries on large tables.

### See also

[numspecifics](#)

## mdlaux.rownemakey

Returns a string that the pre-computed NEMAKEY from the domain index secondary table.

### Syntax

```
mdlaux.rownemakey(molIndex, rowid, option)
```

Parameter	Description
<i>molIndex</i>	The name of a molecule domain index. It cannot be a molecule table.
<i>rowid</i>	The ROWID of the row in the molecule table containing the molecule for which the NEMAKEY should be returned.

Parameter	Description
<i>option</i>	<p>Specifies what to generate, and consists of one or more of the following three-letter keywords.</p> <p>The keywords are:</p> <ul style="list-style-type: none"> <li>■ CON - Returns constitutional NEMAKEY (30 characters). The constitutional key does not include any stereochemical information. It should be used to compare two molecules without regard to stereochemistry.</li> <li>■ STE - Returns stereochemical NEMAKEY (30 characters). This key includes stereochemical information for most cases, but will not differentiate mixtures of different types of enhanced stereochemical collections.</li> <li>■ EXA - Returns stereochemical NEMAKEY (30 characters). Returns no key if a FLEXMATCH verification is required, this occurs if the molecule contains data Sgroups or if it contains mixtures of different types of enhanced stereochemical collections. This is the key returned by Cheshire 4.1.</li> <li>■ FLG - Returns three (zero-padded) digits of NEMAKEY and cartridge flag information, as described below.</li> </ul> <p>The specified values are appended to the output string in the order in which the keywords appear in the option parameter. Any characters which are not keywords are appended to the output string as-is.</p> <p>Thus, to generate the stereo NEMAKEY followed by a dash followed by the flags, use 'STE-FLG' for the option parameter. If the option parameter is not present, is NULL, or is blank, then the default is to return 'EXA'. (The option parameter is optional; if it is not specified, it is the same as if NULL were specified.)</p>

**Return value**

A VARCHAR2 that contains the pre-computed NEMAKEY for a particular row, in response to the keywords used in the option parameter.

**Example**

The following example uses `mdlaux.rownemakey` to return the NEMAKEY string for a specific row containing a molecule:

```
select mdlaux.rownemakey('sample2d_mdlix',
    (select rowid from sample2d
     where corp_id = 'MUSE00500062'))
from dual;
```

**Comments**

The domain index must have NEMA key registration and searching enabled. Otherwise, the `ROWNEMAKEY` function will return NULL. If you do not know whether or not NEMA key searching is enabled, use the `mdlaux.molnemakey` function instead.

**See also**

[mdlaux.molnemakey](#)

*BIOVIA Direct Developers Guide > Using Direct > NEMAKEY searching and key generation*

**mdlaux.sequencetext**

Generates the biopolymer sequence text for a molecule.

**Syntax**

```
mdlaux.sequencetext(molIndexOrTable, molecule)
```

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule table that contains a single molecule domain index, or the name of a molecule domain index.
<i>molecule</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A CLOB containing the biopolymer sequence text, a string of single letters representing the individual monomers, for example amino acids.

The operator returns NULL if the structure is not an SCSR sequence molecule.

**Usage**

```
select mdlaux.sequencetext(molIndexOrTable, molecule) from tablename where condition;
```

**Example**

```
select mdlaux.sequencetext(null, 'c:\ct18_human.mol') as "SequenceText" from dual;
```

```
SequenceText
```

```
-----
```

```
MSPSSMCSPVPLLAASGQNRMTQGQHFLQKV
```

**mdlaux.setmolname**

Given a molfile or Chime CLOB, sets the molecule name and returns a modified molfile or Chime CLOB which contains the molecule name.

**Syntax**

```
mdlaux.setmolname(molecule, molname)
```

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Molfile string (CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (CLOB)</li> <li>■ SMILES string</li> <li>■ InChI string</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul> <p>Note: The molecule parameter cannot be a filename nor a VARCHAR2 string containing a molfile or Chime string.</p>
<i>molname</i>	The name of the molecule.

**Return value**

A temporary CLOB that contains the modified molfile or Chime string containing the specified molecule name. If the input was a Chime string, the output is a Chime string, otherwise it is a molfile string.

**Usage**

The typical usage for the `mdlaux.setmolname` function is to merge a separately stored molecule name into a molecule as it is fetched from the table. For example:

```
SELECT MDLAUX.SETMOLNAME(MOLFILE(ctab), 'New molecule name')
FROM moltable
WHERE extreg = 'ABC-123';
```

If you want to write the molfile containing the molecule name, you can use the `writebinaryfile` operator. For example:

```
select writefile(
    (select mdlaux.setmolname(molfile(ctab), 'New molecule name')
     from sample2d
     where cdbregno=128),
    'c:\BIOVIA\direct\testmolrxn\cwtest.mol')
from dual
```

**See also**

[mdlaux.getsavedmolname](#)

[mdlaux.molname](#)

**mdlaux.sgroupfields**

Returns a character string containing Sgroup field names and types for either the specified index or the global environment.

**Syntax**

```
mdlaux.sgroupfields(molIndexOrTable)
```

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule index, or the name of a table which contains exactly one molecule index. (The schema may be included, e.g. 'schema.table'.) If the value of this parameter is NULL, sgroupfields returns Sgroup field information for the global environment.

**Usage**

```
select mdlaux.sgroupfields(molIndexOrTable) from dual;
```

**Return value**

A VARCHAR2 that contains the Sgroup field names and types. This function returns NULL if there are no defined Sgroup fields

Each field is on its own line, lines are terminated by line-feed (\n) characters. Field names occupy the first 30 characters of each line, then one space, then the field type. For example:

```
AtomData          NUMERIC
BondData          TEXT
```

**mdlaux.smiles**

Returns a SMILES string representation of a molecule.

**Syntax**

```
mdlaux.smiles(molecule)
```

Parameter	Description
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>noncanonical</i>	(Optional) To generate noncanonical SMILES strings, use the argument 'noncanonical'. The default SMILES string is canonical.

**Return value**

A temporary CLOB that contains the SMILES string. The mdlaux.smiles function returns NULL if the SMILES string cannot be generated. Use mdlaux.errors to see the related error message.



**Usage**

```
select mdlaux.smiles(molecule) from dual;
select mdlaux.smiles(molecule, 'noncanonical') from dual;
```

**Example**

The following example shows the SMILES string for a molecule:

```
SQL> select mdlaux.smiles('f:/work/mols/muse1.mol') from dual;
```

```
MDLAUX.SMILES('F:/WORK/MOLS/MUSE1.MOL')
```

```
-----
CN1C(=O)N(C)c2ncn(C)c2C1=O
```

The following example shows the non-canonical SMILES string for a molecule:

```
SQL> select mdlaux.smiles('f:/work/mols/muse1.mol', 'noncanonical') from
dual;
```

```
MDLAUX.SMILES('F:/WORK/MOLS/MUSE1.MOL', 'NONCANONICAL')
```

```
-----
c12c(ncn1C)N(C)C(N(C)C2=O)=O
```

**Comments**

- There are limitations to the generation of SMILES strings. Not all BIOVIA molecule features can be handled. If the specified molecule cannot be handled, the mdlaux.smiles function returns NULL. Use mdlaux.errors to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also:**

[smiles](#)

*BIOVIA Direct Developers Guide > Using Direct > Getting the SMILES string*

*BIOVIA Direct Developers Guide > Using Direct > Limitations to the generation of SMILES string*

**mdlaux.smilestomolfile**

Returns a molfile string (CLOB) from a SMILES string (CLOB).

**Syntax**

```
mdlaux.smilestomolfile(smiles)
```

Parameter	Description
<i>smiles</i>	A CLOB containing a SMILES string

### Return value

A temporary CLOB that contains the converted molfile string. If the SMILES string cannot be converted, the `mdlaux.smilestomolfile` function returns NULL. Use `mdlaux.errors` to see related error message.

### Usage

```
select mdlaux.smilestomolfile(smiles) from dual;
```

### Example

The following example shows the converted molfile from a SMILES string:

```
select mdlaux.smilestomolfile('B1=NB=NB=N1') from dual;
```

```
MDLAUX.SMILESTOMOLFILE('B1=NB=NB=N1')
-----
-OEChem-12170814082D

6 6 00 0 0 0 0 0999 V2000
2.0000-2.50430.0000 B0 0 0 0 0 0 0 0 0 0 0 0 0
2.0000-3.50950.0000 N0 0 0 0 0 0 0 0 0 0 0 0 0
2.8675-4.00700.0000 B0 0 0 0 0 0 0 0 0 0 0 0 0
3.7350-3.50950.0000 N0 0 0 0 0 0 0 0 0 0 0 0 0
3.7350-2.50430.0000 B0 0 0 0 0 0 0 0 0 0 0 0 0
2.8675-1.99660.0000 N0 0 0 0 0 0 0 0 0 0 0 0 0
1 6 1 0 0 0 0
1 2 2 0 0 0 0
2 3 1 0 0 0 0
3 4 2 0 0 0 0
4 5 1 0 0 0 0
5 6 2 0 0 0 0
M END
```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[smiles](#)

*BIOVIA Direct Developers Guide > Using Direct > Conversion of SMILES strings to molfile*

## mdlaux.xhelm

Returns a XHELM string representation of a biopolymer sequence molecule.

### Syntax

```
mdlaux.xhelm(molIndexOrTable, molecule)
```

Parameter	Description
<i>molIndexOrTable</i>	The name of a molecule index, or the name of a table which contains exactly one molecule index. The schema may be included, e.g. 'schema.table'. If the value of this parameter is NULL the global environment is used.
<i>molecule</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a) .</li> <li>■ Direct molecule object (BLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

A temporary CLOB that contains the XHELM string. The helm operator returns NULL if the XHELM string cannot be generated, for example if the molecule is not a biopolymer or if an error occurs. Use `mdlaux.errors` to see the related error message.

### Usage

```
select mdlaux.xhelm(molecule) from dual;
select mdlaux.xhelm(molIndexOrTable, molecule) from dual;
```

### Example

The following example shows the XHELM strings for a molecule:

```
select mdlaux.xhelm('f:/work/mols/a20a1_human.mol') from dual;
MDLAUX.SMILES('F:/WORK/MOLS/A20AL_HUMAN.MOL')
```

```
-----
PEPTIDE1{M.K.L.F.G.F.R.S.R.R.G.Q.T.V.L.G.S.I.D.H.L.Y.T.G.S.G
.Y.R.I.R.Y.S.E.L.Q.K.I.H.K.A.A.V.K.G.D.A.A.E.M.E.R.C.L.A.R.R
.S.G.D.L.D.A.L.D.K.Q.H.R.T.A.L.H.L.A.C.A.S.G.H.V.K.V.V.T.L.L
.V.N.R.K.C.Q.I.D.I.Y.D.K.E.N.R.T.P.L.I.Q.A.V.H.C.Q.E.E.A.C.A
.V.I.L.L.E.H.G.A.N.P.N.L.K.D.I.Y.G.N.T.A.L.H.Y.A.V.Y.S.E.S.T
.S.L.A.E.K.L.L.F.H.G.E.N.I.E.A.L.D.K.V}$$$PEPTIDE1{ChainName
:Putative ankyrin repeat domain-containing protein 20A-like
protein MGC26718}|PEPTIDE1{ChainDescription:chain}$
```

### Comments

- XHELM strings can only be generated for biopolymer sequence molecules which do not contain any modified residues. Other molecules will return a NULL value from the helm operator.

- The environment is used to specify biopolymer template definitions and XHELM monomer definitions. The XHELM monomer definitions are always taken from the global environment even if a local environment is specified with the `molIndexOrTable` parameter.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

### See also

*BIOVIADirect Developers Guide > Using Direct > Getting the HELM string*

# Chapter 4:

## Reaction-Specific Operators and Functions

---

This chapter contains the reference listings for the functions and operators that are useful when working with reactions.

<a href="#">Reaction-Specific Operators</a>	147
<a href="#">Reaction-Specific Functions</a>	193

### Reaction-Specific Operators

In some cases, Direct offers both a function and an operator with the same name that behave identically to each other. For example, the `readfile` operator and the `mdl aux.readfile` function have the same functionality. In these cases, the description of the operator appears under this section. The Reaction-Specific Functions section lists the name of the function and then references the description under this section.

If both a function and an operator are available, use the function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators.

<a href="#">hasnostructs</a>	147
<a href="#">ncomponents</a>	148
<a href="#">rinchi</a>	149
<a href="#">rinchiauxinfo</a>	151
<a href="#">rinchikey</a>	153
<a href="#">rss</a>	154
<a href="#">rsshighlight</a>	160
<a href="#">rsstimeout</a>	161
<a href="#">rxn</a>	162
<a href="#">rxnautomap</a>	165
<a href="#">rxnautomapchange</a>	168
<a href="#">rxnautomapstatus</a>	169
<a href="#">rxnchime</a>	170
<a href="#">rxnctrsim</a>	171
<a href="#">rxnfile</a>	172
<a href="#">rxnflexmatch</a>	173
<a href="#">rxnflexmatchtimeout</a>	176
<a href="#">rxngzip64</a>	177
<a href="#">rxnimage</a>	178
<a href="#">rxnkeys</a>	180
<a href="#">rxnmol</a>	182
<a href="#">rxnmolsim</a>	183
<a href="#">rxnsim</a>	184
<a href="#">rxnsmiles</a>	189
<a href="#">rxnstringsegment</a>	190

### hasnostructs

Returns a NUMBER that indicates whether any component of the specified reaction is a nostruct (“no-structure”). A no-structure is a chemical structure that consists of zero fragments, that is, zero atoms

and zero bonds.

### Syntax

`hasnostructs(rxn)`

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. The value of this parameter can also be a reaction object.

### Return value

A NUMBER that indicates whether the specified reaction includes a no-structure. The return value will be 1 if the reaction does include a nostruct and 0 if the reaction does not include a no-structure.

### Usage

```
select hasnostructs(rxn)
      [, other-column-data]
from tablename
where condition;
```

### Comments

Applications can use the `hasnostructs` operator to “test” reactions before using them in an RSS query. Performing an RSS query using a reaction that contains a no-structure results in an error condition.

### See also

[rss](#)

## ncomponents

Returns the number of reactants or products in a reaction.

### Syntax

`ncomponents(rxn, comptype)`

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. The value of this parameter can also be a reaction object.
<i>comptype</i>	A NUMBER that indicates whether the components to count are reactants or products in the reaction. The possible values are: 1 - Counts the reactant molecules 2 - Counts the product molecules

### Return value

A NUMBER equal to the number of reactants or products in the reaction. `ncomponents` returns NULL if *comptype* is invalid, or if there is an error.

If *comptype* equals 1, and if the reaction does not contain any reactant, `ncomponents` returns 0 (zero).

If *comptype* equals 2, and if the reaction does not contain any product, `ncomponents` returns 0 (zero).

**Usage**

```
select ncomponents(rxn, comptype)
[, other-column-data]
from tablename
where condition;
```

Alternatively, in PL/SQL:

```
numvalue := mdlaux.ncomponents(rxn, comptype);
```

**Example**

The following example uses `ncomponents` to return the number of reactants in the matched reactions:

```
select rxnregno,
       ncomponents(rxn, 1)
from isisrx
where rss(rxn,
          '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn'
        )=1;
```

The following PL/SQL example uses `ncomponents` and the constant `mdlaux.product` to return the number of products in a reaction:

```
DECLARE
  products NUMBER;
  rxnval BLOB;
BEGIN
  select rxn into rxnval from isisrx where rxnregno=10;
  products := mdlaux.ncomponents(rxnval, mdlaux.product);
END;
```

**Comments**

- In PL/SQL, you can use the following constants for *comptype*:
  - `mdlaux.reactant` - Equivalent to 1, counts the reactant molecules
  - `mdlaux.product` - Equivalent to 2, counts the product molecules
- The typical usage of `ncomponents` is in a reaction table trigger which is intended to extract the component molecules from a reaction and insert them into a molecule table. Use `ncomponents` in conjunction with the `rxnmol` operator to determine the number of reactants and products to extract.

**See also**

[rxnmol](#)

*BIOVIA Direct Developers Guide > Using Direct > Working with Molecules in a Reaction*

**rinchi**

Returns an IUPAC standard International Chemical Identifier (standard “RInChI”) string for the specified reaction.

**Syntax**

```
rinchi(reaction, options)
```

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a reaction object.
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options.</p> <p>If no additional options are provided in the call to RINCHI, the standard RInChI string is generated.</p> <p>If any of the following options are specified, a non-standard RInChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbS - Absolute stereo (default)</li> <li>■ SReI - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RInChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A temporary CLOB that contains the RInChI string. The output string length will exceed 4000 characters for very large reactions. If the RInChI string cannot be generated, the `rinchi` operators returns NULL. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select rinchi(reaction, options) from dual;
```

**Example**

The following example shows the RInChI string for a reaction, using the default option:

```
select rinchi('/work/rxns/test.rxn') from dual;
```

```
RINCHI('/WORK/RXNS/TEST.RXN')
```

```
-----
RInChI=1.00.1S/C2H7ClSi/c1-4(2)3/h4H,1-2H3!C6H10O3/c1-3-9-6(8)4-5(2)7/h3-4H2,1-2H3<>C8H17ClO3Si/c1-5-11-8(10)6-7(2)12-13(3,4)9/h7H,5-6H2,1-4H3/t7-/m1/s1/d+
```



### Comments

- There are limitations to the generation of RInChI strings. Not all BIOVIA reaction features can be handled. The `rinchi` function returns NULL if the specified reaction cannot be handled. Use `mdlAux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[mdlAux.rinchi](#)  
[rinchikey](#)

## rinchiauxinfo

Returns the auxiliary information (AuxInfo) that is computed along with the IUPAC International Chemical Identifier (InChI) string for a molecule.

### Syntax

`rinchiauxinfo(rctab [, options])`

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a reaction object.
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options. If no additional options are provided in the call to RINCHIAUXINFO, the standard RinChI AuxInfo string is generated. If any of the following options are specified, a non-standard RinChI AuxInfo string is generated.</p> <ul style="list-style-type: none"> <li>■ <b>NEWPSOFF</b> - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ <b>FixedH</b> - Include Fixed H layer (default is 'not')</li> <li>■ <b>RecMet</b> - Include reconnected metals results (default is 'not')</li> <li>■ <b>SAbs</b> - Absolute stereo (default)</li> <li>■ <b>SRel</b> - Relative stereo</li> <li>■ <b>SRac</b> - Racemic stereo</li> <li>■ <b>SUCF</b> - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ <b>SNon</b> - Exclude stereo</li> <li>■ <b>SUU</b> - Include omitted unknown/undefined stereo</li> <li>■ <b>SLUUD</b> - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ <b>KET</b> - Account for keto/enol tautomerization (default is off)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RinChI AuxInfo string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A temporary CLOB that contains the RinChI AuxInfo string. The output string length will exceed 4000 characters for very large reactions. If the RinChI AuxInfo string cannot be generated, the `rinchiauxinfo` operators returns NULL. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select rinchiauxinfo(reaction, options) from dual;
```

**Example**

The following example shows the RinChI AuxInfo string for a reaction, using the default option:

```
select rinchiauxinfo(rctab) from dual;
```

```
RINCHIAUXINFO(RCTAB)
```

```
-----
RAuxInfo=1.00.1/0/N:2,3,4,1/E:
(1,2)/rA:4nSiCCCl/rB:s1;s1;s1;/rC:.7895,.1596,0;.0183,1.4909,0;-.6599,-.369,
0;1.0538,-
1.3563,0;!0/N:9,6,8,1,3,2,7,5,4/rA:9nCCCOOCC/rB:s1;s1;s2;d2;s3;d3;s4;s8;/r
C:-1.2727,-.933,0;.0655,-.1637,0;-
2.6067,-.1637,0;1.3995,-.933,0;.0655,1.375,0;-3.9367,-.933,0;-
2.6067,1.375,0;2.7336,-.1637,0;4.0677,-.933,0;<0/N:13,5,10,11,9,1,2,3,12,7,
6,4,8/E:
(3,4)/it:im/rA:13CCCCOOSiCCCClC/rB:s1;s1;P2;s2;s3;d3;s4;s6;s8;s8;s9;/rC
:.2778,.3641,0;-1.1928,1.2072,0;.2778,-1.3365,0;-1.1928,2.9126,0;-
2.6635,.3641,0;1.7485,-2.1892,0;-1.1928,-2.1892,0;.1437,3.8132,0;1.8635,-
3.7988,0;.6467,4.7617,0;.5461,2.8168,0;-0.6276,4.5605,0;2.7881,-4.3354,0;
```

**Comments**

- There are limitations to the generation of RinChI AuxInfo strings. Not all BIOVIA reaction features can be handled. The `rinchiauxinfo` function returns NULL if the specified reaction cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[mdlaux.rinchiauxinfo](#)  
[rinchi](#)

## rinchikey

Returns an IUPAC International standard Chemical Identifier (standard “RInChI”) key for the specified reaction. The RInChI key is a 27-character hashed form of the RInChI string. The `rinchikey` operator generates the key by first generating the RInChI string, and then calling an RInChI library operator to convert the string into the 27-character key.

### Syntax

```
rinchikey(reaction, options)
```

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the binary chemical structures. The field name is normally CTAB. The value of this parameter can also be a reaction object.
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options. If no additional options are provided in the call to RINCHI, the standard RInChI string is generated. If any of the following options are specified, a non-standard RInChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is ‘not’)</li> <li>■ RecMet - Include reconnected metals results (default is ‘not’)</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown (‘u’) and undefined (‘?’) are different (default for both is ‘?’)</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RInChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

### Return value

A VARCHAR2 that contains the 27-character RInChI key. The `rinchikey` operator returns NULL if the RInChI string cannot be generated. Use `md1aux.errors` to see the related error message.

### Usage

```
select rinchikey(reaction,options)
      [, other-column-data]
from tablename
where condition;
```

### Example

The following example shows the RInChI key the reactions in a table, using the default option:

```
select rinchikey(ctab) from rxntable;
RINCHIKEY(CTAB)
-----
MVPPADPHJFYWMZ-UHFFFAOYSA-N
```

**Comments**

There are limitations to the generation of RInChI strings. Not all BIOVIA reaction features can be handled. The `rinchikey` operator returns NULL if the specified reaction cannot be handled. Use `mdlaux.errors` to see the related error message.

**See also**

[mdlaux.rinchikey](#)  
[rinchi](#)

**rss**

Finds reactions that contain one (and only one) of the following:

- The reaction substructure that you specify in the query. A reaction substructure is a portion of a reaction with your choice of atom and bond query features, mapped atoms, and restrictions on the reacting centers.
- The molecule component that you specify in the query. Depending on the contents of the `rss-flags` parameter, `rss` will search for the molecule as a product or as a reactant component of the reaction.

If query contains reaction information, `rss` will search for a reaction substructure. In that case, the `rss-flags` parameter is optional.

If query contains molecule information, `rss` will search for a molecule component. In that case, the `rss-flags` parameter is required.

**Syntax**

```
rss(rxn, query [, rss-flags], rss-number])
```

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. <i>rxn</i> can also be a BLOB that contains a reaction object.
<i>query</i>	<ul style="list-style-type: none"> <li>■ Reaction substructure or molecule to search for. <i>query</i> can use one of the following formats:</li> <li>■ File path of a rxnfile (VARCHAR2). The rxnfile must be located on the server where Direct is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Filepath of a molfile (VARCHAR2). The molfile must be located on the server where Direct is installed.</li> <li>■ Molfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"><li>■ SMILES string (VARCHAR2 or CLOB)</li><li>■ InChI string (VARCHAR2 or CLOB)</li><li>■ Reaction object created by a previous operation (BLOB)</li><li>■ Molecule object created by a previous operation (BLOB)</li><li>■ Reaction SMILES string (VARCHAR2 or CLOB)</li><li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li></ul> <p>Notes:</p> <ul style="list-style-type: none"><li>■ If <i>query</i> contains any NOSTRUCT molecules, the search will fail.</li><li>■ <i>query</i> cannot be NULL.</li></ul>

Parameter	Description
<i>rss-flags</i>	<p>Any or a combination of the following values, separate multiple options with a space:</p> <ul style="list-style-type: none"> <li>■ <b>NOFS</b> - The presence of 'NOFS' causes sss to refrain from using the fastsearch index when performing the search. If 'GENERICS' is also specified, the 'NOFS' option is ignored.</li> <li>■ <b>IgnoreChargesInPiSystems</b> – When absent, substructure mapping of pi systems, i.e. of haptic bonds, takes total charge into account and will not allow a query that has a charged pi system or metal connected to the pi system to map to a target which is uncharged. When the option is present, total charge in the pi system and metal attached to the pi system is ignored in both query and target. This allows the radical representation of ferrocene to map to the charged representation of ferrocene; without the option these two will not match.</li> <li>■ <b>RingHomologyGroupsOnlyMapTerminalRings</b> – When this option is present, ring homology query atoms will map only to terminal ring assemblies in the target. The atoms will not map to ring assemblies which have any non-ring attachments.</li> <li>■ <b>InterpretRAtomsLiterally</b> - When this option is present an R atom in the query will only match an R atom in the target, it does not have its normal meaning of matching any atom including hydrogen.</li> <li>■ <b>InterpretXAtomsLiterally</b> - When this option is present an X atom in the query will only match an X atom in the target, it does not have its normal meaning of matching any atom including hydrogen.</li> <li>■ <b>InterpretQueryAtomsLiterally</b> - When this option is present an A, Q, X or M atom in the query will only match a corresponding A, Q, X or M atom in the target, it does not have its normal meaning as an atom query feature.</li> <li>■ <b>IgnoreStereo</b> - When this option is present all atom stereochemistry in the query will be ignored during matching. This includes enhanced stereochemistry and higher-order stereochemistry. Double bond stereochemistry is still matched when marked in the query.</li> </ul> <p>If query is a reaction, the only valid values for rss-flags are the options listed above. If query is a molecule rss-flags must include one of the following options to specify how the molecule should be searched:</p> <p><b>REACTANT</b> - Find all reactions which contain this molecule as a substructure in one of the reactants. The query is converted into a reactant-only reaction and used as an RSS query.</p> <p><b>REACTANT NOT PRODUCT</b> - Find all reactions which contain this molecule as a substructure in one of the reactants, and do not contain this substructure in any of the products. The query is converted into a reactant-only reaction and used as an RSS query. Each hit is then validated to ensure it does not contain the substructure in any product.</p> <p><b>PRODUCT</b> - Find all reactions which contain this molecule as a substructure in one of the products. The query is converted into a product-only reaction and used as an RSS query.</p> <p><b>PRODUCT NOT REACTANT</b> - Find all reactions which contain this molecule as a substructure in one of the products, and do not contain this substructure in any of the reactants. The query is converted into a product-only reaction and used as an RSS query. Each hit is then validated to ensure it does not contain the substructure in any reactant.</p> <p>Notes:</p> <ul style="list-style-type: none"> <li>■ If the query is a molecule and rss-flag is set to either <b>PRODUCT</b>, <b>REACTANT</b>, <b>PRODUCT</b></li> </ul>

Parameter	Description
	NOT REACTANT, or REACTANT NOT PRODUCT, the performance is slower than an equivalent search using SSS and a JOIN or IN clause that converts molecule results to reactions. See an example in the Comments section.
<i>rss-number</i>	A NUMBER equal to the rss-number parameter used with the ancillary operator <code>rsshighlight</code> . This parameter only applies if you use <code>rsshighlight</code> .

### Return value

The NUMBER 1 indicates that the query matched the reaction, the number 0 indicates that the query did not match the reaction. When you use `rss` in a WHERE clause, always test the return value for a result of 1.

### Usage

```
select column-data
from tablename
where rss(rxn, query, rss-flags)=1
[operator other-conditions];
```

The `rss` operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a reaction is really a reaction substructure search hit from a complex WHERE clause.

```
select rss(rxn, query, rss-flags)
[, other-column-data]
from tablename
where condition;
```

### Example

The following example uses `rss` to find reactions that contain a reaction substructure:

```
select count(*)
from samplerx_reaction
where rss(
  rctab,
  '/opt/BIOVIA/Direct/examples/rxnfiles/query1.rxn'
)=1;
```

The query used in this simple example is contained in a rxnfile that is located in the examples/rxnfiles directory of the Direct installation. For more examples, see [Reaction Substructure Search](#).

### Comments

- To negate the results of `rss`, use the SQL operator NOT. For example, to count all reactions that do not contain a specific reaction substructure:

```
select count(*) from isisrx
where not rss(rxn,
  '/opt/BIOVIA/Direct/examples/rxnfiles/query1.rxn'
)=1;
```

- To highlight the substructure in the resulting structures, use the `rsshighlight` operator. You can use any number as the `rss-number` parameter, but it must match the `rss-number` parameter used with

rsshighlight. For example:

```
select rsshighlight(2)
from samplerx_reaction
where rss(rctab,
  '/opt/BIOVIA/Direct/examples/rxnfiles/query1.rxn',
  2
)=1;
```

- To check for errors from the rss operator, call the function `mdl aux.errors`.
- If there is no domain index on a reaction column, an rss search will execute more slowly than if a domain index is present and Oracle chooses to use it. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command `EXPLAIN PLAN`. For details about creating the reaction domain index, see “Creating Reaction Tables” in the *BIOVIA Direct Administration Guide*.
- A mapped reaction query improves the performance of `rss`. As a general guideline, the more information you provide in your query, the better the search performance.
- `rss` supports attached data (Sgroup data). If the query to be searched contains attached data, any attached data in the query will be matched with those in the target. The following are the valid query operators for attached data:
  - `<`
  - `>`
  - `<=`
  - `>=`
  - `=` like between
  - exists or is not null
  - null or is null

Numeric attached data in the reaction query or reaction to be registered can be integers, floating point numbers, or two numbers separated by white space. Two numbers separated by white space are considered a range.

Direct trims off leading or trailing white space in the text query data. However, Direct does not trim white space in the text registration data. Text matching uses Oracle, which is normally case-dependent. For example, 'ABC' will not match 'abc'. To support case-independent matching, the attached data query text may be contained within the parentheses of `UPPER()` or `LOWER()` Oracle functions. The text in both query and target will then be upper-cased (or lower-cased) prior to matching. For example, if case-dependent matching is desired, when defining the attached data query in BIOVIA Draw, specify the exact text:

Enter query text: John Smith

To enable case-independent matching include `UPPER()` (case doesn't matter), for example: Enter query text: `upper(john smith)`

### Notes:

- In order to use the `UPPER()` (or `LOWER()`) functions, your text cannot include a closing parenthesis.
- If you use the between query operator and you want to enable case-independent matching, you must use the `UPPER()` (or `LOWER()`) function on both operands, not just one



If you want to search for quotation marks using a query with a leading quotation mark, enclose the entire query in quotation marks. You can either use double quotation marks to enclose the query (see Alternative 1 in the following table), or use single quotation marks to enclose the query and escape the embedded quotation mark with another single quotation mark (see Alternative 2 in the following table). The following table shows two alternatives for entering the query text for the desired query:

Desired query	Alternative 1	Alternative 2
'23-x'b	""23-x'b"	'''23-x''b'

For more information about attached data see *Attached Data* in *BIOVIA Chemical Representation*.

- `rss` does not match reactions that contain Rgroup queries, generic structures, or polymer Sgroups. Direct currently does not support these features. If a reaction contains polymer Sgroups, the cartridge returns the following error:

MDL-0427: Rxnfile containing polymer Sgroups is not supported

If a reaction contains Rgroup queries or generic structures, the cartridge returns one or more of the following errors:

MDL-0279: CTlib error: [RDRFIL]RDRFIL:Error reading reaction generic molfile (1)

MDL-0279: CTlib error: [RDRFIL]getV2000Rxnfile:Error reading reaction generic molfile (2)

MDL-0041: Unable to read rxnfile, error=3002

- If the RSS query is a product molecule, the performance is slower than an equivalent search using SSS and a JOIN or IN clause that converts molecule results to reactions. This applies to RSS searches using a molecule as the query structure, and the `rss-flag` set to either “product”, “reactant”, “product not reactant”, or “reactant not product”.

For example, the following query attempts to find all reactions which have the query molecule substructure in one of the products:

```
select rxnid from reaction_table where rss(rctab, 'query.mol',
'product')=1;
```

The above query performs slower than the following query which uses an IN clause and an sss search:

```
select rxnid from reaction_table where rxnid in
(select rxnid from product_table p, molecule_table m
where p.molid = m.molid and sss(m.ctab, 'query.mol')=1);
```

- Multi-threaded reaction substructure search can be enabled. By default, reaction substructure search is not multi-threaded. To set the number of threads used during substructure searching, use the administrative function `mdlAux.setProperty('NTHREADS', numberOfThreads)`. For details, see *Command Reference* in the *BIOVIA Direct Administration Guide*.

#### See also

[rsshighlight](#)

[Reaction Substructure Search](#)

*BIOVIA Direct Developers Guide > Using Direct > Searching for reactions*

*BIOVIA Direct Developers Guide > About Direct > Direct Domain Indexes*

## rsshighlight

Returns a Chime representation of a reaction that:

- Matches a reaction substructure query. `rsshighlight` is an ancillary operator of the `rss` operator. To use `rsshighlight`, you must also use `rss` within the same SQL statement.
- Contains highlight information for the matched reaction substructure. The highlighted reaction can be rendered by BIOVIA Draw or a version of ISIS/Draw, Chime, or Chime Pro that supports molecule highlighting.

### Syntax

`rsshighlight(rss-number)`

Parameter	Description
<i>rss-number</i>	A NUMBER equal to the <code>rss-number</code> parameter that is used with the <code>rss</code> operator.

### Return value

A CLOB that contains the Chime representation of a candidate reaction, and contains highlight information for the matched reaction substructure. `rsshighlight` stores the return value in a temporary CLOB. The Chime string uses the V3000 format, and contains highlight information as CTlib collection objects.

### Usage

```
select rsshighlight(rss-number)
[, other-column-data]
from tablename
where rss(rxn, query, rss-number)=1
[operator other-conditions];
```

### Example

The following example returns a Chime string, as a result of a reaction substructure search, that contains highlight information for a reaction substructure.

```
select rxnmdlnumber,
       rsshighlight(3)
from samplerx_reaction
where rss(
       rctab,
       '/opt/BIOVIA/direct/examples/rxnfiles/query1.rxn',
       3
       )=1;
```

**Note:** The number 3 is used to correlate the `rss` operator in the WHERE clause with the `rsshighlight` operator in the SELECT clause. This could be any number as long as the values in the two operators match.

### Comments

- The `rss-number` parameters for `rsshighlight` and `rss` operators must match. If the `rss-number` parameters do not match, or if you use `rsshighlight` without using `rss`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

- You can convert the highlighted Chime string to a rxnfile. The `rsshighlight` operator returns a Chime string because the Chime structure renderer can be typically used to display the highlighted reactions. To display the highlighted reaction in a rxnfile string instead of a Chime string, use the `mdlaux.chimetoclob` function. For example:

```
select rxnmdlnumber,
       mdlaux.chimetoclob(rsshighlight(3))
from samplerx_reaction
where rss(rctab,
         '/opt/BIOVIA/direct/examples/rxnfiles/query1.rxn',
         3
        )=1;
```

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();}
```

- Because the `rsshighlight` operator is always used with the `rss` operator, highlighting a reaction always involves a reaction substructure search. However, if you simply want to highlight a reaction without searching a table, you can directly map a query to the target, such as the following example:

```
select rsshighlight(1)
from dual
where rss(mol('/home/users/rxns/target.rxn'),
         '/home/users/rxns/query.rxn',1)=1;
```

The following example shows that you can also put `rss` with the `rsshighlight` operator in the `SELECT` clause. In this example, if the query is not a reaction substructure of the specified target, `rsshighlight` returns the unhighlighted target reaction.

```
select rss(mol('/home/users/rxns/target.rxn') "RSSResult",
         '/home/users/rxns/query.rxn',1),
       rsshighlight(1) "RSSHighlight"
from dual;
```

#### See also

[rss](#)

### rsstimeout

Returns a the timeout status value from an `rss` search.

`rss` will return as matches those candidates for which the matching algorithm times out. Such candidates may or may not be actual matches. This ancillary operator gives information about that timeout status.

#### Syntax

`rsstimeout(rss-number)`

Parameter	Description
<i>rss-number</i>	A NUMBER equal to the <i>rss-number</i> parameter that is used with the <i>rss</i> operator.

**Return value**

A NUMBER that indicates the status of the reaction substructure search. The possible values are:

Value	Description
0	The <i>rss</i> search did not time out.
1	The <i>rss</i> search timed out.
NULL	The target was not a match to the query.

**Usage**

```
select rsstimeout(rss-number)
[,other-column-data]
from tablename
where rss(rxn, query, rss-number)=1
[operator other-conditions];
```

**Example**

The following example returns the timeout status while searching the table.

```
select extreg,
       rsstimeout(3) "Timeout"
from moltable
where rss(
       rxncol,
       '/opt/BIOVIA/direct/examples/rxnfiles/query1.mol',
       3
       )=1;
```

**Note:** The number 3 is used to correlate the *rss* operator in the WHERE clause with the *rsstimeout* operator in the

SELECT clause. This could be any number as long as the values in the two operators match.

**Comments**

- The *rss-number* parameters for *rsstimeout* and *rss* operators must match. If the *rss-number* parameters do not match, or if you use *rsstimeout* without using *rss*, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

**See also**

[rss](#)

**rxn**

Converts a VARCHAR2 or CLOB reaction into a BLOB reaction object.

**Syntax**

```
rxn(structure)
```

Parameter	Description
<i>structure</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

**Return value**

A BLOB that contains the reaction object. The reaction object is a packed, binary representation of a reaction.

**Usage**

```
insert into tablename(
    extreg, rxncol,
    [, other-column-data]
)
values (
    rxnregno,
    rxn(rxnfile-structure)
    [, other-value-data]
);
```

```
insert into tablename(
    extreg, rxncol,
    [, other-column-data]
)
values (
    rxnregno,
    rxn(chime-structure)
    [, other-value-data]
);
```

```
update tablename
set rxncol = rxn(rxnfile-structure)
    [, other-column=other-value]
where condition;
```

```
update tablename
```

```
set rxncol = rxn(chime-structure)
[,other-column=other-value]
where condition
```

### Example

The following registration example uses rxn to cast into a BLOB the contents of a rxnfile:

```
insert into samplerx_reaction(
    rxnmdlnumber,
    rctab
)
values (
    'NEWRXN',
    rxn('/opt/BIOVIA/direct/examples/rxnfiles/newrxn1.rxn')
);
```

The following update example uses rxn to cast into a BLOB the contents of a rxnfile:

```
update samplerx_reaction
set rctab =
    rxn('/opt/BIOVIA/direct/examples/rxnfiles/newrxn2.rxn')
where rxnmdlnumber = 'RXCI94006733';
```

The following PL/SQL example uses the package function name mdlaux.rxn to cast into a BLOB the contents of a rxnfile, and then insert it into a table:

```
DECLARE
    rxnval blob;
BEGIN
    rxnval :=mdlaux.rxn(
        '/opt/BIOVIA/direct/examples/rxnfiles/query1.rxn');
    insert into samplerx_reaction(rxnmdlnumber, rctab)
    values('RXCI94006733', rxnval);
END;
```

**Note:** The reactions used in these examples are contained in a rxnfile that are located in the examples/rxnfiles directory of the Direct installation. For more examples, see [Reaction Registration](#).

### Comments

- To register or update a reaction, use the rxn operator to convert a VARCHAR2 or aCLOBreaction to a BLOB.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

- Warnings appear when a reaction is modified during registration. The following four warnings occur when the stereochemistry perception determines that an atom marked as a stereocenter is not

actually a valid stereocenter. This might occur because a trivalent nitrogen stereocenter was removed, which caused a remaining stereocenter to be invalid because of symmetry.

MDL-2010: Warning: Removed invalid Chiral flag  
 MDL-2011: Warning: Removed invalid non-tetrahedral stereo center  
 (s) MDL-2012: Warning: Removed invalid atom stereo center(s)  
 MDL-2129: Warning: Flattening 3D molecule to 2D  
 MDL-2134: Warning: Removed wedge from atom that is not a stereo  
 center  
 MDL-2141: Warning: Removed invalid double-either bond(s) MDL-5092:  
 Warning: Added implicit hydrogens to metal atom

#### Tips:

To preserve the original molecule or reaction, add a new column to the table and store the original molfile, rxnfile, or Chimestring in it.

#### See also

#### [Reaction Registration](#)

*BIOVIA Direct Developers Guide* > Using Direct > Inserting, Updating, and Deleting Reactions

### rxnautomap

Automatically assigns atom-to-atom mapping in a reaction (“automaps a reaction”), and returns a CLOB that contains the automapped reaction. The automapped reaction contains a unique mapping number for each corresponding atom in the reactants and products in the reaction.

#### Syntax

rxnautomap(rxn, mode)

Parameter	Description
<i>rxn</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>mode</i>	<p>A VARCHAR2 string that specifies how the reaction will be automapped. mode is not case-sensitive. The possible values are:</p>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ <b>RegenAlter</b> - Computes the atom-atom maps and bond change marks for the reaction, using any existing bond marks to guide the mapping. RegenAlter assumes the existing marks might be wrong and can be altered. This mode is used by the Regenerate AAMappings command in REXEC.</li> <li>■ <b>Clear</b> - Removes all existing atom-atom maps and bond marks from the reaction. Clear does not perform any mapping.</li> <li>■ <b>Default</b> - Computes the atom-atom maps and bond change marks for the reaction, using the existing bond marks. Default assumes the existing marks are absolutely correct. This mode is used by ISIS/Base when mapping reactions.</li> <li>■ <b>RegenKeep</b> - equivalent to Default.</li> <li>■ <b>NULL</b> or empty string - equivalent to Default.</li> <li>■ <b>RegenAuto</b> - Computes the atom-atom maps and bond change marks for the reaction, without further input. RegenAuto disregards the existing maps.</li> </ul>

### Return value

A CLOB that contains the `rxnfile` representation of the automapped reaction. The CLOB includes line-feed characters (0x0a) that separate the lines within the rxnfile. `rxnautomap` stores the return value in a temporary CLOB.

`rxnautomap` returns an automapped reaction if `rxnautomapstatus` returns a non-zero value. If `rxnautomapstatus` returns 0 (zero), `rxnautomap` returns NULL.

### Usage

```
select rxnautomap(rxn, mode)
      [, other-column-data]
from tablename
where condition;
```

```
select column-data
from tablename
where search-operator(
      rxn,
      rxnautomap(rxn, mode)
      [, search-parameter]
)=1;
```

### Example

The following example uses the `rxnautomap` operator to automap a reaction in a file:

```
select rxnautomap(
      '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
      'default')
from dual;
```

The following example uses the `rxnautomap` operator to automap a reaction query structure, and use it for a reaction substructure search:

```
select rxnmdlnumber
from samplerx_reaction
```



```

where rss(
  rctab,
  rxnautomap('/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
    'regenauto'
  )=1;

```

### Comments

After calling `rxnautomap`, you can get the:

- Status of the automap operation. To get the automap status, use the `rxnautomapstatus` operator.
- Number of changes performed by the automap operation. To get the automap changes, use the `rxnautomapchange` operator.

The following example automaps a specific reaction, and then gets the automap status and the number of changes:

```

select rxnautomap(
(select rctab from samplerx_reaction where rxnmdlnumber='RXCI94070168'),
'regenauto')
from dual;
select rxnautomapstatus(0) from dual;
select rxnautomapchange(0) from dual;

```

- Avoid using a combination of the `rxnautomap`, `rxnautomapstatus`, and `rxnautomapchange` operators within one SQL statement. Oracle might not call operators in the order they appear in a SQL statement. Even if the `rxnautomap` operator appears first in the SELECT statement, Oracle can call `rxnautomapstatus` and `rxnautomapchange` first.
- The following example uses the `rxnautomap`, `rxnautomapstatus`, and `rxnautomapchange` operators in one SELECT statement. This example attempts to automap all reactions in a table, and stores all the mapped reactions and mapping status in a new table. In this example, Oracle actually calls `rxnautomapstatus` and `rxnautomapchange` before `rxnautomap`. The result is that the values of the mapping status are actually associated with the previous row in the table.

### IMPORTANT!

IMPORTANT! In the following example, Oracle does not execute the operators in the order they appear in the SELECT statement. This example results in corrupted data. Do not use this example in your application. This example results in corrupted data!

```

create table automapped (
  extreg varchar2(20), rxn blob,
  mapstatus number, bondschanged number);
insert into automapped
  (select rxnmdlnumber, rxn(rxnautomap(rctab, null)), rxnautomapstatus(0),
  rxnautomapchange(0)
  from samplerx_reaction);

```

- In a PL/SQL application, use the `mdlaux.automap` function. This function returns three values: the mapped reaction, the mapping status, and the number of changes. `mdlaux.automap` is more efficient because it eliminates additional calls to the server.
- The `rxnautomap` operator ignores all attached data (Sgroup data) when it automaps a reaction.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[rxnautomapchange](#)

[rxnautomapstatus](#)

## rxnautomapchange

Returns the number of changes performed in the last automap operation.

### Syntax

rxnautomapchange(0)

Parameter	Description
0	This can be any number. This parameter is not used.

### Return value

A NUMBER that indicates the number of changes performed in the last automap operation. The following are the possible values:

Value	Description
0	No change. The last automap operation did not change atom-atom maps, bond marks or inversion/retention flags. The rxnautomap operator returns a reaction similar to the input reaction.
>0	The number of atom-atom maps and bond marks which the user had specified, and which were changed to some other value by rxnautomap. The user should examine the resulting mapped reaction.
-1	No change in the existing atom-atom maps and bond marks, but new atom-atom maps, bond marks, or inversion/retention flags were added. If the automap mode for the rxnautomap operator is RegenAlter, this could also mean that the original atom-atom maps were modified.

### Usage

```
select rxnautomapchange(0) from dual;
```

### Example

The following example automaps a specific reaction, and uses rxnautomapchange to return the number of changes in the automapped reactions:

```
select rxnautomap(
  '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
  'regenalter')
from dual;
select rxnautomapchange(0) from dual;
```

**Comments**

- Avoid using a combination of the `rxnautomap`, `rxnautomapstatus`, and `rxnautomapchange` operators within one SQL statement. Oracle might not call operators in the order they appear in a SQL statement. Even if the `rxnautomap` operator appears first in the SELECT statement, Oracle can call `rxnautomapstatus` and `rxnautomapchange` first.
- In a PL/SQL application, use the `mdlaux.automap` function. This function returns three values: the mapped reaction, the mapping status, and the number of changes. `mdlaux.automap` is more efficient because it eliminates additional calls to the server.

**See also**

[rxnautomap](#)  
[rxnautomapstatus](#)

BIOVIA Chemical Representation > Reaction Representation

**rxnautomapstatus**

Returns the status of the last `automap` operation.

**Syntax**

```
rxnautomapstatus(0)
```

Parameter	Description
0	This can be any number. This parameter is not used.

**Return value**

A NUMBER that indicates the status of the last `automap` operation. The following are the possible values:

Value	Description
0	The <code>rxnautomap</code> operator failed, and returns NULL.
1	The <code>rxnautomap</code> operator succeeded, and returns the automapped reaction.
2	The <code>automap</code> mode that is specified with the <code>rxnautomap</code> operator is invalid. <code>rxnautomap</code> returns the input (unchanged) reaction.
3	The <code>rxnautomap</code> operator attempted to map the reaction, but failed. It returns the input (unchanged) reaction.
>4	The <code>rxnautomap</code> operator succeeded, but the mapping might be incorrect and must be examined by the user. <code>rxnautomap</code> returns the automapped reaction. The actual numeric return value has no significance. A typical value is 39.

**Usage**

```
select rxnautomapstatus(0) from dual;
```

**Example**

The following example automaps a specific reaction, and uses `rxnautomapstatus` to return the status of the `automap` operator:

```
select rxnautomap(  
    '/opt/BIOVIA/direct2021/query.rxn',  
    'regenalter')  
from dual;  
select rxnautomapstatus(0) from dual;
```

**Notes:**

The query used in this simple example is contained in a rxnfile that is located in the examples/rxnfiles directory of the Direct installation.

**Comments**

- Avoid using a combination of the rxnautomap, rxnautomapstatus, and rxnautomapchange operators within one SQL statement. Oracle might not call operators in the order they appear in a SQL statement. Even if the rxnautomap operator appears first in the SELECT statement, Oracle can call rxnautomapstatus and rxnautomapchange first.
- In a PL/SQL application, use the mdlaux.automap function. This function returns three values: the mapped reaction, the mapping status, and the number of changes. mdlaux.automap is more efficient because it eliminates additional calls to the server.

**See also**

[rxnautomap](#)

[rxnautomapchange](#)

BIOVIA Chemical Representation > Reaction Representation

### rxnchime

Converts a BLOB reaction object into a CLOB that contains the Chime string representation of a reaction.

**Syntax**

```
rxnchime(rxn)
```

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. <i>rxn</i> can also be a BLOB that contains a reaction object.

**Return value**

A CLOB that contains the Chime string representation of a reaction. rxnchime stores the return value in a temporary CLOB.

**Usage**

```
select rxnchime(rxn)  
    [, other-column-data]  
from tablename  
where condition;
```

**Example**

The following example uses the rxnchime operator to return the Chime string representation of a reaction:

```
select rxnchime(rctab)
```

```
from samplerx_reaction
where rxnmdlnumber='RXCI94070168';
```

### Comments

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[rxnfile](#)

[Fetching Reactions Using the Chime Format](#)

BIOVIA Direct Developers Guide > Using Direct > Fetching Reactions

## rxnctrsim

Returns the reacting center similarity value from a similarity search. Higher values indicate greater similarity.

### Syntax

```
rxnctrsim(sim-number)
```

Parameter	Description
<i>sim-number</i>	A NUMBER equal to the sim-number parameter that is used with the rxnsim operator.

### Return value

A NUMBER ranging from 0 to 100 that represents the reacting center similarity value.

### Usage

```
select rxnctrsim(sim-number)
[, other-column-data]
from tablename
where rxnsim(rxn, query, simtype, sim-number)=1 [operator other-conditions];
```

### Example

The following example returns the reacting center similarity values for reactions that are at least 80% similar to the reacting centers in the query, and at least 20% similar to the molecules in the query:

```
select rxnmdlnumber,
       rxnctrsim(1) "RxnCtrSim"
from samplerx_reaction
where rxnsim(
    rctab,
    '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
    '80 20',
```

```
1
) = 1;
```

### Comments

- The `sim-number` parameters for `rxnctrsim` and `rxnsim` operators must match. If the `sim-number` parameters do not match, or if you use `rxnctrsim` without using `rxnsim`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

- If the query specified in `rxnsim` does not contain reacting center features such as bond marks, the reacting center component of similarity is ignored during a search.
- If the query specified in `rxnsim` (or if the candidate) does not contain reacting center features, `rxnctrsim` returns NULL. This indicates that the similarity was ignored during the search. For example, if the specified similarity threshold is 80%, you should only see hits of 80% or greater, or hits where the `rxnctrsim` value is NULL (ignored).

### See also

[rxnsim](#)

[rxnmolsim](#)

## rxnfile

Converts a BLOB reaction object into a CLOB that contains the `rxnfile` representation of a reaction.

### Syntax

```
rxnfile(rxn)
```

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. <i>rxn</i> can also be a BLOB that contains a reaction object.

### Return value

A CLOB that contains the `rxnfile` representation of a reaction. The CLOB includes line-feed characters (0x0a) characters that separate the lines within the `rxnfile`. `rxnfile` stores the return value in a temporary CLOB.

### Usage

```
select rxnfile(rxn)
       [, other-column-data]
from tablename
where condition;
```

### Example

The following example uses `rxnfile` to return the CLOB that contains the `rxnfile` representation of a reaction:

```
select rxnfile(rctab)
from samplerx_reaction
where rxnmdlnumber='RXCI94070168';
```

The following example uses `rxnfile` and `writebinaryfile` to write a reaction to a `rxnfile`:

```
select writefile(
  rxnfile(rctab),
  '/opt/BIOVIA/direct/examples/rxnfiles/myrxn.rxn')
from samplerx_reaction
where rxnmdlnumber='RXCI94070168';
```

### Comments

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[rxnchime](#)

[writebinaryfile](#)

[Fetching Reactions Using the Rxnfile Format](#)

*BIOVIA Direct Developers Guide > Using Direct > Fetching Reactions*

## rxnflexmatch

Finds reactions that are an exact match of the structure that you specify in the query. `rxnflexmatch` accepts flexmatch-parameters that allow you to restrict or relax the definition of an exact match.

### Syntax

`rxnflexmatch(rxn, query, rxnflexmatch-parameters [, rxnflexmatch-number])`

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. <i>rxn</i> can also be a BLOB that contains a reaction object.
<i>query</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> <li>■ Note: <i>query</i> cannot be NULL.</li> </ul>
<i>rxnflexmatch-parameters</i>	<p>A VARCHAR2 string that contains the following: [subset] flexmatch-switches where:</p> <ul style="list-style-type: none"> <li>■ subset - Indicates that the candidate can contain more reactants and/or products than the query</li> <li>■ flexmatch-switches - a switch or a combination of switches that allows you to selectively restrict or relax the search criteria based on the query structure. The string cannot be NULL.</li> <li>■ 'all' finds the exact match, and is the most restrictive match.</li> </ul> <p>For details about the rxnflexmatch parameters, see the “Exact Search (Flexmatch)” chapter in <i>BIOVIA Chemical Representation Guide</i>.</p>
<i>rxnflexmatch-number</i>	A number that is equal to the rxnflexmatch-number parameter used with the rxnflexmatchtimeout operator. This parameter only applies if you use rxnflexmatchtimeout.

**Return value**

The NUMBER 1 indicates that the query matched the reaction, the number 0 indicates that the query did not match the reaction. When you use rxnflexmatch in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where rxnflexmatch(rxn, query, flexmatch-parameters)=1
[operator other-conditions];
```

The rxnflexmatch operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a reaction is really a flexmatch search hit from a complex WHERE clause.

```
select rxnflexmatch(rxn, query, flexmatch-parameters)
[, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses the rxnflexmatch parameter all to search for an exact match of a specific reaction structure:

```
select rxnmdlnumber
from samplerx_reaction
where rxnflexmatch(
    rctab,
```



```

'/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
'all'
)=1;

```

**Note:** The query used in this simple example is contained in a rxnfile that is located in the examples/rxnfiles directory of the Direct installation. For more examples, see [Reaction Flexmatch Search](#).

### Comments

- To negate the results of rxnflexmatch, use the SQL operator NOT. For example:

```

select count(*)
from samplerx_reaction
where not rxnflexmatch(
  rctab,
  '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
  'all'
)=1;

```

- To find a match, each query component (reactant or product) must match a different component in the candidate structure. For example, a query structure A + A -> B will not match the candidate A -> B. To find such results, use a simpler query and specify subset in the flexmatch-parameter. For example:

```

select rxnmdlnumber
from samplerx_reaction
where rxnflexmatch(
  rctab,
  '/opt/BIOVIA/direct/examples/rxnfiles/query3.rxn',
  'subset all'
)=1;

```

- NOSTRUCT reactant or product molecules in the query reaction are legal, but will match only NOSTRUCT molecules in the target. Thus the query '-> B' with flags of 'subset match=all' where B is a NOSTRUCT molecule will match any reaction which has one or more NOSTRUCT products.
- Prefix the switches with 'SUBSET' to allow more reactants and/or products in the hit than were in the query, or to use a query with zero reactants or products. For example, when searching for a reaction that has a single stereospecific product there can be a reaction in the table which has multiple products, one for each of several stereoisomers.
- To check for errors from the rxnflexmatch operator, call the function mdlaux.errors.
- If there is no domain index on a reaction column, a rxnflexmatch search will execute more slowly than if a domain index is present and Oracle chooses to use it. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command EXPLAIN PLAN.
- rxnflexmatch supports attached data (Sgroup data). If the rxnflexmatch parameter is "all", "match=all", or the combination of specified MATCH switches includes the DAT switch, any attached data in the query will be matched with those in the target. If the query contains attached data, rxnflexmatch performs an exact match of the number or text in the attached data; it even includes any blanks preceding the data in the rxnfile. If the rxnflexmatch parameter is "ignore=dat" or the combination of specified switches excludes the DAT switch, all attached data in the query and target reactions will be ignored. For more information about attached data, see the "Attached Data"

chapter in the *BIOVIA Chemical Representation Guide*. For information on flexmatch parameters, see "Exact Search (Flexmatch)" in the *BIOVIA Chemical Representation Guide*.

- `flexmatch` does not match reactions that contain Rgroup queries, generic structures, or polymer Sgroups. Direct currently does not support these features. If a reaction contains polymer Sgroups, Direct returns the following error:
  - MDL-0427: Rxnfile containing polymer Sgroups is not supported
  - If a reaction contains Rgroup query features, Direct returns one or more of the following errors:
    - MDL-0279: Ctlb error: [RDRFIL]RDRFIL:Error reading reaction generic molfile (1)
    - MDL-0279: Ctlb error: [RDRFIL]getV2000Rxnfile:Error reading reaction generic molfile (2)
    - MDL-0041: Unable to read rxnfile, error=3002
- Clearly specify the `rxnflexmatch` switches. If the specified `rxnflexmatch-parameters` is NULL, an empty string, or a string of blank characters, "MATCH=NONE" is assumed. This is also equivalent to "IGNORE=ALL". For details about the flexmatch switches, see the "Exact Search (Flexmatch)" chapter in *BIOVIA Chemical Representation*.
- When a timeout occurs during a flexmatch search, a NEMA key match will be performed if it is possible to do so. In most cases, this will resolve the timeout.

See also:

[Reaction Flexmatch Search](#)

*BIOVIA Direct Developers Guide > Using Direct > Searching for the reactions*

*BIOVIA Direct Developers Guide > About Direct > Direct Domain Indexes*

*BIOVIA Direct Administration Guide > Creating Reaction Tables*

## rxnflexmatchtimeout

Returns a the timeout status value from an `rxnflexmatch` search.

`rxnflexmatch` will return as matches those candidates which for which the matching algorithm times out. Such candidates may or may not be actual matches. This ancillary operator gives the user information about that timeout status.

### Syntax

`rxnflexmatchtimeout(rxnflexmatch-number)`

Parameter	Description
<i>flexmatch-number</i>	A NUMBER equal to the <code>rxnflexmatch-number</code> parameter that is used with the <code>rxnflexmatch</code> operator.

### Return value

A NUMBER that indicates the status of the reaction flexmatch search. The possible values are:

Value	Description
0	The <code>rxnflexmatch</code> search did not time out.
1	The <code>rxnflexmatch</code> search timed out.
NULL	The target was not a match to the query.

**Usage**

```
select rxnflexmatchtimeout(rxnflexmatch-number)
  [, other-column-data]
from tablename
where flexmatch(rxn, query, v1-v2-options, rxnflexmatch-number)=1
  [operator other-conditions];
```

**Example**

The following example returns the timeout status while searching for a duplicate.

```
select extreg,
  rxnflexmatchtimeout(3) "Timeout"
from rxntable
where rxnflexmatch(
  rxncol,
  '/opt/BIOVIA/direct/examples/rxnfiles/query1.rxn',
  'match=all',
  3
)=1;
```

**Note:** The number 3 is used to correlate the `rxnflexmatch` operator in the WHERE clause with the `rxnflexmatchtimeout` operator in the SELECT clause. This could be any number as long as the values in the two operators match.

**Comments**

The *rxnflexmatch-number* parameters for `rxnflexmatchtimeout` and `rxnflexmatch` operators must match. If the *rxnflexmatch-number* parameters do not match, or if you use `rxnflexmatchtimeout` without using `rxnflexmatch`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

**See also**

[rxnflexmatch](#)

**rxngzip64**

Converts a BLOB reaction object into a CLOB that contains the gzip compressed and base-64 representation of a reaction.

**Syntax**

```
rxngzip64(rxn)
```

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. The value of this parameter can also be a reaction object.

**Return value**

A CLOB that contains the gzip compressed and base-64 encoded string representation of a reaction.  
`rxnchime` stores the return value in a temporary CLOB.

**Usage**

```
select rxngzip64(rxn)
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example uses the `rxngzip64` operator to return the compressed string representation of a reaction:

```
select rxngzip64(rctab)
from samplerx_reaction
where rxnmdlnumber='RXCI94070168';
```

**Comments**

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**rxnimage**

Returns a temporary BLOB containing a PNG, BMP, SVG, or EMF image of the reaction.

**Syntax**

```
rxnimage(reaction [, options])
```

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the binary chemical reaction structures. The field name is normally RCTAB. The value of this parameter can also be a reaction object.
<i>options</i>	Optional. A VARCHAR2 argument to control the type of image created, its size, and other preferences. Specify this argument as a string of comma separated options, each option takes the form keyword=value. Possible options are: <ul style="list-style-type: none"> <li>■ imagetype=png - Creates a PNG image (default)</li> <li>■ imagetype=bmp - Creates a BMP</li> <li>■ imagetype=svg - Creates a SVG</li> <li>■ imagetype=emf - Creates a EMF image</li> <li>■ width=number - Specifies a width number, typically 100 to 1000 (default is 500)</li> <li>■ height=number - Specifies a height number, typically 100 to 1000 (default is 500)</li> <li>■ ColorAtomsByType=TRUE   FALSE - Specifies whether to color the atom labels by</li> </ul>

Parameter	Description
	<p>their type. The default is TRUE.</p> <ul style="list-style-type: none"> <li>■ <b>HydrogenDisplayMode=mode</b> - Specifies how to display implicit hydrogen atoms. The default is HYDROGEN_HETERO. Valid mode values are: <ul style="list-style-type: none"> <li>■ HYDROGEN_OFF - Does not display implicit hydrogen atoms</li> <li>■ HYDROGEN_HETERO - Displays implicit hydrogens on heteroatoms.</li> <li>■ HYDROGEN_TERMINAL - Displays implicit hydrogens on terminal atoms.</li> <li>■ HYDROGEN_TERMINAL_AND_HETERO - Displays implicit hydrogens on terminal atoms and heteroatoms.</li> <li>■ HYDROGEN_ALL - Displays implicit hydrogens on all atoms.</li> </ul> </li> <li>■ <b>BackgroundColor=color</b> - Specifies the background color. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is white.</li> <li>■ <b>ForegroundColor=color</b> - Specifies the color of the shape or text. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is black.</li> <li>■ <b>ChiralityLabels=ANDtext,ABStext,ORtext,MIXEDtext</b> – Specifies the chirality label text to display for structures with stereocenters. The default is “AND Enantiomer,,OR Enantiomer,Mixed”. You must include the double quote characters and four comma-separated fields within the quotes. An empty field will not display anything for that type of chirality, for example the default text does not display anything for a structure that has only absolute stereocenters. ■</li> <li>■ <b>DisplayRS=TRUE   FALSE</b> – Specifies whether to display R and S stereocenter labels on the structure. The default is FALSE. ■</li> <li>■ <b>DisplayEZ=TRUE   FALSE</b> – Specifies whether to display E and Z double bond labels on the structure. The default is FALSE.</li> <li>■ <b>DisplayAtomMappings=TRUE   FALSE</b> - Specifies whether atom mapping numbers should be displayed. Default is TRUE.</li> <li>■ <b>ReactingCenterDisplay=mode</b> - Specifies how reacting center bonds should be displayed. Default is HASHES. Valid mode values are: <ul style="list-style-type: none"> <li>■ None - No special display for reacting center bonds.</li> <li>■ HASHES - Displays single or double lines across reacting center bonds.</li> <li>■ COLOR - Displays reacting center bonds in red.</li> <li>■ THICKNESS - Displays reacting center bonds thicker than other bonds.</li> <li>■ ALL - Same as HASHES but also displays a circle on bonds that do not change.</li> </ul> </li> <li>■ <b>PolAtomDisplayMode=POL_STYLE_BEAD   POL_STYLE_TEXT</b> - Specifies how to indicate the atom(s) that bind the structure to a polymer (atoms of type 'Pol'). Default is POL_STYLE_BEAD. Valid values are: <ul style="list-style-type: none"> <li>■ POL_STYLE_BEAD - Displays polymer atoms as shaded circles that resemble beads</li> <li>■ POL_STYLE_TEXT - Displays polymer atoms with the label text Pol.</li> </ul> </li> </ul> <p>The following shows an example that specify image options:</p> <pre>rxnimage(rxn, 'imagetype=png,width=300,height=100')</pre>

**Return value**

A BLOB that contains the binary image data. The `rxnimage` operator returns NULL if an image cannot be generated for the reaction. Use `mdlaux.errors` to see related error message.

**Usage**

```
select rxnimage(reaction [, options])
      [, other-column-data]
from tablename
where condition;
```

**Example**

The following example inserts images into a new column for all reaction in a table:

```
alter table rxntable add (imagefile blob);
update rxntable set imagefile=rxnimage(rctab);
```

**Comments**

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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 "blob":

```
if ( ((oracle.sql.BLOB)blob).isTemporary() ){
    ((oracle.sql.BLOB)blob).freeTemporary();
}
```

- Applications that use this function in a SQL SELECT statement must be aware that the temporary LOBs are only freed when the statement ends. If the statement selects many rows Oracle may run out of temporary space needed to store the LOBs. To work around this you can increase the temporary tablespace size, or you can convert the SELECT into a PL/SQL function which computes the image and then frees the BLOB immediately.

**See also**

[mdlaux.rxnimage](#)

**rxnkeys**

Returns the RSS keys that would be registered for a reaction as a printable string.

**Syntax**

```
rxnkeys(reaction, print)
```

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the reactions. <i>reaction</i> can also be a reaction object. If <i>reaction</i> is a reaction object, <code>rxnkeys</code> will use the global key definitions.
<i>print</i>	Specifies what is to be printed, and how. The specified print string should follow the format " <i>outputFormat, delimiterType</i> ". It can contain the following keywords and options, separated by whitespace. Extra characters are ignored, thus 'DEC' or 'DECIMAL' would be allowed. The following values control: What keys are output:

Parameter	Description										
	<ul style="list-style-type: none"> <li>■ CTR - Reacting center keys only.</li> <li>■ MOL - Reaction's Ored molecule keys only.</li> <li>■ ALL - Both, with molecule keys first.</li> </ul> <p>Default: CTR</p> <p>Query or registerable key setting:</p> <ul style="list-style-type: none"> <li>■ QRY - Generate keys as if reaction is query</li> </ul> <p>Default: Generates keys as if reaction is registerable</p> <p>Output format:</p> <ul style="list-style-type: none"> <li>■ BIN - Binary, i.e., '1' and '0'. Delimiter is applied between bits.</li> <li>■ HEX - Hexadecimal, i.e., 'fa03'. Lowest key (key 1) is highest bit in first word, thus 'a000' would set keys 1 and 3. Delimiter is applied between 32-bit words.</li> <li>■ DEC - Decimal key numbers. If only reacting center keys are requested, their numbers start at 1; if both then reacting center keys start immediately after molecule keys.</li> <li>■ WTS or WEI - Decimal key weights. All key positions are output, keys which are not set have a weight of zero.</li> <li>■ SET - Returns only the number of keys set, not the key values.</li> <li>■ TOT - Returns only the total number of keys, not the key values. This is independent of the mol</li> </ul> <p>Default: BIN</p> <p>Type and placement of delimiter character:</p> <ul style="list-style-type: none"> <li>■ DELIM=c - Output key values separated by 'c'.</li> <li>■ LEAD - Include a leading delimiter.</li> <li>■ TRAIL - Include a trailing delimiter.</li> </ul> <p>Default: None</p> <p>Examples:</p> <p>Note: If DECIMAL output (which is of limited utility) string exceeds 4000 characters, Oracle returns an error.</p> <table> <tr> <td>""</td><td>Same as "BIN CTR"</td></tr> <tr> <td>"DEC CTR DELIM=,"</td><td>"23,47,230"</td></tr> <tr> <td>"DEC CTR DELIM=, LEAD TRAIL"</td><td>",23,47,230,"</td></tr> <tr> <td>"BIN ALL"</td><td>"000000101011101..."</td></tr> <tr> <td>"CTR TOT"</td><td>"231" [number of rxnctr keys]</td></tr> </table>	""	Same as "BIN CTR"	"DEC CTR DELIM=,"	"23,47,230"	"DEC CTR DELIM=, LEAD TRAIL"	",23,47,230,"	"BIN ALL"	"000000101011101..."	"CTR TOT"	"231" [number of rxnctr keys]
""	Same as "BIN CTR"										
"DEC CTR DELIM=,"	"23,47,230"										
"DEC CTR DELIM=, LEAD TRAIL"	",23,47,230,"										
"BIN ALL"	"000000101011101..."										
"CTR TOT"	"231" [number of rxnctr keys]										

**Return value**

A VARCHAR2 that contains the RSS keys which would be registered for a molecule as a printable string

**Usage**

```
select rxnkeys(rctab, print)
[, other-column-data]
```

```
from tablename
where condition;
```

### Comments

- There is no package function identical to the rxnkeys operator, as it is indexed. However, there is a package function that accepts the index or table name as an argument: `mdlaux.rxnkeys`.
- Although rxnkeys is formally an indexed operator, there is no indexed implementation of it. Thus, the following search to find all reactions that have no reacting center keys will fail with an error message:

```
SELECT extreg FROM rxntable WHERE RXNKEYS(rxncol, 'CTR SET') = '0';
```

- If you wish to use rxnkeys in a WHERE clause, you must force Oracle to use the non-indexed implementation. For example:

```
SELECT extreg FROM rxntable WHERE RXNKEYS(rxncol, 'CTR SET') || 'X' =
'0X';
or
SELECT extreg FROM rxntable
WHERE TO_NUMBER(RXNKEYS(rxncol, 'CTR SET')) = 0;
```

## rxnmol

Returns a CLOB that contains a specified molecule in a reaction.

### Syntax

```
rxnmol(rxn, comptype, compidx)
```

Parameter	Description
<i>rxn</i>	A BLOB that contains a reaction object, or the name of the BLOB column that contains the reactions.
<i>comptype</i>	A NUMBER that indicates whether the component to return is a reactant or a product in the reaction. The possible values are: 1 - Returns a reactant molecule 2 - Returns a product molecule
<i>compidx</i>	NUMBER that represents the component index that identifies which reactant or product to return. If <i>comptype</i> is 1, <i>compidx</i> is a number ranging from 1 to the number of reactants. If <i>comptype</i> is 2, <i>compidx</i> is a number ranging from 1 to the number of products.

### Return value

A CLOB that contains the molfile representation of the specified reactant or product. The CLOB includes line-feed characters (0x0a) that separate the lines within the molfile. `rxnmol` returns NULL if *comptype* or *compidx* is invalid, or if there is an error.

### Usage

```
select rxnmol(rxn, comptype, compidx)
[, other-column-data]
from tablename
where condition;
```



**Example**

The following example uses `rxnmol` to return the first reactant in a specific reaction:

```
select rxnmol(rctab, 1, 1)
from samplerx_reaction
where rxnmolnumber = 'RXCI94070168';
```

The following example uses `rxnmol` and `ncomponents` to return the last product molecule in a specific reaction:

```
select rxnmol(rctab, 2, ncomponents(rctab, 2))
from samplerx_reaction
where rxnmolnumber = 'RXCI94070168';
```

**Comments**

- In PL/SQL, you can use the following constants for `comptype`:
  - `mdlaux.reactant` - Equivalent to 1, returns a reactant molecule
  - `mdlaux.product` - Equivalent to 2, returns a product molecule
- The typical usage of `rxnmol` is in a reaction table trigger that extracts the component molecules in the registered reaction, and automatically inserts them into a molecule table.
- `compidx` ranges from 1 to the number of reactants or products in the reaction. Use `ncomponents` to determine the maximum possible value for `compidx`.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[ncomponents](#)

*BIOVIA Direct Developers Guide > Using Direct > Working with Molecules in a Reaction*

**rxnmolsim**

Returns the molecule similarity value from a similarity search. The higher the value, the more similarity.

**Syntax**

`rxnmolsim(sim-number)`

Parameter	Description
<i>sim-number</i>	A NUMBER equal to the <i>sim-number</i> parameter that is used with the <code>rxnsim</code> operator.

**Return value**

A NUMBER ranging from 0 to 100 that represents the molecule similarity value.

**Usage**

```
select rxnmolsim(sim-number)
[,other-column-data]
from tablename
where rxnsim(rxn, query, simtype, sim-number)=1
[operator other-conditions];
```

**Example**

The following example returns the molecule similarity values for reactions that are at least 80% similar to the reacting centers in the query, and at least 20% similar to the molecules in the query:

```
select rxnmdlnumber,
       rxnmolsim(2) "MolSim"
from samplerx_reaction
where rxnsim(
  rctab,
  '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
  '80 20',
  2
) = 1;
```

**Note:** The query used in this simple example is contained in a rxnfile that is located in the examples/rxnfiles directory of the Direct installation.

**Comments**

The `sim-number` parameters for `rxnmolsim` and `rxnsim` operators must match. If the `sim-number` parameters do not match, or if you use `rxnctrsim` without using `rxnsim`, you get the following error:

ORA-29908: missing primary invocation for ancillary operator

**See also**

[rxnsim](#)

[rxnctrsim](#)

[Reaction Similarity Search](#)

**rxnsim**

Finds reactions that are similar to your query. You can define both the type and degree of similarity in your query. The type of similarity can be:

- Structural - physical resemblance to the substrate molecules in your query
- Transformational - similar reacting centers

**Syntax**

```
rxnsim(rxn, query, simtype, [sim-number])
```

Parameter	Description
<i>rxn</i>	The name of the BLOB column that contains the reactions. <i>rxn</i> can also be a BLOB that contains a reaction object. If a reaction object is specified, the GLOBAL key definition files will be used to generate the keys used during searching.
<i>query</i>	A molecule that uses one of the following formats:

Parameter	Description
	<ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct reaction object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>■ Reactions used in a similarity query must not include substructure query features.</li> <li>■ <i>query</i> cannot be NULL.</li> </ul>

Parameter	Description
<i>simtype</i>	<p>A VARCHAR2 string that specifies the threshold values of similarity, and follows the following format:  rcmin [rcmax][,] molmin [molmax] [sub super]  where:</p> <ul style="list-style-type: none"> <li>■ <b>rcmin</b> - A number from 0 to 100 that specifies the minimum percentage of similarity to the reacting center of the reaction query. A high value for <b>rcmin</b> finds reactions with nearly identical reacting centers. A lower value results in hits with less related transformations.</li> <li>■ <b>rcmax</b> - A number from 0 to 100 that specifies the maximum percentage of similarity to the reacting center of the reaction query. This is optional. If you do not specify <b>rcmax</b>, <b>rcmax</b> defaults to 100. If you specify <b>rcmax</b>, you must also specify <b>molmax</b> even if you want the default value.</li> <li>■ <b>molmin</b> - A number from 0 to 100 that specifies the minimum percentage of structural similarity to the molecules of the reaction query. A high value for <b>molmin</b> finds reactions with nearly identical reacting molecules. A lower value causes rxnsim to ignore nonreacting groups in a reaction.</li> <li>■ <b>molmax</b> - A number from 0 to 100 that specifies the maximum percentage of structural similarity to the molecules of the reaction query. This is optional. If you do not specify <b>molmax</b>, <b>molmax</b> defaults to 100. If you specify <b>molmax</b>, you must also specify <b>rcmax</b> even if you want the default value.</li> <li>■ <b>sub</b> or <b>SUB</b> - Indicates substructure similarity search. The reaction components that are retrieved contain more structural complexity than the query. That is, the query is a substructure. If you specify <b>sub</b>, you must provide threshold values. If you do not specify <b>sub</b> or <b>super</b>, the reaction components that are retrieved contain the same structural similarity as the query ("normal" similarity.)</li> <li>■ <b>super</b> or <b>SUPER</b> - Indicates superstructure similarity search. The reaction components that are retrieved contain less structural complexity than the query. That is, the query is a superstructure. If you specify <b>super</b>, you must provide threshold values. If you do not specify <b>sub</b> or <b>super</b>, the reaction components that are retrieved contain the same structural similarity as the query ("normal" similarity.)</li> </ul> <p><b>Note:</b> The <i>simtype</i> parameter is required. If you specify an empty string (''), <i>simtype</i> defaults to '80 20', which returns all hits that are at least 80% similar in the reacting center portion, and at least 20% similar in the molecule portion. For example '60 40 SUB'. If the third argument is blank or NULL, a value of '80 20' is used. The min and max values must range from 0 to 100, with min &lt;= max. The first form is equivalent to specifying <b>rcmax</b> and <b>molmax</b> as 100 and 100 in the second form. For example, to return all hits which are at least 80% similar in the reacting center portion, and at least 20% similar in the molecule portion, use '80 20'. To return all hits which are very dissimilar, for example 5% or less similarity in both, use '0 5 0 5'.</p>
<i>sim-number</i>	<p>A NUMBER equal to the <i>sim-number</i> parameter used with the ancillary operators <b>rxnctrsim</b> and <b>rxnmolsim</b>. This parameter only applies if you use <b>rxnctrsim</b> or <b>rxnmolsim</b>.</p>

**Return value**

The NUMBER 1 indicates that the query matched the reaction, the number 0 indicates that the query did not match the reaction. When you use `rxnsim` in a WHERE clause, always test the return value for a result of 1.

**Usage**

```
select column-data
from tablename
where rxnsim(rxn, query, simtype)=1
  [operator other-conditions];
```

```
select rxnctrsim(sim-number)
  [, rxnmolsim(sim-number)]
  [, other-column-data]
from tablename
where rxnsim(rxn, query, simtype, sim-number)=1
  [operator other-conditions];
```

The `rxnsim` operator can also be used in the SELECT clause because it evaluates to a 1 for a hit based on the parameters passed to the result row, or 0 for no hit. Generally, this type of operation can be expected to be as slow as a non-indexed search. Although it is not common usage, it can be used to determine if a reaction is really a similarity search hit from a complex WHERE clause.

```
select rxnsim(rxn, query, simtype)
  [, other-column-data]
from tablename
where condition;
```

**Example**

The following example finds reactions that are at least 60% similar to the reacting centers in the query, and at least 40% similar to the molecules in the query:

```
select rxnmdlnumber
from samplerx_reaction
where rxnsim(
  rctab,
  '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
  '60 40'
) = 1;
```

The following example finds reactions that are very dissimilar, that is, 5% or less similarity in both reacting centers and molecules. The example also returns the corresponding similarity values. Note that the parameter "1" for `rxnmolsim` and `rxnctrsim` matches the last parameter (`sim-number`) for `rxnsim`.

```
select rxnmdlnumber,
  rxnmolsim(1) "MolSim",
  rxnctrsim(1) "RxnCtrSim"
from samplerx_reaction
where rxnsim(
  rctab,
```

```

'/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
'0 5 0 5',
1
) = 1;

```

### Comments

- For the required similarity threshold values (simtype parameter), specify one of the following:

Similarity threshold Value	Description
Empty string ('') or NULL	The default value is '80 20', which returns all hits that are at least 80% similar in the reacting center portion, and at least 20% similar in the molecule portion.
'rcmin molmin'	This is equivalent to 'rcmin 100 molmin 100'. The maximum value for both reacting center and molecule similarity is 100.
'rcmin rcmax molmin molmax'	Specify both minimum and maximum values for both reacting center and molecule similarity.

- The normal similarity calculation uses the Tanimoto coefficient, except that set intersection/union set counts are replaced with the sum of the key weights for the bits in the set. The 'SUB' calculation uses the keys of the query in the denominator, thus corresponds to a "substructure similarity". That is, you will get close to 100% similarity if the query exists as a substructure within the candidate. The 'SUPER' calculation is the reverse, it uses the keys of the candidate in the denominator, thus corresponds to a "superstructure similarity". That is, you will get close to 100% if the candidate exists as a substructure within the query. See ISIS docs for more information.
- NOSTRUCT molecules in the query (or target) reaction are essentially ignored during the similarity computation. A query reaction in which all of the component molecules are NOSTRUCTS is not allowed, and will fail with an error message.
- To get the reacting center similarity values for the resulting reactions, use the rxnctrsim ancillary operator. You can use any number as the sim-number parameter, but it must match the sim-number parameter used with rxnctrsim.
- To get the molecule similarity values for the resulting reactions, use the rxnmolsim ancillary operator. You can use any number as the sim-number parameter, but it must match the sim-number parameter used with rxnmolsim.
- To check for errors from the rxnsim operator, call the function mdlaux.errors.
- rxnsim does not match reactions that contain Rgroup queries, polymer Sgroups, or attached data (Sgroup data). Direct currently does not support these features.
- If there is no domain index on a reaction column, an rxnsim search will execute more slowly than if a domain index is present and Oracle chooses to use it. To check if the domain index is part of the execution plan for the SQL statement, use the Oracle command EXPLAIN PLAN. For details about creating the reaction domain index, see *BIOVIA Direct Administration Guide* > Creating Reaction Tables.

### See also

[rxnctrsim](#)

[rxnmolsim](#)

[Reaction Similarity Search](#)

Examples of [Reaction Similarity Search](#)

*BIOVIA Direct Developers Guide > Using Direct> Searching for reactions*

*BIOVIA Direct Developers Guide > About BIOVIA Direct > Direct Domain Indexes*

## rxnsmiles

Returns a SMILES string representation of a reaction.

### Syntax

`rxnsmiles(reaction)`

Parameter	Description
<i>reaction</i>	The name of the BLOB field that contains the binary reaction.

### Return value

The `rxnsmiles` operator returns a non-canonical reaction SMILES string. If the reaction SMILES string cannot be generated, a NULL is returned. Use `mdlaux.errors` to see the related error message.

### Usage

```
select rxnsmiles(reaction) from tablename;
```

### Example

The following example shows the SMILES string for the reaction in a table:

```
SQL> select rxnsmiles(rctab) from samplerx_reaction where rxnmdlnumber =
'RXCI92014607';
```

```
RXNSMILES(RCTAB)
```

```
-----
```

```
[C:1]1(=[O:4])[CH2:3][CH2:6][CH2:7][CH2:5][CH2:2]1>>[CH:1]1([OH:4])[CH2:3]
[CH2:6][CH2:7][CH2:5][CH2:2]1
```

### Comments

- There are limitations to the generation of SMILES strings. Not all BIOVIA molecule features can be handled. If the specified reaction cannot be handled, the `rxnsmiles` operator returns NULL. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[mdlaux.rxnsmiles](#)

*BIOVIA Direct Developers Guide > Using Direct> Retrieving Related Reaction Information*

## rxnstringsegment

Returns a VARCHAR2 string that contains up to 4000 characters of a reaction.

Direct provides this operator to emulate the `rxnstringsegment` operator of the reaction cartridge prior to version 6.0. `rxnstringsegment` is equivalent to `stringsegment`. See [stringsegment](#) for details.

### Syntax

```
rxnstringsegment([tempclob-number,] rxnclob)
```

```
rxnstringsegment(tempclob-number)
```

Parameter	Description
<i>tempclob-number</i>	A NUMBER from 0 to 4 that specifies the temporary CLOB that stores the input <code>rxnclob</code> . If you do not specify <code>tempclob-number</code> in your initial call to <code>rxnstringsegment</code> , the default is 0. If the reaction contains more than 4000 characters, use <code>rxnstringsegment(tempclob-number)</code> to get the next portion of the CLOB.
<i>rxnclob</i>	A CLOB that contains the reaction to be copied to the temporary CLOB. The reaction can use either the <code>rxnfile</code> , Chime string format or Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).

### Return value

The `rxnstringsegment([tempclob-number,] rxnclob)` syntax returns VARCHAR2 data that contains the first 4000 characters of a reaction. The reaction can use either the `rxnfile` or Chime string format.

The `rxnstringsegment(tempclob-number)` syntax returns VARCHAR2 data that contains the next 4000 characters.

Both syntax will return NULL if there are no more characters left in the reaction.

### Usage

```
select rxnstringsegment(tempclob-number, rxnclob)
       [, other-column-data]
from   tablename
where  condition;
```

```
select rxnstringsegment(rxnclob)
       [, other-column-data]
from   tablename
where  condition;
```

```
select rxnstringsegment(tempclob-number)
from   dual;
```

### Example

The following example uses `rxnstringsegment` to get the first 4000 characters of the Chime representation of a reaction:



```
select rxnstringsegment(1, rxnchime(rctab))
from samplerx_reaction
where rxnmdlnumber='RXCI94070168';
```

The next example uses `rxnstringsegment` to return the next portion of the same Chime string from the preceding example. The Chime string is less than 4000 characters, so this statement returns a NULL:

```
select rxnstringsegment(1)
from dual;
```

The following is another example that uses `rxnstringsegment` to get the first 4000 characters of the rxnfile representation of a reaction. The default number for the temporary CLOB is 0 (zero):

```
select rxnstringsegment(rxnfile(rctab))
from samplerx_reaction
where rxnmdlnumber='RXCI92071538';
```

The next example uses `rxnstringsegment` to return the next portion of the same rxnfile. Note that the number for the temporary CLOB is 0 (zero):

```
select rxnstringsegment(0)
from dual;
```

### Comments

- Use `rxnstringsegment` if the application that uses Direct does not support CLOBs. If the application that uses the reaction cartridge supports CLOBs, use the operator `rxnchime` or `rxnfile` instead of `rxnstringsegment`.
- Repeatedly call `rxnstringsegment` to retrieve reactions that exceed 4000 characters. The initial call must use the syntax:

```
rxnstringsegment([tempclob-number], rxnclob)
```

The subsequent calls must use the following syntax. Repeatedly call `rxnstringsegment` using this syntax until `rxnstringsegment` returns NULL or a string whose length is less than 4000. Use the same `tempclob-number` that you used in the initial call.

```
rxnstringsegment(tempclob-number)
```

- For the initial call to `rxnstringsegment`, the `tempclob-number` parameter can be an arbitrary number from 0 to 4. For subsequent calls to `rxnstringsegment`, the `tempclob-number` parameter must be the same as what you used in the initial call. If you did not specify a `tempclob-number` parameter in the initial call, the `tempclob-number` parameter in the subsequent call must be 0 (zero).
- Although it is possible to write a query that contains multiple calls to `rxnstringsegment`, we do not recommend it. Oracle does not necessarily call the operators in the order they appear in the SELECT statement. For example, do not call `rxnstringsegment(tempclob-number, clob)` and `rxnstringsegment(tempclob-number)` in a single SELECT statement, where `tempclob-number` is the same for both calls. The syntax `rxnstringsegment(tempclob-number, clob)` saves the data in a temporary CLOB, and the syntax `rxnstringsegment(tempclob-number)` reads the contents of the temporary CLOB. If Oracle calls `rxnstringsegment(tempclob-number)` first, you will get the contents of an old temporary CLOB that might have been created in a separate operation. It is more reliable to issue separate SELECT statements for each segment, as shown in the following example. Note that this example uses the default number 0 (zero) for the temporary CLOB:

```

DECLARE
  strseg VARCHAR2(4000);
  bigstr VARCHAR2(32000);
  numval NUMBER;
BEGIN
  SELECT RXNSTRINGSEGMENT(RXNFILE(rctab)) INTO bigstr
  FROM samplerx_reaction WHERE rxnmdlnumber='RXCI92071538';
  LOOP
    SELECT RXNSTRINGSEGMENT(0) INTO strseg FROM dual;
    IF strseg IS NULL THEN EXIT; END IF;
    bigstr := bigstr || strseg;
  END LOOP;
END;

```

- Use the same number to identify the same temporary CLOB in one Oracle session. The rxnstringsegment operator shares the package-level, session-duration temporary CLOBs with the tempclob and writetempclob operators. Make sure that you use the correct, corresponding numbers in order to reference multiple temporary CLOBs within one Oracle session. The following example shows how corresponding numbers are used to reference multiple temporary CLOBs:

```

DECLARE
  str1 varchar2(4000);
  str2 varchar2(4000);
  bigstr1 varchar2(32000);
  bigstr2 varchar2(32000);
BEGIN
  --Fetch the first segments of the:
  --unhighlighted Chime string (tempCLOB#1)
  --highlighted Chime string (tempCLOB#2)
  select rxnstringsegment(1, rxnchime(rctab)),
         rxnstringsegment(2, rsshighlight(99))
  into bigstr1, bigstr2
  from samplerx_reaction
  where rss(rctab,
    '/opt/BIOVIA/direct/examples/rxnfiles/query.rxn',
    99
  )=1;
  --Fetch the rest of the Chime strings:
  --unhighlighted Chime string (tempCLOB#1)
  --highlighted Chime string (tempCLOB#2)
  loop
    select rxnstringsegment(1), rxnstringsegment(2)
    into str1, str2
    from dual;

    if str1 is NULL and str2 is NULL then exit; end if;
    bigstr1 := bigstr1 || str1;
    bigstr2 := bigstr2 || str2;
  end loop;
END;

```

See also

[rxnchime](#)

[rxnfile](#)[tempclob](#)[writetempclob](#)[Examples of Fetching Reactions Using the Rxnfile Format](#)[Examples of Fetching Reactions Using the Chime Format](#)*BIOVIA Direct Developers Guide > Using Direct > Fetching Reactions as String Segments*

## Reaction-Specific Functions

In some cases, Direct offers both a function and an operator with the same name. The function and operator behave identically to each other. For example, the `readfile` operator and the `mdlaux.readfile` function have the same functionality. In these cases, the description of the operator appears under Reaction-Specific Operators. This section lists the name of the function and then references the description in Reaction-Specific Operators.

If both a function and an operator are available, use the function name instead of the operator name in situations where the operator is not allowed. For example, you must use the package function name in a PL/SQL assignment statement, because PL/SQL assignment statements do not accept operators.

<a href="#">mdlaux.automap</a>	193
<a href="#">mdlaux.hasnostructs</a>	196
<a href="#">mdlaux.rinchi</a>	196
<a href="#">mdlaux.rinchiauxinfo</a>	198
<a href="#">mdlaux.rinchikey</a>	200
<a href="#">mdlaux.rinchitorxnfile</a>	201
<a href="#">mdlaux.rxnimage</a>	204
<a href="#">mdlaux.rxnkeys</a>	206
<a href="#">mdlaux.rxnsmiles</a>	209
<a href="#">mdlaux.smilestorxnfile</a>	210

### mdlaux.automap

Automatically assigns atom-to-atom mapping in a reaction (“automaps a reaction”), and returns:

- A CLOB that contains the automapped reaction. The automapped reaction contains a unique mapping number for each corresponding atom in the reactants and products in the reaction.
- The number of changes performed in the automapped reaction
- The status of the automap operation

`mdlaux.automap` also provides the option to clear the maps from a reaction.

#### Syntax

```
mdlaux.automap(rxn, mode, mappedrxn, changes)
```

Parameter	Description
<i>rxn</i>	<p>The reaction to be automapped.</p> <p><i>rxn</i> can use one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Full path and file name of a rxnfile (VARCHAR2). The rxnfile must be located on the server where Direct is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ InChI string</li> <li>■ SMILES string</li> <li>■ Reaction column name (BLOB)</li> <li>■ Reaction object created by a previous operation (BLOB)</li> <li>■ Reaction SMILES string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> </ul>
<i>mode</i>	<p>A VARCHAR2 string that specifies how the reaction will be automapped. <i>mode</i> is not case-sensitive.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>■ <b>RegenAuto</b> - Computes the atom- maps and bond change marks for the reaction, without further input. RegenAuto disregards any existing maps or bond change marks.</li> <li>■ <b>RegenAlter</b> - Computes the atom-atom maps and bond change marks for the reaction, using any existing bond marks to guide the mapping. RegenAlter assumes the existing marks might be wrong and can be altered. This mode is used by the Regenerate AAMappings command in REXEC.</li> <li>■ <b>Clear</b> - Removes all existing atom-atom maps and bond marks from the reaction. Clear does not perform any mapping.</li> <li>■ <b>Default</b> - Computes the atom-atom maps and bond change marks for the reaction, using the existing bond marks. Default assumes the existing marks are absolutely correct. This mode is used by ISIS/Base when mapping reactions.</li> <li>■ <b>RegenKeep</b> - equivalent to Default.</li> <li>■ <b>NULL</b> or empty string - equivalent to Default.</li> </ul>
<i>mappedrxn</i>	<p>A temporary CLOB that has been allocated, is not NULL, and is empty.</p> <p>If the automap operation succeeds, <i>mappedrxn</i> contains the resulting mapped reaction. The mapped reaction is a <i>rxnfile</i> string, with each line terminated by a line-feed character LF (0x0a)</p>
<i>changes</i>	<p>A <b>BINARY_INTEGER</b> variable that will contain the number of changes performed, if the automap operation succeeds.</p>

Parameter	Description
	<p>The possible values are:</p> <ul style="list-style-type: none"> <li>■ 0 - No change. The <code>automap</code> operation did not change atom-atom maps, bond marks or inversion/retention flags. The <code>mdl aux . automap</code> function returns <code>mappedrxn</code> as the input reaction.</li> <li>■ &gt; 0 - The number of atom-atom maps and bond marks which the user had specified, and which were changed to some other value by <code>mdl aux . automap</code>. The user should examine the mapped reaction in <code>mappedrxn</code>.</li> <li>■ -1 - No change in the existing atom-atom maps and bond marks, but new atom-atom maps, bond marks, or inversion/retention flags were added. If the <code>automap mode</code> (mode) for the <code>mdl aux . automap</code> function is <code>RegenA1ter</code>, this could also mean that the original atom-atom maps were modified.</li> </ul> <p>The <i>changes</i> parameter is not used when <i>mode</i> is <code>C1ear</code>. In <code>C1ear</code> mode, the <i>changes</i> parameter is always returned as -1.</p>

### Return Value

A `BINARY_INTEGER` that indicates the status of the `automap` operation. The possible values are:

Value	Description
0	The <code>mdl aux . automap</code> function failed. <code>mdl aux . automap</code> returns an empty CLOB in <code>mappedrxn</code> .
1	The <code>mdl aux . automap</code> function succeeded. <code>rdcaux . automap</code> returns the automapped reaction in <code>mappedrxn</code> .
2	The specified mode parameter is invalid. <code>mdl aux . automap</code> returns the input (unchanged) reaction in <code>mappedrxn</code> .
3	The <code>mdl aux . automap</code> function attempted to map the reaction, but failed. <code>mdl aux . automap</code> returns the input (unchanged) reaction in <code>mappedrxn</code> .
>=4	The <code>mdl aux . automap</code> function succeeded, but the mapping might be incorrect and must be examined by the user. <code>mdl aux . automap</code> returns the automapped reaction in <code>mappedrxn</code> . The actual value has no significance. A typical return value is 39.

### Usage

This function must be called from PL/SQL.

```
status := mdl aux . automap(
    rxn, mode,
    mappedrxn,
    changes);
```

### Example

The following example uses the `rdcaux . automap` function to automap a reaction in a file:

```
DECLARE
    mappedrxn    CLOB;
    status       BINARY_INTEGER;
```

```
changes          BINARY_INTEGER;
BEGIN
  dbms_lob.createtemporary(
    mappedrxn,
    FALSE,
    dbms_lob.call);
  status := mdlaux.automap (
    '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn',
    NULL,
    mappedrxn,
    changes);
  if (status = 0) then
    dbms_lob.freetemporary(mappedrxn);
    mappedrxn := NULL;
  end if;
  -- Add statements here to use mappedrxn, status, changes
END;
```

### Comments

- Use `mdlaux.automap` only in PL/SQL. Because the `mdlaux.automap` function requires variable input parameters for its return values, you can only use it in PL/SQL. For applications that do not use PL/SQL, use the `rxnautomap`, `rxnautomapchange`, and `rxnautomapstatus` operators. The `mdlaux.automap` function returns three values: the mapped reaction, the mapping status, and the number of mapping changes. `mdlaux.automap` is more efficient because it eliminates additional calls to the server.
- Check the resulting automapped reaction. If the mapping changed, or if the mapping status is greater than 3, the user should inspect the resulting mapped structure to verify its correctness.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[rxnautomap](#)

[rxnautomapchange](#)

[rxnautomapstatus](#)

### mdlaux.hasnostructs

This package function is equivalent to `hasnostructs`. For more information, see [hasnostructs](#).

### mdlaux.rinchi

Returns an IUPAC standard International Chemical Identifier (standard "RInChI") string for the specified reaction.

#### Syntax

`mdlaux.rinchi(reaction, options)`

Parameter	Description
<i>reaction</i>	<p>A reaction that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where Direct is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Direct reaction object (BLOB)</li> <li>■ Gzip compressed and base-64 encoded rxnfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options.          If no additional options are provided in the call to RINCHI, the standard RInChI string is generated.          If any of the following options are specified, a non-standard RInChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RInChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A temporary CLOB that contains the RInChI string. The output string length will exceed 4000 characters for very large reactions. If the RInChI string cannot be generated, the `mdlAux.rinchi` functions returns NULL. Use `mdlAux.errors` to see the related error message.

**Usage**

```
select mdlAux.rinchi(reaction, options) from dual;
```

**Example**

The following example shows the RInChI string for a reaction, using the default option:

```
select mdlAux.rinchi('/work/rxns/test.rxn') from dual;
```

```
MDLAUX.RINCHI('/WORK/RXNS/TEST.RXN')
```

```
-----
RInChI=1.00.1S/C2H7ClSi/c1-4(2)3/h4H,1-2H3!C6H10O3/c1-3-9-6(8)4-5(2)7/h3-4H2,1-2H3<>C8H17ClO3Si/c1-5-11-8(10)6-7(2)12-13(3,4)9/h7H,5-6H2,1-4H3/t7-/m1/s1/d+
```

### Comments

- There are limitations to the generation of RInChI strings. Not all BIOVIA reaction features can be handled. The `mdlaux.rinchi` function returns NULL if the specified reaction cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

### See also

[mdlaux.rinchikey](#)

[rinchi](#)

## mdlaux.rinchiauxinfo

Returns the auxiliary information (AuxInfo) that is computed along with the IUPAC International Chemical Identifier (InChI) string for a molecule.

### Syntax

```
mdlaux.rinchiauxinfo(any-reaction [, options])
```

Parameter	Description
<i>reaction</i>	<p>A reaction that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where Direct is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Direct reaction object (BLOB)</li> <li>■ Gzip compressed and base-64 encoded rxnfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options. If no additional options are provided in the call to RINCHIAUXINFO, the standard RInChI AuxInfo string is generated. If any of the following options are specified, a non-standard RInChI AuxInfo string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> </ul>



Parameter	Description
	<ul style="list-style-type: none"> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RinChI AuxInfo string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

### Return value

A temporary CLOB that contains the RinChI AuxInfo string. The output string length will exceed 4000 characters for very large reactions. If the RinChI AuxInfo string cannot be generated, the `mdlaux.rinchiauxinfo` functions returns NULL. Use `mdlaux.errors` to see the related error message.

### Usage

```
select mdlaux.rinchiauxinfo(reaction, options) from dual;
```

### Example

The following example shows the RinChI AuxInfo string for a reaction, using the default option:

```
select mdlaux.rinchiauxinfo('/work/rxns/test.rxn') from dual;
```

```
MDLAUX.RINCHIAUXINFO('/WORK/RXNS/TEST.RXN')
```

```
-----
RAuxInfo=1.00.1/0/N:2,3,4,1/E:
(1,2)/rA:4nSiCCCl/rB:s1;s1;s1;/rC:.7895,.1596,0;.0183,1.4909,0;-.6599,-.369,
0;1.0538,-
1.3563,0;!0/N:9,6,8,1,3,2,7,5,4/rA:9nCCCCOOCOCC/rB:s1;s1;s2;d2;s3;d3;s4;s8;/r
C:-1.2727,-.933,0;.0655,-.1637,0;-
2.6067,-.1637,0;1.3995,-.933,0;.0655,1.375,0;-3.9367,-.933,0;-
2.6067,1.375,0;2.7336,-.1637,0;4.0677,-.933,0;<>0/N:13,5,10,11,9,1,2,3,12,7,
6,4,8/E:
(3,4)/it:im/rA:13CCCCOCOOSiCCCCl/rB:s1;s1;P2;s2;s3;d3;s4;s6;s8;s8;s8;s9;/rC
:.2778,.3641,0;-1.1928,1.2072,0;.2778,-1.3365,0;-1.1928,2.9126,0;-
2.6635,.3641,0;1.7485,-2.1892,0;-1.1928,-2.1892,0;.1437,3.8132,0;1.8635,-
3.7988,0;.6467,4.7617,0;.5461,2.8168,0;- .6276,4.5605,0;2.7881,-4.3354,0;
```

**Comments**

- There are limitations to the generation of RInChI AuxInfo strings. Not all BIOVIA reaction features can be handled. The `mdlaux.rinchi_auxinfo` function returns NULL if the specified reaction cannot be handled. Use `mdlaux.errors` to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();
}
```

**See also**

[mdlaux.rinchi](#)  
[rinchi\\_auxinfo](#)

**mdlaux.rinchikey**

Returns an IUPAC International standard Chemical Identifier (standard "RInChI") key for the specified reaction. The RInChI key is a 27-character hashed form of the RInChI string. The `mdlaux.rinchikey` function generates the key by first generating the RInChI string, and then calling an RInChI library function to convert the string into the 27-character key.

**Syntax**

`mdlaux.rinchikey(reaction, options)`

Parameter	Description
<i>reaction</i>	<p>A reaction that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where Direct is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a) , or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct Reaction object (BLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded rxnfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>(Optional) Specifies a complete set of RInChI library options.</p> <p>If no additional options are provided in the call to RINCHI, the standard RInChI string is generated.</p> <p>If any of the following options are specified, a non-standard RInChI string is generated.</p> <ul style="list-style-type: none"> <li>■ NEWPSOFF - Both ends of wedge point to stereocenters (Narrow End of Wedge Points to Stereocenter OFF)</li> <li>■ FixedH - Include Fixed H layer (default is 'not')</li> <li>■ RecMet - Include reconnected metals results (default is 'not')</li> <li>■ SAbs - Absolute stereo (default)</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ SRel - Relative stereo</li> <li>■ SRac - Racemic stereo</li> <li>■ SUCF - Use Chiral Flag where On means Absolute stereo and Off means Relative</li> <li>■ SNon - Exclude stereo</li> <li>■ SUU - Include omitted unknown/undefined stereo</li> <li>■ SLUUD - Stereolabels for unknown ('u') and undefined ('?') are different (default for both is '?')</li> <li>■ KET - Account for keto/enol tautomerization (default is off)</li> <li>■ 15T - Account for 1-5 tautomerization (default is off)</li> <li>■ SaveOpt - Save non-default options in the RInChI string (shows up as a three-character suffix, a backslash followed by two letters)</li> </ul>

**Return value**

A VARCHAR2 that contains the 27-character RInChI key. The `mdlaux.rinchikey` function returns NULL if the RInChI string cannot be generated. Use `mdlaux.errors` to see the related error message.

**Usage**

```
select mdlaux.rinchikey(reaction, options) from dual;
```

**Example**

The following example shows the RInChI key for a reaction, using the default option:

```
select mdlaux.rinchikey('/work/rxns/muse2.rxn') from dual;
```

```
MDLAUX.RINCHIKEY('/WORK/RXNS/MUSE2.RXN')
```

```
-----
UHOVQNZZJSORNB-UHFFFAOYSA-N
```

**Comments**

There are limitations to the generation of RInChI strings. Not all BIOVIA reaction features can be handled. If the specified reaction cannot be handled, the `mdlaux.rinchikey` function returns NULL. Use `mdlaux.errors` to see the related error message.

**See also**

[mdlaux.rinchi](#)  
[rinchikey](#)

**mdlaux.rinchitorxnfile**

Returns the rxnfile string representation of a reaction InChI (IUPAC International Chemical Identifier) string and optional InChI AuxInfo string.

**Syntax**

```
mdlaux.rinchitorxnfile(inchi [, auxinfo])
```

Parameter	Description
<i>inchi</i>	A VARCHAR2 or CLOB containing a reaction InChI string.
<i>auxinfo</i>	An optional VARCHAR2 or CLOB containing a reaction InChI AuxInfo string. Adding the optional AuxInfo argument will provide more information to the conversion including the original atom coordinates, the output reaction will generally be more similar to the reaction from which InChI was calculated.

**Return value**

A temporary CLOB that contains the converted rxnfile string. If the InChI input cannot be converted, the `mdlaux.rinchitorxnfile` function returns NULL. Use `mdlaux.errors` to see related error message.

**Usage**

```
select mdlaux.rinchitorxnfile(inchi) from dual;
select mdlaux.rinchitorxnfile(inchi, auxinfo) from dual;
```

**Examples**

The following example shows the converted rxnfile from an InChI string for a simple reaction:

```
select mdlaux.rinchitorxnfile('RINCHI=1.00.1S/C2H6O/c1-2-3/h3H,2H2,1H3<>C2H7N/c1-2-3/h2-3H2,1H3/d+') from dual;
```

```
MDLAUX.RINCHITORXNFILE('RINCHI=1.00.1S/C2H6O/C1-2-3/H3H,2H2,1H3<>C2H7N/C1-2-3
```

```
-----
-
$RXN

      SciTegic      0622201530

      1  1
$MOL
Reactant1
      SciTegic06222015302D

      3  2  0  0  0  0          999 v2000
          0.0000      0.0000      0.0000 C    0  0
          -0.8660      0.5000      0.0000 C    0  0
          -1.7321      0.0000      0.0000 O    0  0
      1  2  1  0
      2  3  1  0
M  END
$MOL
Product1
      SciTegic06222015302D

      3  2  0  0  0  0          999 v2000
0.0000      0.0000      0.0000 C    0  0
          -0.8660      0.5000      0.0000 C    0  0
          -1.7321      0.0000      0.0000 N    0  0
      1  2  1  0
```

```

      2  3  1  0
M  END

```

This example includes the reaction AuxInfo argument. Note how the coordinates in the computed rxnfile are now taken from the AuxInfo string:

```

select mdlaux.rinchitorxnfile('RINCHI=1.00.1S/C2H6O/c1-2-
3/h3H,2H2,1H3;<>C2H7N/c1-2-3/h2-
3H2,1H3/d+', 'RAuxInfo=1.00.1/0/N:1,2,3/rA:3nCCO/rB:s1;s2;/rC:2,-
5.7188,0;3.0232,-5.128,0;4.0463,-
5.7188,0;<>0/N:1,2,3/rA:3nCCN/rB:s1;s2;/rC:9.0938,-.7188,0;10.1875,-
5.0313,0;11.3061,-5.6771,0;') from dual;

```

```

MDLAUX.RINCHITORXNFILE('RINCHI=1.00.1S/C2H6O/C1-2-3/H3H,2H2,1H3<>C2H7N/C1-2-
3

```

```

-----
-
$RXN

```

```

SciTegic      0622201608

```

```

      1  1
$MOL
Reactant1
  SciTegic06222016082D

```

```

      3  2  0  0  0  0          999 v2000
      2.0000   -5.7188   0.0000 C    0  0
      3.0232   -5.1280   0.0000 C    0  0
      4.0463   -5.7188   0.0000 O    0  0
      1  2  1  0
      2  3  1  0

```

```

M  END
$MOL
Product1
  SciTegic06222016082D

```

```

      3  2  0  0  0  0          999 v2000
      9.0938   -0.7188   0.0000 C    0  0
     10.1875   -5.0313   0.0000 C    0  0
     11.3061   -5.6771   0.0000 N    0  0
      1  2  1  0
      2  3  1  0

```

```

M  END

```

### Comments

InChI has limitations, for example it does not encode information about whether stereochemistry is relative or absolute nor does it always differentiate between two tautomeric forms of a molecule. Nitro groups in an InChI string are always converted to the uncharged hypervalent nitrogen form in the molfile. You will not always get the same molecule that you started with when converting from molfile to InChI and then from that InChI back to molfile.

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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() )
Unknown macro: { ((oracle.sql.CLOB) clob).freeTemporary(); }
```

See also

[rinchi](#)

[rinchiauxinfo](#)

## mdlaux.rxnimage

Returns a temporary BLOB containing a PNG, BMP, SVG, or EMF image of the reaction.

### Syntax

```
mdlaux.rxnimage(reaction [, options])
```

Parameter	Description
<i>reaction</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>
<i>options</i>	<p>Optional. A VARCHAR2 argument to control the type of image created and its size.. Specify this argument as a string of comma separated options, each option takes the form keyword=value. Possible options are:</p> <ul style="list-style-type: none"> <li>■ imagetype=png - Creates a PNG image (default)</li> <li>■ imagetype=bmp - Creates a BMP image</li> <li>■ imagetype=svg - Creates a SVG image</li> <li>■ imagetype=emf - Creates a EMF image</li> <li>■ width=<i>number</i> - Specifies a width number, typically 100 to 1000 (default is 500)</li> <li>■ height=<i>number</i> - Specifies a height number, typically 100 to 1000 (default is 500)</li> <li>■ ColorAtomsByType=TRUE   FALSE - Specifies whether to color the atom labels by</li> </ul>

Parameter	Description
	<p>their type. The default is TRUE.</p> <ul style="list-style-type: none"> <li>■ <code>HydrogenDisplayMode=OFF   HETERO   TERMINAL   TERMINAL_AND_HETERO   ALL</code> - Specifies how to display implicit hydrogen atoms. The default is HETERO. Valid values are: <ul style="list-style-type: none"> <li>■ OFF - Does not display implicit hydrogen atoms</li> <li>■ HETERO - Displays implicit hydrogens on heteroatoms.</li> <li>■ TERMINAL - Displays implicit hydrogens on terminal atoms.</li> <li>■ TERMINAL_AND_HETERO - Displays implicit hydrogens on terminal atoms and heteroatoms.</li> <li>■ ALL - Displays implicit hydrogens on all atoms.</li> </ul> </li> <li>■ <code>BackgroundColor=color</code> - Specifies the background color. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is white.</li> <li>■ <code>ForegroundColor=color</code> - Specifies the color of the shape or text. Use either the name of the color (red, green, others) or the hexadecimal RGB value (FF0000, 00FF00, others). The default is black.</li> <li>■ <code>ChiralityLabels=ANDtext,ABStext,ORtext,MIXEDtext</code> - Specifies the chirality label text to display for structures with stereocenters. The default is "AND Enantiomer,,OR Enantiomer,Mixed". You must include the double quote characters and four comma-separated fields within the quotes. An empty field will not display anything for that type of chirality, for example the default text does not display anything for a structure that has only absolute stereocenters. ■</li> <li>■ <code>DisplayRS=TRUE   FALSE</code> - Specifies whether to display R and S stereocenter labels on the structure. The default is FALSE. ■</li> <li>■ <code>DisplayEZ=TRUE   FALSE</code> - Specifies whether to display E and Z double bond labels on the structure. The default is FALSE.</li> <li>■ <code>PolAtomDisplayMode=POL_STYLE_BEAD   POL_STYLE_TEXT</code> - Specifies how to indicate the atom(s) that bind the structure to a polymer (atoms of type 'Pol'). Default is POL_STYLE_BEAD. Valid values are: <ul style="list-style-type: none"> <li>■ POL_STYLE_BEAD - Displays polymer atoms as shaded circles that resemble beads</li> <li>■ POL_STYLE_TEXT - Displays polymer atoms with the label text Pol. The following shows an example that specify image options: <code>mdl aux.rxn image(rxn, 'imagetype=png,width=300,height=100,polatomdisplaymode=pol_style_text')</code></li> </ul> </li> <li>■ <code>DisplayAtomMappings=TRUE   FALSE</code> - Specifies whether atom mapping numbers should be displayed. Default is TRUE.</li> <li>■ <code>ReactingCenterDisplay=mode</code> - Specifies how reacting center bonds should be displayed. Default is HASHES. Valid mode values are: <ul style="list-style-type: none"> <li>■ None - No special display for reacting center bonds.</li> <li>■ HASHES - Displays single or double lines across reacting center bonds.</li> <li>■ COLOR - Displays reacting center bonds in red.</li> </ul> </li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>■ THICKNESS - Displays reacting center bonds thicker than other bonds.</li> <li>■ ALL - Same as HASHES but also displays a circle on bonds that do not change.</li> </ul>

**Return value**

A BLOB that contains the binary image data. The `mdlaux.rxnimage` function returns NULL if an image cannot be generated for the molecule. Use `mdlaux.errors` to see related error message.

**Usage**

```
select mdlaux.rxnimage(reaction [, options]) from dual;
```

**Comments**

- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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 "blob":

```
if ( ((oracle.sql.BLOB)blob).isTemporary() ){
    ((oracle.sql.BLOB)blob).freeTemporary();
}
```

- Applications that use this function in a SQL SELECT statement must be aware that the temporary LOBs are only freed when the statement ends. If the statement selects many rows Oracle may run out of temporary space needed to store the LOBs. To work around this you can increase the temporary tablespace size, or you can convert the SELECT into a PL/SQL function which computes the image and then frees the BLOB immediately.

**See also**

[rxnimage](#)

**mdlaux.rxnkeys**

Returns the RSS keys for a reaction as a printable string.

**Syntax**

```
mdlaux.rxnkeys(rxnIndexOrTable, reaction, print)
```

Parameter	Description
<i>rxnIndexOrTable</i>	The name of a reaction index, or the name of a table which contains exactly one reaction index. (The schema may be included, e.g. ' <i>schema.table</i> '.) The value may be NULL, in which case the global environment is used. Use the first argument to control what molecule 2D key definition file and reacting center key definition files are used to compute the keys for the reaction. A NULL value will cause the global ptable to be used.
<i>reaction</i>	A molecule that uses one of the following formats: <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> </ul>



Parameter	Description
	<ul style="list-style-type: none"><li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li><li>■ Direct molecule object (BLOB)</li><li>■ IUPAC name (VARCHAR2 or CLOB)</li><li>■ HELM string (VARCHAR2 or CLOB)</li><li>■ XHELM string (VARCHAR2 or CLOB)</li><li>■ Chime string (VARCHAR2 or CLOB)</li><li>■ SMILES string (VARCHAR2 or CLOB)</li><li>■ InChI string (VARCHAR2 or CLOB)</li><li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li></ul>

Parameter	Description										
<i>print</i>	<p>Specifies what is to be printed, and how. The specified print string should follow the format "<i>outputFormat, delimiterType</i>". It can contain the following keywords and options, separated by whitespace. Extra characters are ignored, thus 'DEC' or 'DECIMAL' would be allowed.</p> <p>Control what keys are output:</p> <ul style="list-style-type: none"> <li>■ CTR Reacting center keys only.</li> <li>■ MOL Reaction's Ored molecule keys only.</li> <li>■ ALL Both, with molecule keys first.</li> </ul> <p>Default: CTR</p> <p>Control query or registerable key setting:</p> <ul style="list-style-type: none"> <li>■ QRY Generate keys as if reaction is query</li> </ul> <p>Default: Generates keys as if reaction is registerable</p> <p>Control the output format:</p> <ul style="list-style-type: none"> <li>■ BIN - Binary, i.e. '1' and '0'. Delimiter is applied between bits.</li> <li>■ HEX - Hexadecimal, i.e. 'fa03'. Lowest key (key 1) is highest bit in first word, thus 'a000' would set keys 1 and 3. Delimiter is applied between 32-bit words.</li> <li>■ DEC - Decimal key numbers. If only reacting center keys are requested, their numbers start at 1; if both then reacting center center keys start immediately after molecule keys.</li> <li>■ WTS or WEI - Decimal key weights. All key positions are output, keys which are not set have a weight of zero.</li> <li>■ SET - Returns only the number of keys set, not the key values.</li> <li>■ TOT - Returns only the total number of keys, not the key values. This is independent of the rxn.</li> </ul> <p>Default: BIN</p> <p>Control type and placement of delimiter character:</p> <ul style="list-style-type: none"> <li>■ DELIM=c - Output key values separated by 'c'.</li> <li>■ LEAD - Include a leading delimiter.</li> <li>■ TRAIL - Include a trailing delimiter. Default: None</li> </ul> <p>Examples:</p> <table> <tr> <td>""</td><td>Same as "BIN CTR"</td></tr> <tr> <td>"DEC CTR DELIM=,"</td><td>"23,47,230"</td></tr> <tr> <td>"DEC CTR DELIM=, LEAD TRAIL"</td><td>“,23,47,230,”</td></tr> <tr> <td>"BIN ALL"</td><td>“000000101011101...”</td></tr> <tr> <td>"CTR TOT"</td><td>“231” [number of rxnctr keys]</td></tr> </table>	""	Same as "BIN CTR"	"DEC CTR DELIM=,"	"23,47,230"	"DEC CTR DELIM=, LEAD TRAIL"	“,23,47,230,”	"BIN ALL"	“000000101011101...”	"CTR TOT"	“231” [number of rxnctr keys]
""	Same as "BIN CTR"										
"DEC CTR DELIM=,"	"23,47,230"										
"DEC CTR DELIM=, LEAD TRAIL"	“,23,47,230,”										
"BIN ALL"	“000000101011101...”										
"CTR TOT"	“231” [number of rxnctr keys]										

**Return value**

A VARCHAR2 that contains the RSS keys which would be registered for a molecule as a printable string

### Usage

The typical usage for the `mdlaux.rxnkeys` function is to get the printable keys for registration (using the same key definition file which is associated with the reaction domain index on the table into which the reaction is being registered). For example:

```
INSERT INTO rxntable (extreg, ctab, rxnkeys) VALUES
('ABC-123', RXN('/test/rxn123.rxn'),
MDLAUX.RXNKEYS('rxntable', '/test/rxn123.rxn',
'ctr dec delim=', 'lead trail'));
```

### Comments

DECIMAL output is of limited utility, since if the string exceeds 4000 characters Oracle will return an error.

### See also

[rxnkeys](#)

## mdlaux.rxnsmiles

Returns a SMILES string representation of a reaction.

### Syntax

`mdlaux.rxnsmiles(reaction)`

Parameter	Description
<i>reaction</i>	<p>A molecule that uses one of the following formats:</p> <ul style="list-style-type: none"> <li>■ Filepath of a rxnfile (VARCHAR2). The rxnfile must be located on the server where BIOVIADirect is installed.</li> <li>■ Rxnfile string (VARCHAR2 or CLOB). Each line in this string must be terminated either by the line-feed character LF (0x0a), or by the carriage-return followed by the line-feed characters CR+LF (0x0d + 0x0a).</li> <li>■ Direct molecule object (BLOB)</li> <li>■ IUPAC name (VARCHAR2 or CLOB)</li> <li>■ HELM string (VARCHAR2 or CLOB)</li> <li>■ XHELM string (VARCHAR2 or CLOB)</li> <li>■ Chime string (VARCHAR2 or CLOB)</li> <li>■ SMILES string (VARCHAR2 or CLOB)</li> <li>■ InChI string (VARCHAR2 or CLOB)</li> <li>■ Gzip compressed and base-64 encoded molfile string (VARCHAR2 or CLOB).</li> </ul>

### Return value

Takes any type of reaction as an argument (filename, file CLOB, binary CTAB BLOB) and returns a non-canonical reaction SMILES string. If the reaction SMILES string cannot be generated, the `mdlaux.rxnsmiles` function returns NULL. Use `mdlaux.errors` to see the related error message.

### Usage

```
select mdlaux.rxnsmiles(reaction) from dual;
```

**Example**

The following example shows the SMILES string for a molecule:

```
SQL> select mdlaux.rxnsmiles('f:/work/rxn.rxn') from dual;

MDLAUX.RXNSMILES('F:/WORK/RXN.RXN')
-----
[CH2:6]1[CH2:5][CH2:4][CH2:3][CH2:2][CH2:1]1>>[CH2:6]1[CH2:5]
[CH2:4][CH2:3][CH2:2][C:1]1=O
```

**Comments**

- There are limitations to the generation of SMILES strings. Not all BIOVIA reaction features can be handled. If the specified molecule cannot be handled, the mdlaux.rxnsmiles function returns NULL. Use mdlaux.errors to see the related error message.
- Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();}
```

**See also**

[rxnsmiles](#)

*BIOVIA Direct Developers Guide > Using Direct > Getting the SMILES string*

*BIOVIA Direct Developers Guide > Using Direct > Limitations to the generation of SMILES string*

**mdlaux.smilestorxnfile**

Returns a reaction string (CLOB) from a SMILES string (CLOB).

**Syntax**

mdlaux.smilestorxnfile(*smiles*)

Parameter	Description
<i>smiles</i>	A CLOB containing a reaction SMILES string

**Return value**

A temporary CLOB that contains the converted rxnfile string. If the reaction SMILES string cannot be converted, the mdlaux.smilestorxnfile function returns NULL. Use mdlaux.errors to see related error message.

**Usage**

```
select mdlaux.smilestorxnfile(smiles) from dual;
```

**Example**

```
SQL> select
mdlaux.smilestorxnfile('[CH2:6]1[CH2:5][CH2:4][CH2:3][CH2:2]
[CH2:1]1>>[CH2:6]1[CH2:5][CH2:4][CH2:3][CH2:2][C:1]1=O') from
```

```

dual;

```

```

MDLAUX.SMILESTORXNFILE(' [CH2:6]1[CH2:5][CH2:4][CH2:3][CH2:2]
[CH2:1]1>>[CH2:6]1[C

```

```

-----
$RXN

```

```

SciTegic      0329131054

```

```

  1 1
$MOL

```

```

SciTegic03291310542D

```

```

6 6 0 0 0          0999 v2000
  1.2990      -0.75000.0000 C      0 0 0 0 0 0 0 0 0 6 0 0
  1.2990       0.75000.0000 C      0 0 0 0 0 0 0 0 0 5 0 0
  0.0000       1.50000.0000 C      0 0 0 0 0 0 0 0 0 4 0 0
 -1.2990       0.75000.0000 C      0 0 0 0 0 0 0 0 0 3 0 0
 -1.2990      -0.75000.0000 C      0 0 0 0 0 0 0 0 0 2 0 0
  0.0000      -1.50000.0000 C      0 0 0 0 0 0 0 0 0 1 0 0
  1 2 1 0
  2 3 1 0
  3 4 1 0
  4 5 1 0
  5 6 1 0
  6 1 1 0
M END
$MOL

```

```

SciTegic03291310542D

```

```

7 7 0 0 0          0999 v2000
  1.2990      -0.75000.0000 C      0 0 0 0 0 0 0 0 0 6 0 0
  1.2990       0.75000.0000 C      0 0 0 0 0 0 0 0 0 5 0 0
  0.0000       1.50000.0000 C      0 0 0 0 0 0 0 0 0 4 0 0
 -1.2990       0.75000.0000 C      0 0 0 0 0 0 0 0 0 3 0 0
 -1.2990      -0.75000.0000 C      0 0 0 0 0 0 0 0 0 2 0 0
  0.0000      -1.50000.0000 C      0 0 0 0 0 0 0 0 0 1 0 0
  0.0000      -2.70000.0000 O      0 0
  1 2 1 0
  2 3 1 0
  3 4 1 0
  4 5 1 0
  5 6 1 0
  6 1 1 0
  6 7 2 0
M END

```

### Comments

Applications using client interfaces such as JDBC must explicitly free the temporary LOB value returned by this function. 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();  
}
```

### See also

[rxnsmiles](#)

[mdlaux.rxnsmiles](#)

*BIOVIA Direct Developers Guide > Using Direct > Conversion of SMILES strings to molfile*

## Chapter 5: Examples

<a href="#">Flexmatch Search</a>	213
<a href="#">Substructure Search</a>	215
<a href="#">Molecule Formula Search</a>	217
<a href="#">Molecule Similarity Search</a>	218
<a href="#">Reading a Molfile</a>	219
<a href="#">Retrieving Molfile Structures</a>	219
<a href="#">Retrieving Chime Structures</a>	220
<a href="#">Structure Registration</a>	220
<a href="#">Reaction Flexmatch Search</a>	222
<a href="#">Reaction Substructure Search</a>	223
<a href="#">Reaction Similarity Search</a>	226
<a href="#">Writing a File</a>	228
<a href="#">Fetching Reactions Using the Rxnfile Format</a>	228
<a href="#">Reading a Rxnfile</a>	230
<a href="#">Fetching Reactions Using the Chime Format</a>	231
<a href="#">Reaction Registration</a>	234

### Flexmatch Search

The following are examples of SQL statements that use flexmatch:

Task	Example
Finding records that match a structure in a molfile	<pre>select cdbregno from sample2d where flexmatch(   ctab,   '/home/users/structx.mol',   'match=all' )=1;</pre>
Finding records that match the tautomers in a structure in a molfile	<pre>select cdbregno from sample2d where flexmatch(   ctab,   '/temp/structmorph.mol',   'fra/sal/tau' )=1;</pre>
Finding records that exactly match a structure in a string variable s1 that contains a structure from a molfile. The molfile string contains embedded newline characters.	<pre>select cdbregno from sample2d where flexmatch(ctab, :s1, 'all')=1</pre> <p>Note: See Oracle SQL documentation for details about binding variables in a SQL statement.</p>
Finding records that match a structure using a variable dcstruc, which contains	<pre>select to_char (cdbregno)</pre>

Task	Example
a Chime string, and dcFcpOpts which contains the flexmatch parameters (a VBScript example)	<pre> from sample2d where flexmatch(   ctab, " &amp;dcStruc &amp; "',' &amp;   dcFcpOpts &amp;'')=1; </pre>
Finding records within a range of cdbregno numbers, and that match a structure in a molfile	<pre> select cdbregno from sample2d where dcbregno btween 200 and 300 and flexmatch(   ctab,   '/home/users/structx.mol',   'match=all' )=1; </pre>
Finding records that exactly match a BLOB structure from another table	<pre> select cdbregno from sample2d where flexmatch(   ctab,   (select ctab     from acd2d_mol     where cdbregno=20),   'all' )=1; </pre>
Finding records that least match a CLOB structure from another table	<pre> select cdbregno from sample2d where flexmatch(   ctab,   (select molfile(ctab)     from acd2d_mol     where cdbregno=20),   'none' )=1; </pre>
Finding records that exactly match a structure that is represented by a SMILES string	<pre> select cdbregno from sample2d where flexmatch(ctab,'c1ccccc1','all')=1; </pre>
Determining if a query that is represented by a SMILES string matches a specific record in the database	<pre> select flexmatch(ctab,'C1CCCCC1','all') from sample2d where cdbregno=27; </pre>
Joining a flexmatch search with a substructure search on the same table	<pre> select b.cdbregno from sample2d a, sample2d b where flexmatch(a.ctab, b.ctab, 'all') = 1 and sss(b.ctab, 'c1ccccc1')=1; </pre>



Task	Example
Joining a flexmatch search with a substructure search on a different table	<pre>select b.cdbregno from sample2d a,      acd2d_mol b where a.cdbregno = b.cdbregno and      flexmatch(a.ctab,b.ctab,'all')=1 and      sss(b.ctab, 'c1ccccc1')=1;</pre>

## Substructure Search

The following are examples of SQL statements that use sss:

Task	Example
Finding structures that contain a substructure in a molfile	<pre>select cdbregno from sample2d where sss(   ctab,   '/home/users/structx.mol' )=1;</pre>
Finding structures that contain a substructure in a string variable s1 that contains a structure from a molfile. The molfile string contains embedded newline characters.	<pre>select cdbregno from sample2d where sss(ctab, :s1)=1;</pre> <p><b>Note:</b> See Oracle SQL documentation for details about binding variables in a SQL statement.</p>
Finding structures that contain a substructure using a string variable dcStruc that contains a Chime string (a VBScript example)	<pre>select to_char(cdbregno) from sample2d where sss(ctab,'"&amp; dcStruc &amp;"')=1</pre>
Counting the structures that contain a substructure that is represented by a SMILES string	<pre>select count(*) from sample2d where sss(ctab,'c1ccccc1')=1;</pre>

Task	Example
Finding records within a range of cdbregno numbers, and that contain a substructure in a temporary table	<ol style="list-style-type: none"> <li>1. Create a temporary table that will contain a CLOB structure from a molfile. For example:  <pre>create table temp_query(query clob);</pre> </li> <li>2. Insert the molfile structure into the temporary table. For example:  <pre>insert into temp_query(query) values(readmol('/home/users/structx.mol')); commit;</pre> </li> <li>3. Perform a substructure search on the molfile structure in the temporary table, for records within the range of cdbregno numbers. For example:  <pre>select cdbregno from sample2d i, temp_query t where cdbregno between 10 and 100 and sss(i.ctab, t.query)=1;</pre> </li> <li>4. Drop the temporary table. For example:  <pre>drop table temp_query;</pre> </li> </ol>
Performing a substructure search, and saving the results into an Oracle table that can be searched by Direct	<ol style="list-style-type: none"> <li>1. Create a table whose rows are populated from a substructure search. For example:  <pre>create table result_sss as select * from sample2d where sss( ctab, '/home/users/structx.mol'  =1;</pre> </li> <li>2. Create an index for the table, using the cdbregno field. For example:  <pre>create index result_sss_idx on result_sss (cdbregno);</pre> </li> <li>3. Create a molecule domain index for the table, using the ctab field. For example:  <pre>create index result_sss_ixmdl on result_sss(ctab) indextype is c\$direct2021.mxixmdl;</pre> </li> <li>4. To test the results, count the number of rows for the newly created table. The count must match the count of rows from the original substructure search. For example:  <pre>select count(*) from result_sss; select count(*) from sample2d where sss( ctab, '/home/users/structx.mol'</pre> </li> </ol>

Task	Example
	<pre>)=1;</pre> <p>5. Drop the temporary table. For example:</p> <pre>drop table result_sss;</pre>
Joining an exact match search with a substructure search on the same table	<pre>select b.cdbregno from sample2d a,      sample2d b where flexmatch(a.ctab, b.ctab, 'all') = 1      and sss(b.ctab, 'c1ccccc1')=1;</pre>
Highlighting the query structure in a Substructure search that returns the molfile format of the matching structure	<pre>select sss_highlight_molfile(1) from sample2d where sss(   ctab, readmol('/home/users/2dquery.mol'),   1 )=1;</pre>
Highlighting the query structure in a substructure search that returns the Chime format of the matching structures	<pre>select ssshhighlight(1) from sample2d where sss(   ctab,   '/home/users/2dquery.mol',   1 )=1;</pre>

## Molecule Formula Search

The following are examples of SQL statements that use `fm1alike` and `fm1amatch`:

Task	Example
Finding records that contain a formula	<pre>select cdbregno from sample2d where fm1alike(ctab, 'C6 H6')=1;</pre>
Finding records that contain a formula in a string variable s1	<pre>select cdbregno from sample2d where fm1alike(ctab, :s1)=1;</pre> <p><b>Note:</b> See Oracle SQL documentation for details about binding variables in a SQL statement.</p>
Finding records that match an exact formula	<pre>select cdbregno from sample2d where fm1amatch(ctab, 'C2 H4')=1;</pre>
Finding records and formulas for a range of cdbregno numbers that contain a formula	<pre>select cdbregno,        molfm1a(ctab) from sample2d</pre>

Task	Example
	<pre>where fmlalike(ctab,'c17h19n1o3')=1 and cdbregno between 10 and 50;</pre>
Finding records for a range of cdbregno numbers that do not contain a formula	<pre>select cdbregno from sample2d where (not fmlalike (ctab,'c17h19n1o3')=1) and cdbregno between 10 and 50;</pre>
Performing a substructure search, and saving the results into an Oracle table that can be searched by Direct	<pre>select cdbregno, molfm1a(ctab), fmlalike(ctab,'c2 h4') from sample2d where cdbregno &lt; 5;</pre>

## Molecule Similarity Search

The following are examples of SQL statements that use similar:

Task	Example
Finding records that are similar to the structure in a molfile, with a similarity degree of 40 percent or more	<pre>select cdbregno from sample2d where similar( ctab, 'c:\temp\structx.mol', '40' ) = 1;</pre>
Finding records that are similar to the structure using a string variable s1 that contains a structure from a molfile. The molfile string contains embedded newline characters.	<pre>select cdbregno from sample2d where similar(ctab,:s1,'40') = 1;</pre> <p><b>Note:</b> See Oracle SQL documentation for details about binding variables in a SQL statement.</p>
Finding records that are similar to the structure in a molfile, within a range of similarity degree	<pre>select cdbregno from sample2d where similar( ctab, readmol('/home/users/structx.mol'), '10 20' ) = 1;</pre>
Counting the records that are similar to a structure that is represented by a query string, with a similarity degree of less than 10	<pre>select count(*) from sample2d where similar(ctab,'c1ccccc1', 'sub 10') = 1;</pre>

## Reading a Molfile

The following are examples of SQL statements that use `readmol` and `readbinaryfile`:

Task	Example
Reading a molfile using the <code>readmol</code> operator.	<pre>select readmol('c:\temp\structx.mol') from dual;</pre>
Reading a molfile using the <code>readbinaryfile</code> operator.	<pre>select readfile('c:\temp\structx.mol') from dual;</pre>
Populating a table queries with the contents of a molfile, and using CLOB structures in the table to perform a substructure search	<p>1. Create a table that will contain a CLOB structure from a molfile. For example:</p> <pre>create table queries(   qid varchar2(32),   query clob ); insert into queries(   qid,   query ) values(   'benzene',   readmol('/home/smith/mols/benzene.mol') ); commit;</pre> <p>2. Grant select access on the new table queries to the database owner. For example:</p> <pre>grant select on queries to dcsamples;</pre> <p>3. Perform a substructure search using the CLOB structures in the table queries. For example:</p> <pre>select cdbregno from sample2d where sss(   ctab,   (select query from queries where    qid='benzene') )=1;</pre> <p>4. Drop the temporary table. For example:</p> <pre>drop table queries;</pre> <p><b>Note:</b> If you do not grant select access on the user table queries to the database owner, you will get the error: table does not exist.</p>

## Retrieving Molfile Structures

The following are examples of SQL statements that use `molfile`, `molfile_string`, and `molfile_string_seg`:

Task	Example
Retrieving a molfile CLOB	<pre>select molfile(ctab) from sample2d where cdbregno = 355;</pre>
Retrieving a molfile string of a small structure	<pre>select molfile_string(ctab) from sample2d where cdbregno=9;</pre>

## Retrieving Chime Structures

The following are examples of SQL statements that use `molchime` and `chime_string`:

Task	Example
Retrieving a molfile CLOB	<pre>select molchime(ctab) from sample2d where cdbregno = 9;</pre>
Retrieving a Chime string of a small structure	<pre>select chime_string(ctab) from sample2d where cdbregno=9;</pre>

## Structure Registration

The following are examples of SQL statements that insert and update structures using the `mol` operator.

Task	Example
Registering a structure and user data into a table, using the molfile format of the structure	<pre>insert into sample2d(   ctab,   corp_id,   f_date ) values(   mol('/home/users/structx.mol'),   'MUSE00100452',   to_date ('13-apr-2000') );</pre>
Updating a structure and the molecule formula in a table	<pre>update sample2d set ctab =   mol(     (select molfile(ctab)      from sample2d      where flexmatch(        ctab,        '/home/users/structx.mol',        'match=all,bon'</pre>

Task	Example
	<pre>         )=1       )     ),     molformula=(makeclob('C6 H6'))   where cdbregno=367; </pre>
Updating a structure and user data in a table	<pre> update sample2d set ctab =   mol(     (select molfile(ctab)      from sample2d      where cdbregno=281)   ),   molname = 'Sodium hydroxide',   corp_id = 'MUSE00500023',   f_date = to_date ('13-apr-2000') where molname = 'Thiamine hydrochloride'; </pre>

The following examples are UPDATE and DELETE statements that use Direct search operators in the WHERE clause. These examples work on Oracle 10, but note that these examples may not work correctly on Oracle 9. In Oracle 9, if the WHERE clause of a DELETE or UPDATE statement uses BIOVIA Direct search operators and Oracle executes the search using the domain index, the SQL fails with an ORA-04091 error. For example, the following SQL returns with an ORA-04091 error:

```
delete from sample2d where fmlamatch(ctab, 'c6 h5 c11')=1;
```

If the search is modified so that it does not use the domain index, by adding a zero to the FMLAMATCH result, then the operation will succeed. The following SQL successfully deletes a row:

```
delete from sample2d where fmlamatch(ctab, 'c6 h5 c11')+0 =1;
```

Task	Example
Updating a structure in a table, using a structure in a molfile	<pre> update sample2d set ctab =   mol('/home/users/structx.mol') where flexmatch(ctab,   'c1cccc1', 'all')=1; </pre>
Deleting rows from a table that match a flexmatch search	<pre> delete from sample2d where flexmatch(   ctab,     '/home/users/structx.mol',     'all'   )=1; </pre>

Task	Example
Deleting rows from a table that match a substructure search	<pre>delete from sample2d where sss( ctab,   '/home/users/structx.mol' )=1;</pre>
Deleting rows from a table that match a formula search	<pre>delete from sample2d where fmla_like(ctab, 'Mn O2')=1;</pre>

## Reaction Flexmatch Search

The following are examples that use rxnflexmatch:

Task	Example
Finding an exact match of a reaction in a rxnfile	<pre>select rxnmdlnumber from samplerx_reaction where rxnflexmatch(   rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   'all' )=1;</pre>
Finding reactions that contain an exact match of a reaction that is stored in a temporary CLOB. This example uses writetempclob to construct a reaction that can contain more than 4000 characters into a temporary CLOB, and uses tempclob to get its contents. For an example of how to do this within a client application, see “Copying string segments into a temporary CLOB” in the <i>BIOVIA Direct Developers Guide</i> .	<pre>select writetempclob(query-string, 0) from dual; select writetempclob(next-query-string, 1) from dual; select writetempclob(last-query-string, 1) from dual; select rxnmdlnumber from samplerx_reaction where rxnflexmatch(   rctab, tempclob(0),   'subset all' )=1;</pre>



Task	Example
<p>Finding records that exactly match a reaction</p> <p>The following variable examples that use rss:</p>	<pre>select rxnmdlnumber from samplerx_reaction where rxnflexmatch(rctab, :s1,'all')=1;</pre>
Task	Example
<p>Finding structures that contain a substructure in a rxnfile</p> <p>string contains embedded newline characters, and only contains up to 4000 characters.</p>	<pre>select rxnmdlnumber from samplerx_reaction where rss(   rctab,</pre>
<p>Finding records that match a Chime reaction that is embedded in a</p>	<p>'/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rx</p> <p>This embed HTML tag defines the Chime box:</p> <pre>' &lt;embed type="chemical/x-mdl-rxnfile" width=400 height=200</pre>
<p>Finding reactions that contain a reaction that is stored in a temporary CLOB. This example uses writetempclob to construct a reaction that can contain more than 4000 characters into a temporary CLOB, and uses tempclob to get its contents. For an example of how to do this within a client application, see "Copying string segments into a temporary CLOB" in the <i>BIOVIA Direct Developers Guide</i>.</p>	<pre>select writetempclob(query-string, 0) from dual; select writetempclob(next-query-string, 1) from dual; select writetempclob(last-query-string, 1) from dual; select rxnmdlnumber from samplerx_reaction where rss(   rctab,   tempclob(0) )=1;</pre>
<p>Finding reactions that contain a reaction that is stored in a temporary CLOB. This example uses writetempclob to construct a reaction that can contain more than 4000 characters into a temporary CLOB, and uses tempclob to get its contents. For an example of how to do this within a client application, see "Copying string segments into a temporary CLOB" in the <i>BIOVIA Direct Developers Guide</i>.</p>	<pre>)=1;select rxnmdlnumber from samplerx_reaction where rss(   rctab,   :s1 )=1;</pre> <p><b>Note:</b> See Oracle SQL documentation for details about binding variables in a SQL statement.</p>
<p>Finding records that contain a Chime reaction that is embedded in a query HTML page (a VBScript example)</p>	<p>'This embed HTML tag defines the Chime box:</p> <pre>' &lt;embed type="chemical/x-mdl-rxnfile" width=400 height=200 queryformbox="document.query.queryrxn" &gt; 'Get the Chime string rxnChime = document.query.queryrxn.value; 'Use the Chime string in the SQL query sqlString = "select rxnmdlnumber</pre>

Task	Example
	<pre>from samplerx_reaction where rss(   rctab,"   &amp; rxnChime &amp; " )=1";</pre>
Highlighting the query in a reaction substructure search that returns the highlighted Chime string	<pre>select rsshighlight(1) from samplerx_reaction where rss(   rctab,  '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn',1 )=1;</pre>
Finding reactions that contain an automapped reaction in a rxnfile	<pre>select rxnmdlnumber from samplerx_reaction where rss(   rctab,   rxnautomap(  '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   NULL) )=1;</pre>

Task	Example
<p>Determining if a BLOB reaction that was converted from a rxnfile contains the structure in another rxnfile. This example uses the package function name for rxn to convert a rxnfile to a BLOB, and uses the package function name for readbinaryfile to read the contents of the second rxnfile. This is a PL/SQL example. If you want to see the output value, execute the following SQL*Plus command before running this sample code:</p> <pre>SQL&gt; set serveroutput on</pre>	<pre>/DECLARE   candidate BLOB;   query CLOB;   match NUMBER; BEGIN   --Convert rxnfile to BLOB   candidate := mdlaux.rxn(  '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn');   --Read rxnfile to a CLOB   query := mdlaux.filetoclob(  '/opt/BIOVIA/direct2021/examples/rxnfiles/query3.rxn');   select rss(     candidate,     query)   into match   from dual;   dbms_output.put_line('Match='    to_char(match)); END; /</pre>
<p>Retrieving multiple (highlighted and unhighlighted) Chime strings for a specific reaction that matches a reaction substructure search. This example assumes that a Chime string can be more than 4000 characters.</p>	<pre>DECLARE   str1 varchar2(4000);   str2 varchar2(4000);   bigstr1 varchar2(32000);   bigstr2 varchar2(32000); BEGIN   --Fetch the first segments of the unhighlighted   --and highlighted Chime strings   select rxnstringsegment(1, rxnchime(rxn)),          rxnstringsegment(2, rsshighlight(99))   into bigstr1, bigstr2   from samplerx_reaction   where rss(rctab,  '/opt/BIOVIA/direct2021/examples/rxnfiles/rssq1.rxn',   99 )=1;   --Fetch the rest of the Chime strings   loop     select rxnstringsegment(1), rxnstringsegment(2)     into str1, str2</pre>

Task	Example
	<pre> from dual; if str1 is NULL and str2 is NULL then exit; end if; bigstr1 := bigstr1    str1; bigstr2 := bigstr2    str2; end loop; END; </pre>
Joining a reaction flexmatch search with a reaction substructure search on the same table	<pre> select b.rxnmdlnumber from samplerx_reaction a,      samplerx_reaction b where rxnflexmatch(   a.rctab,   b.rctab,   'all') = 1 and rss(   b.rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query4.rxn')=1 ; </pre>

## Reaction Similarity Search

The following are examples that use rxnsim:

Task	Example
Finding reactions that are at least 80% similar in the reacting center portion in the query, and at least 20% similar in the molecule portion in the query. This is the default.	<pre> select rxnmdlnumber from samplerx_reaction where rxnsim(   rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   '' )=1; </pre>
Finding reactions that are at least 60% similar in the reacting center portion in the query, and at least 40% similar in the molecule portion in the query. The query is stored in a temporary CLOB. This example uses writetempclob to construct a reaction that	<pre> select writetempclob(query-string, 0) from dual; select writetempclob(next-query-string, 1) from dual; select writetempclob(last-query-string, 1) from dual; select rxnmdlnumber from samplerx_reaction where rxnsim(   rctab,   tempclob(0),   '60 40' )=1; </pre>

Task	Example
can contain more than 4000 characters into a temporary CLOB, and uses <code>tempclob</code> to get its contents. For an example of how to do this within a client application, see “Copying string segments into a temporary CLOB” in the <i>BIOVIA Direct Developers Guide</i> .	
Finding reactions that are at between 20% and 80% similar in the reacting centers and molecule portion in the query.	<pre>select rxnmdlnumber from samplerx_reaction where rxnsim(   rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   '20 80 20 80' )=1;</pre>
Finding reactions that are very dissimilar, that is, at most 3% similar to both reacting centers and molecules of a structure in a rxnfile. This example also shows molecule and reacting center similarity values. The ORDER BY clause orders the results by descending similarity to see the most dissimilar first.	<pre>select rxnmdlnumber,   rxnmolsim(1) "Molecule Similarity",   rxnctrsim(1) "Reacting Center Similarity" from samplerx_reaction where rxnsim(   rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   '0 3 0 3',   1 )=1; order by rxnctrsim(1) desc, rxnmolsim(1) desc;</pre>
Finding reactions that are similar to a substructure in a rxnfile	<pre>select rxnmdlnumber from samplerx_reaction where rxnsim(   rctab,    '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   '80 20 sub' )=1;</pre>
Finding reactions that are similar to a	<pre>select rxnmdlnumber from samplerx_reaction</pre>

Task	Example
superstructure in a rxnfile	<pre>where rxnsim(   rctab,   '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn',   '20 40 20 40 super' )=1;</pre>

## Writing a File

The following are examples that use `writebinaryfile`:

Task	Example
Writing a rxnfile for a specific reaction	<pre>select writefile(   rxnfile(rctab),   '/opt/BIOVIA/direct/examples/rxnfiles/rxn1.rxn') from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
Writing a molfile for the first molecule component in the product of a reaction	<pre>select writefile(   rxnmol(rctab, 2, 1),   '/opt/BIOVIA/direct/examples/rxnfiles/rxn1.mol') from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
Writing a file that contains the Chime string representation of a reaction	<pre>select writefile(   rxnchime(rctab),   '/opt/BIOVIA/direct/examples/rxnfiles/rxnchime.txt' ) from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
Converting a Chime string in a file into the rxnfile format, and writing it to another file	<pre>select writefile(   mdlaux.chimetoclob(     readfile(       '/opt/BIOVIA/direct2021/examples/rxnfiles/rxnchime.txt')),   '/opt/BIOVIA/direct2021/examples/rxnfiles/rxnfile.rxn') from dual;</pre>
Converting the contents of a rxnfile into a Chime string, and writing it to another file	<pre>select writefile(   mdlaux.clobtochime(     readfile(       '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn')),   '/opt/BIOVIA/direct2021/examples/rxnfiles/rxn1.txt') from dual;</pre>

## Fetching Reactions Using the Rxnfile Format

The following are examples that use `rxnfile`, `rxnstringsegment`, `tempclob`, and `writetempclob`:

Task	Example
Retrieving a rxnfile CLOB from a reaction substructure search	<pre>select rxnfile(rctab) from samplerx_reaction where rss(rctab, '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn' )=1;</pre>
Retrieving a rxnfile CLOB representation of a reaction, and saving it to a variable	<pre>DECLARE   rxnfileclob CLOB; BEGIN   select rxnfile(rctab)   into rxnfileclob   from samplerx_reaction   where rxnmdlnumber='RXCI94061946'; END;</pre>
Retrieving a rxnfile CLOB representation of a reaction, and writing it to a rxnfile	<pre>select writefile(   rxnfile(rctab),   '/opt/BIOVIA/direct2021/examples/rxnfiles/rxn1.rxn') from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
Retrieving an automapped rxnfile CLOB from a reaction substructure search	<pre>select rxnautomap( rxnfile(rctab), NULL) from samplerx_reaction where rss(   rctab,   '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn' )=1;</pre>
Highlighting the query structure in a reaction substructure search that returns the highlighted Chime string, and retrieving its rxnfile representation	<pre>select mdlaux.chimetoclob(rsshhighlight(1)) from samplerx_reaction where rss(   rctab,   '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn',   1 )=1;</pre>
Retrieving segments of rxnfile strings from a search. This is a PL/SQL example that: <ul style="list-style-type: none"> <li>■ Gets the first string segment of a reaction</li> </ul>	<pre>DECLARE   strseg VARCHAR2(4000);   bigstr VARCHAR2(32000);   rxnval BLOB;   match NUMBER;</pre>

Task	Example
<p>using rxnstringsegment</p> <ul style="list-style-type: none"> <li>■ Uses rxnstringsegment(0) to get the subsequent segments of the reaction, and accumulate the segments into another string variable</li> <li>■ Uses the package function name of rxn to convert the big string to a BLOB, and compare it with the original reaction</li> </ul> <p>If you want to see the output value, execute the following SQL*Plus command before running this sample code:</p> <pre>SQL&gt; set serveroutput on</pre>	<pre>BEGIN --Get the first string segment of reaction select rxnstringsegment(rxnfile(rctab)) into bigstr from samplerx_reaction where rxnmdlnumber = 'RXCI94061946'; loop --Get the next string segment select rxnstringsegment(0) into strseg from dual; IF strseg IS NULL THEN EXIT; END IF; bigstr := bigstr    strseg; end loop; --Convert the complete reaction string to a BLOB rxnval := mdlaux.rxn(bigstr); --Check if the BLOB matches original reaction select rxnflexmatch(rctab, rxnval, 'all') into match from samplerx_reaction where rxnmdlnumber='RXCI94061946'; dbms_output.put_line ('Match='    to_char(match)); END;</pre>

## Reading a Rxnfile

The following are examples that read a rxnfile:

Task	Example
Performing a reaction flexmatch search based on a structure in a rxnfile	<pre>select rxnmdlnumber from samplerx_reaction where rxnflexmatch( rctab, '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn', 'all' )=1;</pre>
Performing a substructure search based on a structure in a rxnfile	<pre>select rxnmdlnumber from samplerx_reaction where rss( rctab, '/opt/BIOVIA/direct2021/examples/rxnfiles/query4.rxn' )=1;</pre>



Task	Example
Performing a reaction similarity search based on a structure in a rxnfile. This example uses the package function name for rxn to convert a rxnfile to a BLOB, and uses the package function name for readbinaryfile to read the contents of the second rxnfile. This is a PL/SQL example. If you want to see the output value, execute the following SQL*Plus command before running this sample code: SQL> set serveroutput on	<pre> DECLARE   candidate BLOB;   query CLOB;   match NUMBER;   rcsim NUMBER;   molsim NUMBER; BEGIN   candidate := mdlaux.rxn(     '/opt/BIOVIA/direct2021/examples/rxnfiles/query2.rxn') ;    query := mdlaux.readfile(     '/opt/BIOVIA/direct2021/examples/rxnfiles/query3.rxn') ;   select rxnctrsim(1), rxnmolsim(1),          rxnsim(candidate,query, '50 50', 1)          into rcsim,               molsim,               match          from dual;   dbms_output.put_line('Match='    to_char(match));   dbms_output.put_line('Rctr='    to_char(rcsim));   dbms_output.put_line('Mol='    to_char(molsim)); END; </pre>

## Fetching Reactions Using the Chime Format

The following are examples that use rxnchime and rxnstringsegment:

Task	Example
Retrieving a Chime CLOB from a reaction substructure search	<pre> select rxnchime(rctab) from samplerx_reaction where rss(   rctab,   '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn' )=1; </pre>
Retrieving a Chime CLOB representation of a reaction, and saving it to a variable	<pre> DECLARE   chimeclob CLOB; BEGIN   select rxnchime(rctab)   into chimeclob   from samplerx_reaction   where rxnmdlnumber='RXCI94061946'; END; </pre>

Task	Example
Retrieving a Chime CLOB representation of a reaction, and writing it to a rxnfile	<pre>select writefile(   rxnchime(rctab),   '/opt/BIOVIA/direct2021/examples/rxnfiles/rxn1chime.txt') from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
Highlighting the query structure in a reaction substructure search that returns the highlighted Chime string	<pre>select rsshighlight(1) from samplerx_reaction where rss(   rctab,   '/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn',   1 )=1;</pre>
Retrieving a Chime string representation of a reaction. This example assumes that the Chime string is not more than 4000 characters. (Chime strings are generally smaller than rxnfile strings.)	<pre>select rxnstringsegment(   rxnchime(rctab)) from samplerx_reaction where rxnmdlnumber='RXCI94061946';</pre>
<p>Retrieving segments of Chime strings from a search. This example assumes that a Chime string can be more than 4000 characters. This is a PL/SQL example that:</p> <ul style="list-style-type: none"> <li>■ Gets the first string segment of a reaction using rxnstringsegment</li> <li>■ Uses rxnstringsegment(0) to get the subsequent segments of the reaction, and accumulate the</li> </ul>	<pre>DECLARE   strseg VARCHAR2(4000);   bigstr VARCHAR2(32000);   rxnval BLOB;   match NUMBER; BEGIN   --Get the first Chime string segment of reaction   select rxnstringsegment(rxnchime(rctab))   into bigstr   from samplerx_reaction where   rxnmdlnumber='RXCI94061946';   loop     --Get the next string segment     select rxnstringsegment(0)     into strseg     from dual;     IF strseg IS NULL THEN EXIT; END IF;     bigstr := bigstr    strseg;   end loop;   --Convert the complete Chime string to a BLOB   rxnval := mdlaux.rxn(bigstr);   --Check if the BLOB matches original reaction</pre>

Task	Example
<p>segments into another string variable</p> <ul style="list-style-type: none"> <li>■ Uses the package function name of rxn to convert the big string to a BLOB, and compare it with the original reaction</li> </ul> <p>If you want to see the output value, execute the following SQL*Plus command before running this sample code:</p> <pre>SQL&gt; set serveroutput on</pre>	<pre>select rxnflexmatch(rctab, rxnval, 'all') into match from samplerx_reaction where rxnmdlnumber='RXCI94061946'; dbms_output.put_line ('Match='    to_char(match)); END;</pre>
<p>Retrieving multiple (highlighted and non-highlighted) Chime strings for a specific reaction that matches a reaction substructure search. This example assumes that a Chime string can be more than 4000 characters.</p>	<pre>DECLARE   str1 varchar2(4000);   str2 varchar2(4000);   bigstr1 varchar2(32000);   bigstr2 varchar2(32000); BEGIN   --Fetch the first segments of the unhighlighted   --and highlighted Chime strings   select rxnstringsegment(1, rxnchime(rctab)),          rxnstringsegment(2, rsshighlight(99))   into bigstr1, bigstr2   from samplerx_reaction   where rss(rctab,  '/opt/BIOVIA/direct2021/examples/rxnfiles/rssq1.rxn', 99 )=1;   --Fetch the rest of the Chime strings   loop     select rxnstringsegment(1), rxnstringsegment(2)     into str1, str2     from dual;     if str1 is NULL and str2 is NULL then exit;     end if;     bigstr1 := bigstr1    str1;     bigstr2 := bigstr2    str2;   end loop; END;</pre>

## Reaction Registration

The following are examples that insert and update reactions using the rxn operator. These examples also include SQL statements that delete reactions from a database:

Task	Example
Registering a reaction in a rxnfile	<pre>insert into samplerx_reaction(   rxnmdlnumber,   rctab ) values(   'NEWRXN99910',   rxn ('/opt/BIOVIA/direct2021/examples/rxnfiles/query1.rxn') );</pre>
Registering a reaction in a rxnfile (a PL/SQL example)	<pre>DECLARE   rxnval blob; BEGIN   rxnval := mdlaux.rxn(     '/opt/BIOVIA/direct91/examples/rxnfiles/query1.rxn');   insert into samplerx_reaction(     rxnmdlnumber,     rctab)   values(     'NEWRXN99911',     rxnval   ); END;</pre>
Registering a reaction in a Chime string that is stored in a file	<pre>insert into samplerx_reaction(   rxnmdlnumber,   rctab ) values(   'NEWRXN99912',   rxn(     readfile(       '/opt/BIOVIA/direct2021/examples/rxnfiles/rxnchime.txt')     )   );</pre>
Registering a reaction that is stored in a temporary CLOB. This example uses writetempclob to construct a reaction that can	<pre>select writetempclob(rxn-string, 0) from dual; select writetempclob(next-rxn, 1) from dual; select writetempclob(last-rxn, 1) from dual; insert into samplerx_reaction(   rxnmdlnumber   rctab ) values(   'NEWRXN99913',</pre>

Task	Example
contain more than 4000 characters into a temporary CLOB, and uses tempclob to get its contents. For an example of how to do this within a client application, see “Copying string segments into a temporary CLOB” in the <i>BIOVIA Direct Developer’s Guide</i> .	<pre>rxn(tempclob(0)) );</pre>
Updating a reaction with a different reaction	<pre>update samplerx_reaction set rctab =   (select rctab    from samplerx_reaction    where rxnmdlnumber='RXCI94013284') where rxnmdlnumber = 'RXCI94070168';</pre>

## Chapter 6:

# Molecule Searches

---

<a href="#">Flexmatch Search</a> .....	236
<a href="#">Substructure Search</a> .....	237
<a href="#">Similarity Search</a> .....	238
<a href="#">Molecule Formula Search</a> .....	239

## Flexmatch Search

A flexible match (`flexmatch`) search finds structures that match your query molecule exactly, except in ways that you specify with any of the flexmatch switches. The flexmatch switches allow you to selectively restrict or relax the criteria that are used to determine whether a structure is an exact match. To perform a flexmatch search, use the [flexmatch](#) operator.

The flexmatch operator requires you to specify flexmatch parameters, such as the following:

```
select cdbregno
from moltable
where flexmatch(ctab, '/home/user/query.mol', flexmatch-
parameters)=1;
```

where *flexmatch-parameters* is a string that consists of MATCH or IGNORE parameters. MATCH and IGNORE parameters allow you to selectively restrict or relax the search criteria based on a query structure. The format of the string *flexmatch-parameters* is:

```
MATCH|IGNORE=FP1[,FP2,FP3,...]
```

where FP1, FP2, FP3 are the flexmatch switches. For additional information and examples of flexmatch switch settings, see the *Exact Search (Flexmatch)* chapter in *BIOVIA Chemical Representation*.

You can combine more than one flexmatch switch with either the MATCH or the IGNORE parameter (not both). If you specify more than one flexmatch switch, use a comma (,) or a forward slash (/) to separate them. If you do not specify MATCH or IGNORE, the cartridge assumes the MATCH parameter. For example, the following flexmatch parameters are equivalent. Both specify an exact match:

```
flexmatch(ctab, '/home/user/query.mol', 'match=all')
flexmatch(ctab, '/home/user/query.mol', 'all')
```

The MATCH parameter allows you to specify features of the retrieved structures that must match the query. If you want a less restrictive definition of exact match, use MATCH= to enable only those switches that you want to apply. Any switches not listed in MATCH= are ignored. MATCH=NONE is the loosest possible definition. Each switch you add in MATCH= tightens the match criteria, and thus generally permits fewer hits.

To enable a flexmatch switch in the search criteria, do one of the following:

- Include the switch in the MATCH parameter.
- Exclude the switch from the IGNORE parameter.

The IGNORE parameter allows you to specify features of the structures that will be ignored in the query. If you want a relatively tight definition of exact match with the switches, use IGNORE= to specify only those switches that you want to disable. Any switches not listed in IGNORE= are matched.

IGNORE=NONE is the tightest possible definition (It is equivalent to an exact match). Each switch you add in IGNORE= relaxes the match criteria.

**IMPORTANT!** Be careful when using IGNORE, since any switches that you do not explicitly include in the list *will* be turned on.

To disable a flexmatch switch in the search criteria, do one of the following:

- Exclude the switch from the MATCH parameter.
- Include the switch in the IGNORE parameter.

See also

[Flexmatch Search of Generic Structures](#)

[Flexmatch Search of Biopolymer Sequence Structures](#)

## Flexmatch Search of Generic Structures

The flexmatch operator allows searching of generic structures. flexmatch can accept a specific or a generic query molecule and finds all generics or specifics in the table which enumerates to exactly the same set of specifics. Fastsearch is not used, instead the pre-screen consists of the minimum and maximum molecule weights and the number of enumerated specifics.

See also

*Biopolymer Searching and Registration* in the *BIOVIA Direct Developers Guide*

GENERIC option in the [flexmatch](#).

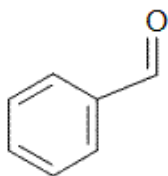
## Flexmatch Search of Biopolymer Sequence Structures

You can search biopolymer sequence structures with the [flexmatch](#) operator.

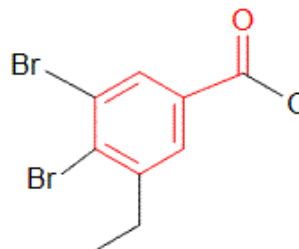
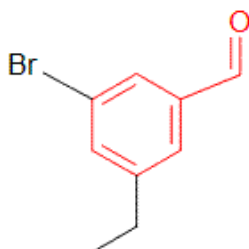
## Substructure Search

A substructure search finds structures that contain your query as a substructure within a larger structure. A substructure is a portion of a larger molecule structure. For example:

A substructure query



Examples of records retrieved



To perform a two-dimensional (2D) substructure search, use the sss operator.

See also

[Substructure search of generic structures](#)

[Substructure search of biopolymer structures](#)

### Substructure Search of Generic Structures

The `sss` operator allows you to search generic structures. `sss` accepts specific structures as the query molecule, and uses Fastsearch to perform a substructure search of both generic and specific structures.

#### See also

*BIOVIA Direct Developers Guide > Using Direct > Searching Generic Structures*

*BIOVIA Direct Developers Guide > Using Direct > Biopolymer Searching and Registration*

[SSS](#)> GENERICS

### Substructure Search of Biopolymer Structures

You can search biopolymer substructures using the `sss` operator.

Because of the potential for a very large number of atoms in a biopolymer, only modified monomer units and cross-linked monomer units are indexed for structure searching. If an `sss` query contains only unmodified and un-cross-linked monomers Direct will use a text search to find all registered biopolymers that have sequence text containing the query sequence text.

For example, the query `leu-leu-leu-leu` would generate the sequence text `LLLL` which would find proteins such as human protein CM036 which has the sequence text:

```
MSEPDTS SGFSGSVENGTFLELFPTSLSTSVDPSSGHL SNVYIYVSIFLSLLAFLLLLLIIALQRLKNIISSSSSYP  
EYPSDAGSSFTNLEVCSISSQRSTFSNLSS
```

#### See also

*BIOVIA Direct Developers Guide > Substructure Searching of Modified and Unmodified Monomers*

### Similarity Search

A similarity search finds structures that are structurally similar to the structure in your query, and returns the degree of similarity between the query and retrieved structures. To perform a similarity search, use the [similar](#) operator. Its ancillary operator [similarity](#) returns a number between 0 and 100 that indicates the degree of similarity between the query structure and the retrieved structure. The higher the value, the more similarity.

The degree of structural similarity is based on the ratio of number of features a stored structure has in common with the query and the total number of features contained in both. For a traditional similarity search, each feature is one of the 960 searchable substructure keys in the database. A searchable key defines a specific structural feature, such as a ring structure or the arrangement of a heteroatom in a molecule. With a fingerprint similarity search, each feature is an Accord or Scitegic fingerprint value. There are 384 Accord fingerprints and a user-defined number of Scitegic fingerprint values. As with traditional similarity search, each fingerprint value defines a specific structural feature. For more information about fingerprints, see "Fingerprint Searching" in the *Direct Administration Guide*.

In contrast to the substructure search, which finds results that are precisely specified by a carefully constructed query structure, similarity search finds a group of structures that are loosely related to the query structure.

### Types of Similarity

The `similar` operator allows you to specify the type of similarity:



- Fingerprint ('fingerprint') - The search uses the predefined Accord or Scitegic fingerprint type. If this option is not specified, the search uses the traditional substructure keys.
- Normal ('normal') - The retrieved molecules can contain the same structural complexity as the query structure. The query structure is neither a substructure nor a superstructure.
- Subsimilar ('sub') - The retrieved molecules can contain more structural complexity than the query structure. They are typically larger than the query structure, which is a substructure. The `similar` operator disregards the attributes of the candidate structure that are not found in the query structure.
- Supersimilar ('super') - The retrieved molecules can contain less structural complexity than the query structure. They are typically smaller than the query structure, which is a superstructure. The `similar` operator disregards the attributes of the query structure that are not found in the candidate structure.

The default is Normal if normal, sub, or super are not specified.

## Molecule Formula Search

A molecule formula search finds records that contain or match the molecule formula that you specify in your query. To find records that contain a molecule formula, use the operator `formula like`. To find records that exactly match a molecule formula, use the operator `formula match`.

To perform a molecule formula search, you specify the atomic symbols and the numbers of atoms in the formula. The format of the formula is:

ATOM(COUNT) ATOM(COUNT) . . . ATOM(COUNT)

where ATOM represents an atomic symbol from the periodic table, and COUNT is an integer or range of integers such as `C(8-9)O2`.

To optimize search performance, include at least one of the atoms C, H, N or O and preferably more than one of these atoms in your query. These are the only atom types which are indexed, and including these atoms can decrease search time.

Your formula query can be:

- An exact formula and no other atoms, such as `C17 H19 N O3`. Use `formula match` for an exact formula search.
- A subformula and any additional atoms, such as `C5 H10`. Use `formula like` for a subformula search.
- A subformula and any additional atoms except a specific atom, such as `C5 H10 N0`, where N0 represents zero nitrogens. Use `formula like` for a subformula search.

Use the operator `formula match` to find records that contain the exact formula in your query. Your exact formula query must include all hydrogens, standard atom symbols, and atom counts. You can use a single space to separate atoms, and capital letters are not required. For example: To find phenol, enter `C6 H6 O`.

Use the operator `formula like` to find records that contain only specific atoms. Your subformula query can include a range of atom counts, and typically excludes the hydrogens. You can use a single space to separate atoms, and capital letters are not required. For example, to find molecules that are sodium salts, enter `Na`. To find molecules that contain twelve carbon atoms, one to two bromine atoms, and two sulfur atoms, enter: `C12 Br(1-2) S2`. To find molecules that contain six carbons, six hydrogens, and any number of other atom types, enter: `C6 H6`.

**Note:** If you enter a single atom within your query but you do not include spaces, you must also enter a value of 1 for the single atom. For example: F1H5.

## Chapter 7:

# Reaction Searches

---

<a href="#">Reaction Flexmatch Search</a>	241
<a href="#">Reaction Substructure Search</a>	242
<a href="#">Reaction Similarity Search</a>	242

### Reaction Flexmatch Search

A flexible match (flexmatch) search finds reactions whose molecules match your query reaction's molecule exactly. You can specify exceptions with any of the flexmatch switches. The flexmatch switches allow you to selectively restrict or relax the criteria that are used to determine whether a structure is an exact match. To perform a reaction flexmatch search, use the `rxnflexmatch` operator.

The `rxnflexmatch` operator requires you to specify flexmatch parameters, such as the following:

```
select rxnmdlnumber
from samplerx_reaction
where rxnflexmatch(
  rctab,
  '/home/user/query.rxn',
  rxnflexmatch-parameters
)=1;
```

where `rxnflexmatch-parameters` is a string that consists of `MATCH` or `IGNORE` parameters. `MATCH` and `IGNORE` parameters allow you to selectively restrict or relax the search criteria based on a query structure. The format of the string `rxnflexmatch-parameters` is:

```
MATCH|IGNORE=FP1[,FP2,FP3,...]
```

where `FP1`, `FP2`, `FP3` are the reaction flexmatch switches.

You can combine more than one flexmatch switch with either the `MATCH` or the `IGNORE` parameter (not both). If you specify more than one flexmatch switch, use a comma (,) or a forward slash (/) to separate them. If you do not specify `MATCH` or `IGNORE`, the cartridge assumes the `MATCH` parameter. For example, the following `rxnflexmatch` parameters are equivalent (Both specify an exact match.):

```
rxnflexmatch(rxn, '/home/user/query.rxn', 'match=all')
rxnflexmatch(rxn, '/home/user/query.rxn', 'all')
```

The `MATCH` parameter allows you to specify features of the retrieved reactions that must match the query. If you want a less restrictive definition of exact match, use `MATCH=` to enable only those switches that you want to apply. Any switches not listed in `MATCH=` are ignored. `MATCH=NONE` is the loosest possible definition. Each switch you add in `MATCH=` tightens the match criteria, and thus generally permits fewer hits.

To enable a flexmatch switch in the search criteria, do one of the following:

- Include the switch in the `MATCH` parameter.
- Exclude the switch from the `IGNORE` parameter.

The `IGNORE` parameter allows you to specify features of the structures that will be ignored in the query. If you want a relatively tight definition of exact match with the switches, use `IGNORE=` to specify only those switches that you want to disable. Any switches not listed in `IGNORE=` are matched.

IGNORE=NONE is the tightest possible definition (It is equivalent to an exact match). Each switch you add in IGNORE= relaxes the match criteria.

**IMPORTANT!** Be careful when using IGNORE, since any switches that you do not explicitly include in the list will be turned on.

To disable a flexmatch switch in the search criteria, do one of the following:

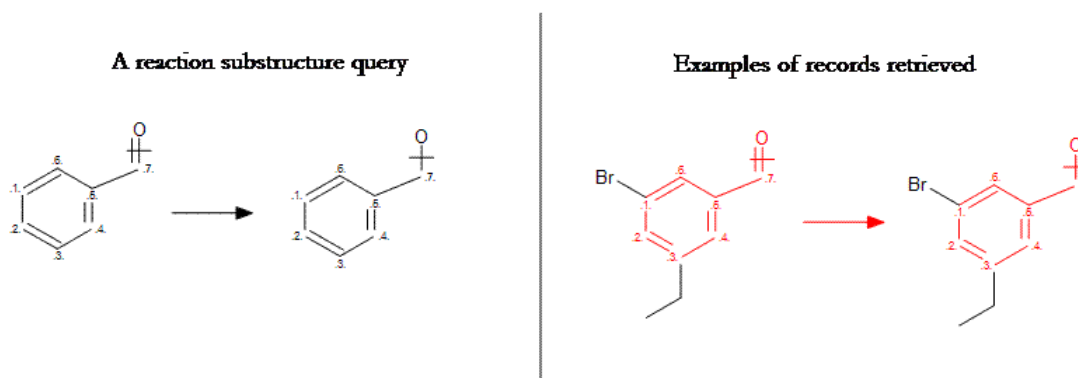
- Exclude the switch from the MATCH parameter.
- Include the switch in the IGNORE parameter.

**See also**

BIOVIA Chemical Representation > Exact Search (Flexmatch)

## Reaction Substructure Search

A reaction substructure search finds reaction records in your database that contain your query as a reaction substructure wholly within a larger reaction. Your reaction substructure query is a two-dimensional representation of a portion of a reaction (a reaction substructure) with mapped atoms and your choice of restrictions on the reacting centers. For example:



To perform a reaction substructure search, use the `rss` operator.

The highlighted portion of the reaction in the example shows the substructure query. Note that the record retrieved does not highlight the query. If you want to highlight the substructure in the records that match the query, use the `rsshighlight` operator.

## Reaction Similarity Search

A reaction similarity search finds reaction records that are similar to your query. You can specify both the type and degree of similarity in your query. To perform a reaction similarity search, use the `rxnsim` operator.

**See also**

[Types of similarity](#)

[Degrees of similarity](#)

## Types of Similarity

The types of similarity:

- Structural - Physical resemblance to the substrate molecules in your query. You specify a structural similarity by specifying degrees of molecule similarity. For more information, see [Degrees of Similarity](#).
- Transformational - Similar reacting centers. You specify a transformational similarity by specifying degrees of reacting center similarity. For more information, see [Degrees of Similarity](#).

In addition, you can also specify one of the following types of similarity:

- Subsimilar ('sub') - The reaction components that are retrieved contain more structural complexity than your query. Your query is a substructure.
- Supersimilar ('super') - The reaction components that are retrieved contain less structural complexity than your query. Your query is a superstructure.
- Normal (neither 'sub' nor 'super') - The reaction components that are retrieved contain the same structural complexity as your query. Your query is not a substructure or a superstructure.

## Degrees of Similarity

The degree of similarity is an arbitrary value between 0 and 100 (these are not actual units). The higher the value, the more similarity. You can specify the degree of:

- Reacting center similarity. A high value finds reactions with nearly identical reacting centers. A lower value results in hits with less related transformations.
- Molecule similarity. A high value finds reactions with nearly identical molecules in the reaction. A lower value causes rxnsim to ignore nonreacting groups in a reaction.

The degree of structural similarity depends on the number of searchable structural keys that a stored reaction component has in common with the query compared to the total number of searchable structural keys BIOVIA sets approximately 960 searchable keys in the database. A searchable key defines a specific structural feature, such as a ring structure or the arrangement of a heteroatom in a molecule.

The degree of transformational similarity depends on number of searchable reacting center keys that a stored reaction has in common with the query compared to the total number of searchable reacting center keys BIOVIA sets approximately 654 searchable reacting center keys in the database.

If the query specified in rxnsim does not contain reacting center features such as bond marks, the reacting center component of similarity is ignored during a search. If the query specified in rxnsim (or if the candidate) does not contain reacting center features, rxnctrsim returns NULL. This indicates that the similarity was ignored during the search. For example, if the specified similarity threshold is 80%, you should only see hits of 80% or greater, or hits where the rxnctrsim value is NULL (ignored).

## Chapter 8:

# Specifying the Query Structure

---

This section explains the possible formats for the query operand for the Direct search operators `flexmatch`, `similar`, and `sss`. When you specify the query structure for these operators, you can use one of the following formats:

<a href="#">Molfile</a> .....	244
<a href="#">Chimestring</a> .....	245
<a href="#">BLOB (Binary Large Object)</a> .....	246
<a href="#">CLOB (Character Large Object)</a> .....	247
<a href="#">HELM String</a> .....	247
<a href="#">SMILES String</a> .....	247

## Molfile

A molfile uses the `.mol` file extension, and contains information about a single molecule structure.

Use one of the following to specify the molfile representation of a query structure:

Molfile representation	Example
filepath of a molfile	<pre>select cdbregno from sample2d where sss(ctab, '/home/johnx/mymol.mol')=1;</pre>
molfile	<pre>select cdbregno,        molname from mysample where flexmatch(        ctab,        (select molfile(ctab)         from sample2d         where cdbregno=1),        'all'       )=1;</pre>
molfile_string	<pre>select cdbregno,        molname from mysample where flexmatch(        ctab,        (select molfile_string(ctab)         from sample2d         where cdbregno=1),        'all'       )=1;</pre>
molfile_string_seg	<pre>select cdbregno,        molname</pre>

Molfile representation	Example
	<pre>from mysample where flexmatch(   ctab,   (select molfile_string_seg(ctab, 1, 3000)    from sample2d where cdbregno=1),   'all' )=1;</pre>
molfile-string	<pre>select cdbregno from sample2d where flexmatch(ctab, :s1, 'all')=1;</pre> <p><b>Note:</b> s1 is a bound variable, up to 4000 characters long, that contains the structure in a molfile. The string includes the embedded newline characters in a molfile. See Oracle SQL documentation for details about binding variables in a SQL statement.</p>

**Notes:**

- For large structures, use `molfile` or `molfile_string_seg` instead of `molfile_string`. `molfile_string` returns only the first 4000 characters of the molfile. If the molfile exceeds 4000 characters, `molfile_string` returns an incomplete value.
- If you use an embedded `SELECT` query to represent the query structure, the embedded query must evaluate to exactly one query structure. In the preceding examples, the embedded `SELECT` queries return only one molfile structure for a specific `cdbregno`.

**See also**

[Retrieving Molfile Structures.](#)

[CLOB \(Character Large Object\).](#)

## Chimestring

A Chime string is an encrypted string that represents a chemical structure.

Use one of the following to specify the Chime string representation of a query structure:

Chime string representation	Example
molchime	<pre>select cdbregno,        molfmla(ctab) from mysample where sss(   ctab,   (select molchime(ctab)    from sample2d    where cdbregno=356) )=1;</pre>
chime_string	<pre>select cdbregno,</pre>

Chime string representation	Example
	<pre> molformula(ctab) from mysample where sss(   ctab,   (select chime_string(ctab)    from sample2d    where cdbregno=1) )=1; </pre>
chime_string_seg	<pre> select cdbregno,   molformula(ctab) from mysample where sss(   ctab,   (select chime_string_seg(ctab, 1, 3000)    from sample2d    where cdbregno=120) )=1; </pre>
chime-string	<pre> select cdbregno from sample2d where flexmatch(ctab, :s1, 'all')=1; </pre> <p><b>Note:</b> s1 is a bound variable, up to 4000 characters long, that contains the Chime string. See Oracle SQL documentation for details about binding variables in a SQL statement.</p>

**Notes:**

- For large structures, use `molchime` or `chime_string_seg` instead of `chime_string`. `chime_string` returns only the first 4000 characters of the Chime string. If the Chime string exceeds 4000 characters, `chime_string` returns an incomplete value. For more examples, see [Retrieving Chime Structures](#).
- If you use an embedded `SELECT` query to represent the query structure, the embedded query must evaluate to exactly one query structure. In the preceding examples, the embedded `SELECT` queries return only one Chime structure for a specific `cdbregno`.

**See also**

[CLOB \(Character Large Object\)](#)

**BLOB (Binary Large Object)**

You can use the BLOB field from a table that stores the binary chemical structures to represent the query structure. This BLOB field is typically called CTAB. Use an embedded query that retrieves the CTAB field to represent the query structure in your search. For example:

```

select cdbregno
from mysample
where flexmatch(
  ctab,

```



```
(select ctab
  from sample2d
  where cdbregno=1),
'all'
)=1;
```

**Note:** If you use an embedded SELECT query to represent the query structure, the embedded query must evaluate to exactly one query structure. In the preceding example, the embedded SELECT query returns only one ctab for a specific cdbregno.

## CLOB (Character Large Object)

You can use the operators `molfile`, `molchime`, or `readmol` to represent the query structure. They return a molfile or Chime representation of the structure, which is of type CLOB. Use one of these operators to represent large structures that exceed 4000 characters. To see examples, see [Molfile](#) or [Chime string](#) representation of a query.

## HELM String

HELM, Hierarchical Editing Language for Macromolecules, is a format that can represent natural and modified biological sequences including 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.

## SMILES String

SMILES<sup>TM</sup>(Simplified Molecular Input Line Entry System) is a language that represents molecules by using ASCII character strings that specify atoms and bonds. You can use a SMILES string to specify a query structure. For more information about SMILES, see <https://www.daylight.com/smiles/index.html>.

# Appendix A:

## RDCAPPS Procedures

---

This section contains information about the procedures in the PL/SQL package RDCAPPS. RDCAPPS contains a set of useful procedures that:

- Reads reactions and one data field from a reaction RDfile, and insert them into a reaction table. See [ReadRxnRDF](#).
- Reads molecules and one data field from a molecule RDfile, and insert them into a molecule table. See [ReadMolRDF](#).
- Creates a sample trigger on a reaction column in a reaction table. The trigger automatically inserts the component molecules in a reaction into a molecule table. See [MakeMolXrefTrigger](#).

<a href="#">Using the RDCAPPS Procedures</a>	248
<a href="#">ReadRxnRDF</a>	248
<a href="#">ReadMolRDF</a>	250
<a href="#">MakeMolXrefTrigger</a>	252

### Using the RDCAPPS Procedures

To use the RDCAPPS procedures, execute these procedures on a SQL\*Plus command line. But before you can use one of these procedures, you must create the RDCAPPS package in your SQL\*Plus session. To create the RDCAPPS package, execute the following SQL script:

```
SQL> @/opt/BIOVIA/direct/examples/rdcapps.sql
```

where /opt/BIOVIA/direct is the location of your Direct installation.

### ReadRxnRDF

ReadRxnRDF reads reactions and one data field in a reaction RDfile, and inserts them into a reaction table.

A reaction RDfile is an BIOVIA file type that uses the .rdf file extension. RDfiles have a hierarchical-file structure that stores reactions and associated data that are exported from the top-level of a database hierarchy. RDfiles also store text and numeric data that is exported from lower levels of a hierarchical database.

ReadRxnRDF requires that the RDfile must contain:

- Reactions (\$RFMT entries), and
- Either one data field which uniquely identifies each reaction (\$DTYPE/\$DATUM entries), or one internal reaction registration number (\$RIREG entries on the \$RFMT line)

#### Syntax

```
ReadRxnRDF(filename, xrgfld, rxntab, rxncol, xrgcol)
```

Parameter	Description
<i>filename</i>	The full path and name of the RDfile.
<i>xrgfld</i>	Specifies what data value to be inserted into the table along with the reaction.

Parameter	Description
	<p>xrgfld can be one of the following:</p> <ul style="list-style-type: none"> <li>■ The name of a leaf-level field which appears in the RDfile</li> <li>■ The keyword 'RIREG' (or 'rireg'), which causes ReadRxnRDF to extract each reaction's internal registry number from the reaction's start-of-record (\$RFMT) line.</li> </ul>
<i>rxntab</i>	The name of the reaction table. This table contains a BLOB column which contains the reactions.
<i>rxncol</i>	The name of the BLOB column that contains the reactions. The rxncol column belongs to the specified rxntab table.
<i>xrgcol</i>	The name of the column which is the destination of the data value specified by the xrgfld parameter.

### Usage

This procedure must be executed from SQL\*Plus:

```
call rdcapps.readrxnrdp(
    filename,
    xrgfld,
    rxntab,
    rxncol,
    xrgcol);
```

### Example

The following is a portion of a sample RDfile, /home/user/rdfiles/rxnsfile.rdf. The examples in this section will use this sample RDfile. "..." indicates deleted portions of the RDfile:

```
$RFMT $RIREG 33
$RXN
...
M END
$DTYPE RXN:VARIATION(1):VARIATION_NO
$DATUM 1
$DTYPE RXN:VARIATION(1):MDLNUMBER
$DATUM RXCI97000001
...
$DTYPE RXN:VARIATION(1):VARIATION_NO
$DATUM 2
$DTYPE RXN:VARIATION(1):MDLNUMBER
$DATUM RXCI97000002
```

The following example stores reactions and their internal regno from the RDfile into a new reaction table.

```
SQL> create table rxntable (rxnregno number(6), rxn blob);
SQL> call rdcapps.readrxnrdp(
    '/home/user/rdfiles/rxnsfile.rdf',
    'rireg',
    'rxntable',
    'rctab',
    'rxnregno');
```

Using the sample RDfile `rxnsfile.rdf`, the preceding example will insert the reaction into `rxntable`, with the value of 33 for `rxnregno`.

The following example stores reactions and their first VARIATION's MDLNUMBER from a RDFile into a new reaction table.

```
SQL> create table rxntable (extreg varchar2(12), rxn blob);
SQL> call rdcapps.readrxnrdf(
    '/home/user/rdf/rxnsfile.rdf',
    'mdlnumber',
    'rxntable',
    'rctab',
    'extreg');
```

Using the sample RDfile `rxnsfile.rdf`, the preceding example will insert the reaction into `rxntable`, with the value of `RXCI97000001` for `extreg`. In this example, the second MDLNUMBER record will be ignored because `ReadRxnRdf` uses the first value it encounters.

### Comments

RDfiles that are written using the ISIS/Host Command Line Interface, and that use a FIELDS file containing the single line:

```
rxnstructure *rxn
```

contain only \$RFMT and \$RIREG entries. BIOVIA databases do not contain a unique identifier for each reaction. Instead of a unique identifier for each reaction, there is a unique identifier for each variation. For these databases, the internal reaction registration number (`regno`) should be used as the identifier.

Use `ReadRxnRDF` to migrate reactions from an ISIS/Host reaction database into a reaction table. For more information, see *Converting Reactions into a Reaction Table* in the *BIOVIA Direct Developers Guide*.

## ReadMoIRDF

`ReadMoIRDF` reads molecule structures and one data field in a molecule RDfile, and inserts them into a molecule table.

A molecule RDfile is an BIOVIA file type that uses the `.rdf` file extension. RDfiles have a hierarchical-file structure that stores molecules and associated data that are exported from the top-level of a database hierarchy. RDfiles also store text and numeric data that is exported from lower levels of a hierarchical database.

`ReadMoIRDF` requires that the RDfile must contain:

- Molecule structures (\$MFMT entries), and
- Either one data field which uniquely identifies each reaction (\$DTYPE/\$DATUM entries), or one internal molecule registration number (\$MIREG entries on the \$MFMT line)

### Syntax

```
ReadMoIRDF(filename, xrgfld, moltab, molcol, xrgcol)
```

Parameter	Description
<i>filename</i>	The full path and name of the RDfile.
<i>xrgfld</i>	Specifies what data value to be inserted into the table along with the molecule.

Parameter	Description
	<p><code>xrgfld</code> can be one of the following:</p> <ul style="list-style-type: none"> <li>■ The name of a leaf-level field which appears in the RDfile</li> <li>■ The keyword 'MIREG' (or 'mireg'), which causes <code>ReadMolRDF</code> to extract each molecule's internal registry number from the molecule's start-of-record (\$MFMT) line.</li> </ul>
<code>moltab</code>	The name of the molecule table.
<code>molcol</code>	The name of the BLOB column that contains the molecules. This column is typically named CTAB.
<code>xrgcol</code>	The name of the column which is the destination of the data value specified by the <code>xrgfld</code> parameter.

### Usage

This procedure must be executed from SQL\*Plus:

```
call rdcapps.readmolrdf(
    filename,
    xrgfld,
    moltab,
    molcol,
    xrgcol);
```

### Example

The following is a portion of a sample RDfile, `/home/user/rdfiles/molfile.rdf`. The examples in this section will use this sample RDfile. "..." indicates deleted portions of the RDfile:

```
...
$MFMT $MIREG 33
IH(4.0) 060501121810.002780.0000033
...
M END
$DTYPE MOL:SYMBOL(1):LINE_NO
$DATUM 1
$DTYPE MOL:SYMBOL(1):SYMBOL
$DATUM ZrCl2(Cp)2
$DTYPE MOL:SYMBOL(2):LINE_NO
$DATUM 2
$DTYPE MOL:SYMBOL(2):SYMBOL
$DATUM ZrCp2Cl2
...
```

The following example stores molecules and their internal regno from the RDfile into a new molecule table. From SQL\*Plus:

```
create table moltable (ctab blob, molregno number(9));
call rdcapps.readmolrdf(
    '/home/user/rdfiles/molfile.rdf',
    'mireg',
    'moltable',
    'ctab',
    'molregno');
```

Using the sample RFile `mol.sfile.rdf`, the preceding example will insert the molecule structure into `moltable`, with the value of 33 for `molregno`.

The following example stores molecules and their first `SYMBOL` value from a RFile into a new table. From SQL\*Plus:

```
create table moltable(first_symbol varchar2(200), ctab blob);
call rdcapps.readmolrdf(
    '/home/user/rdfiles/mol.sfile.rdf',
    'symbol',
    'moltable',
    'ctab',
    'first_symbol');
```

Using the sample RFile `mol.sfile.rdf`, the preceding example will insert the molecule structure into `moltable`, with the value of `ZrCl2(Cp)2` for `extreg`. In this example, the second `SYMBOL` record will be ignored because `ReadMolRdf` uses the first value it encounters.

### Comments

RFiles that are written using the ISIS/Host Command Line Interface, and that use a `FIELDS` file containing the single line:

```
molstructure *structure
```

contain only `$MFM`T and `$MIREG` entries.

## MakeMolXrefTrigger

Creates a sample trigger on a reaction table. When a user inserts or updates a reaction into the reaction table, the sample trigger:

- Automatically inserts the component molecules in a reaction into a molecule table
- Maintains a cross-reference table that correlates each reaction in the reaction table with its molecules in the molecule table. If the cross-reference table does not yet exist, `MakeMolXrefTrigger` creates it.

You can use the sample trigger as a basis for creating your own trigger that more closely meets your business requirements. To customize the trigger, examine the code for the `MakeMolXrefTrigger` procedure in `examples/rdcapps.sql`.

The following lists the Requirements and the Trigger Operations for the reaction table trigger that `MakeMolXrefTrigger` creates.

### Requirements

Verify that the following requirement is met before using `MakeMolXrefTrigger`:

- The trigger on the reaction table must not yet exist. The name of the trigger is specified in the optional `InTriggerName` parameter, or the default name assigned by the `MakeMolXrefTrigger` procedure. See description of the `InTriggerName` parameter. For example, to determine if the trigger already exists:

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

where `MY_RXN_TABLE` is the name of your reaction table. If the trigger already exists, drop it prior to running `MakeMolXrefTrigger`. For example, to drop the trigger:

```
drop trigger my_rxn_table_trigger;
```

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

- The molecule table must contain a column named `CDBREGNO`, and the values in this column must be automatically generated. The reaction table trigger assumes that the `CDBREGNO` number is automatically generated on the molecule table. If the `CDBREGNO` number is not automatically generated on the molecule table, the reaction table trigger will insert `NULL` `CDBREGNO` values.

If you are creating a new molecule table, you must create a trigger on the molecule table that automatically generates `CDBREGNO` numbers. The following example creates a molecule table, its domain index, and a trigger that automatically generates `CDBREGNO` numbers on insert:

```
create table mol_table (cdbregno number(9), ctab blob);
create index mol_table_ix on mol_table (ctab)
  indextype is c$direct2021.mxixmdl;
create sequence mol_table_sequence start with 1;
create trigger mol_table_trigger
  before insert on mol_table
  for each row
  declare
  begin
    if ((inserting and :new.cdbregno is not null) or
        (updating('CDBREGNO'))) then
      raise_application_error(-20100, 'CDBREGNO is not updatable');
    end if;
    if (inserting) then
      select mol_table_sequence.nextval into :new.cdbregno from dual;
    end if;
  end;
```

## Trigger Operations

`MakeMolXrefTrigger` creates a sample trigger that performs the following operations on:

- **Insert**

The trigger decomposes the reaction into its component reactant and product molecules. For each molecule, the trigger performs a flexmatch query against the molecule table. If a molecule is found, the trigger retrieves its `CDBREGNO` number. If a molecule is not found, the trigger inserts the molecule and its newly assigned `CDBREGNO` number into the molecule table. Then the trigger inserts the following values into the cross-reference table:

- The primary key value of the reaction row
- The `CDBREGNO` number of the molecule
- The component type ('R' for reactant, 'P' for product)
- The component index number ranging from 1 to the number of reactants or products

- **Update**

The trigger deletes the old reaction value using the primary key value from the cross-reference table, and inserts the new reaction value.

- **Delete**

Using the primary key value on the reaction table row, the trigger deletes all rows in the cross-reference table that contain the same primary key value. The trigger does not delete molecules from the molecule table.

**Note:** You can use the sample trigger as a basis for creating your own trigger that more closely meets your business requirements. To customize the trigger, examine the code for the `MakeMolXrefTrigger` procedure in `examples/rdcapps.sql`.

## Syntax

```
MakeMolXrefTrigger(InRxnTableName, InRxnTablePKCol, InMolTableName)
MakeMolXrefTrigger(InRxnTableName, InRxnTablePKCol, InMolTableName,
                    InRxnTableRxnCol, InRxnTablePKType, InMolTableMolCol,
                    InFlexmatchSwitches, InTriggerName, InXRefTableName)
```

Parameter	Description
<i>InRxnTableName</i>	The name of the reaction table. The reaction table contains a BLOB column which contains the reactions. <i>InRxnTableName</i> is required.
<i>InRxnTablePKCol</i>	The name of the primary key column in the reaction table. <i>InRxnTablePKCol</i> is required.
<i>InMolTableName</i>	The name of the molecule table that will receive component molecules during reaction insert and update operations. <i>InMolTableName</i> is required.
<i>InRxnTableRxnCol</i>	The name of the BLOB column that holds the reactions. <i>InRxnTableRxnCol</i> belongs to the specified table <i>InRxnTableName</i> . If you do not specify <i>InRxnTableRxnCol</i> , or if you specify NULL: <ul style="list-style-type: none"> <li>■ If <i>InRxnTableName</i> contains only one BLOB column, this column is used.</li> <li>■ If <i>InRxnTableName</i> contains more than one BLOB column, <code>MakeMolXrefTrigger</code> returns an exception. You must rerun <code>MakeMolXrefTrigger</code>, and must specify the column name in <i>InRxnTableRxnCol</i>.</li> </ul>
<i>InRxnTablePKType</i>	The name of the column which is the destination of the data value specified by the <code>xrgfld</code> parameter.
<i>InRxnTablePKType</i>	The Oracle datatype of the primary key column that you specified in the <i>InRxnTablePKCol</i> parameter. <i>InRxnTablePKType</i> is only used when <code>MakeMolXrefTrigger</code> creates the cross-reference table. If you do not specify <i>InRxnTablePKType</i> , or if you specify NULL, <code>MakeMolXrefTrigger</code> determines the datatype from the data dictionary table. <code>MakeMolXrefTrigger</code> produces an error if it fails to determine the datatype for less common datatypes. In this situation, you must rerun <code>MakeMolXrefTrigger</code> with a specified <i>InRxnTablePKType</i> .



Parameter	Description
<i>InMolTableMolCol</i>	<p>The name of the BLOB column that contains the molecules. This column belongs to the specified <i>InMolTableName</i>.</p> <p>If you do not specify <i>InMolTableMolCol</i>, or if you specify NULL:</p> <ul style="list-style-type: none"> <li>■ If <i>InMolTableName</i> contains only one BLOB column, this column is used.</li> <li>■ If <i>InMolTableName</i> contains more than one BLOB column, <i>MakeMolXrefTrigger</i> fails returns an exception. You must rerun <i>MakeMolXrefTrigger</i>, and must specify the column name in <i>InMolTableMolCol</i>.</li> </ul>
<i>InFlexmatchSwitches</i>	<p>The flexmatch switches to be used when the trigger executes a molecule search for a reaction's component molecules.</p> <p>If you do not specify <i>InFlexmatchSwitches</i>, or if you specify NULL, <i>MakeMolXrefTrigger</i> uses 'match=all'.</p>
<i>InTriggerName</i>	<p>The name of the trigger to be created.</p> <p>If you do not specify <i>InTriggerName</i>, or if you specify NULL, <i>MakeMolXrefTrigger</i> uses the first of the following names which fits 30 characters:</p> <p><i>InRxnTableName</i> + '_TRIGGER'</p> <p><i>InRxnTableName</i> + '_TRIG'</p> <p>SUBSTR(<i>InRxnTableName</i>,1,26) + '_TRIG'</p> <p><i>MakeMolXrefTrigger</i> fails if an object with this name already exists. In this situation, you must drop the trigger, and rerun <i>MakeMolXrefTrigger</i>.</p>
<i>InXRefTableName</i>	<p>The name of the cross-reference table that correlates each reaction in the reaction table with its molecules in the molecule table. If you do not specify <i>InXRefTableName</i>, or if you specify NULL, <i>MakeMolXrefTrigger</i> uses the first of the following names which fits 30 characters:</p> <p><i>InRxnTableName</i> + '_MOLXREF'</p> <p><i>InRxnTableName</i> + '_XREF'</p> <p><i>InRxnTableName</i> + '_XRF'</p> <p>SUBSTR(<i>InRxnTableName</i>,1,26) + '_XRF'</p> <p>If the table does not exist, <i>MakeMolXrefTrigger</i> creates it using the following definition:</p> <pre>CREATE TABLE InXRefTableName (   InRxnTablePKCol      InRxnTablePKType,   COMPTYPE              CHAR(1),   COMPIDX               NUMBER(3),   CDBREGNO              NUMBER(10));</pre> <pre>CREATE INDEX &lt;SUBSTR(InXRefTableName,1,26)  '_IX1'&gt; ON InXRefTableName (InRxnTablePKCol);</pre>

Parameter	Description
	<p>CREATE INDEX  &lt;SUBSTR(InXRefTableName,1,26)  '_IX2'&gt;  ON InXRefTableName (CDBREGNO);</p> <p>If the table or its indices require specific storage requirements, you must create them before using MakeMolXrefTrigger. The table you create must use specified the column names and datatypes in the preceding definitions.</p>

## Usage

This procedure must be executed from SQL\*Plus:

```
call rdcapps.makemolxreftrigger(
    InRxnTableName,
    InRxnTablePKCol,
    InMolTableName);

call rdcapps.makemolxreftrigger(
    InRxnTableName,
    InRxnTablePKCol,
    InMolTableName,
    InRxnTableRxnCol,
    InRxnTablePKType,
    InMolTableMolCol,
    InFlexmatchSwitches,
    InTriggerName,
    InXRefTableName);
```

## Example

The following steps show what happens to a new molecule table and a new reaction table after an insert operation to the reaction table that has the sample trigger:

1. Create a new molecule table. From SQL\*Plus: `create table moltable(cdbregno number (9), ctab blob);`
2. Create a new reaction table. From SQL\*Plus: `create table rxntable (rxnregno number (6), rxn blob);`
3. Create the sample trigger on the reaction table. From SQL\*Plus: `call rdcapps.makemolxreftrigger('rxntable', 'rxnregno', 'molview');`
4. Verify that the molecule and cross-reference tables are empty. The following SQL\*Plus commands should return zero for both tables: `select count(*) from moltable;` `select count(*) from rxntable_molxref;`
5. Insert a row to the reaction table. From SQL\*Plus: `insert into rxntable values (1,rxn ('c:\rxns\file.rxn'));`
6. Verify that the molecule and cross-references tables contain molecules and references to the new reaction in the reaction table. The following SQL\*Plus commands should return non-zero for both tables:  
`select count(*) from moltable;`  
`select count(*) from rxntable_molxref;`

## Comments

- If you only specify the required three parameters, `MakeMolXrefTrigger` uses the NULL (default) value for all the optional parameters. If you specify at least one optional value, you must also specify the rest of the optional values. Use NULL if you want to use the default value for an optional parameter.
- Use `MakeMolXrefTrigger` to create a sample trigger that maintains an association between reactions in a reaction table and the component molecules in a molecule table.
- If you use the sample trigger that is created by `MakeMolXrefTrigger`, users must only insert into the molecule table. `MakeMolXrefTrigger` creates a trigger on the reaction table, but not on the molecule table. If you use the sample trigger that maintains a cross-reference table between the reaction table and a molecule table, a user must not update or delete molecules from the molecule table. The users must only insert into the molecule table.
- For multiple reaction tables that share the same molecule table, execute `MakeMolXrefTrigger` for each reaction table.
- To customize the sample trigger on the reaction table, examine the code for the `MakeMolXrefTrigger` procedure in `examples/rdcapps.sql`.