Daylight Chemical Information Systems, Inc.
120 Vantis - Suite 550 - Aliso Viejo, CA 92656
tel +1 949-831-9990 - fax +1 949-831-9902 - www.daylight.com

Daylight Chemistry Cartridge

Table of Contents

Table of Contents

Daylight Chemistry Cartridge

Daylight Version 4.95
Release Date 08/01/11

DAYLIGHT Chemical Information Systems, Inc. Aliso Viejo, CA USA

9. Function/Option Summary

1. Installation

1.1 Detailed Installation Instructions

A quick overview of the installation process can be found in the Daylight Installation Guide. This section is intended to give detailed installation instructions. For the purpose of consistency in the examples, this document will show actual commands and changes that were performed on a Solaris machine named 'snowcrash'. Also installed on this machine are Oracle 9.2 with an instance name of 'snow' and installed in the location '/usr/local/oracle/product/9.2.0', defined as ORACLE_HOME and Daylight owned by the user 'daylight', group 'dba', and installed in the directory /usr/local/daylight/v491, defined as DY_ROOT. When an alternate platform needs something done another way, that platform will be referenced and then a relevant alternate example given.

1.1.1 Installation of the Daylight Tar File

If you do not have Daylight already installed, unpack the Daylight tar distribution that you have received from Daylight and follow the relevant installation instructions. Daylight is usually located in /usr/local/daylight/v491, where 'v491' is the version of the Daylight software installed, and is owned by an account 'thor' or 'daylight'. As stated before this location is refered to as DY_ROOT. It might be useful to have the owning account of the daylight software, also be a member of the 'dba' group with Oracle. The Daylight distribution contains two directories that are relevant to the daycart install. 'dcischem', which has the installation SQL scripts and 'lib' which has the required shared object library for Oracle.

1.1.2 Connecting to Oracle as a DBA User

In this section the following commands must be performed by an Oracle user that has dba privileges. Once you are logged in as the owning Oracle account you can login to oracle as a dba with the following:

```
$ sqlplus "/ as sysdba"

SQL*Plus: Release 9.2.0.1.0 - Production on Wed Sep 18 09:53:01 2002

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Connected to:
```

```
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL>
```

Now that we are logged in as a DBA we can continue.

1.1.3 Space Creation and Creating the Daycart User in Oracle.

It is recommended by Oracle that the 'SYSTEM' tablespace not be used for user code. Decide which tablespace to use to store the cartridge definitions and create or enlarge that tablespace. The cartridge code only requires about 2MB of space, so size isn't a huge issue. In this example, we are creating a tablespace named 'daylight_ts' and giving that tablespace 100 megabytes of space.

```
create tablespace daylight_ts
   datafile '/data/d01/oracle/oradata/snow/daylight_01.dbf'
   size 100M;
```

Now create the user 'c$dcischem' and grant necessary privileges. Execute the following commands in your DBA session:

```
create user c$dcischem
   identified by
      secret
   default tablespace
      daylight_ts
   temporary tablespace
      temp;

grant dba to c$dcischem;
```

Alternately, rather than granting DBA privilege to c$dcischem, one can grant the minimal required privileges:

```
grant
   resource,
   connect,
   create operator,
   create indextype,
   create library,
   create procedure,
   create role,
   create type,
   create public synonym
to
   c$dcischem;

grant
   create table
to
   c$dcischem
with admin option;
```

Exit the DBA session, and edit the file 'create.sql'. Modify the lines that contain the following sql command:

```
create or replace library
   c$dcischemlib
```

```
as
   '/usr/local/daylight/v491/lib/ddlib.so.10';
```

and substitute the absolute path to the appropriate cartridge shared object library. The file ddlib.so is usually found in the $DY_ROOT/lib directory within the normal Daylight distribution.

Daylight ships multiple versions of ddlib.so in each release. One or more of the following files will be found in your Daylight distribution (depending on the operating system):

```
  $DY_ROOT/lib/ddlib.so.10    32-bit ddlib.so for Oracle 10g & 10gR2.
  $DY_ROOT/lib/ddlib.so.11    32-bit ddlib.so for Oracle 11g & 11gR2.
  $DY_ROOT/lib64/ddlib.so.10  64-bit ddlib.so for Oracle 10g & 10gR2.
  $DY_ROOT/lib64/ddlib.so.11  64-bit ddlib.so for Oracle 11g & 11gR2.
```

Also, remove the section of code before the 'create or replace library' command. These 'echo' commands are in place as a reminder for users who neglect to edit this file before trying to run it.

Execute the script 'create.sql'.

```
$ sqlplus 'c$dcischem/secret' @create.sql
```

The output will consist of quite a few notes and messages. One should not see any error messages from sqlplus. If one gets a message about the COMPATIBLE parameter being set incorrectly, see the troubleshooting section below before proceeding further. A sample of the output follows:

```
$ sqlplus 'c$dcischem /secret' @create

SQL*Plus: Release 9.2.0.1.0 - Production on Wed Sep 18 11:28:38 2002

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

Connected.

Library created.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

Role created.

Grant succeeded.
```

[many lines of text omitted]

```
Synonym created.

#########NOTE: Public synonyms "matches" and "contains" #########
######### clash with InterMedia cartridge names if #########
######### InterMedia is installed. Please see the #########
######### daycart manual and FAQ for details. #########
```

```
PL/SQL procedure successfully completed.

Grant succeeded.
```

[many lines of text omitted]

```
Grant succeeded.

Grant succeeded.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
$
```

If you already have a daycart license, you can put the insert statement for c$dcischem.license into the create.sql file after the creation of the table c$dcischem.license. In this example install, we will do it separately, after the create.sql script has run. If not, you'll need to contact Daylight in order to get a license key. We'll need the unique identifier from your machine. If you have installed the daylight tools and thus have testlicense installed you can get it from the command line:

```
$ testlicense -i
   cpu type: i686
   cpu idno: 2341395969
```

alternatively, in oracle's sqlplus once the cartridge is installed, one can execute the following command:

```
$ sqlplus 'c$dcischem /secret'

SQL*Plus: Release 9.2.0.1.0 - Production on Wed Sep 18 08:34:44 2002
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL> select ddpackage.finfo('hostid') from dual;

DDPACKAGE.FINFO('HOSTID')
--------------------
2341395969
```

E-mail the this identifier to Daylight, and we'll e-mail back the required license key, which would look like this (with your data substituted, of course):

```
insert into c$dcischem.license values ('daycart',
  '1d96f3c5d4305d57c0b4bf54eb69111c',
  to_date('01-01-05', 'DD-MM-YY'),
  'Daylight CIS',
  'Santa Fe Research Office',
  '441 Greg Avenue, Santa Fe, NM 87501, USA');
```

NOTE: The license key is not required in order to run the installation scripts. The license key can be added after installation is complete. However, once the cartridge is installed it will not successfully run without the license key.

1.1.4 Configuration of the Oracle Listener

The cartridge uses the 'extproc' functionality on the server within Oracle. The Oracle server contacts the network listener when it wants to run external procedures like the Daylight cartridge. Hence, the listener must always be running in order to use the cartridge.

NOTE: The Oracle Installer will start and configure the listener if you do a full default Oracle installation, however the listener will not be configured to restart automatically after a reboot. It is best to verify the configuration and state of the listener at this point, however don't be suprised if the listener configuration is already correct.

1.1.5 Checking the Listener

The first thing to do is check to see that the listener is working correctly. Using the instance name in the connect string tests the SQLNet listener:

```
$ tnsping snow

TNS Ping Utility for Solaris: Version 9.2.0.1.0 - Production on 28-MAY-03 21:20:25

(c) Copyright 1997 Oracle Corporation.  All rights reserved.

Attempting to contact (ADDRESS=(PROTOCOL=TCP)(HOST=snowcrash)(PORT=1521))
TNS-12541: TNS:no listener
```

If one gets "OK" then the listener is running. In the above example it is not running.

```
$ lsnrctl status

LSNRCTL for Solaris: Version 9.2.0.1.0 - Production on 18-SEP-2002 13:05:00

Copyright (c) 1991, 2002, Oracle Corporation. All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC)))
TNS-12541: TNS:no listener
 TNS-12560: TNS:protocol adapter error
 TNS-00511: No listener
 Solaris Error: 146: Connection refused
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST= snowcrash)(PORT=1521)))
TNS-12541: TNS:no listener
 TNS-12560: TNS:protocol adapter error
 TNS-00511: No listener
 Solaris Error: 146: Connection refused
```

This confirms that the listener isn't started.

```
$ lsnrctl start

LSNRCTL for Solaris: Version 9.2.0.1.0 - Production on 18-SEP-2002 13:05:10

Copyright (c) 1991, 2002, Oracle Corporation. All rights reserved.

Starting /usr/local/oracle/product/9.2.0/bin/tnslsnr: please wait...

TNSLSNR for Solaris: Version 9.2.0.1.0 - Production
System parameter file is /usr/local/oracle/product/9.2.0/network/admin/listener.ora
Log messages written to /usr/local/oracle/product/9.2.0/network/log/listener.log
```

```
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc )(KEY=EXTPROC)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp )(HOST=snowcrash)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC)))
STATUS of the LISTENER
------------------------
Alias LISTENER
Version TNSLSNR for Solaris: Version 9.2.0.1.0 - Production
Start Date 18-SEP-2002 13:05:10
Uptime 0 days 0 hr. 0 min. 0 sec
Trace Level off
Security OFF
SNMP  OFF
Listener Parameter File /usr/local/oracle/product/9.2.0/network/admin/listener.ora
Listener Log File /usr/local/oracle/product/9.2.0/network/log/listener.log
Listening Endpoints Summary...
 (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC)))
 (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=snowcrash)(PORT=1521)))
Services Summary...
Service "PLSExtProc" has 1 instance(s).
 Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "snow" has 1 instance(s).
 Instance "snow", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

Now retry the tnsping. It should return "OK". This verifies that the listener is functional.

1.1.6 Modification of the Listener's Configuration Files

There are two ways to modify the configuration for the listener. One can either edit the configuration files directly or use the network configuration assistant program, which provides a GUI front end to the configuration. This section will describe the file entries.

First change directory to where the oracle installation's files are for the listener:

```
$ cd $ORACLE_HOME/network/admin
```

The three files that are relevant are listener.ora, tnsnames.ora, sqlnet.ora.

Start with sqlnet.ora. This file need not be edited however if the default domain is used then one must change the name of the EXTPROC service in tnsnames.ora.

```
$ more sqlnet.ora
# SQLNET.ORA Network Configuration File: /usr/local/oracle/product/9.2.0/network/admin/sqlnet.o
# Generated by Oracle configuration tools.

NAMES.DIRECTORY_PATH= (TNSNAMES, ONAMES, HOSTNAME)
```

If one is using fully-qualified domains on will find a line like:

```
NAMES.DEFAULT_DOMAIN = daylight.com
```

If this line is present, note the domain name for use in the tnsnames.ora file.

Here is an example tnsnames.ora file on snowcrash. If full domains are being used the service name for EXTPROC_CONNECTION_DATA must be changed to: EXTPROC_CONNECTION_DATA.DAYLIGHT.COM (using your default domain).

```
$ more tnsnames.ora
# TNSNAMES.ORA Network Configuration File: /usr/local/oracle/product/9.2.0/network/admin/tnsnam
# Generated by Oracle configuration tools.

<< Other entries omitted >>

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
       (PRESENTATION = RO)
    )
  )

SNOW =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = snowcrash)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = snow)
    )
  )
```

Here is a sample listener.ora file:

```
$ more listener.ora
# LISTENER.ORA Network Configuration File: /usr/local/oracle/product/9.2.0/network/admin/listen
# Generated by Oracle configuration tools.

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = snowcrash)(PORT = 1521))
      )
   )
 )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /usr/local/oracle/product/9.2.0)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = snow)
      (ORACLE_HOME = /usr/local/oracle/product/9.2.0)
```

```
      (SID_NAME = snow)
    )
 )
```

In this case, the oracle installer created files that are mostly correct. All that is required (for 9.2 and higher versions of Oracle) is to add a line which allows oracle to use shared objects (aka: .so files) that are not in the $ORACLE_HOME/lib directory. Simplest is to allow any .so file on the system. This was the behavior for versions of Oracle prior to 9.2. The SID_LIST_LISTENER section of the listener.ora file has one additional line:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /usr/local/oracle/product/9.2.0)
      (PROGRAM = extproc)
      (ENVS = "EXTPROC_DLLS=ANY")                #### ADDED LINE ####
    )
    (SID_DESC =
      (GLOBAL_DBNAME = snow)
      (ORACLE_HOME = /usr/local/oracle/product/9.2.0)
      (SID_NAME = snow)
    )
 )
```

To verify the changes, stop and start the listener:

```
$ lsnrctl stop

LSNRCTL for Solaris: Version 9.2.0.1.0 - Production on 18-SEP-2002 13:52:17

Copyright (c) 1991, 2002, Oracle Corporation. All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC)))
The command completed successfully
$ lsnrctl start

LSNRCTL for Solaris: Version 9.2.0.1.0 - Production on 18-SEP-2002 13:52:30

Copyright (c) 1991, 2002, Oracle Corporation. All rights reserved.

Starting /usr/local/oracle/product/9.2.0/bin/tnslsnr: please wait...

TNSLSNR for Solaris: Version 9.2.0.1.0 - Production
System parameter file is /usr/local/oracle/product/9.2.0/network/admin/listener.ora
Log messages written to /usr/local/oracle/product/9.2.0/network/log/listener.log

Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc )(KEY=EXTPROC)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp )(HOST=snowcrash)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC)))
STATUS of the LISTENER
------------------------
Alias LISTENER
Version TNSLSNR for Solaris: Version 9.2.0.1.0 - Production
Start Date 18-SEP-2002 13:52:30
Uptime 0 days 0 hr. 0 min. 0 sec
Trace Level off
Security OFF
```

```
SNMP  OFF
Listener Parameter File /usr/local/oracle/product/9.2.0/network/admin/listener.ora
Listener Log File /usr/local/oracle/product/9.2.0/network/log/listener.log
Listening Endpoints Summary...
 (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC)))
 (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=snowcrash)(PORT=1521)))
Services Summary...
Service "PLSExtProc" has 1 instance(s).
 Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
Service "snow" has 1 instance(s).
 Instance "snow", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

One final test of extproc functionality is to use the 'tnsping' utility to verify that the SQLNet service is available:

```
$ tnsping extproc_connection_data
TNS Ping Utility for Solaris: Version 9.2.0.1.0 - Production on 28-MAY-03 21:18:46

(c) Copyright 1997 Oracle Corporation.  All rights reserved.

Attempting to contact (ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC0))
OK (0 msec)
```

1.1.7 Creating an Account to Access Daycart and Test the Installation

Create (or give the privileges to) a user account that uses the Daycart software. As a DBA, grant the daycart role to a non-privileged user to use the cartridge. In this example, the user 'mug' with the password 'coffee' is granted daycart privilege. Simple queries can be used to test the cartridge.

```
$ sqlplus "/ as sysdba"
create user mug
   identified by
      coffee
   default tablespace
      daylight_ts
   temporary tablespace
      temp;

grant connect, daycart to mug;
connect mug/coffee

select ddpackage.finfo('hostid') from dual;

DDPACKAGE.FINFO('HOSTID')
------------------------------------------------------------
80ad7574

select ddpackage.ftestlicense('daycart') from dual;

DDPACKAGE.FTESTLICENSE('DAYCART')
---------------------------------
 1

select smi2cansmi('NCC', 0) from dual;

SMI2CANSMI('NCC',0)
------------------------------------------------------------
CCN
```

If the above three queries don't give the expected results, see the section on troubleshooting.

1.1.8 Revoking Privileges from c$dcischem

The user c$dcischem owns all the cartridge code, however c$dcischem requires no privileges after installation for the cartridge to work. All cartridge functionality operates with the privileges of the current user, not the cartridge owner (see the SQL reference manual for 'CREATE FUNCTION ... AUTHID CURRENT_USER and invokers rights for more details). Also, you might want to change the password for c$dcischem to something non-obvious.

```
revoke dba from c$dcischem;
```

or

```
revoke
   create public synonym,
   create type,
   create role,
   create operator,
   create indextype,
   create table,
   create library,
   create procedure,
   resource,
   connect
from
   c$dcischem;
```

1.2 Installation Troubleshooting

There are a number of details which must be correct for the cartridge to work properly. For a new Oracle installation, these are typically already set correctly, however following is a list of potential issues, which are included here for diagnostic purposes:

In order to use the cartridge functionality, the server must be configured to support *at least* 8.1 compatability. The line: compatible = "8.1.0" (or higher) must be present in your init.ora file. If it isn't, stop your server, edit the file, and restart the server. If, during the installation, you get the following message gazillions of times, it is because this parameter isn't set correctly:

```
SQL> create or replace operator smi2cansmi binding (varchar2, number)
*
ORA-00406: COMPATIBLE parameter needs to be 8.1.0.0.0 or greater
```

If one gets the following error message:

```
SQL> select ddpackage.finfo('hostid') from dual;
*
ERROR at line 1:
ORA-06520: PL/SQL: Error loading external library
ORA-06522: ld.so.1: extprocPLSExtProc:
  fatal: /usr/local/daylight/v491/dcischem/ddlib.so.10:
open failed: No such file or directory
```

the cause is the specification of the c$dcischemlib library. Recheck the lines in the file create.sql and verify that absolute path to the file 'ddlib.so' matches the actual location of the file and that the file is readable by the

Oracle user:

```
create or replace library c$dcischemlib
  as '/usr/local/daylight/v491/dcischem/ddlib.so.10';
```

It will be necessary to completely remove and reinstall the cartridge in order to fix this problem. See the section on 'Cartridge De-installation'.

If a valid license has not been inserted into the c$dcischem.license table, then the ftestlicense function will return:

```
SQL> select ddpackage.ftestlicense('daycart') from dual;

ERROR at line 1:
ORA-29400: data cartridge error
C$DCISCHEM-003: FAILED: License not available
```

In this case, contact Daylight and get a valid license key for the cartridge. Provide the hostid to Daylight (see installation above) and we'll provide to you the insert statement which must be run. If you have inserted the license key, then perhaps the c$dcischem.license table is not visible to the current user. Try:

```
SQL> select * from c$dcischem.license;

PRODUCT KEY EXPIRATION
--------- -------------------------------- ----------
daycart 8befd3d06296e9b89abfb158b431d278 01-JAN-01
```

If this table isn't visible, then perhaps the role 'daycart' hasn't been granted to the current user. Remember that each user must have the 'daycart' role granted to it in order to use the cartridge. The 'daycart' role grants select on c$dcischem.license and c$dcischem.progob, execute on the cartridge functions, and create table privilege.

If the network listener is not running or is misconfigured, then you'll get a message like this:

```
SQL> select smi2cansmi('NCC', 0) from dual
*
ERROR at line 1:
ORA-28575: unable to open RPC connection to external procedure agent
```

when attempting to access any Daylight Cartridge functions. In this case, verify that the listener is running:

```
$ lsnrctl
LSNRCTL> status

Connecting to (DESCRIPTION ...
STATUS of the LISTENER
------------------------
Alias LISTENER
Version TNSLSNR for Solaris: Version 8.1.5.0.0 - Production
Start Date 14-MAR-00 12:00:37
Uptime 0 days 0 hr. 17 min. 15 sec
Trace Level off
Security OFF
SNMP OFF
Listener Parameter File /oracle/o815/network/admin/listener.ora
Listener Log File /oracle/o815/network/log/listener.log
Services Summary...
```

```
PLSExtProc has 1 service handler(s)
dev has 3 service handler(s)
The command completed successfully
```

Note that the listener is running, and that there is a service listed for PLSExtProc. Also, verify that the files: $ORACLE_HOME/network/admin/listener.ora and $ORACLE_HOME/network/admin/tnsnames.ora have the appropriate entries. If necessary, restart the listener to make sure that any configuration file changes have taken effect.

In one case, we've seen a problem with the LD_LIBRARY_PATH not being set for the Oracle instance. This was an non-standard Oracle instance which was upgraded from 8.0.X, so it isn't clear where the problem arose. The cartridge requires the Oracle shared client library:

```
libclntsh.so.8.0 or libclntsh.so.9.0
```

This file is found in $ORACLE_HOME/lib/. It is preferred that the UNIX Oracle userid have the LD_LIBRARY_PATH set in the .profile or .cshrc to include $ORACLE_HOME. If this is not the case, define LD_LIBRARY_PATH and then restart the Oracle instance and listener.

1.3 Verification

The subdirectory 'dcischem/TEST' includes a number of stand-alone SQL scripts which test the cartridge installation. The shell script 'run_tests' will execute all the SQL scripts in turn and compare the output to reference output files. Any discrepencies will be reported. The script 'run_tests' does attempt to log in as 'c$dcischem/secret', so if you've changed the password and privileges on user 'c$dcischem ' you'll need to edit run_tests to reflect this change. For example using the mug user we created above.

The scripts use the 'EXPLAIN PLAN' facility, which requires that the plan table be present for the test user. The script $ORACLE_HOME/rdbms/admin/utlxplan.sql creates the plan table; this is executed as part of the 'run_tests' script.

```
$ run_tests
Testing dd_basic_test ...
Comparing output to reference file ...
dd_basic_test test OK.

Testing dd_exact_test ...
Comparing output to reference file ...
dd_exact_test test OK.
<... >
All tests passed.
```

Also, note that the SQL scripts give useful examples of using the cartridge functionality in each category.

1.4 Upgrading from Previous Versions of Daycart

In general, minor versions of Daycart (eg. 4.71 -> 4.72) do not *require* any database operations during upgrade. Typically these upgrades consist solely of a replacement for the ddlib.so shared library. In cases where Oracle DDL commands need to be executed as part of a minor version upgrade, these will be documented separately as part of the upgrade procedure.

Upgrades between major versions (4.7x -> 4.81, etc.) do require completely removing and replacing the Daycart Oracle packages. In order to perform major upgrades it is necessary to drop all dependant indexes

from the system and recreate them after the instance has been upgraded.

The basic procedure for a major upgrade is:

1. Completely back up the database instance.
2. Run the 'list_indexes' program as sysdba. Provide a name prefix which will be used to generate the 'create' and 'drop' files used in subsequent steps.
3. Edit the '[prefix].create.sql' file created by the 'list_indexes' program with passwords for accounts
4. From SQL, as a dba, run the '[prefix].drop.sql' file.
5. Run the Daycart 'clean.sql' script.
6. Perform the Daycart installation by running the new 'create.sql' script.
7. Verify that the new version of Daycart is correct by running the regression tests in $DY_ROOT/dcischem/TEST.
8. Run the '[prefix].create.sql' script to add the indexes.

Detailed instructions to upgrade your instance are:

First, back up your instance!!! Use an export, a cold backup (tar the database files with the instance shut down), or a hot backup using your favorite Oracle backup utility.

Next use the 'list_indexes' program provided with the release. It is a program which generates two SQL scripts, one to drop and the other to create your Daycart indexes.

Below is a 'list_indexes' example. There are two accounts (mug and glass) that have daycart indexes in this example. The list_indexes program can be found in $DY_ROOT/bin. There are two versions; one for Oracle 9.x one for 10 & 11 (list_indexes.9 and list_indexes.10, respectively). and

```
$ pwd
/usr/local/daylight/v491/bin
$ ls list_index*
list_indexes.9*  list_indexes.10*
$ ./list_indexes.9
Usage: list_indexes <user/pass@sid> <output_file_prefix>
```

If given a user with DBA privilege, list_indexes will find and catalog all Daycart indexes on the system. If a non-privileged user is specified, then only that users indexes will be cataloged.

```
$ ./list_indexes.9 system/secret snow
Total Rows processed: 8
```

Having supplied snow as the file prefix parameter, the two files: snow.create.sql and snow.drop.sql were created.

```
$ more snow.create.sql
-------------------------------------------------------------------------------
-- Index Owner: GLASS
--

CONNECT GLASS/########

-------------------------------------------------------------------------------
-- Index: GLASS.SMI_INDEX_EXACT_SMILES
-- Table: GLASS.SMI_MAIN
-- Type: DDEXACT
```

```
--

CREATE INDEX GLASS.SMI_INDEX_EXACT_SMILES
   ON GLASS.SMI_MAIN(SMILES)
   INDEXTYPE IS C$DCISCHEM.DDEXACT;
--  PARAMETERS (' ');

-------------------------------------------------------------------------------
-- Index: GLASS.SMI_INDEX_BLOB_SMILES
-- Table: GLASS.SMI_MAIN
-- Type: DDBLOB
-- Parameters: fpsize=1024
--

CREATE INDEX GLASS.SMI_INDEX_BLOB_SMILES
   ON GLASS.SMI_MAIN(SMILES)
   INDEXTYPE IS C$DCISCHEM.DDBLOB
   PARAMETERS ('fpsize=1024');
```

[many lines deleted]

```
CONNECT MUG/########

-------------------------------------------------------------------------------
-- Index: MUG.SMI_INDEX_EXACT_SMILES
-- Table: MUG.SMI_MAIN
-- Type: DDEXACT
--

CREATE INDEX MUG.SMI_INDEX_EXACT_SMILES
   ON MUG.SMI_MAIN(SMILES)
   INDEXTYPE IS C$DCISCHEM.DDEXACT;
--  PARAMETERS (' ');
```

[many lines lines deleted, to the end of the file]

here is the begining part of the drop script:

```
$ more snow.drop.sql
-------------------------------------------------------------------------------
-- Index: GLASS.SMI_INDEX_EXACT_SMILES
-- Table: GLASS.SMI_MAIN
-- Type: DDEXACT
--

DROP INDEX GLASS.SMI_INDEX_EXACT_SMILES FORCE;

-------------------------------------------------------------------------------
-- Index: GLASS.SMI_INDEX_BLOB_SMILES
-- Table: GLASS.SMI_MAIN
-- Type: DDBLOB
--

DROP INDEX GLASS.SMI_INDEX_BLOB_SMILES FORCE;
```

[many lines deleted, to the end of the file]

See the manpage for list_indexes(1) in the Daylight documentation for more details on the list_indexes

program.

Edit the 'prefix.create.sql' file. For each index, add optional creation parameters (eg. table_tablespace, etc.) and verify that the columns and tables are correct. Also, note that the creation script runs as multiple different users, so you'll need to either enter the passwords for each user in the script or cut apart the sql scripts for each individual user to invoke.

From SQL, as dba, run the 'prefix.drop.sql' file. This will drop all Daycart indexes.

```
$ sqlplus system/secret @snow.drop.sql

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Dec 17 13:45:52 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production


Index dropped.


Index dropped.


Index dropped.


Index dropped.


Index dropped.


Index dropped.


Index dropped.


Index dropped.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

Run the Daycart 'clean.sql' script. This will drop all daycart packages, indextypes, and operators from the database instance.

```
$ sqlplus 'c$dcischem/secret' @/usr/local/daylight/v473/daycart.9i/clean.sql

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Dec 17 13:54:53 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
```

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

Connected.
Connected.

Synonym dropped.
```

[many lines deleted]

```
Package dropped.


Role dropped.


Library dropped.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

Perform the Daycart installation by running the new 'create.sql' script. Remember to edit create.sql and set the library path. This is covered in the installation section of this document.

```
$ sqlplus 'c$dcischem/secret' @/usr/local/daylight/v491/dcischem/dcischem/create.sql

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Dec 17 13:54:53 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

PL/SQL procedure successfully completed.

Connected.

Library created.
```

[many lines deleted]

```
Grant succeeded.


Grant succeeded.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

Verify that the new version of Daycart is correct by running the regression tests in $DY_ROOT/dcischem/TEST.

Now, run the 'prefix.create.sql' script. This will recreate all Daycart indexes:

```
$ sqlplus 'c$dcischem/secret' @snow.create.sql

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Dec 17 14:31:04 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

Connected.

Index created.


Index created.


Index created.


Index created.

Connected.

Index created.


Index created.


Index created.


Index created.

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

At this point you are finished with doing the upgrade.

1.5 De-Installation

In order to cleanly de-install the cartridge, all dependent indexes must be dropped from the system first. That is, any indexes created using one of the Daylight indextypes (ddexact, ddrole , ddgraph, ddblob) should be dropped. If one does not drop all the indexes first, then they will be marked as invalid. If this occurs, the indexes and their associated index tables ([indexname] || '_DDT') must be dropped manually. The list_indexes program is convenient to generate the 'drop index' commands. See the previous section.

After any dependent indexes have been dropped, one can run the script 'clean.sql'. This script will remove all cartridge index definitions, functions, operators, packages, roles, and synonyms. The script 'clean.sql' will remove everything which is installed by the script ' create.sql' with the exception of the license and program object tables. These tables are preserved since it is convenient to not need to recreate and populate these tables. The user c$dcischem will require DBA privilege to be granted in order to run the script.

Example output follows:

```
$ sqlplus 'c$dcischem/secret' @clean

SQL*Plus: Release 9.2.0.1.0 - Production on Thu Dec 12 13:27:29 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

Connected.

Revoke succeeded.


Revoke succeeded.


Revoke succeeded.


Revoke succeeded.
```

[many lines of text omitted]

```
Package dropped.


Role dropped.


Library dropped.

SQL> select * from tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
LICENSE                        TABLE
PROGOB                         TABLE
PLAN_TABLE                     TABLE

SQL> exit
Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
$
```

It is safe to repeatedly run the scripts create.sql and clean.sql. The create.sql script creates a bunch of objects, types, etc. The clean.sql script removes them. Neither script will modify any other objects, types, or tables owned by c$dcischem or any other user on the system. The only components created by the create.sql script which are not removed by the clean.sql script are the license, progob, salt, and transform tables. These tables are preserved by the clean.sql script for convenience.

When one runs create.sql after clean.sql, one will see warnings that the tables license, progob, salt, and transform already exist, however these warnings don't impact the outcome of the cartridge re-installation.

After running clean.sql , the database will be left in its original pre-cartridge state. One can then delete the license, progob, salt, and transform tables, if desired, to eliminate all remnants of the cartridge.

Note that the user c$dcischem will continue to exist after running clean.sql and any non-cartridge tables, objects, etc. owned by c$dcischem will be unaffected by either clean.sql or create.sql .

Another way to remove the cartridge is to drop the user c$dcischem. Note that this does not remove public synonyms or roles, so these should be dropped manually (or via clean.sql).

```
SQL> drop user c$dcischem cascade;
```

This will remove the user c$dcischem and all dependent objects. It will be necessary to repeat the entire installation procedure if this method is chosen.

2. PL/SQL Functions, SQL Operators

Several stateless PL/SQL functions and SQL operators have been implemented for the cartridge. All of the PL/SQL functions are contained in a package called 'ddpackage' owned by 'c$dcischem'. Privileges required to access these functions are granted by the 'daycart' role.

The default installation (create.sql) creates public synonyms for the package 'ddpackage' and all of the operators described herein. Therefore, any Oracle user can access functions and operators using the synonyms, e.g., select tanimoto(...) from table rather than select c$dcischem.tanimoto(...) from table). Please note that there are no synonyms allowed for indextype specifications; hence one must use full names when creating indexes as a user other than c$dcischem, e.g., create index <name> on small(smi) indextype is c$dcischem.ddblob.

2.1 String Data Handling

Daycart supports both VARCHAR2 and CLOB string datatypes interchangeably in all Daycart functions, operators and index searches. At runtime Oracle uses argument overloading to transparently handle the strings arguments passed into the Daycart functions. However, since Oracle does not transparently handle return types, the user must be aware of the string type returned by a function. In general, Daycart functions return strings with the same string type which was passed into the function as the 'key' string parameter. For example:

```
function ddpackage.fsmi2cansmi (smiles IN VARCHAR2_OR_CLOB, type IN NUMBER) => VARCHAR2_OR_CLOB
```

If the string argument passed into smi2cansmi() is a VARCHAR2, then the function will always return a VARCHAR2. Similarly, if the string argument is a CLOB, the function will always return a CLOB.

There are three Daycart functions which take multiple input strings and also return a string --- partnorm(), atomnorm(), and bondnorm(). These functions use the ntuple-list data as the 'key' argument is the ntuple-list data.

2.2 General Purpose Functions

There are five general purpose functions available that are typically used for debugging or setting session-level options only. Operators and public synonyms are not created for these functions during the installation. Hence, these they must be referenced by their fully-qualified names, e.g., c$dcischem.ddpackage.ftestlicense().

fsetdebug

```
function ddpackage.fsetdebug (value IN NUMBER) => NUMBER
```

Controls the level of messages written to the log. The default logfile is /tmp/extproc.log. The logfile name can be changed with the DAYCART_LOGFILE environment variable in listener.ora. Sets the new value of the logging level to the new 'value' for the current session. Valid values are integers the range of 0 - 9. By default only Error message are written to the log.

```
0 - No logging at all
1 - Error messages only
5 - Warnings and errors
9 - Notes, warnings, and errors
```

fgeterrors

```
function ddpackage.fgeterrors (level IN NUMBER) => VARCHAR2
```

Returns error strings from the error queue for previously failed functions or operators based on the level requested and clears the error queue. By default only message at the level of Errors or above are sent to the screen when a function fails.

```
0 - All messages
1 - Notes
2 - Warnings
3 - Errors
4 - Fatal errors
```

fgetlog

```
function ddpackage.fgetlog => CLOB
```

Returns error strings from the log messages and clears the local buffer of log messages. Behavior is controlled by the session-level option 'log'.

In order to get log messages from the ddpackage.fgetlog() function the 'log' option must be set to either 'LOCAL' or 'BOTH'. From that point log messages will be kept and can be retrieved with the fgetlog() function. By default, the 'log' option is set to 'CENTRAL' and all log messages go to the central logfile, which by default is /tmp/extproc.log. The central logfile is controlled by the environment variable DAYCART_LOGFILE, which can be set in the listener.ora parameter file for extproc.

ftestlicense

```
function ddpackage.ftestlicense (product IN VARCHAR2_OR_CLOB) => NUMBER
```

Checks the license. The license is contained in a special table is created at install time. Currently the only recognized value for product is 'daycart'. Returns 1 if the Daylight cartridge has a valid license and 0 if not.

finfo

```
function ddpackage.finfo (which IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
```

Returns informational strings from DayCart. Valid input parameters are listed below. In addition, finfo can be used to find the value for any dayconvert option. See Section 2.3 Molecule / Reaction Functions for detailed descriptions of the dayconvert options.

`'toolkit_version'`

Returns the current Daylight Toolkit version.
`'daycart_version'`

Returns the cartridge executable version.
`'extproc_pid'`

Returns the extproc process ID for this DayCart session.
`'hostid'`

Returns the hardware hostid used for generation of the license key.
`'debug_level'`

Returns the debug level set for logging messages.
`'session_tag'`

Returns the session tag set for a given session. The session tag, if set, is included in any log messages and can be used to identify the session which generated a log entry.
`'exit_on_fault'`

Returns TRUE or FALSE. When TRUE, the daycart session will keep track of the count of severe errors encountered (ORA-00600, ORA-03113, ORA-03114, ORA-07445) and if more than fifty of these errors are logged, the extproc process for the current session will exit.
`'log'`

Returns where the session log info is written. Choices are NONE, LOCAL, CENTRAL, and BOTH. CENTRAL is the default, and refers to the value of DAYCART_LOGFILE (in the listener.ora) or a default of /tmp/extproc.log. LOCAL logging indicates that any log messages are stored for retrieval by the ddpackage.fgetlog() function.
`'default_delimiter'`

Returns the default delimiter used to separate lines of multi-line output.
`'force_delimiter'`

Returns TRUE or FALSE. When TRUE, the default delimiter is always used for multi-line output. When FALSE, Daycart may attempt to detect the delimiter to use (from other input).
`'vcs_table_cache'`

Returns the value for the state of the VCS cache of salts and transforms. When off, each invocation of a vcs function causes the salt or transform table to be reread.

`'default_fpsize'`

Returns the default fingerprint size used for similarity comparisons and index creation when not otherwise defined.

`'timeout_interrupt'`

Returns the current search interrupt time in seconds. An ongoing ddblob search will return to Oracle from extproc in at most this many seconds, provided that some search results have been found. Unless the search is interrupted by the client the search will continue.

`'timeout_abort'`

Returns the current search abort time in seconds. An ongoing ddblob search will abort and return the hits found so far after this elapsed time.

`'timeout_progress'`

For the last ddblob search, returns zero if the search completed or a positive integer if the search aborted due to timeout_abort. In the event of an abort, the value returned is the number of structures searched; this can be used as the approximate progress of the search (relative to the number of rows in the table being searched).

`'thread_count'`

Returns the number of worker threads which will be allocated in a session for multithreaded searches. Multithreading is used for contains(), isin(), and matches() searches.

`'count_value'`

Returns the number of hits expected (estimated or actual) for the previously-run query. One must set the count_mode using ddpackage.fsetinfo('count_mode={ACTUAL|ESTIMATE}') immediately before executing the query, and then get the count_value immediately after the query. When in count mode, no hits will be returned from any Daycart search but the count_value variable will reflect the number of hits which would have been returned.

fsetinfo

```
function ddpackage.fsetinfo (name_value_pair IN VARCHAR2_OR_CLOB) => NUMBER
```

Allows the user to individually set session-level options in DayCart for the following parameters and all dayconvert options (see Section 2.3 Molecule / Reaction Functions).

`'vcs_table_cache={on|off}'`
`'debug_level={level}'` valid values are integers 0 - 9
`'session_tag={value}'` any string, truncated to 32 chars
`'exit_on_fault={TRUE|FALSE}'`
`'log={place}'` valid values are NONE, LOCAL, CENTRAL, BOTH
`'default_fpsize={nbits}'` valid values are integers 32 - 16384
`'default_delimiter={string}'`
`'force_delimiter={TRUE|FALSE}'`
`'timeout_interrupt={seconds}'`
`'timeout_abort={seconds}'`
`'options={class}'` valid values correspond to options table
`'thread_count={count}'` valid values are integers 0 - 32

```
'count_mode={NONE|ACTUAL|ESTIMATE}'
```

Starting with version 4.93, fsetinfo can be used to globally set options on a session-level basis. An OPTIONS table which can be populated with user-defined sets of parameters is automatically created during installation. Executing fsetinfo using `'options={class}'` will reset all options to their default values and then will set the values associated with the given class. On session startup, the options with class of zero are set.

```
Name                                      Type
----------------------------------------- ----------------------------
NAME                                      VARCHAR2(100)
VALUE                                     VARCHAR2(100)
CLASS                                     NUMBER(7)

NAME = parameter name as listed above or any of the dayconvert options
VALUE = new value
CLASS = set number for options.
```

2.3 Molecule / Reaction Functions

Functions and their respective operations relating to conversion, transformation. property values and normalization and of molecules and reactions are described below.

fsmi2cansmi

```
function ddpackage.fsmi2cansmi (smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER) => VARCHAR2_OR_CLOB
operator smi2cansmi (smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER)=> VARCHAR2_OR_CLOB
```

Returns a canonical SMILES string from an input SMILES. Type is either 0 or 1, for unique or absolute SMILES, respectively.

fsmi2xsmi

```
function ddpackage.fsmi2xsmi (smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, explicit IN NUMBER)=> VARCHAR2_OR_CLOB
operator smi2xsmi (smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, explicit IN NUMBER) => VARCHAR2_OR_CLOB
```

Returns an exchange SMILES string which is semantically identical to the input SMILES but which does not use Daylight-specific aromaticity conventions. Type is either 0 or 1, for unique or absolute SMILES, respectively. When 'explicit' is 1, it supplies hydrogen count and other atomic properties explicitly for every atom. Note that exchange SMILES are not canonical; the same input molecule or reaction may return different exchange SMILES depending on the input order to this function.

fsmi2netch

```
function ddpackage.fsmi2netch (smiles IN VARCHAR2_OR_CLOB) => NUMBER
operator smi2netch (smiles IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the net charge of the input molecule or reaction.

fsmi2hcount

```
function ddpackage.fsmi2hcount (smiles IN VARCHAR2_OR_CLOB) => NUMBER
operator smi2hcount (smiles IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the total hydrogen count for the input molecule or reaction.

fsmi2mf

```
function ddpackage.fsmi2mf (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
operator smi2mf (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
```

Returns the molecular formula string for the input molecule or reaction.

fsmi2amw

```
function ddpackage.fsmi2amw (smiles IN VARCHAR2_OR_CLOB) => NUMBER
operator smi2amw (smiles IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the average molecular weight for the input molecule or reaction. The weight used for any atoms which do not have specified isotopes is the average atomic weight. The weight used for atoms with a specified isotope is the high precision molecular weight for that atom. For example, "c1ccccc1" returns 78.1184, while "[1H][12c]1[12c]([1H])[12c]([1H])[12c]([1H])[12c]([1H])[12c]1[1H]" returns 78.0469502.

fsmi2pmw

```
function ddpackage.fsmi2pmw (smiles IN VARCHAR2_OR_CLOB) => NUMBER
operator smi2pmw (smiles IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the high-precision molecular weight for the input molecule or reaction. The weight used for any atoms which do not have specified isotopes is the high precision weight for the most abundant isotope. The weight used for atoms with a specified isotope is the high precision molecular weight for that atomic isotope. For example, "c1ccccc1" returns 78.0469502, the high precision molecular weight. "BrBr" returns 157.836675, while "[81Br][81Br]" returns 161.832582.

Both smi2amw() and smi2pmw() return the same values for SMILES with fully specified isotopes; they only differ in their handling of SMILES with unspecified isotopic weights. Note also that the unique canonical SMILES (eg. smi2cansmi('smiles', 0)) returns the structure with all isotopic information removed; this is useful for consistent handling of partially specified isotopic information in conjunction with the two functions.

fsmi2graph

```
function ddpackage.fsmi2graph (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
operator smi2graph (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
```

Returns the hydrogen- and charge-suppressed canonical graph string for the input molecule or reaction.

fvcs_desalt

```
function ddpackage.fvcs_desalt(smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, class IN NUMBER(7,0)) => VARCHAR2_OR_CLOB
operator vcs_desalt(smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, class IN NUMBER(7,0))=> VARCHAR2_OR_CLOB
```

Removes molecule fragments found in the c$dcischem.salts table from the input SMILES. Type is either 0 or 1, for unique or absolute SMILES, respectively. The class value is the class of salts entries used from the salts table. All of the structures in the salts table with the given class are checked against the input SMILES and if found, they are removed.

fvcs_normalize

```
function ddpackage.fvcs_normalize(smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, class IN NUMBER(7,0)) => VARCHAR2_OR_CLOB
operator vcs_normalize(smiles IN VARCHAR2_OR_CLOB,
        type IN NUMBER, class IN NUMBER(7,0)) => VARCHAR2_OR_CLOB
```

Performs a SMIRKS-based structure normalization on the input SMILES. All of the SMIRKS from the c$dcischem.transform table with the given class are applied to the input molecule. The resulting molecule is output as a canonical SMILES. Type is either 0 or 1 for unique or absolute SMILES, respectively.

fatomnorm
fbondnorm
fpartnorm

```
function ddpackage.fatomnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB
operator atomnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB

function ddpackage.fbondnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB
operator bondnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB

function ddpackage.fpartnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB
operator partnorm (smiles IN VARCHAR2_OR_CLOB,
        list IN VARCHAR2_OR_CLOB, ntuple IN NUMBER, isotype IN NUMBER) => VARCHAR2_OR_CLOB
```

Returns a potentially reordered N-tuple string for the given list input parameter. The list string is a comma-separated list of data which is associated in order with the atoms, bonds or parts of the input SMILES. The list string is reordered based on the canonical atom, bond, or part ordering of the input SMILES. ntuple is the number of comma-separated values per atom, bond, or dot-separated part, and isotype is 0 for unique SMILES canonicalization and 1 for absolute SMILES canonicalization.

fgen_molecules
fgen_reactions

```
function ddpackage.fgen_molecules(smiles IN VARCHAR2_OR_CLOB,
        smirks IN VARCHAR2_OR_CLOB,direction IN NUMBER, limit IN NUMBER,
        type IN NUMBER) => VARCHAR2_OR_CLOB
operator gen_molecules(smiles IN VARCHAR2_OR_CLOB,
        smirks IN VARCHAR2_OR_CLOB, direction IN NUMBER, limit IN NUMBER,
        type IN NUMBER) => VARCHAR2_OR_CLOB

function ddpackage.fgen_reactions(smiles IN VARCHAR2_OR_CLOB,
        smirks IN VARCHAR2_OR_CLOB, direction IN NUMBER, limit IN NUMBER,
        type IN NUMBER) => VARCHAR2_OR_CLOB
operator gen_reactions(smiles IN VARCHAR2_OR_CLOB,
        smirks IN VARCHAR2_OR_CLOB, direction IN NUMBER, limit IN NUMBER,
```

```
        type IN NUMBER) => VARCHAR2_OR_CLOB
```

Applies the transform created from the given SMIRKS to the input molecules. The argument direction indicates that the transform is applied in the forward (0) or reverse (1) direction. The limit parameter is the maximum count of specific molecules to return and 0 indicates no limit. The resulting molecules or reactions are output as a set of newline-delimited canonical SMILES. The gen_molecules() function and operator returns the molecules which result from a transformation; the gen_reactions() function and operator return the complete reaction. The type indicates either 0 or 1 for unique or absolute SMILES, respectively.

The results are returned as a single string. If more than one molecule or reaction is formed, then each is on a separate line of the output, delimited by the default_delimiter string (see ddpackage.finfo().

fdayconvert

```
function ddpackage.fdayconvert (data IN VARCHAR2_OR_CLOB,
        ifmt IN VARCHAR2, ofmt IN VARCHAR2, type IN NUMBER,
        ptable_class IN NUMBER) => VARCHAR2_OR_CLOB
operator dayconvert (data IN VARCHAR2_OR_CLOB,
        ifmt IN VARCHAR2, ofmt IN VARCHAR2, type IN NUMBER,
        ptable_class IN NUMBER) => VARCHAR2_OR_CLOB
```

Returns the input chemical information in a different format. See the Daylight Conversion Manual for additional information. Note: The 'type' and 'ptable_class' parameters are optional as described below.

The 'ifmt' and 'ofmt' parameters are used to designate the input and output formats, respectively. Both parameters need to be identified by particular letter sequence. Valid combinations of input and output formats for conversion are as follows where tdtsma and tdtsmrk are tdt versions with smarts and smirks, respectively:

```
        smi ---> mol/sdf/rdf
        tdt ---> mol/sdf/rdf
        mol or sdf ---> smi/ism/sma/tdt/tdtsma
        rdf ---> smi/ism/sma/smrk/tdt/tdtsma/tdtsmrk
```

Either mol or sdf can be used for rgfile input.

Delimiters for interpreting multi-line input data are detected from the input stream. Input delimiters may be the strings "\n", "\r", or "\r\n". On output, the delimiter chosen for multi-line output depends on the input delimiter detected and the settings of "force_delimiter" and "default_delimiter". See the description of these items in the documentation for the ddpackage.finfo(). function.

The 'type' parameter has been included for backwards compatibility and no longer controls the inclusion of isomeric information in the conversion output. Conversion output from a smiles form to an MDL form will always include the isomeric information while conversion from an MDL form to SMILES is controlled by the inclusion of ism versus smi for ofmt. However, inclusion of a value for the type parameter (0 or 1) is required for versions 4.92;

```
select ddpackage.fdayconvert ('CCCOC', 'smi', 'sdf', 1) from dual;
```

In contrast for versions 4.93 and later, a value for the type parameter or is only required if a ptable_class value is given.

```
select ddpackage.fdayconvert ('CCCOC', 'smi', 'sdf') from dual;
```

```
OR

select ddpackage.fdayconvert ('CCCOC', 'smi', 'sdf', 1, 10) from dual;
```

The ptable_class parameter is optional. If the 'ptable_class' parameter is provided, it indicates that valence and charge information in the user-defined PTABLE table information is to be used instead of that provided in the default p-table. The specific information to be used is based upon the class number supplied.

```
Name                                     Type
---------------------------------------- ----------------------------
AT_NO                                    NUMBER
SYMBOL                                   VARCHAR2(8)
AT_MASS                                  NUMBER
VALENCE_CHARGE_LIST                      VARCHAR2(4000)
CLASS                                    NUMBER

AT_NO = atomic number
SYMBOL = atomic symbol
AT_MASS = atomic mass
VALENCE_CHARGE_LIST = list of valence and charge, e.g., '2,-1,3,0'
        for valence 2 with -1 change or valence 3 with 0 charge
```

The following are a series of dayconvert related options. The value for any of these options can be found using the finfo() function. Values for these options can be changed on a per session basis for an individual option or a group of options using the fsetinfo() function as described in Section 2.2 General Purpose Functions.

`'conv_smi_is_ism={TRUE|FALSE}'`

Default is `FALSE`. When set to `TRUE`, conversions to tdt format include the isomeric SMILES in the root $SMI datatype

`'conv_add_3d={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE`, conversion to tdt format includes 3D coordinates in the output file if present in the input.

`'conv_add_2d={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE`, conversion to tdt format includes 2D coordinates in the output file if present in the input.

`'conv_use_3d={TRUE|FALSE}'`

Default is `FALSE`. When `TRUE`, conversion from tdt format includes 3D coordinates in the output if present in the input.

`'conv_split_fields={TRUE|FALSE}'`

Default is `FALSE`. Controls handling of multi-line data from sdf and tdt files. `TRUE` causes these to be split into separate dataitems.

`'conv_id_field={id_field}'`

Sets the id field to be used. The default behavior for molecules being converted to tdt format is to use the first line of each header block as the id field. The default behavior for reactions is to use the value following $RIREG as the id.

`'conv_prefix={prefix}'`

Sets the designated prefix to be parsed from the datatype names for conversion of rdf files. The default is none.
`'conv_implicit_chirality={TRUE|FALSE}'`

Default is `FALSE`. Alters the way in which chirality is determined in order to detect implicit chiral centers. Useful for some natural products. For a bond A-hash-B, the interpretation is that B is below A from the perspective of A and A is above B from the perspective of B. `TRUE` causes both ends of the chiral bonds to be used in the determination of chiral centers.
`'conv_ring_cistrans={TRUE|FALSE}'`

Default is `FALSE`. Setting this option as `TRUE` ignores cis/trans stereochemistry for all ring double bonds.
`'conv_db_explicit_h={TRUE|FALSE}'`

Default is `FALSE`. When `TRUE` requires that double bonds have all hydrogens explicitly indicated for conversion from non-query MDL format to SMILES.
`'conv_chi_explicit_h={TRUE|FALSE}'`

Default is `FALSE`. When `TRUE` chiral atoms have all hydrogens explicitly indicated for conversion from non-query MDL format to SMILES.
`'conv_fix_radical_rings={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE` allows for the certain types of five, six and seven-membered radical rings to be converted to aromatic. When `FALSE` keeps the ring as specified in the input MDL file. In order for a ring to be converted all atoms in the ring must be carbon and doublet radical. In addition no atom in the ring may have a charge.
`'conv_nametag={tag}'`

Sets the tdt datatype tag used for the id. Default is $NAM.
`'conv_comment_smi={TRUE|FALSE}'`

Default is `FALSE`. If `TRUE` output SMILES is written into the comment line of the header block in each connection table.
`'conv_smi_tuples={TRUE|FALSE}'`

Default is `TRUE`. When `FALSE` the tuple information associated with the isomeric SMILES field in a tdt is printed in the output MDL file.
`'conv_day_hcount={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE` the program uses both explicit H and H-count field for the conversion of query MDL files.
`'conv_day_stereo={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE` the program uses stereochemistry as specified for the conversion of query MDL files. When `FALSE` the program uses stereochemistry as specified and unspecified.
`'conv_day_chih={TRUE|FALSE}'`

Default is `TRUE`. When `TRUE` chiral hydrogens are required to be explicit for the conversion of query MDL files. When `FALSE` implicit hydrogens are used.

fsmi2scbits

```
function ddpackage.fsmi2scbits (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
operator smi2scbits (smiles IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
```

Returns ASCII encoded scaffold data for a given molecule or reaction. This data is not normally visible within Daycart. The function is provided as a convenience when creating ddblob indexes. It allows one to precompute and store the scaffold data. One can then use the 'initsccolumn' parameter for ddblob creation to avoid recomputing the scaffold data.

2.4 Fingerprint Functions

This section describes functions and their respective operations relating to generation of and information concerning fingerprints.

fsmi2fp

```
function ddpackage.fsmi2fp (smiles IN VARCHAR2_OR_CLOB,
        min IN NUMBER, max IN NUMBER, nbits IN NUMBER) => VARCHAR2_OR_CLOB
operator smi2fp (smiles IN VARCHAR2_OR_CLOB, min IN NUMBER,
        max IN NUMBER, nbits IN NUMBER) => VARCHAR2_OR_CLOB
```

Returns the ASCII fingerprint for a given molecule or reaction. Min and max are the minimum and maximum pathlengths, respectively, and size is the number of bits in the fingerprint.

fsmi2xfp

```
function ddpackage.fsmi2xfp (smiles IN VARCHAR2_OR_CLOB,
        min IN NUMBER, max IN NUMBER, nbits IN NUMBER) => VARCHAR2_OR_CLOB
operator smi2xfp (smiles IN VARCHAR2_OR_CLOB, min IN NUMBER,
        max IN NUMBER, nbits IN NUMBER) => VARCHAR2_OR_CLOB
```

Returns the ASCII difference fingerprint for a given molecule or reaction. Min and max are the minimum and maximum pathlengths, respectively, and size is the number of bits in the fingerprint.

ffoldfp

```
function ddpackage.ffoldfp (fpstr IN VARCHAR2_OR_CLOB,
        nbits IN NUMBER, dens IN NUMBER) => VARCHAR2_OR_CLOB
operator foldfp (fpstr IN VARCHAR2_OR_CLOB, nbits IN NUMBER,
        dens IN NUMBER) => VARCHAR2_OR_CLOB
```

Folds the given fingerprint to the minimum appropriate size or density, whichever is limiting, and returns the new, folded fingerprint.

fbitcount

```
function ddpackage.fbitcount (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
operator bitcount (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the number of bits on in the fingerprint.

fnbits

```
function ddpackage.fnbits (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
operator nbits (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the total size of the fingerprint, in bits. In the current Daylight toolkit this will always be a power of two.

fisfp

```
function ddpackage.fisfp (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
operator isfp (fpstr IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the string is a fingerprint, 0 otherwise. The syntax of a fingerprint can never be confused with a valid SMILES. This is the bit of cleverness which allows us to overload the searching functions.

2.5 Comparison Functions

Functions and their respective operations relating to a variety of direct comparisons between input smiles, smarts, or fingerprints are described below.

fexact

```
function ddpackage.fexact (a IN VARCHAR2_OR_CLOB, b IN VARCHAR2_OR_CLOB) => NUMBER
operator exact (a IN VARCHAR2_OR_CLOB, b IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the two input strings are identical, 0 otherwise. The operator is optionally backed by the ddexact indextype.

fgraph

```
function ddpackage.fgraph (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator graph (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the two input SMILES share the same canonical graph, 0 otherwise. The operator is optionally backed by the ddgraph indextype.

ftautomer

```
function ddpackage.ftautomer (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator tautomer (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the two input SMILES share the same canonical graph, net charge, and total hydrogen count, 0 otherwise. The operator is optionally backed by the ddgraph indextype.

fusmiles

```
function ddpackage.fusmiles (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator usmiles (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the two input SMILES share the same unique canonical smiles, 0 otherwise. The operator is optionally backed by the ddgraph indextype.

fasmiles

```
function ddpackage.fasmiles (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator asmiles (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if the two input SMILES share the same absolute canonical smiles, 0 otherwise. The operator is optionally backed by the ddgraph indextype.

fcomponent

```
function ddpackage.fcomponent (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator component (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if smiles2 (a molecule SMILES) is a component of smiles1 (any SMILES), otherwise returns 0. Also returns false if smiles2 is not a single-component query. The last example in the table below illustrates this problem. The operator is optionally backed by the ddrole indextype. A component is a single dot-separated part of a larger molecule or reaction SMILES. Some examples follow:

smiles1	smiles2	Returns
CCC	CCC	1
CCC.CCCN	CCC	1
CCC.CCCN	CCCN	1
CCC>>CCCN	CCC	1
CCC>>CCCN	CCCN	1
CCC.CCCN	CCC.CCCN	0

freactant
fagent
fproduct

```
function ddpackage.freactant (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
operator reactant (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER

function ddpackage.fagent (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB)=> NUMBER
operator agent (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB)=> NUMBER

function ddpackage.fproduct (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB)=> NUMBER
operator product (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns 1 if smiles2 (a molecule SMILES) is a component of smiles1 (a reaction SMILES) with the appropriate role, otherwise returns 0. The operators are optionally backed by the ddrole indextype.

fcontains

```
function ddpackage.fcontains (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator contains (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
```

Returns 1 if smiles1 contains smiles2; that is, smiles2, assuming opened valences for all hydrogens, is a substructure of smiles1. The operator is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits (based on tanimoto similarity) will be returned that meet the search criteria.

fisin

```
function ddpackage.fisin (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator isin (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is
optional)
```

Returns 1 if smiles2 contains smiles1; that is, smiles1, assuming opened valences for all hydrogens, is a substructure of smiles2. This functionality is identical to 'contains()' with the arguments swapped. The operator is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits (based on tanimoto similarity) will be returned that meet the search criteria.

fmatches

```
function ddpackage.fmatches (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator matches (smiles1 IN VARCHAR2_OR_CLOB,
        smiles2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
```

Returns 1 if the smarts expression matches the given SMILES, 0 otherwise. The operator is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits (based on bits set in the target fingerprint) will be returned that meet the search criteria.

fmatchcover

```
function ddpackage.fmatchcover (smiles IN VARCHAR2_OR_CLOB,
        smarts IN VARCHAR2_OR_CLOB) => NUMBER
operator matchcover (smiles IN VARCHAR2_OR_CLOB,
        smarts IN VARCHAR2_OR_CLOB) => NUMBER
```

Returns the ratio of atoms in the target SMILES matched by the given query to the number of atoms in the target as a number between zero and one.

feuclid

```
function ddpackage.feuclid (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator euclid (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
```

Returns the euclidean distance between two fingerprints or SMILES. If both parameters are fingerprints and are not the same size (nbits()), then the larger will be folded automatically to match the size of the smaller before comparison. If one parameter is a SMILES, its fingerprint is generated automatically at the size of the other parameter. If both parameters are SMILES, then a fingerprint size of 512 bits is used. The returned value is a floating point number between 0.0 and 1.0. This is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits will be returned that meet the search criteria.

ftanimoto

```
function ddpackage.ftanimoto (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator tanimoto (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
```

Returns the tanimoto distance between two fingerprints or SMILES. If both parameters are fingerprints and are not the same size (nbits()), then the larger will be folded automatically to match the size of the smaller before comparison. If one parameter is a SMILES, its fingerprint is generated automatically at the size of the other parameter. If both parameters are SMILES, then a fingerprint size of 512 bits is used. The returned value is a floating point number between 0.0 and 1.0. This is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits will be returned that meet the search criteria.

ftversky

```
function ddpackage.ftversky (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, alpha IN NUMBER, beta IN NUMBER,
        count IN NUMBER) => NUMBER   (count is optional)
operator tversky (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, alpha IN NUMBER, beta IN NUMBER,
        count IN NUMBER) => NUMBER   (count is optional)
```

Returns the tversky distance between two fingerprints or SMILES. If both parameters are fingerprints and are not the same size (nbits()), then the larger will be folded automatically to match the size of the smaller before comparison. If one parameter is a SMILES, its fingerprint is generated automatically at the size of the other parameter. If both parameters are SMILES, then a fingerprint size of 512 bits is used. The returned value is a floating point number between 0.0 and 1.0. This is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits will be returned that meet the search criteria.

ffingertest

```
function ddpackage.ffingertest (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
operator fingertest (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, count IN NUMBER) => NUMBER   (count is optional)
```

Returns 1 if all of the bits in fp_or_smi2 are also present in fp_or_smi1. That is, the fingerprint from fp_or_smi2 represents a possible substructure of fp_or_smi1. If both parameters are fingerprints and are not the same size (nbits()), then the larger will be folded automatically to match the size of the smaller before comparison. If one parameter is a SMILES, its fingerprint is generated automatically at the size of the other parameter. If both parameters are SMILES, then a fingerprint size of 512 bits is used. The returned value is a floating point number between 0.0 and 1.0. This is optionally backed by the ddblob indextype.

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits (based on the number of bits set in the target) will be returned that meet the search criteria.

fsimilarity

```
function ddpackage.fsimilarity (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, expression in VARCHAR2,
        count IN NUMBER) => NUMBER   (count is optional)
operator similarity (fp_or_smi1 IN VARCHAR2_OR_CLOB,
        fp_or_smi2 IN VARCHAR2_OR_CLOB, expression in VARCHAR2,
        count IN NUMBER) => NUMBER   (count is optional)
```

Returns the similarity or distance between two fingerprints or SMILES, based on the computed expression. If both parameters are fingerprints and are not the same size (nbits()), then the larger will be folded automatically to match the size of the smaller before comparison. If one parameter is a SMILES, its fingerprint is generated automatically at the size of the other parameter. If both parameters are SMILES, then a fingerprint size of 512 bits is used. The returned value is a floating point number. This is optionally backed by the ddblob indextype.

The value of expression can be any legal expression based on the counts of bits in the corresponding fingerprints (see the man page for expression(5)). Optionally, the expression string can be preceded with either "distance=" or "similarity=". These are used in by the index implementation but ignored in the functional form. The following examples are all identical:

```
fsimilarity(smi1, smi2, 'TANIMOTO')
fsimilarity(smi1, smi2, 'tanimoto')
fsimilarity(smi1, smi2, 'similarity=tanimoto')
fsimilarity(smi1, smi2, 'c/(a+b+c)')
fsimilarity(smi1, smi2, 'similarity=c/(a+b+c)')
```

The optional count argument is ignored in the functional forms. It is used in the index implementation as the desired number of hits to be returned. When used, the best 'count' hits will be returned that meet the search criteria.

2.6 Program Object Functions

There are two lower-level functions which invoke program objects. It is expected that they will almost always be called from a PL/SQL wrapper layer, which would be responsible for packaging the communication between Oracle and the program object in a meaningful way. Hence, the default cartridge installation does not create operators for these two functions.

fprogob

```
function ddpackage.fprogob (name IN VARCHAR2_OR_CLOB,
        message IN VARCHAR2_OR_CLOB) => VARCHAR2_OR_CLOB
```

Communicates with a program object running on the Oracle server. The parameter 'name' is the symbolic name of the program object. The table c$dcischem.progob contains the mappings of symbolic names to actual executable programs. The function fprogob() proceeds as follows:

1. Looks up the symbolic name in the table c$dcischem.progob,
2. If the executable is not already running, starts the program object,
3. Parses the 'message' parameter into a sequence of strings,
4. Sends the sequence of strings to the program object,
5. Takes the returned sequence of strings and converts it into a newline delimited VARCHAR2_OR_CLOB,
6. Returns the VARCHAR2_OR_CLOB string.

Valid delimiters for the lines in the 'message' parameter include the normal line-termination characters for UNIX, Mac, and PC: '\n', '\r', and '\r\n'. The function properly handles all three termination cases. The returned VARCHAR2_OR_CLOB string is delimited by UNIX line-termination '\n'.

3. Extensible Indexes:

3.1 General Comments

The default installation (create.sql) creates public synonyms for the package 'ddpackage' and all of the operators. The one place where synonyms aren't allowed is in indextype specifications; indextypes must be refered to by their full names when creating indexes as a user other than c$dcischem (eg. create index <name> on small(smi) indextype is c$dcischem.ddblob).

All four of the extensible indexes create Oracle tables to maintain their required internal data. The ddexact, ddgraph, and ddrole indextypes use regular Oracle tables to store the data with a regular Oracle index for fast retrieval; the ddblob indextype uses an Oracle table with a rows containing BLOB data. These internal tables and indexes are maintained automatically by the index code; there is normally no need for a user or application developer to know about these auxillary tables.

There are two situations where the developer or DBA must pay attention to these auxillary tables. First, on index creation, the auxillary data tables and indexes are created in the users default tablespace, using default storage parameters. The auxillary table creation can be modified by using numerous optional parameters to the 'create index' command. Second, if an index gets corrupted, the developer or DBA may need to delete the auxillary data table and index manually.

3.2 Exact Lookup Indextype (ddexact)

This indextype supports the exact() operator. This is the least interesting index, as it implements a simple string comparison between the indexed column and the query. In that sense, the built in BTree index within Oracle has the similar functionality. The BTree index even supports the "greater than" and "less than" operators, which this index doesn't.

The main advantages of the ddexact indextype is that it supports the direct comparison of CLOB datatypes in SQL; the built-in BTree index does not. Also, in some situations, the BTree index has limits on the length of VARCHAR2 string which may be indexed.

This indextype uses an auxillary data table within Oracle to maintain its index data. This table is kept in sync with the base table in real time and strictly obeys the Oracle transaction model for all DML operations. The name of the auxillary data table is: <index name> || '_DDT'. An Oracle BTree index is also created. The index is named: <index name> || '_DDI'.

The index can be created on a VARCHAR2 or CLOB column with the following general statement:

```
SQL> create index <index name> on <table> ( <column> )
       indextype is c$dcischem.ddexact;
```

The valid operator comparison using the index are:

```
 exact(...) = 1
 exact(...) = 0
```

Note that the "!=" comparison does not use the index and will use the functional form. All other predicate comparisons will return the error: "C$DCISCHEM-024: Incorrect predicate for index operator".

This index makes no assumptions about the content of the indexed column. The applications are responsible for normalizing both the data in the indexed column and and queries to the exact() operator. This allows the ddexact indextype to properly handle both Unique and Absolute SMILES without complication. In the following example, the USMILES column is normalized using smi2cansmi(smiles, 0) and the ASMILES column is normalized using smi2cansmi(smiles, 1). Note also that when performing the exact() query on a column, the query is normalized using the same method as the column data.

```
SQL> create table test (id number, usmiles varchar2(4000),
       asmiles varchar2(4000));

SQL> desc test;
Name                                  Null?    Type
------------------------------- -------- ----------------------------
ID                                            NUMBER
USMILES                                       VARCHAR2(4000)
ASMILES                                       VARCHAR2(4000)

SQL> insert into test values (1234, smi2cansmi('Cl/C=C/Cl', 0),
                                    smi2cansmi('Cl/C=C/Cl', 1));

SQL> select * from test;

      ID  USMILES   ASMILES
--------  -------   -------
  1234    ClC=CCl   Cl/C=C/Cl

SQL> create index usmi_index on test(usmiles) indextype
       is c$dcischem.ddexact;
SQL> create index asmi_index on test(asmiles) indextype
       is c$dcischem.ddexact;

SQL> select * from test
       where exact(usmiles, smi2cansmi('Cl/C=C/Cl', 0)) = 1;
```

```
      ID  USMILES   ASMILES
--------  -------   -------
  1234    ClC=CCl   Cl/C=C/Cl

SQL> select * from test
       where exact(asmiles, smi2cansmi('Cl/C=C/Cl', 1)) = 1;

      ID  USMILES   ASMILES
--------  -------   -------
  1234    ClC=CCl   Cl/C=C/Cl
```

NOTE: Because this indextype accesses the base table during a index scan, operations which modify the base table based on the results of the index scan aren't allowed. This is similar to the case where a trigger on a column attempts to modify the column data. The message which one receives is the following:

```
SQL> delete from <table> where exact(smiles, 'CCN') = 1;

ERROR at line 1:
ORA-29903: error in executing ODCIIndexFetch() routine
ORA-29400: data cartridge error
ERROR ORA-04091: table C$DCISCHEM.T1 is mutating, trigger/function may
not see it
C$DCISCHEM-101: CARTRIDGE ERROR
ORA-06512: at "C$DCISCHEM.DDEXACT_IM", line 159
ORA-06512: at line 1
```

In order to perform this function, one must use two steps. One option would be:

```
SQL> create table <temptable> (zapcol) as
        select rowid from <table> where exact(smiles, 'CCN') = 1;
SQL> delete from <table> where rowid in
        (select zapcol from <temptable>)
```

3.3 Graph and Tautomer Indextype (ddgraph)

This indextype supports the graph(), tautomer(), usmiles() and asmiles() operators. It creates a tabular index of the graph and tautomeric information (hydrogen count and net charge) for rapid retrieval. The indexed search only requires an index lookup based on the graph of the query, followed by a secondary comparison of the net charge, hydrogen count, unique smiles, or absolute smiles, depending on the search type.

This indextype uses a special data table within Oracle to maintain its index data. This table is kept in sync with the base table in real time and strictly obeys the Oracle transaction model for DML operations. The name of the auxillary data table is: <index name> || '_DDT'. An Oracle BTree index is also created. The index is named: <index name> || '_DDI'.

The index can be created on a VARCHAR2 or CLOB column with the following general statement:

```
SQL> create index <index name> on <table> ( <SMILES column> )
       indextype is c$dcischem.ddgraph;
```

The valid operator comparison using the index are:

```
graph(...) = 1
graph(...) = 0
tautomer(...) = 1
tautomer(...) = 0
```

```
usmiles(...) = 1
usmiles(...) = 0
asmiles(...) = 1
asmiles(...) = 0
```

Note that the "!=" comparison does not use the index and will use the functional form. All other predicate comparisons will return the error: "C$DCISCHEM-024: Incorrect predicate for index operator".

Unlike the ddexact index, the ddgraph indextype always interprets the column as SMILES. Invalid SMILES are rejected. There is no concern with the ddgraph indextype as to whether the SMILES are Unique or Absolute, as stereochemical and isomeric information are discarded for the graph() and tautomer() lookups and the isomeric information is handled properly for the usmiles() and asmiles() lookups.

NOTE: As with the ddexact indextype, this indextype accesses the base table during a index scan. See the previous section on the limitations to performing modifications of the base table based on the results of an index scan.

3.4 Role Indextype (ddrole)

This indextype supports component level searches for exact molecules within molecules or reactions by role. It creates a tabular index of each component in every SMILES by role. Component level searches are implemented simply by a index lookup of the component, followed by validation of the role.

This indextype uses a special data table within Oracle to maintain its index data. This table is kept in sync with the base table in real time and strictly obeys the Oracle transaction model for all DML operations. The name of the auxillary data table is: <index name> || '_DDT'. An Oracle BTree index is also created. The index is named: <index name> || '_DDI'.

The index can be created on a VARCHAR2 or CLOB column with the following general statement:

```
SQL> create index <index name> on <table> ( <SMILES column> )
       indextype is c$dcischem.ddrole;
```

The valid operator comparison using the index are:

```
reactant(...) = 1
agent(...) = 1
product(...) = 1
component(...) = 1
```

Note that the "!=" comparison does not use the index and will use the functional form. All other predicate comparisons will return the error: "C$DCISCHEM-024: Incorrect predicate for index operator". Also note that, unlike the other three indextypes, in 4.81 the 'inverse' queries are not implemented (eg. component = 0). These will be added in a later release.

A component is defined as a single SMILES part of a larger molecule or reaction SMILES. A component is a useful concept for finding a molecule within a larger mixture, or finding a molecule which appears anywhere within a reaction. In all cases queries must be single component SMILES, else no hits are found. See the section on the component() operator for examples.

Like the ddexact index, the ddrole index makes no assumptions about the canonicalization of the SMILES in the index column. Hence, the column must be normalized by the application to the appropriate SMILES and

the query on the column must match that normalization (see the explanation re. ddexact above).

NOTE: As with the ddexact indextype, this indextype accesses the base table during a index scan. See the previous section on the limitations to performing modifications of the base table based on the results of an index scan.

3.5 SMARTS and Similarity Search Indextype (ddblob)

This index caches the SMILES and fingerprints in a table of binary large objects for rapid searching. Substructure and similarity searches are implemented through this blob-based image of the structural data. The blobs are persistant; they are automatically stored in the Oracle database and retrieved for processing. The blobs are kept synchronized with the base table in real time and strictly obey the Oracle transaction model for all DML operations. The name of the auxillary data table which contains the BLOB data is: <index name> || '_DDT'.

The index can be created on either a SMILES or fingerprint column, provided that the fingerprints are all the same size. If the index is created on a SMILES column, all seven searches are implemented on that column: contains(), isin(), matches(), tanimoto(), tversky(), euclid(), similarity(), and fingertest(). If the index is created on a fingerprint column, the four fingerprint-specific searches (tanimoto(), tversky(), euclid(), and fingertest()) are supported for that column.

The query for tanimoto(), tversky(), euclid(), similarity(), and fingertest() can be either a fingerprint or a SMILES. If a SMILES, it will be used to create a normal fingerprint of the appropriate size for the index which will be used for comparison. If the query is a fingerprint, the size of the fingerprint must match the size of the fingerprint in the column (or in the blob for a SMILES index). The indexes and operators never perform automatic folding of fingerprints during searches.

The index can be created on a VARCHAR2 or CLOB column with the following general statements:

```
-- Default internal fingerprint size of 512 bits

SQL> create index <index name> on
       <table> ( <SMILES column> ) indextype is c$dcischem.ddblob;

-- Specify internal fingerprint size of 64 bits

SQL> create index <index name> on
       <table> ( <SMILES column> ) indextype is c$dcischem.ddblob
       parameters ('fpsize=64');

-- Rather than generating the fingerprints, perform the initial index
   creation from the fp column

SQL> create index <index name> on
       <table> ( <SMILES column> ) indextype is c$dcischem.ddblob
       parameters ('initfpcolumn=<FP column>');

-- Rather than generating the scaffold data, perform the initial index
   creation from the sc column

SQL> create index <index name> on
       <table> ( <SMILES column> ) indextype is c$dcischem.ddblob
       parameters ('initsccolumn=<SC column>');

-- Use stored fingerprints and scaffold data to create the index
```

```
SQL> create index <index name> on
       <table> ( <SMILES column> ) indextype is c$dcischem.ddblob
       parameters ('initfpcolumn=<FP column> initsccolumn=<SC column>');

-- Fpsize taken from size of data in FP column (fpsize parameter, if it were
   specified, must match the size of data in the column)

SQL> create index <index name> on
       <table> ( <FP column> ) indextype is c$dcischem.ddblob;
```

The index creation command takes three optional parameters: fpsize, initfpcolumn, and initsccolumn. The 'fpsize' parameter is the number of bits that the blob index will use for fingerprint creation.

When indexing a SMILES column, the ddblob indextype will use the 'fpsize' parameter for the creation size for fingerprints within the internal blob index or a default value of 512 bits if the 'fpsize' is not specified.

When indexing a fingerprint column, all fingerprints within that column must match the 'fpsize' parameter. If the 'fpsize' parameter is not specified, the size of the first fingerprint encountered by the index code is used. Any fingerprints which do not match this size are rejected by the index.

The 'initfpcolumn' and 'initsccolumn' parameters give the names of the columns which contain the initial set of fingerprints and scaffolds for index creation on a SMILES column. Normally, when creating a ddblob index on a column of SMILES, fingerprints and scaffold data must be generated for every SMILES in the table. The 'initfpcolumn' and 'initsccolumn' options allow the first creation to occur by loading the data from a separate columns in the same base table.

These two options, if provided, are *only* used during index creation. They are only for convenience. Inserts, deletes, and updates of the SMILES column occur via the normal mechanism. Inserts, deletes, and updates of the given FP and SC columns do not affect the SMILES index after index creation (there is no ongoing transactional integrity between the given FP and SC columns and the index). The options are solely for index creation efficiency; the pre-generated fingerprints and scaffold data can be stored in the base table with the SMILES to decrease computational overhead for ddblob index creation. Fingerprints and scaffold data which are present in the given columns are assumed to be correct. Null data in the columns cause the fingerprints and scaffold data to be computed by the ddblob indextype code.

The valid operator comparison using the index are:

```
contains(...) = 1
contains(...) = 0
matches(...) = 1
matches(...) = 0
isin(...) = 1
isin(...) = 0
fingertest(...) = 1
fingertest(...) = 0
```

Tanimoto(), tversky(), and euclid() functions return values in the range of 0.0 - 1.0, inclusive. Similarity() returns numeric values based on the expression provided. These index operators support the full range of comparison functions: ">", ">=", "<", "<=", and "=".

Note that the "!=" comparison does not use the index and will use the functional form. All other predicate comparisons will return the error: "C$DCISCHEM-024: Incorrect predicate for index operator".

The index operators all take an optional numeric parameter which is the desired count of rows returned. The query will return up to the number of rows desired. The rows returned are those "nearest" to the query which match the predicate criteria. Note that "nearest" means larger values for tanimoto() and tversky() and smaller values for euclid(). In the case of contains() and isin(), the value used for the nearest comparison is the computed tanimoto() value. In the case of matches(), the value used is the number of bits set in each target hit.

For the similarity() operator, the default behavior is to 'figure out' whether the given expression is a distance (eg. smaller is better) or a similarity (eg. larger is better) from the expression. If needed, this can be overridden with an addition to the expression argument. The valid values are:

```
similarity(arg1, arg2, 'similarity=')
similarity(arg1, arg2, 'distance=')
```

Note that these are only used internally for the index implementation when the optional 'count' parameter is provided. This allows the index to return the 'count' best hits, taking into account the sense of the comparison.

```
-- Returns the ten best hits.  These will be the ten highest tanimoto
-- values in the table.

select * from test where tanimoto(smiles, 'NCCc1ccccc1', 10) >= 0.0;

-- Returns the ten best hits.  These will be the ten smallest euclidean
-- distance values.

select * from test where euclid(smiles, 'NCCc1ccccc1', 10) >= 0.0;

-- Returns the ten best hits, provided that they are all greater than
-- 0.9 tanimoto.  Otherwise, just returns those.

select * from test where tanimoto(smiles, 'NCCc1ccccc1', 10) > 0.9;

-- Returns the ten best hits below 0.5 tanimoto.

select * from test where tanimoto(smiles, 'NCCc1ccccc1', 10) <0.5;
```

For a simple table, one need not create a fingerprint column or use the fingerprint indexes. One only needs to create an index on a SMILES column to enable all structural searching functions.

```
-- We specified a preferred size for the internally generated fingerprints
-- as 1024 bits, overriding the default of 512 bits.  These fingerprints
-- are internal to the blob and aren't visible.

SQL> create index asmi_blob_index on test(asmiles) indextype is
       c$dcischem.ddblob parameters ('fpsize=1024');

SQL> select * from test where matches(asmiles, '[N;H1,H2]ccC=O') = 1;
SQL> select * from test where tanimoto(asmiles, 'c1ccccc1') > 0.8;

-- A fingerprint can be used directly as a query for tanimoto(),
-- euclid(), fingertest() and tversky() instead of a SMILES.  The
-- fingerprint size must match the size of the fingerprints used in
-- index creation.  In this case, 1024 bits.

SQL> select * from test where
       tanimoto(asmiles, smi2fp('OC(=O)CS', 0, 7, 1024)) > 0.8;
```

If one wants structural searches on multiple SMILES columns within a single table, then an index must be created on each SMILES column in the table. Furthermore, we can create and index fingerprint columns also. The index automatically will adjust fingerprint operations based on the size of the data in the fingerprint column. So, for example, if a column of fingerprints is added to our test table and populated:

```
SQL> alter table test add (fp varchar (15));
SQL> update test set fp = smi2fp(usmiles, 0, 7, 64);

-- This index, on the FP column, uses the size of the fingerprints in
-- the column (64 bits, in this case) in index creation.

SQL> create index fp_blob_index on test(fp) indextype is
       c$dcischem.ddblob;

-- As with an index on a SMILES column, a SMILES query or a fingerprint
-- can be used as a query for tanimoto(), euclid(), fingertest(),
-- similarity(), and tversky().

SQL> select smi2fp('OC(=O)CS', 0, 7, 64) from dual;

SMI2FP('OC(=O)CS',0,7,64)
-------------------------
IM9vA4w127g.2

-- The following two queries give identical results.

SQL> select * from test where tanimoto(fp, 'IM9vA4w127g.2') > 0.8;
SQL> select * from test where tanimoto(fp, 'OC(=O)CS') > 0.8;
```

There are some additional subtlties explained using the following example queries:

```
-- This screens and searches using the index; internally it uses 1024 bit
-- fingerprints.  The output is generated using the functional form, at
-- its default of 512 bits.  Remember that the functional forms will use a
-- fingerprint size of 512 bits unless one or both of the parameters is a
-- fingerprint.  The tanimoto values output by the functional form may not
-- match our search criteria based on the folding-related differences
-- between 1024-bit and 512-bit fingerprints.

SQL> select smiles, tanimoto(asmiles, 'NCCc1ccccc1') from test
       where tanimoto(asmiles, 'NCCc1ccccc1') > 0.8;

-- This screens and searches using the index; internally using 1024 bit
-- fingerprints.  The output is generated using the functional form; this
-- time calculated at 1024 bits.  By passing a fingerprint of 1024 bits
-- to the functional form of tanimoto(), we force its use of 1024 bits.
-- In this case, the tanimoto values output by the functional form will
-- exactly match those used by the index.

SQL> select smiles, tanimoto(asmiles, smi2fp('NCCc1ccccc1', 0, 7, 1024))
       from test where tanimoto(asmiles, 'NCCc1ccccc1') > 0.8;

-- This searches using the index f1, using the 128 bit fingerprints from
-- the actual column for comparison.  Again, the tanimoto values computed
-- by the functional form are calculated at 512 bits and will likely not
-- match those used by the index for the query.

SQL> select smiles, tanimoto(fp, 'NCCc1ccccc1') from test
       where tanimoto(fp, 'NCCc1ccccc1') > 0.8;
```

3.6 Alter Index Command

The alter index command provides the ability to perform maintainence operations on the indexes themselves. At this time the only supported maintainence operations are: renaming an index and rebuilding an index.

Indexes of all four extensible indextypes can be renamed with the *alter index ... rename* command:

```
SQL> alter index <oldname> rename to <newname>;
```

Renames an extensible index and the auxillary tables which are associated with it.

The *alter index ... rebuild* command causes the index to be dropped and recreated from scratch. For the ddblob indextype, the special parameters clause "parameters ('compress=true')" causes the ddblob indextype to be repacked, rather than rebuilt. For the ddblob indextype, when a row is deleted from the base table on which a ddblob index exists unused space is left in the index data structure. Over time, this unused space will cause a decrease in search performance. After many delete or update operations it is recommended that the index either be rebuild or compressed. The compress operation is typically faster than a full rebuild. Generally, after 20% of the rows in a table have been deleted or updated it is advantagous to compress a ddblob index.

```
SQL> alter index <indexname> rebuild;
```

Rebuilds the index. Reclaims unused space, rebuilds all hashtables, checks consistancy of the index.

3.7 Import / Export of Daycart Indexes

Oracle supports the import and export of domain-specific indexes. The Daycart indexes can be exported and imported along with other database objects.

The default behavior of Oracle is that when one exports a table which has a domain index created on one of it's columns, the *description* of that domain index is stored in the export file with the base table. On import, Oracle will attempt to recreate the domain index on the target machine after the base table data has been loaded.

In order for this to work properly, several guidelines must be followed:

- Derived data tables that Daylight uses do not need to be exported explicitly. All of the tables with names ####_DDT should **not** be exported with the base table data. Only the base tables should be exported.
- If one performs a user or database export, the derived tables will be skipped automatically by Oracle. This is appropriate; the derived data will be regenerated during the import on the target instance.
- the Daylight cartridge must be installed on the target Oracle instance before beginning the import operation.

Import and export of Daycart indexes is generally transparent if one doesn't attempt to deal explicitly with the derived tables. Note than an import operation will include index creation time, so long runtimes during the import step are not unusual.

3.8 Index Creation Options

A large number of optional parameters can be passed as part of a Daycart index creation. These are passed in

the 'parameters' field of the create index function:

```
SQL> create index my_daycart_index on my_table (my_column)
        indextype is c$dcischem.ddexact
        parameters ('table_tablespace=users logging=false');
```

The parameters are a whitespace-separated list of name=value pairs. The double-quote character can be used to include whitespace in a parameter value.

There are several parameters which are specific to the ddblob indextype and which control the fingerprint generation for that indextype. Those parameters are discussed in the ddblob indextype section. The parameters discussed here are used to modify the storage parameters for the auxillary tables and indexes which are created. Each of the parameters described in this section result in the modification of "CREATE TABLE" or "CREATE INDEX" commands within the Daycart code.

The following options apply to all Daycart indextypes (ddexact, ddgraph, ddrole, ddblob) and modify the characteristics of the auxillary data table created for the index:

table_tablespace=<value>
: Sets the tablespace for the INDEX_DDT table on creation. See the Oracle documentation for "CREATE TABLE ... TABLESPACE <value> ..." for more information.

table_logging=<TRUE|FALSE>
: Sets the logging/nologging value for the INDEX_DDT table on creation. A value of TRUE enables logging, FALSE disables logging. See the Oracle documentation for "CREATE TABLE ... LOGGING for more information.

table_cache=<TRUE|FALSE>
: Sets the cache/nocache value for the INDEX_DDT table on creation. A value of TRUE enables caching, FALSE disables caching. See the Oracle documentation for "CREATE TABLE ... CACHE for more information.

table_initrans=<value>
table_pctfree=<value>
table_pctused=<value>
table_maxtrans=<value>
: Sets initrans, pctfree, pctused, and maxtrans values for the INDEX_DDT table on creation. See the Oracle documentation for "CREATE TABLE" for more information on each of these physical attributes and their use.

table_storage=<value>
: Sets storage clause for the INDEX_DDT table on creation. Since the storage clause often contains multiple delimited values this clause often requires additional quoting. Use double-quotes within the parameters value. For example:

```
CREATE INDEX my_index on my_table(my_column)
indextype is c$dcischem.ddexact
parameters ('table_storage="minextents 2 maxextents 5"');
```

The following options apply to the ddexact, ddgraph, and ddrole indextypes and modify the characteristics of the auxillary index which is created as part of the Daycart index:

index_tablespace=<value>
: Sets the tablespace for the INDEX_DDI index on creation. See the Oracle documentation for "CREATE INDEX ... TABLESPACE <value> ..." for more information.

index_logging=<TRUE|FALSE>

Sets the logging/nologging value for the INDEX_DDI index on creation. A value of TRUE enables logging, FALSE disables logging. See the Oracle documentation for "CREATE INDEX ... LOGGING for more information.

index_initrans=<value>

index_pctfree=<value>

index_maxtrans=<value>

Sets initrans, pctfree, pctused, and maxtrans values for the INDEX_DDI index on creation. See the Oracle documentation for "CREATE INDEX" for more information on each of these physical attributes and their use.

index_storage=<value>

Sets storage cluase for the INDEX_DDI index on creation. Since the storage clause often contains multiple delimited values this clause often requires additional quoting. The proper quoting syntax is double-quotes for the index_storage value. For example:

```
SQL> CREATE INDEX my_index on my_table(my_column)
       indextype is c$dcischem.ddexact
       parameters ('index_storage="minextents 2 maxextents 5"');

internally results in:

CREATE INDEX MY_INDEX_DDI ... STORAGE ( minextents 2 maxextents 5 ) ...
```

The following options only apply to the ddblob indextype and modify the characteristics of the BLOB datatypes which are created as part of the Daycart index:

lob_tablespace=<value>

Sets the tablespace for the BLOB datatypes on creation. See the Oracle documentation for "CREATE TABLE ... LOB (...) STORE AS (TABLESPACE <value>) ..." for more information.

lob_cache=<TRUE|FALSE>

Sets the cache/nocache value for the BLOB datatypes on creation. A value of TRUE enables caching (and forces logging), FALSE disables caching.See the Oracle documentation for "CREATE TABLE ... LOB (...) STORE AS (CACHE) ... " for more information. The default behavior, for efficiency sake, is that logging is disabled during index creation.

lob_logging=<TRUE|FALSE>

Sets the logging/nologging value for the BLOB datatypes on creation. A value of TRUE enables logging, FALSE disables logging. See the Oracle documentation for "CREATE TABLE ... LOB (...) STORE AS (LOGGING) ... " for more information. The default behavior, for efficiency sake, is that logging is disabled during index creation.

lob_storage=<value>

Sets storage cluase for the BLOB datatypes on creation. Since the storage clause often contains multiple delimited values this clause often requires additional quoting. The proper quoting syntax is double-quotes for the index_storage value. For example:

```
SQL> CREATE INDEX my_index on my_table(my_column)
       indextype is c$dcischem.ddblob
       parameters ('index_storage="minextents 2"');

internally results in:

CREATE TABLE MY_INDEX_DDT ... LOB (...) STORE AS ( STORAGE (minextents 2) ) ...
```

3.9 Partitioned Domain Indexes

Beginning with version 4.91, DayCart supports indexing of range partitioned tables within Oracle. Partitioned tables are handled automatically by the Daycart indextypes. All DDL operations on partitioned tables are supported. These include adding, removing, exchanging, merging and splitting partitions. For each partition of a range-partitioned table within Oracle, Daycart maintains a domain index and backing table named: <name>_<part>_ddt. Any changes to the base table are automatically propagated to the index table(s).

When creating a Daycart index against a range-partitioned table, use the "local" clause on the indextype to indicate that a partitioned index is desired. One can also provide names for the index partitions:

```
SQL< create index idx_test_blob on test(isomer)
     indextype is ddblob local
     (partition pe1, partition pe2, partition pe3)
```

The names "pe1", "pe2" and "pe3" are used for the creation of the backing index tables (eg. *_DDT). When a DayCart partitioned indextype is modified using the "alter index" command, the underlying *_DDT table is modified by DayCart. The user does not need to perform any other maintanence within DayCart.

Individual partitions within a range-partitioned index can be rebuilt with:

```
SQL> alter index idx_test_blob rebuild partition pe2;
```

This is required if an index partition becomes unusable. When partitions of the base table are split or merged, the indexes are marked as unusable and a rebuild is required.

Individual partitions within a range-partitioned index can be renamed with:

```
SQL> alter index idx_test_blob rename partition pe2 to pe2x;
```

Most other operations on partitioned tables are transparently handled by Daycart. More information about range partitioned tables and Oracle indextypes can be found in the Oracle documentation at: http://otn.oracle.com.

One note for Oracle version 10g:

Oracle 10g now allows for a short recovery time of dropped tables and indexes. Basically this works just like a recycle bin on a desktop computer. Dropped tables and indexes are kept in an "Oracle internal Recycle bin state" until the space is needed again. This means that while the space used is free for new create statements, the tablespace will show the space taken by objects which start with "$BIN...". You can issue the SQL command "purge recyclebin" to remove these ghost objects from the object tables. More information can be found in the Oracle documentation at http://otn.oracle.com.

4. Using Program Objects

This section describes the configuration and use of program objects from within Oracle.

Program objects are stand-alone executable programs which communicate via standard input and standard output using the "Pipetalk" protocol (See the Daylight Theory Manual for more details). The cartridge supports the ability to execute external program objects on the same machine as the Oracle server, and to communicate with the program objects in a robust, general way.

The table c$dcischem.progob is the dictionary of program objects available to the Oracle server. In order to use a program object, it must be defined in this dictionary table.

```
SQL>  desc progob
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
 NAME                                               VARCHAR2(60)
 PATH                                               VARCHAR2(4000)
 ARGS                                               VARCHAR2(4000)
```

For example, consider the clogp program object. The code for the clogp example is included in the CONTRIB subdirectory. It can be defined to the system with:

```
SQL> insert into progob values
   ('clogp', '/oracle/progob/clogptalk.sh', NULL);
```

This defines a program object with the symbolic name 'clogp'. It corresponds to the absolute path: '/oracle/progob/clogptalk.sh', which must be an executable program object. In this case, the arguments to the program object (whitespace delimited) are NULL.

The file '/oracle/progob/clogptalk.sh' looks like the following:

```
#!/bin/sh

DY_ROOT=/usr/local/daylight/v481
export DY_ROOT

DY_LICENSEDATA=/usr/local/daylight/dy_license.dat
export DY_LICENSEDATA

LD_LIBRARY_PATH=/usr/lib:$DY_ROOT/lib
export LD_LIBRARY_PATH

$DY_ROOT/bin/clogptalk
```

When Oracle executes the program object, the environment is empty. Hence, all required environment variables must be defined in a shell wrapper before calling the actual program object.

Since program objects are executed under the Oracle userid, a strict security policy must be defined and obeyed to prevent access to unauthorized Oracle privileges. The policy, checked and enforced by the cartridge code *before* executing any external program object, is the following:

1. If the program object is executable and SUID, then it is OK. SUID requires privileges of the destination UID, so this means that whoever turned on the SUID bit has either root or owner privilege on the file.
2. If the program object is executable by and owned by Oracle, and the directory is not group or world writeable, then OK.
3. Otherwise, fail.

Program objects are persistant on the server and are executed on a per-user-session basis. The first time a program object is accessed within a user session it is started, and the program object will continue to run until the user session is terminated. If the program object crashes during the user session, the cartridge will restart it and retry the transaction. Because of the ability to restart program objects, program objects ideally should be stateless and communications to the program object from Daycart should assume that the program is stateless.

Program object communication through the ddpackage.fprogob() function is via VARCHAR2 or CLOB datatypes. The string types can contain one or more delimited lines of data. Because of this line-oriented data communication, most program objects will require a PL/SQL wrapper function to convert from Oracle datatypes to line-oriented data and to parse returned results. Continuing the clogp example, consider a simple function which will calculate clogp:

```
create function fclogp (sosdata in varchar2) return number
 as
  v1 varchar2(4000);
  rc number;
  off1 number;
  off2 number;
 begin
  v1 := c$dcischem.ddpackage.fprogob('clogp', sosdata);
  off1 := instr(v1, ' ', 1, 1);
  off2 := instr(v1, ' ', 1, 2);
  rc := to_number(substr(v1, off1, off2 - off1));
  return rc;
 end;
```

Executing the fprogob() function directly using the clogptalk program object results in a full line of data being returned from the program object for each SMILES:

```
SQL> select ddpackage.fprogob('clogp', 'c1ccccc1') from dual;

DDPACKAGE.FPROGOB('CLOGP','C1CCCCC1')
--------------------------------------------------------------------------------
c1ccccc1 2.142 0 LogPstar: 2.13
```

The wrapper function fclogp() simply parses out the second field, converts it to a number, and returns it. The wrapper function hides the details from the user and allows the function to return the desired numeric value.

```
SQL> select fclogp('c1ccccc1') from dual;

FCLOGP('C1CCCCC1')
------------------
             2.142
```

More complex examples (eg. MOLFILE -> SMILES conversion) are included in the CONTRIB directory.

5. Optimizer Support

The cartridge will provide estimates of the costs of extensible index queries to the Oracle optimizer. The Oracle optimizer uses these cost estimates for its decision making with respect to the execution plan for a given SQL query. The execution plan chosen by Oracle can have a dramatic effect on the time and resources required to perform a particular query. The cartridge has the ability to provide selectivity and resource estimates of varying precision to Oracle, depending upon the configuration of the database index.

The default cartridge installation includes the code for performing cost estimates, but does not enable the cartridge optimizer code. All costs for index queries default to zero. Any Oracle query will preferrentially execute the extensible index query first.

INSTALLATION

In order to activate the optimizer code for the Daylight cartridge, one must associate statistics with the package and indextypes with the following code:

```
SQL> connect c$dcischem/secret
SQL> associate statistics with packages ddpackage using ddoptimizer;
SQL> associate statistics with indextypes ddexact using ddoptimizer;
SQL> associate statistics with indextypes ddgraph using ddoptimizer;
SQL> associate statistics with indextypes ddrole using ddoptimizer;
SQL> associate statistics with indextypes ddblob using ddoptimizer;
```

To test the optimizer code, execute the verification SQL script in $DY_ROOT/dcischem/TEST/dd_optimizer_test.sql. The results can be checked against dd_optimizer_test.ref_out.

```
sqlplus -s 'c$dcischem/secret' <dd_optimizer_test.sql > test.out
diff test.out dd_optimizer_test.ref_out
```

There should be no differences found.

Once the optimizer code is enabled, one can view the cost estimates provided to Oracle and the execution plan chosen using the "explain plan" command in SQL.

The command "analyze index <indexname> compute statistics" will collect and store information about the size of the index table for the ddgraph, ddrole, and ddexact indextypes. This size information will then be used to refine the cost estimates provided to the Oracle optimizer. The ddblob indextype does not require a separate "analyze index" command; the blob index always knows its exact size.

Note that the cost and selectivity estimates are not going to be reasonable in all cases; developers may find that forcing a particular execution order will result in better query performance. Oracle SQL supports a number of hints which allow the developer to control the execution order. See the section on "Optimizer Hints" in the Oracle SQL Reference manual for more details.

DAYCART API DESCRIPTION: FINDING THE COST

The optimizer features allow the user access to the data used within the optimizer to determine SQL statement execution paths.

To understand the execution path of a SQL statement, the user needs to understand the actual costs involved when DayCart functions are called. The costs referred to here are the actual CPU and I/O costs, or at least the best estimation of those.

The following functions allow the user to query the actual CPU and IO costs and selectivities provided to Oracle by the ddoptimizer package. These are the values which the SQL execution engine uses internally to decide between multiple execution plans.

```
FUNCTION ddoptimizer.fgetsel (
        operator_name IN VARCHAR2,
        query_string IN VARCHAR2_OR_CLOB,
        direction IN NUMBER,
        hitcount IN NUMBER) => NUMBER
```

Returns the query selectivity as a percentage, the range is 1 - 100. Operator_name is the index function (eg. 'exact', 'contains'). Query_string is the SMILES, SMARTS, or fingerprint query. The direction indicates whether to find matches (1) or non-matches(0), and the hitcount, for 'nearest', queries, is the number of hits desired.

```
FUNCTION ddoptimizer.fgetfunccpucost (
        operator_name IN VARCHAR2) => NUMBER
FUNCTION ddoptimizer.fgetfunciocost (
        operator_name IN VARCHAR2) => NUMBER
```

Returns the CPU cost for the functional execution path for the given operator.

```
FUNCTION ddoptimizer.fgetindexcpucost (
          index_name IN VARCHAR2,
          operator_name IN VARCHAR2,
          selectivity IN NUMBER,
          hitcount IN NUMBER) => NUMBER
FUNCTION ddoptimizer.fgetindexiocost (
          index_name IN VARCHAR2,
          operator_name IN VARCHAR2,
          selectivity IN NUMBER,
          hitcount IN NUMBER) => NUMBER
```

These return the CPU and IO costs for the index execution path for the given operator, against the given index, where index_name is the Daycart index being used, operator_name is the index function, selectivity is the percentage selectivity of the query (from fgetsel()), and hitcount is the number of hits desired for 'nearest' queries, or zero.

DAYCART API DESCRIPTION: CUSTOMIZING THE COST

In addition to finding the estimated costs involved, you can also adjust those settings as needed. Because all databases, data, and usage is different, the user is the person best equipped to set the actual costs. These functions provide the user with the tools to adjust the costs involved and maintain an optimal SQL statement execution plan.

The ddpackage.finfo() and ddpackage.fsetinfo() functions are general utilities to get and set parameters within Daycart. Parameters can be modified on a per-session basis. The following parameters apply to the ddoptimizer package:

```
-ddoptimizer_index_cpu_factor: Scaling factor controlling the
    CPU cost via the index path (default: 100)
-ddoptimizer_index_io_factor: Scaling factor controlling the
    IO cost via the index path (default: 100)
-ddoptimizer_func_cpu_factor: Scaling factor controlling the
    CPU cost via the functional path (default: 100)
-ddoptimizer_func_io_factor: Scaling factor controlling the
    IO cost via the functional path (default: 100)
```

These parameters can be modified before every query, if necessary. Unfortunately they can not be made to apply to parts of a compound query. Consider the following query:

```
select * from my_table where
  contains(smi, 'c1ccccc1') = 1
```

```
and
matches(smi, '[CX3]') = 1
```

The scaling factors apply to both the contains() and matches() queries with the SQL session; it is not possible to use the scaling factors to bias the execution plan to choose either predicate preferentially.

EXAMPLES OF USE:

The following examples assume a ddblob and ddgraph index on the table my_table as follows:

```
create index my_graph_index on my_table(smi) indextype is ddgraph;
create index my_blob_index  on my_table(smi) indextype is ddblob;
```

For these examples, my_table held 101227 rows.

Query 1: (basic query)

```
select * from my_table where contains(smi, 'c1ccccc1') = 1;
```

The following ddoptimizer functions will provide the complete statistics for this query:

```
sel  := ddoptimizer.fgetsel('contains', 'c1ccccc1', 1, 0);
fcpu := ddoptimizer.fgetfunccpucost('contains');
fio  := ddoptimizer.fgetfunciocost('contains');
icpu := ddoptimizer.fgetindexcpucost('my_blob_index', 'contains', sel, 0);
iio  := ddoptimizer.fgetindexiocost('my_blob_index', 'contains', sel, 0);
```

The following results are with scaling factors set to 100 across the board:

```
Cost of select... contains('c1ccccc1'):       12
CPU cost of contains() function:              700
IO  cost of contains() function:              28
CPU cost of my_blob_index index               30492
IO  cost of my_blob_index index               175
```

And now if we set the scaling factors to 200 across the board by using the commands:

```
SQL> select ddpackage.fsetinfo('ddoptimizer_index_cpu_factor=200') from dual;
SQL> select ddpackage.fsetinfo('ddoptimizer_index_io_factor=200') from dual;
SQL> select ddpackage.fsetinfo('ddoptimizer_func_cpu_factor=200') from dual;
SQL> select ddpackage.fsetinfo('ddoptimizer_func_io_factor=200') from dual;
```

The results change for the new scale:

```
Cost of select... contains('c1ccccc1'):       12
CPU cost of contains() function:              1400
IO  cost of contains() function:              56
CPU cost of my_blob_index index               60983
IO  cost of my_blob_index index               349
```

It can also be useful to specify the functional form of a query in order to eliminate consideration of the index path for the execution plan. Consider the following query:

```
SQL> select * from <table> where
        graph(smiles, 'c1ccccc1') = 1 and
        contains(smiles, 'C') = 1;
```

By default, (assuming that both required indexes exist) this query will use indexes for both the graph and contains search, and then will merge the results. In most cases, it would be much more efficient to perform the graph index search, then perform the functional version of contains() on the handful of hits returned from the graph search. The following query will force the contains() operation to occur functionally:

```
SQL> select * from <table> where
        graph(smiles, 'c1ccccc1') = 1 and
        ddpackage.fcontains(smiles, 'C') = 1;
```

In order to disable the cartridge optimizer code, any computed statistics must be dropped with "analyze index <indexname> delete statistics". Then, The cartridge optimizer code can be disabled with the following commands:

```
SQL> connect c$dcischem/secret
SQL> disassociate statistics from indextypes ddblob;
SQL> disassociate statistics from indextypes ddgraph;
SQL> disassociate statistics from indextypes ddrole;
SQL> disassociate statistics from indextypes ddexact;
SQL> disassociate statistics from packages ddpackage;
```

Once the cartridge optimizer code is disabled, cost estimates revert to their default values of zero cost.

6. Error Codes

This section lists the possible error codes returned from the cartridge and diagnositic actions. Under most error conditions Oracle will return several different error codes after a DayCart error. These usually include ORA-06512 and ORA-29400 in the error stack along with the relevant C$DCISCHEM-### error.

6.1 General Errors

C$DCISCHEM-001: Unable to Get Session MEMORY
: Internal error. Try quitting the user session and restarting. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-002: Debug level out of range: 0 - 9
: The parameter passed to the fsetdebug() function was out of range.

C$DCISCHEM-003: FAILED: License not available
: A cartridge function failed because of a license error. Check that the user has the daycart role granted to him, that the user has select privilege on the c$dcischem.license table, and that the license table has a valid cartridge license.

C$DCISCHEM-004: Return string too long
: For reaction transform functions, the enumerated set of molecules or reactions overflowed the returned string type. To avoid this error, coerce the function to return a CLOB type by calling it with a CLOB for the first parameter (smiles).

C$DCISCHEM-005: License not available for Partitioning option
: The available Daycart license does not include partitioning capabilities. An enterprise-level Daycart license is required..

C$DCISCHEM-006: Can't read ptable
: The ptable is unable to be read for a conversion operation. Check for the existance and select permission on the c$dcischem.ptable table.

C$DCISCHEM-007: Conversion failed
: The conversion function failed.

6.2 Invalid Arguments to Cartridge Functions

C$DCISCHEM-021: Not a valid QUERY: %.*s
: The query string was not a valid SMARTS, SMILES, or fingerprint, as required.

C$DCISCHEM-022: Not a valid input SMILES: %.*s
: The SMILES parameter was not parseable as a valid molecule or reaction.

C$DCISCHEM-023: Not a valid fingerprint: %.*s
: The fingerprint parameter was not parseable as a valid molecule or reaction.

C$DCISCHEM-024: Incorrect predicate for index operator
: The comparison operation is not supported for this index. For example, the predicate "contains(smiles, query) = 0" is not recognized by the index code. See the documentation for each indextype for the supported predicate forms.

C$DCISCHEM-025: Can't execute contains/isin/matches query on FP column
: The comparison operation is not supported for this index. The contains(), isin(), and matches() operations are only supported for SMILES-based ddblob indexes.

6.3 Index Errors

C$DCISCHEM-031: Index name too long; maximum 27 chars
: The name given for an index creation or index alter is too long. Oracle typically allows 31 character names, however since DayCart appends four characters to the name to use for it's backing data tables and indexes, DayCart indexes must have shorter names.

C$DCISCHEM-032: Backing table creation failed: %s

C$DCISCHEM-033: Backing table rename failed: %s

C$DCISCHEM-034: Backing table alter failed: %s
: An operation on backing data table (name is given) failed. This may be because of permissions or because a table with that name already exists. Typically, additional messages in the error stack will help isolate this problem.

C$DCISCHEM-035: Backing index creation failed: %s

C$DCISCHEM-036: Backing index rename failed: %s
: The creation of the backing data index (name is given) failed. This may be because of permissions or because an index with that name already exists. Typically, additional messages in the error stack will help isolate this problem.

C$DCISCHEM-035: Partition alter failed: (%s)
: The alter index command of a range partitioned table failed. Typically, additional messages in the error stack will help isolate this problem.

6.4 Internal Index Errors

C$DCISCHEM-041: Unable to Get Session Handle
: Internal error. Try quitting the user session and restarting. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-042: Unable to QUERY table

C$DCISCHEM-043: Unable to Insert Row, ROWID = %s

C$DCISCHEM-044: Unable to Delete Row

C$DCISCHEM-045: Refresh number corrupt
: Possibly corrupt index. Try quitting the user session and restarting. Also, drop and recreate the index. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-046: Unable to Convert Start parameter

C$DCISCHEM-047: Unable to Convert Stop parameter

C$DCISCHEM-048: Cannot Generate KEY

C$DCISCHEM-049: Cannot Set Context Value

C$DCISCHEM-050: Cannot Assign value to Self

C$DCISCHEM-051: Unable to get KEY value

C$DCISCHEM-052: Unable to get max number of rows for fetch

C$DCISCHEM-053: Unable to append DUMMY rowid

C$DCISCHEM-054: Unable to append rowid
: Internal error. Try quitting the user session and restarting. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-055: Query of base table failed

C$DCISCHEM-056: Fetch of base table failed
: Possibly corrupt index or base table. Try quitting the user session and restarting. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-057: Access of blob data failed
: Possibly corrupt index. Try quitting the user session and restarting. Also, drop and recreate the index. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-058: Access of blob header failed

C$DCISCHEM-059: Blob write failed

C$DCISCHEM-060: Rebuild of index failed

C$DCISCHEM-061: Insert of index row failed

C$DCISCHEM-062: Index table access failed
: Possibly corrupt index. Drop drop and recreate the index. Report this problem to Daylight if it reoccurs.

C$DCISCHEM-063: Index search aborted with I/O error
: This typically occurs when the client performs an abort of a ddblob-based search (with ctl-c in SQL, or killing the client process). It is a harmless side-effect of the extproc cleanup routines and can be ignored if it is indeed related to an aborted search. If not due to an aborted search, try quitting the user session and restarting. Also, drop and recreate the index. Report this problem to Daylight if it reoccurs.

6.5 Program Object Errors

C$DCISCHEM-071: Couldn't stat() program object

The program object was not found. Check that the value of "path" in the table c$dcischem.progob matches the absolute path to the desired program object.

C$DCISCHEM-072: Couldn't stat() directory

The directory which supposedly contains the program object was not found. Check that the value of "path" in the table c$dcischem.progob matches the absolute path to the desired program object.

C$DCISCHEM-073: Program object is not executable

The program object does not have executable privilege set.

C$DCISCHEM-074: Program object directory must be owned by Oracle

For security reasons, the program object must be owned by Oracle or be SUID. Check the file permissions for the program object.

C$DCISCHEM-075: Program object directory group or world writable

For security reasons, the directory containing the program object must not be group or world writable. Check the file permissions for the directory.

C$DCISCHEM-076: Access of c$dcischem.progob table failed

The table c$dcischem.progob was not readable by the current user. The user must have select privilege on the table. Check that the role daycart has been granted to the user.

C$DCISCHEM-077: Program object not found in table c$dcischem.progob

The symbolic name for the program object (the first argument to fprogob()) was not found in the c$dcischem.progob table. Every program object requires a row in the c$dcischem.progob table specifying the absolute path to the program object and any arguments to the program object. The "name" parameter must match the first argument.

C$DCISCHEM-078: dt_alloc_program() failed

The program object didn't start. Verify that the desired program object runs properly from the command line before retrying. Also, make sure that any required environment variables are set by the program object itself.

C$DCISCHEM-079: dt_alloc_program() failed for restart

The program object crashed, and Oracle wasn't able to restart it. Try quitting the user session and restarting.

C$DCISCHEM-080: Progob failed dt_converse

The program object didn't respond properly to a message from Oracle. Verify that the desired program object runs properly from the command line before retrying.

6.6 Generic Errors

C$DCISCHEM-101: CARTRIDGE ERROR

This is a generic error. Typically, there was an Oracle error during a cartridge function. The error stack should provide additional information.

C$DCISCHEM-102: CARTRIDGE WARNING

This is a generic warning. Typically, there was an Oracle warning during a cartridge function. The error stack should provide additional information.

C$DCISCHEM-103: CARTRIDGE INFO

This is a generic informational message. Typically, there was an Oracle informational message during a cartridge function. The error stack should provide additional information.

7. Tuning Hints

This section contains notes on Cartridge Tuning.

- INIT.ORA:

  There are several INIT.ORA parameters which affect query performance for the cartridge. These primarily impact the ddblob index, which is the most resource-intensive of the four indextypes.

  db_file_multiblock_read_count: The number of blocks read per I/O request in a sequential scan. Larger is better, up to a value of 32.

  db_block_buffers: Number of buffers in the buffer cache. The blob index uses the buffer cache, so if the buffer cache is larger than the blob index, then no physical disk I/O is required for a search. The size of the blob data can be obtained for an index by examining the log data for a search at the debug_level of 9 (select c$dcischem.ddpackage.fsetdebug(9) from dual;). An entry of the form "Data LOB size for search: 9726184" gives the LOB size in bytes.

  log_buffer: The size of the redo log buffer. Index creation is a long transaction which benefits from a large log_buffer. Also, see the section on logfile sizes.
- LOB caching:

  The default behavior of the ddblob indextype is to enable LOB caching for all ddblob indexes created in the system. This means that the Oracle Server will attempt to cache BLOB data in memory like normal tabular data. Provided that the SGA is sufficiently large (db_block_buffers), searches over ddblob indexes will not require any physical disk I/O.

  The downside of LOB caching is that BLOB data will displace other tabular data from the buffer cache during a search. On a system with small memory availability, this can cause all queries to slow down substantially as the BLOB data can fill the buffer cache for every search. In this scenario, it is preferred to disable BLOB caching on a per-index basis.

  Each ddblob index has an auxillary table associated with it. That auxillary table has two BLOB columns. The following SQL commands will alter the CACHE status for the auxillary table:

```
SQL> create index test_index on test(asmiles) indextype is c$dcischem.ddblob;

Index created.

-- the auxillary table is named <index_name> || '_DDT'

SQL> desc test_index_ddt;

 Name                    Null?    Type
----------------------- -------- --------------
 N1                               NUMBER
 N2                               NUMBER
 HASH                             BLOB
 DAT                              BLOB

SQL> alter table test_index_ddt modify lob (dat) (nocache);

Table altered.
```

  One can turn caching back on with "alter ... modify lob (dat) (cache)". Note that the HASH BLOB is used for inserts, deletes, and updates. It is small compared to the DAT BLOB, so should remain cached for best overall performance. The DAT BLOB is larger and is used for searching, so select its cache behavior based on overall throughtput and tuning requirements.

LOGFILES:

Index creation time for the ddblob index is sensitive to logfile availability as it does a lot of I/O which needs to be logged. I added a total of four logfile groups to minimize checkpoint blocking. If you see entries in the instance logfile admin/<sid>/bdump/alert_<sid>.log like:

```
  Current log# 1 seq# 15 mem# 0: /oracle/db/oradata/dev/redo02.log
Thread 1 cannot allocate new log, sequence 16
Checkpoint not complete
  Current log# 1 seq# 15 mem# 0: /oracle/db/oradata/dev/redo02.log
```

during index creation, then increasing the number and size of your redo logs will improve index creation and modification performance substantially.

```
SQL> select * from v$logfile;

    GROUP# STATUS    MEMBER
---------- ------- ---------------------------------
         1         /oracle/db/oradata/dev/redo01.log
         2         /oracle/db/oradata/dev/redo02.log
```

Something like the following sequence will work. This will give you a total of 16 MB of log file space spread over four separate

```
-- Create two new logfiles

SQL> alter database add logfile group 3
       ('/oracle/db/oradata/dev/redo03.log') size 4000k;
SQL> alter database add logfile group 4
       ('/oracle/db/oradata/dev/redo04.log') size 4000k;

-- Advance the current log file into the new ones

SQL> alter system switch logfile;
SQL> alter system switch logfile;

-- Get rid of the original two logfiles

SQL> alter database drop logfile group 1;
SQL> alter database drop logfile group 2;

-- Remove the files redo01.log and redo02.log
   from the system and re-create them.

SQL> quit
$ rm /oracle/db/oradata/dev/redo01.log
$ rm /oracle/db/oradata/dev/redo02.log

$ sqlplus sys/pw

SQL> alter database add logfile group 1
       ('/oracle/db/oradata/dev/redo01.log') size 4000k;
SQL> alter database add logfile group 2
       ('/oracle/db/oradata/dev/redo02.log') size 4000k;

SQL> select * from v$logfile;

    GROUP# STATUS    MEMBER
```

- ```
  ---------- ------- --------------------------------
             1         /oracle/db/oradata/dev/redo01.log
             2         /oracle/db/oradata/dev/redo02.log
             3         /oracle/db/oradata/dev/redo03.log
             4         /oracle/db/oradata/dev/redo04.log
  ```
- MULTITHREADING EXTPROC ON SOLARIS:

  Beginning with version 4.93, Daycart supports multithreaded searches for contains(), matches(), and isin() queries. These three in particular are the queries where the performance tends to be limited by CPU availability.

  On Solaris, in order to get the best performance with multithreading searches it is desirable to use the memory management library which Sun provides for this purpose (libmtmalloc.a). The 'mtmalloc' library is optimized by Sun for multithreaded applications. It minimizes inter-thread contention for shared heap data (eg. malloc() calls). By default, Oracle does not use this library when building extproc.

  One can relink the extproc executable to include libmtmalloc.a on Solaris and see a significant improvement in performance for searches (20+%). The steps are as follows:

  1. Find the make.log file in your Oracle installation. Typically it is in $ORACLE_HOME/install/.
  2. In make.log, locate the block of messages pertaining to extproc and copy the link command into a separate script file. Typically, a string search for 'extproc' will find the command. The desired line starts with:

     ```
     /usr/ccs/bin/ld -o ..../extproc -L ...
     ```
  3. Edit the link line and add the argument '-lmtmalloc' *before* the sysliblist entry. Change the name of the output target to extproc_mt.
  4. Run the link command.
  5. Change the SQLNet listener.ora file to use the new binary and restart SQLNet.

  An example link command, with the edits highlited, follows:

  ```
  /usr/ccs/bin/ld -o /oracle/products/o92e/bin/extproc_mt \
           -L/oracle/products/o92e/rdbms/lib/ \
           -L/oracle/products/o92e/lib/ \
           -dy /oracle/products/o92e/lib/WS6U1/crti.o \
           /oracle/products/o92e/lib/WS6U1/crt1.o  \
           /oracle/products/o92e/rdbms/lib/hormc.o \
           /oracle/products/o92e/rdbms/lib/defopt.o \
           /oracle/products/o92e/rdbms/lib/homts.o  \
           -lagtsh -lpls9 -lplp9 -lthread  -lclntsh \
           -lxml9 -lcore9 -lunls9 -lnls9 \
           /oracle/products/o92e/lib/libgeneric9.a \
           -lmtmalloc \
           `cat /oracle/products/o92e/lib/sysliblist` \
           -R /opt/SUNWcluster/lib:/oracle/products/o92e/lib  \
           -Y P,:/opt/SUNWcluster/lib:/usr/ccs/lib:/usr/lib  \
           -Qy -lc -laio -lposix4 -lkstat -lm \
           /oracle/products/o92e/lib/WS6U1/crtn.o -lvsn9
  ```

Previous Index
  ```

8. Release Notes

Version 4.95:

- Added support for Oracle 11g and 11gR2.
- Added initsccolumn parameter to ddblob index creation, allows rapid creation of ddblob index from stored scbits and fingerprints using the smi2scbits() function.
- Fixed bug on [A] primitive on SMARTS.
- Fixed buffer overflow bug in role searching.

Version 4.94:

- The ddblob indextype now computes additional statistics about each molecule/reaction. These are used to improve screening for contains() and matches() searches.
- Added the exit_on_fault, log, and session_tag options.
- Fixed bugs in multithreading.

Version 4.93:

- Overloaded the fdayconvert() function and dayconvert() operator so the unused "type" parameter is optional
- Added new logging options. The finfo()/fsetinfo() option 'log' controls the disposition of log data: into a session-level queue (which can be retrieved by ddpackage.fgetlog(), and/or the central logfile, which by default is /tmp/extproc.log, but can be overridden with the listener.ora environment variable DAYCART_LOGFILE.
- Added new scaffold-based structure analysis for screening. Improves contains() and matches() search performance for some queries.

Version 4.92:

- Implemented a time-based interrupt and abort mechanism for long-running ddblob index searches. New session-level parameters are "timeout_interrupt", "timeout_abort", and "timeout_progress".
- Added overloaded form of matches() function with an optional count parameter for the number of hits to return.
- Added matchcover() function.
- Added new conversion functions which allow the specification of an alternate ptable file.

Version 4.91:

- Daycart now supported on Oracle 9.2 and higher only.
- Conversion toolkit functionality.

- Reaction transform functionality.
- Support for range-partitioned tables.
- New optimizer tuning functionality.
- smi2pmw() function - high-precision molecular weight.
- user_similarity() function and index operators. Allows searching with a user-provided similarity measure.
- Added a mug/coffee demo user under contrib.
- Added a mechanism to get the toolkit error queue from Oracle. Helpful in debugging SMILES parsing problems or other failures.

Version 4.82:

- NOTE: a script (upgrade_481-482.sql) is provided. It performs the package replacement and table creation needed to add the vcs-oriented functions to Daycart. Existing Daycart indexes do NOT need to be dropped and recreated during the upgrade.
- New functions fvcs_desalt(), fvcs_normalize() to assist structure normalization across vendor databases.
- New function fsetinfo() which allows setting session-level Daycart parameters.
- Bug fix for matches() query against certain cyclic structures (fingerprint toolkit).

Version 4.81:

- Supported on Solaris 7, 8, 9. 32- and 64-bit databases. Red Hat Linux 7.x, 32-bit databases.
- Supported on Oracle 8.1.7, 9.0, 9.2. Standard and Enterprise editions.
- Added 'alter index ... rename' for all indexes.
- Added 'alter index ... rebuild' for ddblob index. This reclaims unused index space.
- Added fcomponent(), component() and the component search to the ddrole index.
- Added fpartnorm() function and partnorm() operator.
- The toolkit error queue is now dumped to the /tmp/extproc.log file.
- Fixed a bug; tanimoto() search on an empty, indexed table caused a failure.
- Tested, verified and documented import and export of Daylight indextypes.
- Added fisin(), isin() and the isin search to the ddblob index.
- Added huge numbers of index-creation options in the parameters field of the create index. These map directly to the Oracle physical_attributes and storage clauses used internally in "CREATE TABLE" and "CREATE INDEX" commands.
- Added initfpcolumn option for ddblob creation on a SMILES column.
- Much better predicate handling for searching.
- CLOB datatypes now legal for most string operations.
- The fversion() and fhostid() functions has been deprecated in favor of a more general function ddpackage.finfo().
- Fixed a bug where the inability to write to the logfile (/tmp/extproc.log) caused crashes of extproc.

Version 4.72:

- Version 4.72 added support for SGI Irix 6.4+ and Red Hat Linux 6.2.
- Fixed a bug on some Solaris architectures; the tanimoto() index function would intermittently fail.
- Unified floating point handling to use doubles internally. This eliminated architecture differences in floating point results.
- Fixed a bug with fprogob(), where the arguments were incorrectly truncated to 4000 characters.

Version 4.71:

- Version 4.71 was the first fully supported version of the cartridge. It was supported on Sun Solaris 2.7 and Oracle 8.1.5 - 8.1.7, Standard and Enterprise editions.