HUMAN HEALTH

ENVIRONMENTAL HEALTH

Oracle Cartridge 19.1

Installation and Configuration Guide

PerkinElmer®
For the Better

Last Updated: December 07, 2020

Table of Contents

Introduction to Oracle Cartridge 19.1

Overview

Chemical structure information can be difficult to manage effectively without using special software, and is even more challenging to incorporate in today's increasingly sophisticated and technologically complex business logic and security models. Fortunately, Oracle data cartridge architecture provides a mechanism to extend the capabilities of Oracle databases. Data cartridges allow domain-specific data - including chemical data - to be directly integrated into existing Oracle databases. The **Oracle Cartridge 19.1** utilizes this technology to bring state-of-the-art chemical intelligence to Oracle databases, allowing the manipulation of chemical structure and reaction data from within Oracle.

Oracle Cartridge 19.1 also allows the storage of chemical structures in an Oracle database, and provides the ability to perform fast retrieval of that data in chemically meaningful ways, to the benefit of everyone in the enterprise of Chemical and Biological Research Informatics and Knowledge Management, including bench chemists. application developers, and database developers.

Oracle Cartridge 19.1 Requirements

This chapter explains the hardware and software required to install and use the Oracle Cartridge **19.1**.

Hardware and Software Considerations

The Oracle Cartridge installation program must be run on a Client computer running the Microsoft Windows Operating System on an Intel or Intel-compatible computer. That computer must also have the 64-bit Oracle Client installed.

An Oracle database or instance resides on the Host computer, which can be either: an Intel- or Intel-compatible computer running on the Microsoft Windows or Linux Operating Systems.

Client Computer Hardware and Software

The Cartridge installation program runs on a Client computer under Microsoft Windows. The following versions of Microsoft Windows (64-bit only) are supported:

- Windows Server 2016 Standard
- Windows Server 2019 Standard

The Cartridge supports the following Oracle Standard or Enterprise Editions versions:

- 12c (12.1.0.2) both single instance and Oracle Multitenant option
- 19c

The client computer through which the Cartridge is installed and the corresponding server computer must run compatible versions of Oracle. Ensure that your Oracle client is of the same or an older version than the database.

Note: *These requirements are only relevant during installation.*

Host Computer Hardware and Software Requirements

The Oracle Cartridge supports the following Host database server configuration. In most cases, the Oracle Cartridge will work on later OS versions than those listed, but not on earlier versions.

- Windows Server 2016 Standard
- Windows Server 2019 Standard

Note: *Only 64-bit Operating Systems are supported.*

- The Cartridge is tested on 64-bit Red Hat Enterprise Linux, version 7.

Note: *On 64-bit Operating Systems, a 64-bit version of Oracle is normally expected with GCC version 4.6.0 or later installed.*

System User Privileges

The **Oracle Cartridge Installation** program requires the DBA to have `Read-Write` privileges on the Oracle file directories located on the **Oracle** Host computer system.

To perform the installation, a user requires:

- Write privileges on both the Client, and **Oracle** Host computer for access to the Oracle file directories.
- `SQL*Plus` access to the **Oracle** Host computer Oracle instance, by using either the `SYS` account or through another user account with comparable access privileges.

Note: *Some Oracle DBAs (Database Administrators) are reluctant to allow any access privileges in the* `$ORACLE_HOME` *to anyone but trusted personnel. As a result, you might require the assistance of the ODBA at your firm to resolve access privilege issues.*

Consequently, most of the installation should be performed by or in cooperation with your company **ODBA**.

Host Computer Memory Requirements

The **Oracle** Host computer must have sufficient internal memory and enough swap space to effectively run the Oracle version (refer to Oracle documentation) at your site. This database server must also have sufficient disk storage space so the chemical databases can be handled with fast substructure searches by the **Oracle Cartridge**.

Database Server Requirements (If planned to be used with PerkinElmer Enterprise applications)

Hardware Requirement	Minimum Specification	Recommended Specification
Memory (RAM) Processor	8GB 2x2.0 GHz or higher quad core processor (8 cores total)	64 GB 4x3.0 GHz or higher quad core processor (16 cores total)
Free Space Required on Hard Drive	150 GB	250 GB + 1 GB per User per Year

In many applications, you will find that there is no optimal size – increasing the cache further will improve performance further, so your goal is to choose a size that is compatible with your resources that yields satisfactory performance.

Host Computer Disk Space Requirements

The Host computer on which the Oracle databases reside require available hard disk space capacity of between 100 - 200 MB/user/year (100 MB minimum) assuming typical rates of database growth. Domain indexes created by the Cartridge require around 10K - 20K bytes per indexed structure, depending on indexing options and the sizes of the structures.

Communications Between the Client and Oracle Host Computer

It is recommended that you use Oracle Network Services (`SQL*Net` or `Net`) and `FTP` (**File Transfer Protocol**) to communicate with the Oracle Host computer when performing an installation in `LINUX` operating systems.

However, it might become necessary for you to manually copy files from the **Client** computer running the installer to the Oracle Host computer.

This would be necessary if:

- The **Oracle** Host computer is a server not enabled for `FTP` file transfers (perhaps for security reasons).
- You are installing on a Windows RAC, for nodes other than the first one.

- You are performing a manual installation. This might be done for reasons of security or to permit Cartridge files to be placed in a non-standard location. Some DBAs require review of scripts before they are installed in a production server, and they might not permit Cartridge support files to be placed in the Oracle directory tree. Also, some RAC installations prefer to place a single copy of Cartridge support files in shared cluster or network storage rather than install duplicate copies on every node.

To manually copy files from the Client computer running the installer to the Oracle Host machine in order to complete an installation:

1. Identify the local `temp` directory specified during installation. By default, this is typically named: *C:\Documents and Settings\<username>\Local Settings\Temp*. In newer versions of Windows, the temp directory is: *C:\Users\<username>\AppData\Local\Temp*.
   A dialog box will display the installation details (from directory, to directory, and filenames).
2. From that directory, locate the four files to be copied, as follows:
   a. `CSChemFinderCustomElements.txt`
   b. `CsCartridge.dll` (for Windows, or `.so` for various `LINUX` Operating Systems)
   c. `NameToStructure.dat`
3. Copy the files in to the following directory:

   $ORACLE_HOME/lib/CsCartridge on `LINUX`

   OR

   $ORACLE_HOME\bin\CsCartridge on Microsoft Windows
4. Once the files are copied to the correct directory on the **Oracle** Host computer, close the dialog box to complete the installation.
5. After this copy is performed, the installation is complete; and if possible, the installer program continues with the post-installation automatic testing.

Other Scenarios Where Manual File Copying is Necessary

There are other scenarios where files need to manually be copied and edited. The installer will display a dialog box that provides the full details of each situation. Situations such as:

- When the **Oracle** Host computer cannot be contacted via `SQL*Net`.When this situation occurs, a dialog box displays fields for you to identify the Oracle instance and machine type. An `SQL` script is then written to the local `temp` directory. You will copy this script to the Host computer, where you will run it in `SQL*Plus`.
- The system libraries on the **Oracle** Host computer are a version that does not support the Cartridge shared library. Instructions will be provided abou updating their libraries.
- The temporary configuration may prevent other Oracle services at your site from functioning properly. After proper operation of the Cartridge has been demonstrated, it will be necessary to merge the configurations in a way that permits both the Cartridge and other services to operate together.

Installing Oracle Cartridge 19.1

Overview

The PerkinElmer Oracle Cartridge has an installation program, named `SetupCartridge.exe`, the icon of which follows:

See "Setup Problems and Error Handling " on page 71, if you face difficulty in starting the installation program with SetupCartridge.exe.

Prior to running the installation program, you should consider which of the four installation modes you are planning to use for the installation process.

1. **Full Install Mode** – The Full Install radio button is set by default. Previous Oracle Cartridge installations are completely replaced with this new installation, including stored procedures and files. Any existing index created by a previous installation of the Cartridge becomes orphaned. While the indexes still exist, they are unusable; as a result, all indexes must be dropped and recreated using the `drop index` command discussed in "Index Creation and Maintenance" on page 26.
2. **Partial Install Mode (object library replacement only)** – **Generally, this option should not be used.** This mode does not affect stored procedures or objects in the Oracle database. It only replaces the shared library that contains the external functions and procedures. All previously created indexes are still valid and can still be used without any further action. A Partial install must only occur when reinstalling the exact same version of the Oracle Cartridge as was previously installed.
3. **Manual installation (generate SQL script)** – **Generally, this option should not be used.** This does not perform a complete installation, it only creates an `SQL` script. When this script is executed by using an appropriate tool (such as `SQL*Plus`), it results in the same database objects as a normal install. Sites with high security requirements can opt to create the `SQL` script, review it, and then manually execute it.

When selecting the manual install mode, and leaving the password for the `SYS` user empty, the Oracle Cartridge does not connect to any Oracle service. A dialog box appears. The user is prompted to select a platform and Oracle version for which to generate a script, and the installer extracts the support files, and places them in the directory specified in the installation dialog.

> ***Note:*** *It is recommended only for trusted personnel, because the generated script may need manual modification. The `LIBRARY` statement in the generated `SQL` script must be changed to reflect the actual environment prior to execution.*

4. **No install, test an existing installation** – This post-installation test confirms that the Oracle client to sever communications and the external process agent are all configured properly. It is performed automatically as part of the Full and Partial install modes and does not need to be run a second time in those cases.

Using the Oracle Cartridge Installation Program

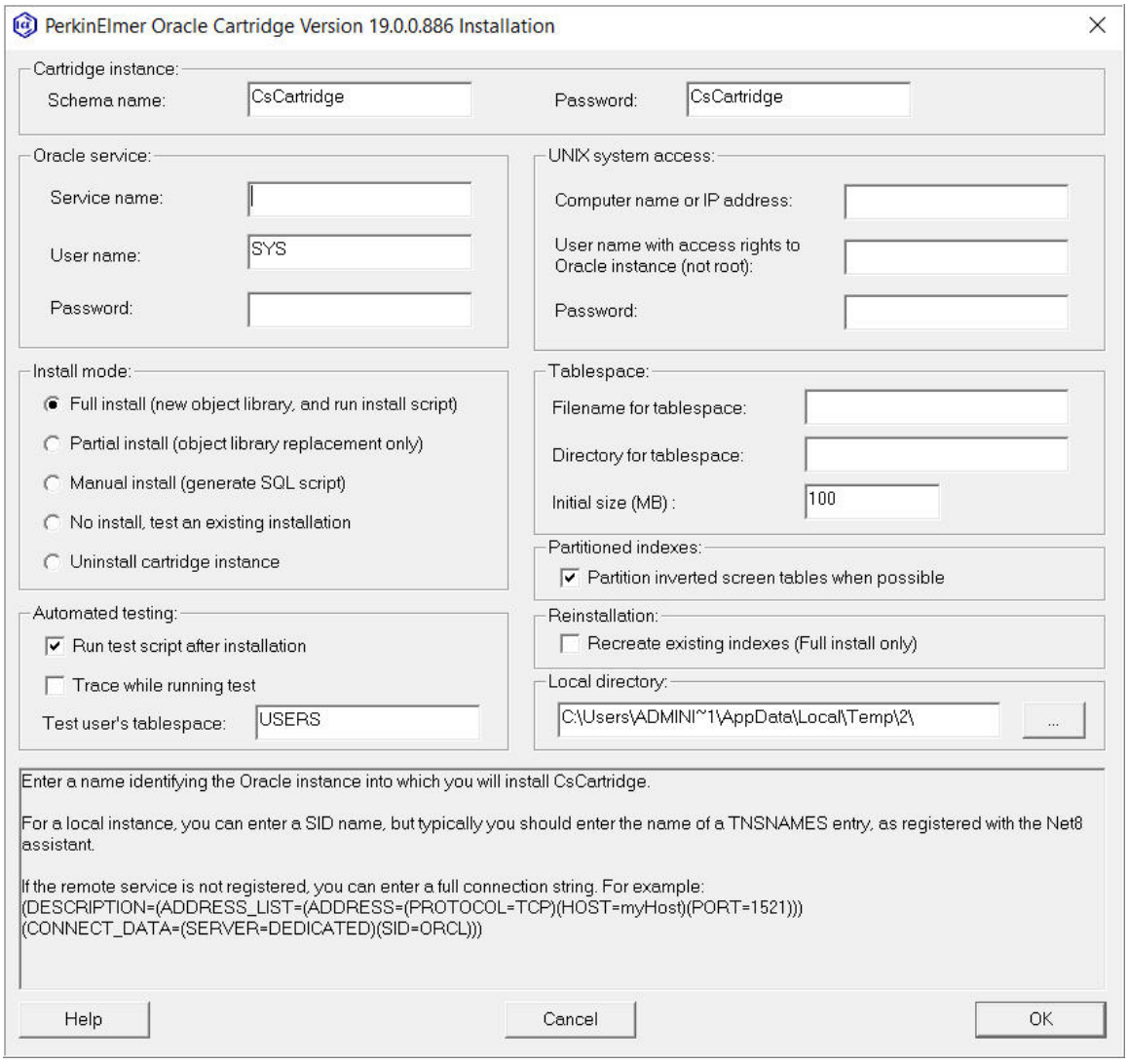
This section explains how to run the installation program on a personal computer running an Oracle Client with access to the Oracle server.That personal computer will be referred to as the Client computer. See "Oracle Cartridge Requirements" for more information about the system requirements for the Client.

To install PerkinElmer Oracle Cartridge for the first time:

1. Log in to the Client computer.
2. Double-click the **SetupCartridge.exe** file icon.

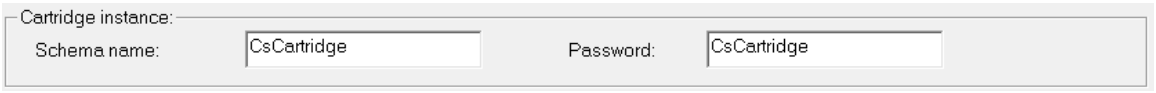
3. The **PerkinElmer Oracle Cartridge** installer screen then displays on your computer desktop. When you click on a text box, the information box at the bottom of the screen displays help information about the value you must enter.

Installation Program Screen



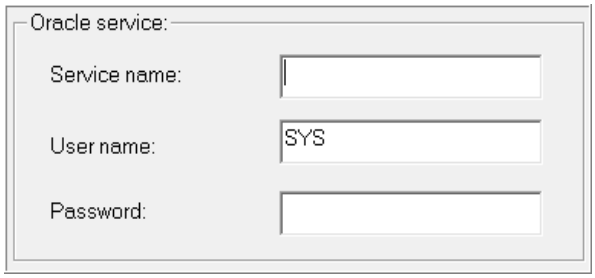
Cartridge Installer Screen Field Descriptions

The default name of the Cartridge schema is "CsCartridge". Enter a non-default value if you plan to perform multiple installations in the same Oracle database:



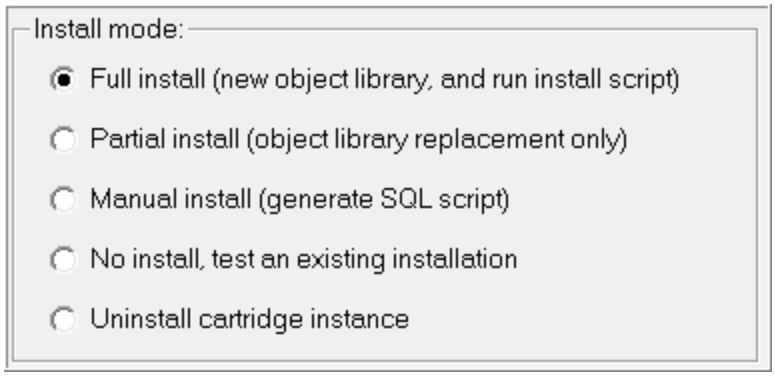
Schema Name	Each Cartridge instance (that is, Oracle schema) should be assigned a unique name. For example: if you plan to test different versions of the Oracle Cartridge, name one instance **CsCartridge**, and then another similarly (for example **CsCartridgeTest)** so they are named uniquely. Changes to the schema name should always be carefully considered. For example, the default behavior of other PerkinElmer applications (including E-Notebook and Inventory) is to assume that the schema name will be CsCartridge as shown here. If the schema name is changed, then similar changes will likely be required in all other applications that communicate with that instance of the Oracle Cartridge.
Password	The Cartridge schema will be protected by this password. For the security of your Cartridge installation, it is recommended that you replace the default “CsCartridge” value with a password of your own choice. This password is not needed by applications that utilize Cartridge operators. It is used in the `SQL CONNECT` statement that allows the administrator to log into the Cartridge schema to modify settings and objects within it.

The Oracle Service dialog box has three fields used by the installation program to specify the Oracle connect string.

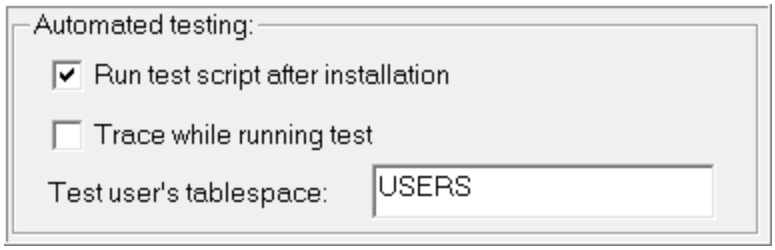


Service Name	The **Oracle Service Name** should be entered as it is named by the **Oracle Net Manager**. All the Oracle objects will be stored in the database referred to by this particular Service Name. This name will typically be an instance name defined in the client tnsnames.ora file. An unambiguous abbreviation or machine name may also be entered; the full service name will be confirmed through a pop-up message.
User name	The Oracle User who performs the installation. By default the `SYS` user, because only the `SYS` user can grant the privileges that need to be granted to the Cartridge schema. An `SQL` script named `CreateCSCInstallUser.sql` is provided along with the Oracle Cartridge installation program to create a user other than `SYS` – just for the purpose of performing the installation - thus making the process of granting privileges to the user more transparent.

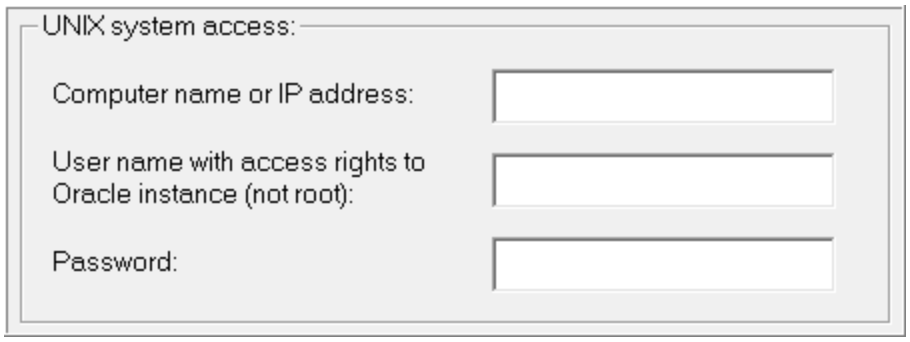
Password	Enter the password for the **Oracle User** in this field.



Full Install: (New object library & run install script)	The Full Install radio button is set by default. Previous Oracle Cartridge installations are completely replaced by this new installation, including stored procedures and files. This option should be used even when installing to `LINUX` systems without ftp access. In such cases, the Full installation will pause until the required files are copied to the server manually, and then may be resumed to complete the post-installation testing.
Partial Install: (object library replacement only)	This mode does not affect stored procedures or objects in the Oracle database. It only replaces the shared library containing the external functions and procedures.
Manual Install: (Generate `SQL` script)	Generally, this option should not be used unless the Client computer is unable to communicate with Oracle on the Host computer, or your Oracle DBA requires review of any scripts installed into the Oracle database. This is not a complete installation; this mode only generates an `SQL` script. When this script is executed by using an appropriate tool (such as `SQL*Plus`), it results in the same database objects as a normal install. The primary purpose of this option is to allow an Oracle DBA review the scripts before they are installed into the Oracle database or to permit modification of the script; for example, to enable installation of Cartridge files in a non-standard location. If there is no need for such a review, the Full Install mode should be used - unless the Client computer is unable to communicate with Oracle on the Host as discussed above.
No Install, test an existing installation	This runs the post-installation test, which can be performed after a manual installation of the Oracle Cartridge to ensure that the Oracle Host has been configured properly.
Uninstall cartridge instance	This can be used to uninstall the Cartridge instance. This mode removes all Cartridge objects, including domain indexes from the Oracle database, and removes the Cartridge files from the server. However, tables and CSCUSER schema are not dropped.

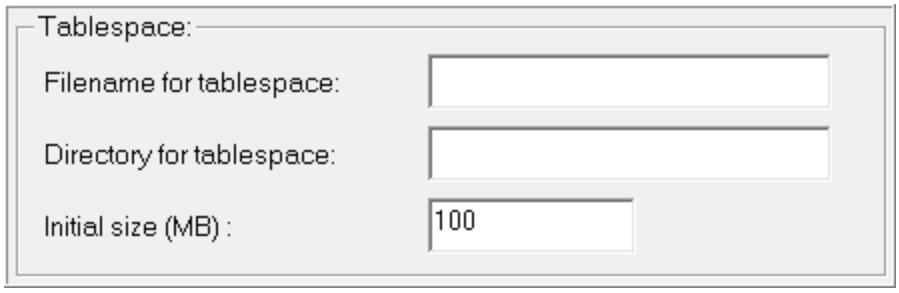


Run test script after installation	This check box allows users to enable or disable the automated test the installer runs after a successful installation. The test is enabled by default. It should be disabled when an Oracle Service is not properly configured to run external procedures. Since the Oracle Cartridge will not function unless the Oracle Host can run external procedures, this item should normally remain checked. A log of the test results is saved in a file `Cartridge Install Test Log.txt` in the specified "Local Directory".
Trace while running test	This check box allows you to perform troubleshooting during an installation. If a former installation or the automated test failed, and the Cartridge is not in working condition, then the tracing may reveal some clues to the problem. The default is no tracing.
Test user's tablespace	The default tablespace is `USERS`. A schema named `CSCUSER` is created in this tablespace to perform the test.

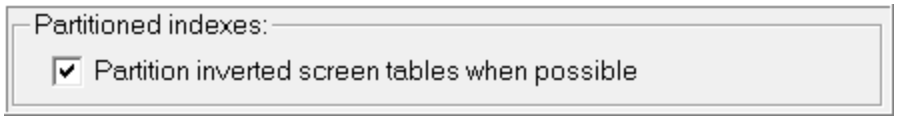


LINUX System Access	Use this group of fields to create an `ftp` connection to the `LINUX` Operating System on the Host computer. For installations on Microsoft Windows operating systems, or for `LINUX` installations where the server does not run an `ftp` service, these fields should be left empty. When a full install or a partial install is performed, the Oracle Cartridge installer program must copy the `CsCartridge.so` shared library and support files to the `LINUX` server. It does this as a Client of the `ftp` service run by the `LINUX` system. To use the `ftp` service the installer program requires the Computer name or IP address of the computer, a User Name and a Password.
Computer name or IP address	The name of the computer or the IP address of the computer that runs the Oracle service. For an RAC installation, where Cartridge files are not placed in a shared file system accessed by all nodes, enter a comma-separated list of RAC nodes onto which files will be copied.

User name with access rights to Oracle instance (not root)	The user name for a `LINUX` account that has write access privileges to the directories where the `Oracle` software is located. This is usually Oracle, but can also be any other user with the required access rights. For an RAC installation, the same login credentials will be used for all the nodes.
Password	The correct password for this user name.

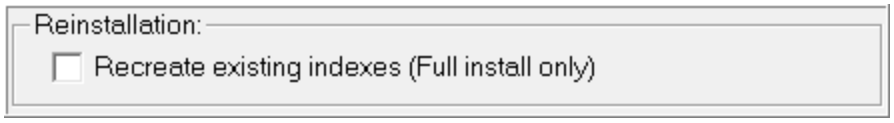


Filename for tablespace:	Normally this box should be left blank. A default filename will be used. This filename will consist of "T_" prefixed to the Cartridge instance schema name for non-ASM file storage, or is generated automatically by ASM. An ASM alias name can be entered here.
Directory for tablespace	If the directory is not specified, then the tablespace file is created in the Oracle file directory location, which is usually in the `$ORACLE_HOME` directory tree. If a fully qualified directory name is used here, then it must be ensured that the Oracle user specified above has the access rights necessary to create the specified directory and file. For an ASM installation, enter a partial or full ASM storage specifier here; the diskgroup name alone is typically sufficient. For example: +RACDB_DATA1. In general, you can allow ASM to generate the remainder of the path and filename for the appropriate datafile type.
Initial size (MB)	The default is 100 MB. This tablespace holds all Oracle Cartridge objects including, by default, all indexes created by the Cartridge.

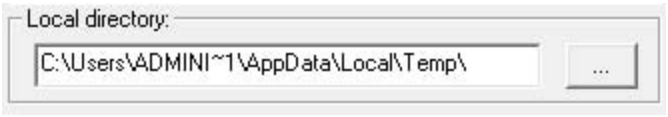


Partition inverted screen tables when possible	This check box is checked by default to allow use of partitioning when available in the Oracle instance. This is generally the case with Oracle Enterprise Edition, but the Cartridge cannot determine whether partitioning is actually licensed at a site. If it is not licensed, the box should be unchecked during Cartridge installation. The Cartridge partitions inverted-screen tables, when possible, in order to obtain better search performance, which is most noticeable in very large tables. If the box is unchecked, partitioning will not be used even when available. This is a global setting, and it overrides any attempt to enable partitioning during `CREATE INDEX` through `PARAMETERS('USEPARTITIONS=YES')`. In fact,

	`'USEPARTITIONS=NO'` is the only meaningful setting for this parameter, to prevent use of partitioning in a specific index at a site where it would otherwise be permitted.

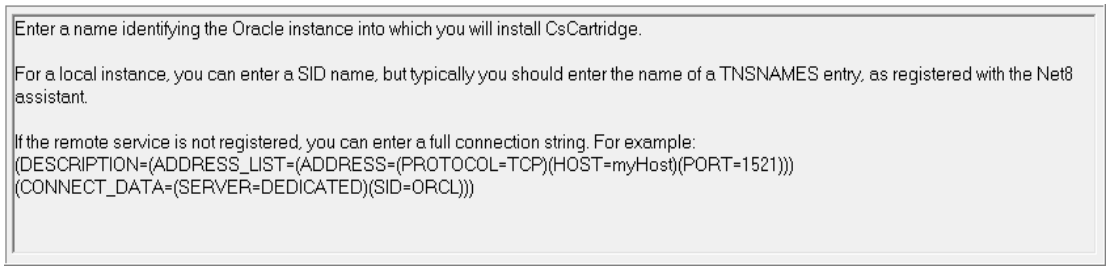


Recreate existing indexes (Full install only)	This check box may be checked during a reinstallation of the Cartridge. When checked, the Cartridge creates an `SQL` script to drop all the existing indexes created by a previous version of the Cartridge, and recreate them with the same parameters as they had previously. It can execute that script if requested. Carefully review the newly generated `CREATE INDEX` statements; either options might have changed since the previous installation was performed or the state of an index might have been changed after creation. The default is not to recreate existing indexes regardless of the fact that all indexes created an older version of the Cartridge must be dropped and recreated. Index creation is a time-consuming process, and great care should be used when the Cartridge is reinstalled.

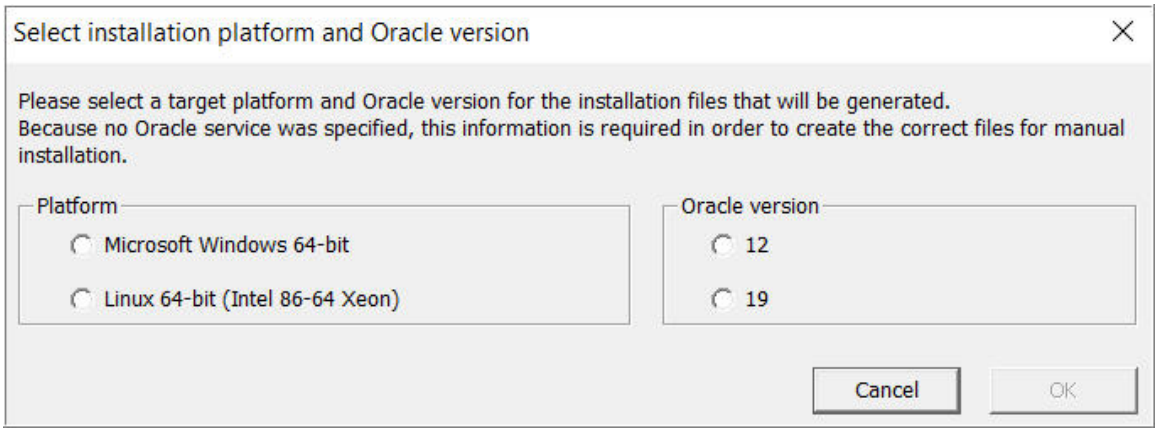


Local Directory	This is the directory where the Oracle Cartridge installer program stores all the files created during installation (the install log, install script, and files). When instructed to do so, manually copy files from this directory to the computer running the Oracle Service, if the installer was unable for some reason to copy those files. The default is:
	
	On Windows 2012 and 2008 Servers:
	
	`C:\Users\ADMINI~1\AppData\Local\Temp\2\`
	
	On a Windows 2003 Server:
	
	`C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\1`
	
	The folder mentioned above is the TEMP directory of the Windows user (the Administrator) running the installer. The user performing the installation must have write privileges to this directory. The directory chooser button to the right allows you to navigate to a different directory.

The **Status** field displays progress messages during the course of the installation. It also displays help when you click to enter information in a text field.



- Click the **OK** button. Installation begins when you click the **OK** button. However, if you had not specified Oracle service name and selected **Manual install (generate SQL script)** in the **Install mode** section, then the **Select platform and Oracle version to install on** dialog box appears when you click the **OK** button:

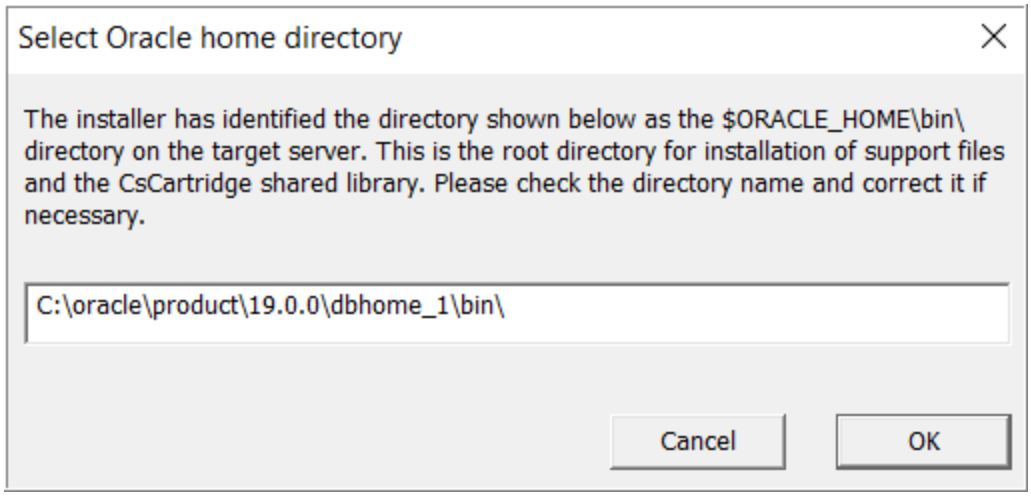


- Select a radio button to specify the operating system on your Oracle Host database server.
- Next, select a radio button to specify the version of the Oracle Server you will be using on the Host computer to run the Oracle Cartridge.

Note: *This method is useful if the files for manual installation are to be generated without being able to connect to the target system.*

- Click the **OK** button to generate the installation script.

The **Select Oracle Home Directory** box appears after you click the **OK** button, unless in your particular installation mode the Client computer is unable to communicate with the Host Oracle database.



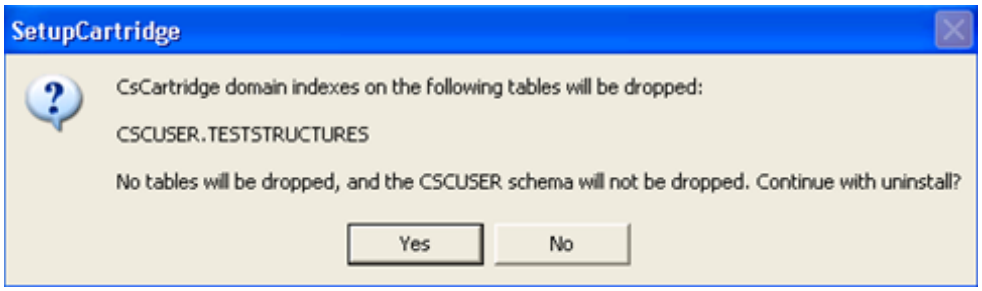
Even though the installer program makes its best effort to establish the `$ORACLE_HOME` directory (as shown in the "Select Oracle home directory" screen displayed above), the retrieved information might be incorrect when there are several Oracle databases on the same Host computer. If the `$ORACLE_HOME` file directory is different at your site from the one displayed by the installer program, you must change the file directory.

Uninstalling the Oracle Cartridge

It is recommended that uninstallation of the Oracle Cartridge should be performed using the **Uninstall cartridge instance** radio button in the **Install mode** section.

To uninstall the Oracle Cartridge:

1. Log in to the Client computer.
2. Execute the file. The **PerkinElmer Oracle Cartridge** installer screen is displayed.
3. Specify Oracle service name and password in the **Oracle service** section. If the Cartridge was installed in an Oracle instance on a remote LINUX server then enter relevant details in the **LINUX system access** section.
4. Select **Uninstall cartridge instance** in the **Install mode** section.
5. Click **OK**. A confirmation message appears:



6. Click **Yes**. The cartridge is uninstalled.

If manual uninstallation is required, use the following procedure:

1. Log in to the Host computer with as a user with Oracle System Administrator privileges.
2. Drop all Cartridge domain indexes. See "Troubleshooting" for information about listing all indexes that were created by the Oracle Cartridge.
3. Drop the Cartridge user and tablespace with the following SQL commands:

   a. `DROP USER CsCartridge CASCADE;`

   b. `DROP TABLESPACE T_CsCartridge INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;`

   Note: *If the Cartridge instance at your particular site is not named* `CsCartridge`*, then use the correct Cartridge schema name.*

4. On some Windows systems, despite the `AND DATAFILES` clause in the second of the above SQL commands, you may find that Oracle did not delete the `T_CSCARTRIDGE.DBF` file from the directory where it stores tablespace files. If this happens, delete the file manually (you might need to wait a few minutes before Windows permits you to do so). If this file remains, and you attempt to reinstall the Cartridge, you will receive an error message:"ORA-00959: tablespace 'T_CSCARTRIDGE' does not exist".

Reinstalling Oracle Cartridge

Reinstalling the Oracle Cartridge can be accomplished by following the same steps for Full Installation program.

If the *Full Installation* option was chosen over an existing installation, then all the existing Cartridge indexes must be recreated.

If the **Recreate existing indexes** check box is selected during a reinstallation, the installer saves information about and drops all existing indexes before replacing the Cartridge instance. It creates a `SQL` script to recreate the indexes and will prompt you about whether to run it after the installation. You may prefer to check it over, edit it, and run it manually later. The script will prompt you to enter the password for each schema in which it executes a `CREATE INDEX SQL` statement.

The Partial Installation option also can be chosen with reinstallation. This option should be chosen only if you know that the data model used by the Cartridge did not change from the installed version. While Partial Installation is a fast option, primarily because existing indexes do not have to be rebuilt, it can also leave the resulting database in a nonfunctional hybrid state if performed incorrectly or incompletely.

Multiple Installations

Under normal circumstances, the *Oracle Cartridge* is installed into the `CsCartridge` schema. In addition, the data files are stored in the `$ORACLE_HOME\bin\CsCartridge` directory on Windows, or the `$ORACLE_HOME/lib/CsCartridge` directory on `LINUX`.

To install more than one instance of the cartridge on your system, enter a non-default Cartridge instance schema name when you run the installer. If the new name is different from one already installed, it will be added as a new instance rather than replacing an existing one. You might do this to test or compare different versions of the Cartridge.

Under normal circumstances, this feature should not be invoked unless the Cartridge administrator has sufficient knowledge of the Cartridge and Oracle configuration to maintain two Cartridge instances. Most Client programs do

not ask the end user to choose a Cartridge instance, but use the default **CsCartridge** name. This is the name we use throughout this documentation.

However, ad-hoc `SQL` can easily be used against two installs, and Client programs can be written to access Cartridge installations. For testing or during a transition, Oracle permits you to create indexes with different Cartridge instances on the same field of a table.

Installing Cartridge Support Files in a Non-standard Location

The default location for Cartridge support files is in a subdirectory of `$ORACLE_HOME`. Security restrictions at some sites do not permit additions to the Oracle directory structure; and in a RAC, administrators might prefer to access a single copy of the Cartridge files from shared network or cluster storage instead of installing separate copies on all nodes. A manual installation is required to place the files in a non-standard location, and the listener must be reconfigured to enable the Oracle External Procedure Agent (extproc) to dynamically load the Cartridge shared library from a path outside of `$ORACLE_HOME`.

To install Cartridge support files in a non-standard location:

1. Run the installer and select the Manual install mode. Do not enter an Oracle service name, and leave the Unix system access boxes blank. If necessary, change the Cartridge instance name. For a RAC installation, or if you must control the tablespace filename and path, enter appropriate values in the Tablespace boxes. The installer will place your installation files in the path specified in the Local directory box.
2. With a popup dialog, the installer will prompt for the type of platform and Oracle version, so that it can prepare the correct installation files.
3. A final popup box will inform you that the files were not copied to the server computer. At the end of the message, it lists the four files that must be copied and the SQL script that you must customize and run to complete the installation. For a LINUX installation, the message also gives the file permissions that should be set on each file after it is copied. Although this message instructs you to copy the files to the default location (e.g: `$ORACLE_HOME/lib/CsCartridge`), you will actually copy them to a non-standard location of your choice.
4. On the server, create the directory that will hold the Cartridge support files. If possible, the directory owner should be Oracle. However, if security restrictions do not permit this, make sure Oracle has rights to access files in this directory. Copy the four support files from the client to the server; and on LINUX, use chmod to give them appropriate permissions. Again, it is simplest if the files are owned by Oracle; but in any case, make certain Oracle has read and execute permissions on the shared library file and read permission on the others.
5. Edit the SQL script as follows:
   a. Find the `Create or Replace library ChemCentral` command, which is about 30 lines from the beginning of the file.
   b. Edit this command, replacing `$ORACLE_HOME/lib/CsCartridge` with the actual path of the directory into which you have copied the Cartridge support files. Make sure this command specifies the full pathname of the CsCartridge shared library file on the server computer.
   c. Find the command `INSERT INTO GLOBALS VALUES('LIBRARY_DIRECTORY', '');` near the end of the file. Edit this command, inserting the directory path for the Cartridge support files between the final two single quotes.
6. Generally, you can run the SQL script for the Cartridge directly from the client computer, but if this is not possible, copy the edited script to the server and run it from there. Connect to the server instance as user “SYS

AS SYSDBA". If this is not permitted, connect as a user with installation rights (created with the CreateCSCInstallUser.sql script). Run the installation script with the SQL*Plus command "@CsCartridge.sql".

7. You or your Oracle DBA must configure the listener to permit execution of the Cartridge shared library. This requires that the listener.ora file contains a `SID_LIST_LISTENER` block, and within it, you must edit or add an ENVS specification that includes the environment setting `EXTPROC_DLLS=<pathname>`, where `<pathname>` is either the full path to the Cartridge shared library or, less securely, is the word `ANY`. In some cases, ENVS must also set the `LD_LIBRARY_PATH` environment variable (PATH on Windows). After editing listener.ora, you must stop and restart the listener. See "Performance Tuning" on page 66 for more information on configuring the Oracle listener. Following is an example of a simple, but complete, listener.ora file for some particular LINUX system (not yours):

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = csserver.<hostname>.com)(PORT = 1521))
    )
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /u01/app/oracle/product/10.2.0/db_1)
      (PROGRAM = extproc)
      (ENVS="EXTPROC_DLLS=ANY,LD_LIBRARY_PATH=/u01/cs/lib/CsCartridge:$LD_
LIBRARY_PATH")
    )
  )
```

8. Test the Cartridge instance you have just created by connecting to it with SQL*Plus and issuing the command

```
execute dumptext('');
```

If this gives an error, see the discussion of the appropriate error code in "Setup Problems and Error Handling " on page 71. To give the Cartridge instance a more complete test, run the "No install, test an existing installation" option in the installer.

Oracle Cartridge Program Files

The Oracle Cartridge installs the following Program files on their respective Operating Systems, as shown in the table that follows:

Operating System	File Name	File Location
Microsoft Windows	`CsCartridge.dll`	`%ORACLE_HOME%\bin\instance_name`

Operating System	File Name	File Location
Other Linux systems	CsCartridge.so	$ORACLE_HOME/lib/instance_name
All Systems	CSChemFinderCustomElements.txt	
All Systems	NameToStructure.dat	

Establishing System Privileges for Oracle Cartridge Users

The **CsCartridge.sql** script creates a new user with the name of CsCartridge and a password also set to **CsCartridge**. The system privileges that follow are granted to CsCartridge:

- CREATE SESSION
- CREATE TABLE
- CREATE LIBRARY
- CREATE PROCEDURE
- CREATE OPERATOR
- CREATE INDEXTYPE
- CREATE TYPE
- CREATE SEQUENCE
- CREATE VIEW
- SELECT ANY TABLE
- UNLIMITED QUOTA ON the T_CsCartridge tablespace
- SELECT ON SYS.V_$PARAMETER
- SELECT ON SYS.V_$SYSSTAT
- SELECT ON SYS.DBA_JOBS
- SELECT ON SYS.ALL_INDEXES
- SELECT ON SYS.ALL_IND_COLUMNS
- SELECT ON SYS.ALL_OBJECTS
- SELECT ON SYS.ALL_TAB_COLUMNS
- SELECT ON SYS.ALL_TYPE_ATTRS
- SELECT ON SYS.V_$INSTANCE

The **CsCartridge** user grants EXECUTE privileges to PUBLIC on all the functions, operators, and types designed to be accessible from external accounts. When a user indexes a table, it grants all privileges on the tables it creates in its schema.

Upgrading the Cartridge

Typically Oracle Cartridge 19.1 will be installed as an upgrade component using the installation procedure below. If you are performing a new installation please refer to the procedure in the chapter, "Installing Oracle Cartridge 19.1" on page 10.

Collect Index Related Data before Uninstalling the Oracle Cartridge

Execute the following query in SQL*PLUS to get the list of all existing indexes. This must be done as Cartridge uninstallation will drop all existing indexes. The list of indexes along with the parameters will help recreate the indexing commands to be executed after the installation of the 19.1 version.

```
SELECT * FROM CSCARTRIDGE.ALL_CSC_INDEXES;
```

A sample output for this query would be as follows:

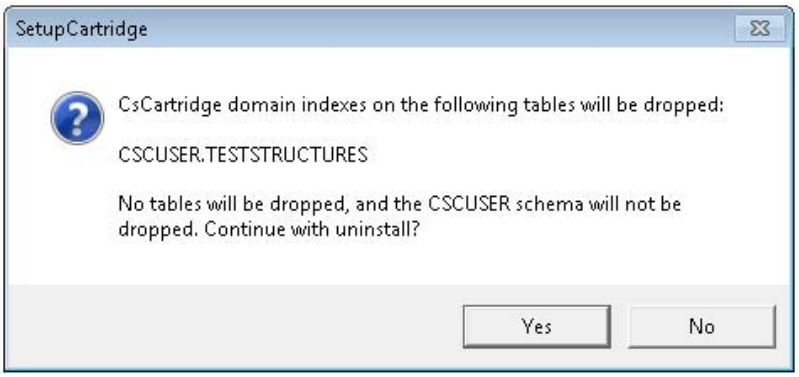
	OWNER	INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_TYPE	OPTIONS
1	EN1310	SX	ELN_CHEMICAL_STRUCTURES	CDXML_INDEX	CLOB	FAST_DELETE=YES
2	CSCJSER	SX	TESTSTRUCTURES	M	CLOB	(null)

Uninstall the Old Version of the Oracle Cartridge

We recommend that you uninstall the Oracle Cartridge using the Uninstall cartridge instance radio button in the Install mode section.

To uninstall Oracle Cartridge:

1. Log in to the Client computer.
2. Execute the SetupCartridge.exe file. The Oracle Cartridge installer screen displays.
3. Specify the Oracle service name and password in the Oracle service section. If the Cartridge was installed in an Oracle instance on a remote Linux server then enter relevant details in the Linux system access section.
4. Select Uninstall cartridge instance in the Install mode section.
5. Click **OK**. A confirmation message appears:



a. Click Yes. The cartridge is uninstalled.

b. Ensure that the CsCartridge folder in the following location is removed:

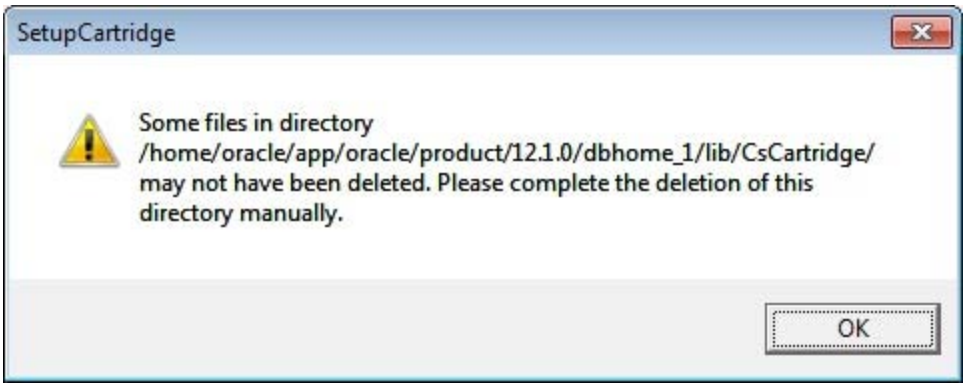
`$ORACLE_HOME/lib/CsCartridge` on Linux or `$ORACLE_HOME\bin\CsCartridge` on Windows.

Uninstall the Oracle Cartridge from Linux

During un-installation of the Oracle Cartridge running on a Linux operating system, if the FTP option is not enabled the following message displays:

Note: *The Oracle path provided in the message varies based on the path selected by the user.*

You must manually delete the files/folders in the directory location as indicated by the message.



Manual Uninstallation Procedure

1. Log in to the Host computer as a user with Oracle System Administrator privileges.
2. Drop all Cartridge domain indexes. See "Setup Problems and Error Handling " on page 71 for information about listing all indexes that were created by the Oracle Cartridge.
3. Drop the Cartridge user and tablespace with the following SQL commands:

- `DROP USER CsCartridge CASCADE;`
- `DROP TABLESPACE T_CsCartridge INCLUDING CONTENTS AND DATAFILES`
- `CASCADE CONSTRAINTS;`

Note: *If the Cartridge instance at your particular site is not named CsCartridge, then use the correct Cartridge schema name.*

4. On some Windows systems, despite the `AND DATAFILES` clause in the second of the above SQL commands, you may find that Oracle did not delete the `T_CSCARTRIDGE.DBF` file from the directory where it stores tablespace files. If this happens, delete the file manually (you might need to wait a few minutes before Windows permits you to do so). If this file remains, and you attempt to reinstall the Cartridge, you will receive an error message:"`ORA-00959: tablespace 'T_CSCARTRIDGE' does not exist`".

Note: *If you plan to upgrade the Cartridge instance alone which is present as part of the E-Notebook installation (i.e., without upgrading the E-Notebook instance), it is necessary to give the following GRANTS to the*

CSCARTRIDGE user after upgrading the Cartridge to version 19.1, as a user with SYSDBA privileges.

```
GRANT SELECT ON ENDB.ELN_CHEMICAL_QUERY_CELLS TO CSCARTRIDGE;
GRANT SELECT ON ENDB.ELN_CHEMICAL_STRUCTURES TO CSCARTRIDGE;
```

Recreate the Indexes Previously Present

The list collected in "Collect index related data before uninstalling the Oracle Cartridge" should be used to construct index creation commands. The default index creation command would be as given below:

```
CREATE index ix ON tab(fld) INDEXTYPE IS CsCartridge.MoleculeIndexType
[PARAMETERS ('keyword=value [,keyword= value ...')];
```

where:

- `ix` - Index name. The name is arbitrary, but must be short in length. The Oracle Cartridge concatenates this name with the user name to create different tables. Table names are restricted to 20 characters in length.
- `tab` - Table name with a field designated to store chemical structure information.
- `fld` - Field name designated to store chemical structure information.
- `PARAMETERS` - String containing the keywords=value pairs, delimited by commas.

A sample Create Index statement for the output given in the "Collect index related data before uninstalling the Cartridge" section is as follows:

```
Create index EN1310.SX on EN1310.ELN_CHEMICAl_STRUCTURES(CDXML_INDEX) indextype
is cscartridge.moleculeindextype parameters ('FAST_DELETE=YES');
```

The remaining indexes need to be recreated as applicable. Recreating indexes could be time consuming based on the data to be indexed. Refer to "Index Creation and Maintenance" for more details on monitoring indexes and more.

Index Creation and Maintenance

Index creation and maintenance is a critical step in the usability of the Cartridge. The CREATE INDEX command, along with its associated parameters, is used to create indexes. Before describing the index creation process, this chapter describes the CREATE INDEX command and all its associated parameters.

After you create an index, you must maintain the index periodically to ensure proper functioning of the Cartridge — periodic maintenance is a requirement in all the modes. The requirements for the Cartridge are similar to the requirements set forth by Oracle for any other type of indexed non-chemical data field.

Overview

The Oracle Cartridge uses **Data Definition Language (DDL)** statements to create and manipulate indexes. Examples include (but are not limited to) the `CREATE` and `DROP` statements.

Chemical structures must be stored in one of two types of data type fields: a **Binary Large Object** (BLOB) or **Character Large Object** (CLOB). More specifically:

- `BLOB` fields are required if you want to store binary data. `CDX` documents and `SKC` files are an example of binary data that you might want to store in a `BLOB` file.
- `CLOB` fields are ASCII-text files that can contain:
  - `CDXML` documents
  - `SMILES` strings
  - molfiles
  - Rxnfiles
  - Chemical names recognized by PerkinElmer's Name to Structure
  - InChI strings

Note: *For more information about the `CDX` and `molfile` formats, refer to "Exporting and Importing Using File Formats" in the ChemDraw User's Guide. For additional help in creating SMILES strings, refer to "SMILES and SMIRKS Strings" in the ChemDraw User's Guide. The ChemDraw ActiveX control, as well as a number of other applications, can save documents in any of the above formats.*

Parameter Options that Specify Index Creation

If an index is to be created on the chemical structure column, the table must not be an Index-Organized Table (IOT). Currently, creation of a Chemistry Cartridge domain index is not supported on an IOT.

Also, if the table is partitioned, creation of a global domain index is supported, but a local partitioned domain index is not currently supported.

The Oracle Cartridge is completely functional, even when it is used with no indexes. Without indexes, however, chemical searching on large tables will be very slow.

Creating indexes is a critical step in the usability of the Cartridge. The six parameter options that follow differ from all the other parameters offered by the Oracle Cartridge because they control what indexes are created within the Cartridge domain index.

Five of the parameters are similar in that they accept `YES/NO/INDEX` values, and can improve performance for the following types of searches:

- NORMAL: Regular substructure and full structure searching without recognition of tautomerization. This index is created by default.
- SIMILAR: Similarity searching.
- FULLEXACT: Full structure searching with recognition of tautomerization.
- SKELETAL: Substructure searching with recognition of tautomerization.
- COUNTS: Further optimizations affecting databases with large numbers of aliphatic atoms ($-CH_2-CH_2-$ chains).

A sixth parameter only accepts YES|NO|DISTANCE|ANGLES values, and can improve performance for 3D searches:

- THREE_D: 3D searching with distance and angle constraints. The YES option creates indexes for both distance and angle constraints; the DISTANCE and ANGLE options create indexes for one or the other.

For each of the five parameters other than NORMAL and SIMILAR, NO creates no index of that type at all, while YES creates a simpler index that results in much better performance than with NO index (except for SIMILAR). The INDEX option creates the fastest index possible for the type.

To create an index that supports the fastest substructure, exact, tautomer, and similarity searches, use the abbreviation ALL=INDEX in place of NORMAL=INDEX, SIMILAR=INDEX, FULLEXACT=INDEX, SKELETAL=INDEX. The disadvantage of this setting is that, with it, the index takes longer to create and maintain, and it occupies more disk space. COUNTS and THREE_D are not included automatically when you specify ALL, and they should be specified separately if your table will contain structures that could benefit from their inclusion.

In more detail, the six parameters are as follows:

- NORMAL=[YES|INDEX]
  This option instructs the Cartridge whether to create and maintain screen tables for regular substructure and full structure searching without recognition of tautomerization. To enable searching that supports recognition of tautomerization, see FULLEXACT and SKELETAL, below. The default is INDEX.
- SIMILAR=[NO|INDEX]
  This option instructs the Cartridge whether to create and maintain screen tables for similarity search. The default is NO.
- FULLEXACT=[YES|NO|INDEX]
  This option instructs the Cartridge whether to create and maintain screen tables for full structure search allowing tautomer matches as well. The default is NO.
- SKELETAL=[YES|NO|INDEX]
  This option instructs the Cartridge whether to create and maintain screen tables for substructure search which honors tautomer matches as well. The default is NO.
- COUNTS=[YES|NO|INDEX]
  This option specifies additional structure screening by aliphatic atom counts. The default is YES.
- THREE_D=[YES|NO|DISTANCE|ANGLE]
  If YES, this option instructs the Cartridge to maintain tables for 3D searching. The default is NO.

CREATE INDEX Command for Fast Structure Searches

The field containing structure information must be indexed to enable fast substructure searches:

```
CREATE index ix ON tab(fld) INDEXTYPE IS CsCartridge.MoleculeIndexType
[PARAMETERS('keyword= value [,keyword= value ...')];
```

where:

- ix – Index name. The name is arbitrary, but must be short in length. The Oracle Cartridge concatenates this name with the user name to create different tables. Table names are restricted to 20-characters in length.
- `tab` – Table name with a field designated to store chemical structure information.
- `fld` – Field name designated to store chemical structure information.

`PARAMETERS` – String contains the `keyword=value` pairs, delimited by commas.

The `PARAMETERS` clause is optional. In addition to the six keywords discussed above, the other supported keywords are as follows and as listed in the next section:

- `TABLESPACE=tablespace_name`
  When present, the Cartridge-generated tables are not to be created in the default **T_CsCartridge** tablespace, but rather in the `tablespace_name` tablespace provided. The tablespace creator must allocate a reasonable quota to the **CsCartridge** User on that tablespace.
- `SKIP_POPULATING=[YES|NO]`
  When present, the indexes are created normally; however, the data stored in the table will not be used to populate the Cartridge-created tables. The User must populate these tables manually. **Generally, this option should be used only for troubleshooting.** Use this option only when an earlier attempt to create an index has failed. The Cartridge-created tables can then be populated by one or more `UPDATE` statements by using the syntax as follows:

```
UPDATE tab SET fld = fld;
```

One table can contain any number of fields designed to store chemical structure information, and some or all of them can be indexed by the Cartridge. To remove an index, especially after errors occurred during its creation and it was not populated successfully, use the `SQL` command:

```
DROP INDEX ix
```

- `RAISE_ERROR=[YES|NO]`
  This option ensures the Cartridge should raise an error if a document is passed to it that cannot be interpreted as any of the supported data formats. When set to `NO`, no exception will be raised, and an entry will be stored into the `CsCartridge.schema_index_E` table. The default is YES. This setting applies during inserts and updates of indexed structures; during CREATE INDEX, exceptions are not raised and any errors are logged in the schema_index_E table.
- `FAST_DELETE=[YES|NO]`
  If `YES`, this option allows the user to prefer faster delete operations and update operations. The default is `YES`. The Oracle Cartridge is optimized to make substructure search, and to a lesser extent, index creation and appending records to a database fast. Deleting a record (s) is slow. When a record is updated by an application, the Oracle Cartridge deletes records and inserts new records, so that can also run at a slow speed. If the `FAST_DELETE=YES` option is chosen, the Oracle Cartridge creates additional indexes to make the delete and update options run at a faster pace. The price to be paid is slower index creation. The `FAST_DELETE=YES` option is recommended for tables where changes are expected to be frequent.
- `USEPARTITIONS=[YES|NO]`

Note: *The use of the USEPARTITIONS keyword is discouraged. If partitioning is not available, the "Partition inverted screen tables when possible" box should be unchecked during installation to disable partitioning for all indexes created by the Cartridge.*

Some Oracle users have partitioning available in their system but cannot use it, because they do not have the license. The cartridge does not know if they are legally restricted from using it, and if the system reports that partitioning is enabled, then partitioning is used.
This option can be used to control the behavior of the cartridge. If set to NO, index-organized tables are used instead of partitioned tables even if partitioning is available.

- SYNCHRONIZATION=[YES|NO]
  The default is NO. In this mode, which is the standard indexing mode, structures are processed by the Cartridge when inserted or deleted, but time-consuming modifications to the index tables are deferred (primarily in the inverted-screen tables created by setting NORMAL, FULLEXACT, or SKELETAL =INDEX as described earlier). This permits inserts and deletes to be completed more quickly, but over time, searching will become slower as the deferred changes accumulate. Periodically, it is necessary to synchronize the index to complete the deferred operations. For more information, see "Maintaining the Index" on page 33. To keep the index always synchronized, specify SYNCHRONIZATION=YES when creating the index. INSERT and UPDATE operations are slower in synchronization mode.

Note: *Standard mode is different from deferred maintenance mode. In standard mode, new structures are processed but not fully indexed. In deferred maintenance mode, the new structures are not processed at all, but are only tagged in the index for later batch processing; as a result INSERT and UPDATE are almost instantaneous. In both modes, synchronization must be performed periodically to maintain search performance, but deferred maintenance mode requires more frequent synchronization.*

- AUTOSYNC=[YES|NO]
  In order to exclude an index from automatic synchronization, add the parameter 'AUTOSYNC=NO' to the CREATE INDEX command.

Related Parameters

This section describes additional CREATE INDEX parameters that affect how searching will be performed, or that can further optimize certain types of searches. These parameters are as follows:

- REACTION_SCREENING[1]=[YES|NO]
  If YES is chosen, this option instructs the Oracle Cartridge whether to create and maintain screen tables to further increase the speed of substructure searches in reactions. It also provides for better similarity searches against reactions. The default is NO.
- THREE_D_RESOLUTION=[HIGH|LOW]
  This option instructs the Oracle Cartridge what level the screening should be carried out. Default is HIGH. This is disregarded if the THREE_D parameter is absent or NO. Setting THREE_D_RESOLUTION to LOW makes three-dimensional screening faster and reduces the size of a three-dimension-enabled database. However, it also decreases the quality of the screening. Only the first significant digits of the distances and angles are stored.
- CUSTOM_SCREENS=[YES|NO]
  If YES, this option instructs the Oracle Cartridge to use the structures from the schema's CUSTOM_SCREENS table, or if that does not exist, from the CsCartridge CUSTOM_SCREENS table. The default is No. With this option, the screens in the CUSTOM_SCREENS table will be added to all indexes created by the Cartridge. For more details, see the "Custom screening" section in "Oracle Cartridge Advanced Techniques".

[1]This parameter is not used by Cartridge 19.1. However, it was used in earlier versions of the Cartridge and it may be re-enabled in a future version of the Cartridge.

- `UNIQUE=[YES|NO|STEREO|RELATIVETETSTEREO]`
  The default is NO. If NO, this option will instruct the Cartridge to allow duplicate structures. Otherwise, duplicate structures are not allowed, and duplicates are defined as follows:
  - If `YES`, then two structures are considered identical if they have the same connectivity, even if they have different stereochemistry.
  - If `STEREO`, then two structures are considered identical if they have the same connectivity and also have the same stereochemistry.
  - If `RELATIVETETSTEREO`, then stereocenters specified using wedged bold and hashed bonds are interpreted as indicating absolute stereochemistry, while stereocenters specified using unwedged bold and hashed bonds are interpreted as indicating relative stereochemistry. Accordingly, if `UNIQUE` is `RELATIVETETSTEREO`, two structures are not considered as identical if one is specified with wedged bonds and one is specified the same in all ways except for the presence of unwedged bonds. This depiction style is rare, and as a result is often confusing to chemists. The default is `NO`.
- `SALTSPLITTING=[YES|NO]`
  If `YES`, this option sets the mode for applying salt splitting/removal from the structures stored. If salt splitting is requested, then the salt parts and the predefined incidental fragments are disregarded from substructure search. The default is `NO`. By default, salts and incidental fragments are not removed from queries; that has to be specified as a search option.

  📝 ***Note:*** *This CREATE INDEX parameter has the same name as a MOLECULECONTAINS searching parameter documented in the Chemical Queries chapter.*

Recommended Parameters to Optimize Structure Searches

The PerkinElmer Oracle Cartridge has four parameters in particular that affect the performance of structure searches:

- `NORMAL` – Regular substructure search
- `SIMILAR` – Similarity search
- `FULLEXACT` – Tautomer aware structure search
- `SKELETAL` – Tautomer aware substructure search

At least one of these screens should be present if searching is to be performed; if not, then the corresponding search will return correct results but will operate more slowly.

There are three different values that can be assigned to these parameters: `NO`, `YES` or `INDEX`:

- If `NO` is chosen, then a particular kind of screening will not be available. However, the `NORMAL` parameter cannot be set to `NO`.
- If `YES` is chosen, then screening bit vectors will be stored and used for that type of search to improve performance.
- If `INDEX` is chosen, then an inverted screen table will be maintained by the Cartridge making the screening phase even faster.

  Choosing the `INDEX` option results in fastest performance for substructure searches. However, the trade-off is slower maintenance, and the occasional need for synchronization and analysis of the index.

Choosing the INDEX option is the default for normal screens, and NO is the default for the remaining screens. INDEX is recommended for large tables which do not change frequently and in which high structure search speed is required.

***Note:** The settings chosen when the index is created cannot be modified later by using the ALTER INDEX command.*

Three-dimensional (3D) data is stored only when the THREE_D option is set to YES when the index is created. This setting cannot be changed later by the ALTER INDEX command. If the THREE_D option is set to DISTANCE, then the Cartridge stores and maintains only atom-atom distances information; if it is set to ANGLE, then only angles defined by three atoms or a centroid are stored.

The REACTION_SCREENING option directs the Cartridge to maintain screens for both the reactants and the products of a reaction in addition to the regular screens. This information is used, when similarity search is to be restricted to either the reactants or to the products. Similarity searches when both reactants and products are specified are not considered valid, and result in "invalid query structure" error raised.[1]

The screens maintained by the Cartridge when REACTION_SCREENING is enabled are also used for substructure searching. This usage is mostly transparent to the user, because the only difference is, that substructure searches for reactions are faster.[2]

Index Creation

When a MoleculeIndexType index is created, the Oracle Cartridge creates several tables and populates those tables with data from the records in the table on which the index is created. If the table has millions of records, this process could take several hours, depending on system speed and resources.

The indexing process can fail if there is insufficient space in the tablespace used by the Cartridge. Usually, the index will require 10 to 20 Kb per row of the original table. Indexing reactions take more space than indexing molecules. Also, space requirements will increase on including more inverted screens with the FULLEXACT or SKELETAL=INDEX options for faster tautomer searching.

If you want to index a large table (particularly one with over one million rows), planning is required to ensure that there is sufficient disk and table space to hold the index. Create a separate tablespace to contain the large table and its index, or for better performance, separate tablespaces for the table and index. Use Oracle version 11g or later, which supports the CREATE BIGFILE TABLESPACE keyword for better management of large tablespace files, and use the TABLESPACE parameter in CREATE INDEX to place the index in the appropriate tablespace.

During the CREATE INDEX operation, Oracle also uses the temporary tablespace and requires that it has space (either allocated or available for expansion) equivalent to approximately 15% of the size of the index.

---

[1]This paragraph is not applicable to Cartridge 19.1 because the REACTION_SCREENING parameter is not used by Cartridge 19.1. However, it was used in earlier versions of the Cartridge and it may be re-enabled in a future version of the Cartridge.

[2]This paragraph is not applicable to Cartridge 19.1 because the REACTION_SCREENING parameter is not used by Cartridge 19.1. However, it was used in earlier versions of the Cartridge and it may be re-enabled in a future version of the Cartridge.

Use the MONITOR command to track the progress of an index-creation operation. This can be helpful when creating an index containing over a million records. It will display the number of records processed during index creation. The MONITOR command must be executed from a different session. The form of the command is:

```
SELECT CsCartridge.MONITOR FROM DUAL;
```

More details about the use of MONITOR command are given below.

Example of Index Creation

```
CREATE TABLE moltable(mol CLOB);

CREATE INDEX mx ON moltable(mol) INDEXTYPE IS CsCartridge.MoleculeIndexType;
```

It is important to remember when creating an index in a different schema than the logged-on user, that both the index name and the table name must be qualified by the schema name. Otherwise, the Cartridge will not connect its tables to the original table. For example, this syntax:

```
CREATE INDEX mx ON USER1.moltable(mol)...
```

is incorrect. The correct syntax is:

```
CREATE INDEX USER1.mx ON USER1.moltable(mol)...
```

Note: *Creating an index may be time consuming if the table already contains a large number of records when the index is created.*

Monitor Command

During index creation, files named by the process ID number (PID) of the Cartridge are created in the system temporary directory in a subdirectory named CsCartridge_CsCartridge_PIDS. These files contain counts of records indexed. Normally, such files are deleted at the end of index creation, but some might be left behind if a Cartridge process ends abnormally. The MONITOR command displays the contents of all such files, which can be confusing if many obsolete files are present. You can use system utilities, such as Task Manager on Windows or the ps –ef command on other systems, to identify the PID of the current index-creation Oracle extproc process in which the Cartridge operates. However, it is simpler to remove extraneous obsolete files before you begin monitoring so that only information from current processing is displayed by MONITOR.

To delete obsolete PIDS files, issue the following SQL command before you begin creating a new index:

```
SELECT CsCartridge.MONITOR('CLEAR') FROM DUAL;
```

Note that if a Cartridge indexing process ends abnormally, an error record is normally inserted into the error (_E) table of the index, and a new process is started automatically to resume indexing at the next record. Therefore the total number of records processed is the sum of the counts in all current PIDS files. For more information about the error table, see “When creating an index fails” in the “Troubleshooting Procedures' chapter. If any indexing errors occur, the index will be created successfully, but Oracle will report that it was created with warnings. (It is extremely unusual for an indexing error to cause the process to end abnormally, but in such cases you can help to improve future Cartridge versions by reporting the record to PerkinElmer.)

After all records are processed, `MONITOR` will no longer show activity, but the indexing process is not quite complete. Oracle sub-indexing of the tables in the Cartridge domain index and collection of statistics is performed at this point; the duration of this process is typically about 15% of the total `CREATE INDEX` time.

Unique Index

When the index is created, you can control whether or not to allow duplicate structures. Due to the ambiguities of defining the identity of a chemical structure, many options are available.

Those options can later be changed according to the needs of an application for the duration of a transaction. For that purpose, the `ChangeOption` stored `SQL` procedure can be used. Of course it can be used for other purposes as well, but that was the main reason for its implementation.

The syntax of the `ChangeOption` procedure is:

```
ChangeOption(schema_name, index_name, identifier_name, value);
```

All parameters are strings enclosed in single quotes. To allow duplicates on a given index call the `ChangeOption` procedure:

```
CALL CsCartridge.ChangeOption('my_schema', 'my_index', 'UNIQUE', 'NO');
```

To restrict duplicates to identity defined by same stereochemistry:

```
CALL CsCartridge.ChangeOption('my_schema', 'my_index', 'UNIQUE', 'STEREO');
```

If you want to make these settings permanent, execute a commit command. Otherwise, the settings will affect only the current transaction.

The UNIQUE parameter can be used during index creation to determine whether duplicate structures will be allowed and how stereochemistry should be considered in defining duplicate structures. An index is built even if duplicate structures are found.

When a duplicate structure is found, a record is inserted into the *UserName_IndexName*_E table with reference to the two records found as duplicate. The record found as duplicate is not indexed, thus it will not be found when the table is searched by index. To ensure that duplicates are completely eliminated the duplicate records should be manually deleted. That can be done by using the entries in the `UserName_IndexName_E` table.

Example of Duplicate Elimination

```
CREATE INDEX mx ON moltable(mol) INDEXTYPE IS CsCartridge.MoleculeIndexType
PARAMETERS('unique=stereo');

DELETE FROM moltable WHERE rowid IN (SELECT rid FROM CsCartridge.UserName_mx_E
WHERE text LIKE '%unique constraint violation');

COMMIT;
```

Maintaining the Index

For best searching performance, PerkinElmer recommends that you use the `INDEX` option for the appropriate screening tables, such as `NORMAL` and `FULLEXACT` as described in the documentation for `PARAMETERS` above. In this case, the Cartridge maintains “inverted” tables in order to implement faster substructure and 3D search.

However, maintenance of inverted tables can be time consuming and may result in users waiting for each other when simultaneously updating the same table.

In the default standard indexing mode, to avoid such delays when updating the database, the Cartridge does not update its inverted tables when you update an indexed table. Instead, it stores a record describing the update. The accumulated changes are merged into the inverted tables only when the DBA explicitly requests that "synchronization" be performed on the index. Alternatively, PerkinElmer provides an Oracle job that can be scheduled to automatically perform synchronization at specified intervals.

The parameter `'SYNCHRONIZATION=YES'` can be used in a `CREATE INDEX` or `ALTER INDEX` command to place the index in synchronized mode. In this mode, all indexing is performed immediately, and periodic synchronization is not required. PerkinElmer recommends standard rather than synchronized mode because generally the improved insert/update performance more than compensates for any slight reduction in searching performance. In a typical environment with standard mode, close to 100,000 updates can accumulate before there is a significant impact on performance, and a scheduled synchronization job can ensure that the total is always well below this level.

A third index-maintenance mode, deferred maintenance mode, further improves update performance at a cost of requiring more frequent synchronization. It is described in the following section and can be requested only after an index has been created.

Synchronization does not quite fully update the index; some statistics that influence the execution of a substructure search are still not updated. You must occasionally issue the following command to recalculate these statistics:

```
ANALYZE INDEX ix COMPUTE STATISTICS;
```

The execution of this command is time consuming, but it is required only rarely when a large portion of a database is replaced, when the database is close to its expected full size, or after the size of a large indexed table has doubled.

Synchronization and computing statistics are the responsibility of the DBA. Failure to perform this maintenance will result in poorer performance.

Executing the `ANALYZE INDEX` command does not analyze tables maintained within the Cartridge, but that usually is not necessary. If the tables maintained by the Cartridge are to be directly referenced by non-Cartridge applications, then the DBA should occasionally call the provided procedure `CsCartridge.AnalyzeIndexTables()` from the Cartridge schema. The procedure has two parameters, the name of the user's schema, and the name of the index. The following is an example of the usage of the `AnalyzeIndexTables` command:

execute `AnalyzeIndexTables('myschema', 'myindex');`

The `ANALYZE` command can also be used to verify the integrity of a Cartridge index. Normally, this command should return without any message. If it throws an exception, the domain index is in unstable condition, and the index may need to be rebuilt using the following command:

```
ANALYZE INDEX ix VALIDATE STRUCTURE;
```

Buffer Pools

The Cartridge allows you to control the assignment of its tables to different Oracle buffer pools. Tables can be assigned to either to the `DEFAULT`, the `KEEP` or the `RECYCLE` pool. The explanation of the usage of these pools go beyond the scope of this document; only the command to control those assignments are covered in this document. The following SQL command ensures that Oracle keeps the Cartridge tables in the `KEEP` buffer:

```
ALTER INDEX ix PARAMETERS('KEEP=YES')
```

`KEEP=NO` in the `PARAMETERS` clause assigns the tables to the `DEFAULT` pool. With `KEEP=YES` or `KEEP=OPT`, it is necessary to configure the `KEEP` and `RECYCLE` buffer pools properly. See "Performance Tuning" chapter for details.

Deferred Maintenance Mode

Use deferred maintenance mode in situations where a large number of updates to a table must be performed quickly. In this mode, the Cartridge postpones indexing of updated records until synchronization is performed. Insertions, updates, and deletions are recorded in a list of deferred updates, and they are almost instantaneous because no further processing is done immediately.

Enter deferred maintenance mode with the following SQL command:

```
ALTER INDEX ix PARAMETERS('DEFER');
```

Terminate deferred maintenance mode with the following SQL command:

```
ALTER INDEX ix PARAMETERS('SYNCHRONIZE');
```

Deferred maintenance mode remains in effect after synchronization with the `Synchronize_Index()` procedure described below. However, to continue in deferred maintenance mode after you synchronize with `ALTER INDEX`, you must again alter the index to put it into deferred mode.

This mode is transparent to the application developer and the user because deferred records are searched as if nonindexed. Searches will become slower when a few thousand deferred records await synchronization. To maintain performance, PerkinElmer recommends that synchronization should be performed more frequently in deferred maintenance mode than in standard mode.

Note: *Deferring or synchronizing an index does not affect CsCartridge sessions that are already in progress. After you perform those operations, you should restart any active sessions that use CsCartridge. For example, if your site uses ELN, you should stop all ELN clients, then stop and restart the IIS server on the middle tier because the server caches sessions to the Cartridge.*

You can determine whether an index is currently in deferred maintenance mode with the following SQL command:

```
SELECT value FROM CsCartridge.<schema-name>_<index-name>_O WHERE id='DEFERRED';
```

The DEFERRED option value displayed by the above command is 'YES' if the index is in deferred maintenance mode.

Bulk Loading

Similar to deferred maintenance mode, bulk loading mode improves speed by postponing Oracle updates. However, bulk loading mode affects only the current session while deferred maintenance mode affects all Cartridge users. In bulk loading mode, new index data are buffered in memory and are written out to Oracle tables only at the end of the batch update, in a single roundtrip to the server.

Bulk loading mode is controlled by `PARAMETERS('SUSPEND')` and `PARAMETERS('RESUME')` clauses in an `ALTER INDEX SQL` command. These settings affect only the current session. After the suspend command, inserts are not executed, and new data are buffered in memory. When the resume command is issued, all the buffered data are stored by the Cartridge.

This feature is useful in batch data maintenance applications.

Recommendations for Index Maintenance

Search performance is best with a fully synchronized and analyzed domain index. The Cartridge will perform most slowly when the field containing the structure data is not indexed at all because the Cartridge can search on unindexed fields, but could take thousands of times as long to return the results. When an index is in deferred maintenance mode, the records that are not synchronized behave the same as records in a non-indexed table. In standard mode, records which are not synchronized are processed, and searching them is much faster but not as fast as when fully synchronized.

In summary, expect the following search performance, from best to worst:

1. Indexed column right after index creation or synchronization and analysis
2. Indexed column in standard mode after record inserts and updates
3. Indexed column in deferred mode after record inserts and updates
4. Non-indexed column

In this connection, note that the `FAST_DELETE` option does not affect search speed. It affects only the speed of DML commands. If `FAST_DELETE=YES` (the default), then index creation will be slower, delete and update operations will be faster, and synchronization will be slower. If `FAST_DELETE=NO`, then index creation will be faster, deletion and modification of synchronized records will be much slower, and synchronization will be faster.

In standard mode, synchronization should be done when about 50,000 to 100,000 records are pending synchronization, and in deferred maintenance mode it should be done when about 5,000 to 10,000 records are pending. The number of records to be synchronized in standard mode can be displayed with the SQL command:

```
SELECT count(*) FROM CsCartridge<schemaName>_<indexName> WHERE isnew = 'Y';
```

In deferred maintenance mode, display the count with the command:

```
SELECT count(*) FROM CsCartridge<schemaName>_<indexName>D;
```

The deferred count could be up to twice the number of deferred records because an update operation enters two records in the _D index table.

As described in the “Maintaining the Index” section, for optimum performance, analyzing an index is recommended after a table has grown by at least 10%, or about 10% of its records have changed and the old and new data are not relatively homogeneous. In any case, it should be performed after the table size has doubled or a large fraction of its records have changed.

The Cartridge provides three ways to synchronize an index: an `ALTER INDEX` command, the `Synchronize_Index()` stored procedure, or a scheduled job that uses the stored procedure. To maintain Cartridge performance, the DBA must establish a maintenance routine based on one of these methods unless all Cartridge domain indexes are kept in synchronized mode. As noted earlier, standard or deferred maintenance mode will generally provide the best overall performance.

📝 ***Note:*** *Even in synchronized mode, the ANALYZE procedure should still be performed from time to time.*

The simplest way to synchronize an index is to connect as the table owner and issue the SQL command:

```
ALTER INDEX ix PARAMETERS('SYNCHRONIZE');
```

This can be done easily by the table owner and typically completes in between a few seconds and a minute or two, depending on system configuration and resources. A disadvantage of using the ALTER INDEX command on publicly accessible tables is that during its execution the index is in a disabled state. As a result, a search performed at the same time will run very slowly, as though unindexed, and DML statements to the indexed table will fail with the **Oracle error ORA-29861: domain index is marked LOADING/FAILED/UNUSABLE state**. In addition, this synchronization command ends deferred maintenance mode, so it is less suitable for use on a deferred-maintenance index.

The stored procedure is better suited for centralized maintenance by the DBA, for use on deferred-maintenance indexes, and for use on tables that must be kept available at all times. In order to use it, connect as the CsCartridge user and issue the SQL command:

EXECUTE Synchronize_Index('schemaName', 'indexName');

When invoking this procedure schemaName and indexName must be in single quotes. The procedure runs just as fast as `ALTER INDEX`, but it does not invalidate the index. Therefore, concurrent searches will not be slowed down and concurrent DML is permitted.

The most convenient way to ensure timely synchronization is with a scheduled job. Such a job can be set to create the least possible disruption by running at a time of low resource usage. If scheduled relatively frequently, each execution will run more quickly, and the total number of records pending synchronization can be kept well below levels that might affect performance. While a DBA might prefer to design his own scheduled job, the Cartridge does provide a procedure that can be used for this purpose.

In order to initiate the preconfigured synchronization job, connect as the CsCartridge user and issue the SQL command:

```
EXECUTE StartSyncJob;
```

The job created by `StartSyncJob` is submitted using the `DBMS_JOB` package and can be controlled with procedures in that package to change its run time or interval, temporarily disable it, etc. By default it runs daily at midnight.

The identifier of the job can be displayed with the following SQL command:

```
SELECT value FROM CsCartridge.globals WHERE id = 'SYNCJOB_START';
```

To remove this job from the system, issue the SQL commands:

```
EXECUTE DBMS_JOB.REMOVE(jobIdentifier);

UPDATE globals SET value = '-1' WHERE id = 'SYNCJOB_START';

COMMIT;
```

where jobIdentifier is the value retrieved with the previous `SELECT` statement. Automatic maintenance can be re-enabled by rerunning StartSyncJob; note, however, that it will submit a new job only if the `SYNCJOB_START` identifier recorded in the `CsCartridge.GLOBALS` table does not reference a valid job.

The preconfigured synchronization job uses the `Synchronize_Index()` procedure to synchronize every index created by the Cartridge instance in which it runs. In order to exclude an index from automatic synchronization, add the parameter `'AUTOSYNC=NO'` to the `CREATE INDEX` command when it is created, or change the parameter with `ALTER INDEX`.

Transportable Tablespace

The Transportable Tablespace feature of Oracle allows an indexed table to be moved or copied from one computer or database instance to another faster than the traditional method using the EXP and IMP utilities. In the traditional method, only the table and the CREATE INDEX command are copied, and it is necessary to perform a time-consuming recreation of the index on the imported table. With a transportable tablespace, the table and its domain index can be copied intact.This can reduce the deployment time for a chemical database from hours to minutes.

The Cartridge provides preparation functions for indexes in transportable tablespaces. A distribution center can create a single file containing all the tables and indexes in a tablespace. When this file is loaded into an Oracle instance on another computer, it creates a copy of the original tablespace with all the original tables and indexes.Using Oracle 11g and later, you can convert tablespaces to work on other systems, although this adds some time to the deployment process. As a result, the time consuming creation of chemical domain indexes need to be done only once, at the distribution center.

Cartridge domain indexes do, however, contain data that are organized differently on computer architectures with different word size and byte order. Consequently, PerkinElmer recommends that transportable tablespaces be used for deployment only among systems running on the same hardware architecture.

Preparing for the Usage of Transportable Tablespaces

When exporting a table containing an indexed column the information maintained by the Cartridge is not exported – only the parameters of the `CREATE INDEX` statement, so the `imp` utility can execute that statement at the time the table is imported. This requires re-processing all the records in the table, and once again building all the information the Cartridge maintains.

Any table that is created by the Cartridge during a CREATE INDEX command, is marked as a secondary object, which means they are never exported, even when a full schema is exported by the `exp` utility. To ensure that the tables maintained by the Cartridge are also exported, the Oracle Cartridge tables cannot be created with `CREATE INDEX`. Instead, use the following stored procedure to create a "detached" index having tables that can be exported:

```
execute CreateDetachedIndex (indexName, schemaName, tableName, columnName,
params, usePartitions);
```

where:

- `indexName` is the name of the index to be created.
- `schemaName` is the name of the schema where the table resides.
- `tableName` is the table containing the CLOB or BLOB structure column to be indexed.
- `columnName` is the name of the column to be indexed.
- `params` is the same string passed to the `CREATE INDEX` statement in the `PARAMETERS` clause.
- `usePartitions` can be set to `YES` or `NO`. This parameter can be used to overrule the Oracle Cartridge's choice of using partitions in the inverted screen table.

All the six parameters listed above are of data type VARCHAR2, and hence must be enclosed in single quotes.

The `CreateDetachedIndex` call must be made while the user is connected as the **CsCartridge** user or schema in which the Cartridge was installed.

Preparing an Indexed Column Table for Transportation

When preparing a table for transportation by using transportable tablespace, the following steps must be performed:

1. Create the table to be transported in a separate tablespace.
2. Populate the table with the data to be distributed or transported.
3. Create the index data for the chemical structure column by calling the `CreateDetachedIndex` stored procedure. It is very important that the `PARAMETERS`clause specifies the tablespace for the index; the index's tablespace is the same as the table's tablespace; or another also planned to be part of the distribution. The `CreateDetachedIndex` procedure throws an exception if the tablespace is not specified. For the `usePartitions` parameter use `NO` unless you are sure that partitioned tables can be imported on the system to which you will transport the data.
4. Mark the tablespace as read-only by connecting as SYS (SYS/<password> AS SYSDBA), and issuing the following command:

   ```
   ALTER TABLESPACE <Space> READ ONLY;
   ```

5. Export the tablespace by using the `exp` utility and issuing the following command: For Oracle 11g or above, you can use the expdp utility. This requires a directory object pointing to a physical directory with the necessary permissions on the database server.

   An example is given below:

```
CONN / AS SYSDBA
CREATE OR REPLACE DIRECTORY temp_dir AS '/tmp/';
GRANT READ, WRITE ON DIRECTORY temp_dir TO SYSTEM;
```

   An example command is as follows:

```
expdp userid=system/passwd directory=<temp_dir>transport_tablespaces=<test_
data>dumpfile=<test_data.dmp> logfile=<test_data_exp.log>
```

   ```
   This requires a directory object pointing to a physical directory with the
   necessary permissions on the database server.
   ```

   An example is given below:

```
CONN / AS SYSDBA
CREATE OR REPLACE DIRECTORY temp_dir AS '/tmp/';
GRANT READ, WRITE ON DIRECTORY temp_dir TO SYSTEM;
```

6. Copy the following files to the target system:

- The created dump file, which holds metadata for the tablespace
- The tablespace data file

Importing the Transportable Tablespace

To import the transportable tablespace, perform the following steps:

If you are using 11g or above you should use the impdp utility. This requires a directory object pointing to a physical directory with the necessary permissions on the database server.

For example,

```
CONN / AS SYSDBA
CREATE OR REPLACE DIRECTORY temp_dir AS '/tmp/';
GRANT READ, WRITE ON DIRECTORY temp_dir TO SYSTEM;
```

1. Type the following at the command prompt:

```
IMP TRANSPORT_TABLESPACE=y FILE=dumpFileName DATAFILES=('dataFileName')
TABLESPACES=<Space> BUFFER=10000000impdp userid=system/password directory=<temp_
dir>dumpfile=<test_data.dmp> logfile=<test_data_imp.log> transport_
datafiles='/u01/app/oracle/oradata/DB11GB/test_data01.dbf' Buffer=
```

where:

- `dumpFile` is the file transported from the Client computer, which can be placed anywhere.
- `dataFileName` is the tablespace data file that was transported from the client computer; this should have been placed in the directory that contains other tablespace files for the target instance.
- For full documentation of the `dp` utility refer to the Oracle documentation. https://oracle-base.com/articles/misc/transportable-tablespaces#destination
- The BUFFER parameter must be large enough to hold the longest row anywhere among the original data or index tables. The default buffer value is system dependent and may often not be large enough to handle typical user cases.

2. To mark the tablespace with read-write privileges, connect as SYS (SYS/password. AS SYSDBA), and issue the following command:

```
ALTER TABLESPACE tSpace READ WRITE;
```

3. To recreate the Cartridge index from the detached files that you just imported, connect to the schema that owns the imported table and execute the following SQL stored procedure, specifying the appropriate Cartridge schema.

```
execute CsCartridge.AttachDetachedIndex(indexName, tableName, columnName);
```

> ***Note:*** *The host operating system and the Cartridge version must be the same as those used to create the detached index.*

Support Functions

Transforming Structure Data between Different Formats

The Oracle Cartridge supports the retrieval of chemical structure information in many formats, regardless of the format of the original data. Not only does the Cartridge support common formats such as `CDX`, `SMILES`, and `RXNfile` formats, but also rarer ones as well. Some derived data may also be generated, including molecular weights and chemical formulas. These conversion functions are grouped together in the `ConvertCDX` package.

Although each of functions has a name starting with `CDXTo`, they will function identically for data stored in both `CDX` and non-`CDX` formats.

In the discussions for each of the functions to follow, the `fld` parameter is the name of a `BLOB` or `CLOB` field where chemical structure data is stored.

These functions would typically be used within a `SQL` statement such as

```
SELECT CsCartridge.ConvertCDX.CDXToMolfile(mol) from .....
```

Retrieving Common File Formats

Chemical data may be retrieved in `CDXML`, `molfile`, `rxnfile`, `MOLFiles`, and `SMILES` formats, as well as `Base64-encoded CDX`.

CDXToCDXML(fld)

CDXToMolfile(fld)

CDXToMolfile(fld, option)

CDXToRxnfile(fld)

CDXToRxnfile(fld, option)

CDXToSMILES(fld)

CDXToB64(fld)

If the optional option parameter is supplied in the function call and has the value 'V3000', the molfile or rxnfile will be returned in V3000 rather than V2000 format. The `CDXToB64` function retrieves `Base64-encoded CDX` data. Base64 encoding is a simple and straightforward standard that converts 8-bit binary data into 6-bit textual data without the risk of losing data. Retrieving `CDX` data in this way avoids the problems associated with manipulating binary data in a `SQL` environment.

Each of these functions returns `CLOB` type data.

Note: *These functions return temporary CLOB type data. Each temporary CLOB created by the Cartridge remains in the server memory until the session is terminated. As such, repeated use of these functions within a single session will result in ever-increasing memory usage on the Host computer, eventually impacting the database server performance. Therefore, it is advised, that these functions should only be used in situations where a Client session is certain to terminate shortly thereafter.*

For backward compatibility with older versions of the Cartridge, and for use in situations where the client session might not be terminated, some functions are also available with a second parameter, as follows:

CDXToCDXML(fld, i)

CDXToMolfile(fld, i)

CDXToMolfile(fld, i, option)

CDXToRxnfile(fld, i)

CDXToRxnfile(fld, i, option)

CDXToSMILES(fld, i)

CDXToB64(fld, i)

For each function, the second parameter i contains a 1-based index referring to a 4,000-character long portion of the resulting text being returned. This is unfortunate, but necessary because Oracle limits the size of `VARCHAR2` results to 4,000 characters. When using these functions, successive 4,000-character portions should be requested until one is returned with fewer than 4,000 characters, and all returned portions must be concatenated to retrieve the desired result.

Each of these functions returns `VARCHAR2` type data.

Chemical data may also be retrieved in CML (**Chemical Markup Language**) and InChI (**IUPAC International Chemical Identifier**) formats.

CDXToCML(fld, option)

CDXToInChI(fld, option)

CDXToInChIKey(fld, option)

For each of these three functions, the second option parameter is included solely for future expansion. In the Oracle Cartridge 19.1, an empty string should be passed as the second parameter; any other value will be ignored. The `CDXToCML` function returns `VARCHAR2` data; the `CDXToInChI` function returns `CLOB` data; and the `CDXToInChIKey` function returns InChIKeys as `VARCHAR2` data.

Retrieving Molecular Weights

CDXToMolWeight(fld)

CDXToMolWeight(fld, option)

The second parameter may contain the text `MASSNATIVE` or `MASSMONOISOTOPIC`. If omitted, the default value is `MASSNATIVE`.

The `MASSNATIVE` (or omitted) option returns the average molecular weight of the structure, where atomic masses of each atom are based on the natural abundance of all isotopes of the element, so the mass of a carbon atom C would be reported as 12.011.

The `MASSMONOISOTOPIC` option returns the exact mass of the structure, where atomic masses are based on the most common isotope for the element, so the mass of a carbon atom C would be reported as 12.

Retrieving Chemical Formulas

CDXToFormula(fld, option)

The second parameter, option, may contain the empty string, or it may contain the text `SORTABLE` or `HTML`.

With the second parameter as an empty string, a plain-text formula is returned, such as `C6H6`

With the second parameter as `HTML`, a formula is returned with `HTML` formatting, such as `C<sub>6</sub>H<sub>6</sub>`. This is useful for client applications that wish to display a formula with properly-formatted superscripts and subscripts. Although it is possible in many cases for the client application to reparse the unformatted formula to add formatting, it can be very difficult to apply accurate formatting in all cases, especially in the presence of charges, radicals, or isotopes.

With the second parameter as SORTABLE, a modified formula is returned in which elements are one or two characters wide and numbers are three characters wide with zero-fill. For example, C034H044O007S002Si001.

This function returns `VARCHAR2` data.

For reactions, mass balancing components are added, in brackets, to the reactants and/or products in the formula to represent atoms not shown explicitly in the structure diagram. For nonlinear and cyclic reactions that cannot be represented by a linear formula, a gross formula for the entire reaction is returned.

Generating Chemical Names

CDXToName(fld, option)

The second parameter, option, may contain the empty string, or it may contain the text `HTML`.

With the second parameter as an empty string, a plain-text name is returned, such as `tert-butylbenzene`.

With the second parameter as `HTML`, a name is returned with `HTML`, such as *tert*-`butylbenzene`.

This function returns `VARCHAR2` data.

Generating Canonical Identifiers

It can often be useful to assign a unique identifier to each chemical substance. This is particularly helpful when identifying duplicate entries within extremely large data sets.

In many ways, the canonical identifier produced by the Cartridge serves much the same purpose as an InChI string. In both cases, the chemical meaning of a structure is identified and condensed into a concise alphanumeric string. Canonical codes are not affected by the particular drawing style used to represent a structure. That is, benzene will produce the same canonical code whether it is drawn with alternating single and double bonds, or drawn with a delocalized circle, or drawn with thick lines or green lines or any fashion that can still be recognized as benzene.

Note: *Canonical codes are constant only within a single version of the Oracle Cartridge. While a single version of the Cartridge will always produce the same canonical code for a structure or reaction, different versions of the Cartridge are guaranteed to produce different canonical codes. Accordingly, canonical codes should only be used for transient operations, and must never be stored in a database for future use.*

CDXToCanonical(fld)

CDXToCanonical(fld, option)

The second parameter, option, may contain `NORMAL` or `STEREO`, or it may be omitted. If omitted, `NORMAL` is the default.

When used with the `STEREO` option, the canonical code considers stereochemistry, so (R)-butan-2-ol and (S)-butan-2-ol will produce different canonical codes. With the `NORMAL` option (or with the second parameter omitted), the canonical code disregards stereochemistry, so (R)-butan-2-ol and (S)-butan-2-ol will produce the same canonical codes. This function returns `VARCHAR2` data.

Identifying Chemical Reaction Data

For the most part, the Oracle Cartridge operates interchangeably with chemical structures and chemical reactions. Some records can store chemical structures in a given field, while other records store chemical reactions in the same field, and the Oracle Cartridge will search both formats at the same time.

In some cases, however, it is important to know whether a given record does contain reaction information. For example, it is possible to export a `rxnfile` only for fields that do contain reactions.

CDXToIsReaction(fld)

Returns 1 if the record contains a reaction, and 0 otherwise.

Calculating Tanimoto Similarity

CDXToSimilarity(fld, query4000, query8000)

The `CDXToSimilarity` function returns the Tanimoto similarity value calculated by comparing the data stored in the specified field to the query specified in the second and third parameters. The query must be specified the same way as for the MoleculeContains operator discussed in Chapter 6.

This function returns an integer value in the range 0…100.

Note: *When you perform a MoleculeContains similarity search, retrieve the computed values with the Similarity () ancillary operator instead of computing it again with CDXToSimilarity().*

Generating Screening Information

CDXToScreen(fld, srcSelection, formSelection)

The srcSelection parameter identifies the type of screen to generate, with acceptable values `NORMAL`, `FULLEXACT`, `SKELETAL`; only `NORMAL` is recommended for use with the Similarity operator. If the requested screen does not exist, or if the input is incorrect, an empty string will be returned.

The `formSelection` parameter specifies the form of the output, with acceptable values as `LIST`, `BITSTRING`, or the empty string, where `BITSTRING` is the default value.

The `CDXToScreen` function is intended for use with the Similarity (fld, option) operator. It is not required with the Similarity Ancillary operator.

Analyzing Cartridge Performance

When analyzing server performance, it is often helpful to measure how much time is occupied by operations within the Cartridge. For example, if the Cartridge is found to be using very little time, then other factors such as insufficient memory or excessive disk usage might be responsible for perceived slowness.

CDXToTiming(fld, option)

The option provides fine-grained analyses of various aspects of Cartridge performance, with acceptable values as `ALL`, `CANONICAL`, `MISC`, `GETCDX`, `COMPRESS`, `SCREENS`; only `ALL` is recommended for general use.

This function returns an integer representing the elapsed time, in milliseconds, required to process the given record.

Additional diagnostic information about Cartridge search with a specified query and target structure can be obtained with the SearchAnalysis function:

```
CDXToSearchAnalysis(fld, query4000, query8000, options, i)CDXToSearchAnalysis
(fld, query, options, i)
```

The query and options are specified the same way as for the MoleculeContains operator described in Chapter 6. The query can be either a pair of VARCHAR2 values, each no longer than 4,000 characters, or an XArray, BLOB, or CLOB.

This function returns a VARCHAR2, and the final parameter i is a 1-based index specifying which 4,000-character portion of the result is to be returned (since a VARCHAR2 cannot hold more than 4,000 characters). A typical report is longer than 4,000 characters, so to obtain the full report, the function should be called repeatedly with increasing values of i until an empty string is returned.

Most search information is accessible only in debug versions of Cartridge, so the report from this function contains minimal information in production. It is intended primarily for Oracle Cartridge diagnostic use.

Optimizing Chemical Structures for Distribution

In order to operate seamlessly with many different types of input, the Oracle Cartridge converts all types of chemical structure data into a single optimized format for use internally. That conversion is normally performed invisibly, and as required. When preparing a table for distribution, however, it can be useful to perform that conversion a single time before distributing the table, rather than repeating the conversion at every separate destination.

CDXToMST(fld, option)

The `CDXToMST` function returns a `BLOB`, which is an augmented `CDX` document. The new `CDX` document contains the original `CDX` data unmodified, so that it will be maintained transparently by ChemDraw and other programs. The new `CDX` document also contains data calculated by the Cartridge, yet not stored in the index structures. So, if a database is to be distributed as an Oracle dump file, then the following preparation is recommended:

UPDATE structures SET cdx = CsCartridge.ConvertCDX.CDXToMST(cdx, ''); COMMIT;

1. Create a dump file of the structures table.
2. Verify that it does not have a Cartridge index.
3. Using the installer program, import the dump file using the imp program

4. Create the domain index on the cdx column.

With this procedure the Cartridge will use the data created in Step 1 instead of processing the cdx document again. This cuts the time used to create an index by about 20 percent.

The `CDXToMST` function is not recommended for use in any other situations.

📝 ***Note:*** *It is even faster to distribute already-indexed tables by means of a transportable tablespace.*

Generating Checksums

The Cartridge also provides an opportunity to check data integrity by retrieving a md5 checksum for any `CLOB` or `BLOB` field. There is a separate CheckSum package devoted to calculating checksums.

Invoke this feature in a `SELECT` statement that uses the following format:

```
SELECT CsCartridge.CheckSum.Calculate(fld) FROM ...
```

This function returns `VARCHAR2` data.

Operators for Fast Retrieval of Properties

In addition to the `CDXTo...()` functions described above, the Oracle Cartridge also provides similar functionality as operators.

Operators typically will retrieve information more quickly than functions, because the operators are able to take advantage of precalculated data within the Cartridge indexes.

However, operators are more limited than functions in that they cannot be used in a `WHERE` clause. If an operator is mistakenly included in a `WHERE` clause, an error message will result: **ORA-29902: error in executing ODCIIndexStart() routine**

For those reasons, operators are used within a `SELECT` clause to format fields for output. For example:

SELECT CsCartridge.MolWeight(mol), CsCartridge.Formula(mol, ''), CsCartridge.Canonical(mol) FROM moltable;

The following operators are available:

MolWeight(fld)

Behavior is identical to the `CDXToMolWeight` function above, returning the average molecular weight.

Formula(fld, option)

Behavior is identical to the `CDXToFormula` function above. Fastest performance is available only when the option field remains empty, however.

Canonical(fld)

Behavior is identical to the `CDXToCanonical` function above, returning the stereochemically-unaware canonical code.

Similarity(fld, option)

Unlike the CDXToSimilarity function above, the Similarity operator has two parameters, the first being a BLOB or CLOB field referring to the target structure, and the second being a VARCHAR2 field. The latter contains a screen

bitvector for the query structure as generated by the CDXToScreen function. A sample SQL statement illustrating the usage of the Similarity operator follows:

```
SELECT id, CsCartridge.Similarity(mol, (select CsCartridge.ConvertCDX.CDXToScreen
('c1ccccc1',

'NORMAL', '') from dual)) "similarity"

FROM moltable

WHERE CsCartridge.MoleculeContains(mol , 'c1ccccc1', '', 'SIMILAR=

YES,SIMTHRESHOLD=90' )=1;
```

This `SQL` statement retrieves all records from `moltable` with a structure in the `mol` field that has a Tanimoto similarity of at least 90% relative to benzene and displays the Tanimoto value. The benzene query appears once as part of the `MoleculeContains` function in the `WHERE` clause, and again as part of the `CDXToScreen` function call in the `SELECT` statement returning the screen bitvector.

Note: *The usage of the Similarity (fld, option) operator is now deprecated and the ancillary form, Similarity (n), should be used instead when possible. For more information, see "Ancillary operators" below.*

Ancillary Operators

Operators ancillary to the `MoleculeContains()` operator provide an efficient way to retrieve data generated during search operations. An ancillary operator should appear in the select list with a single literal argument, which must match an ancillary argument appended to the primary `MoleculeContains()` operator in the predicate. Since multiple `MoleculeContains()` operators could appear in a complex predicate, the ancillary argument allows the correct association to be made between the primary and ancillary operators.

Similarity()

The ancillary `Similarity()` operator provides a convenient and efficient way to retrieve the Tanimoto coefficients calculated during a similarity search. For example, the following query retrieves two rows, with associated similarity scores, from the test database created in the CSCUser schema during the Cartridge installation test:

```
select id, CsCartridge.Similarity(1) similarity from teststructures where
CsCartridge.MoleculeContains(m, (select m from teststructures where id=3),
 'similar=yes,simthreshold=90', 1)=1;
```

ID=3 in the table is benzene. The ancillary argument ("1" in this case) is arbitrary, but must be a literal and must be unique for each ancillary association in the query.

Highlight()

The ancillary `Highlight()` operator allows retrieval from a substructure search of CDX files in which color has been applied to the query that was hit within the original full structure. The same type of ancillary argument as described above for the `Similarity()` operator must be used to associate the primary and ancillary operators. In addition, since recording highlight data slows the search slightly, you must request it by adding `'HIGHLIGHT=YES'` to the `MoleculeContains()` option list. For example, the following query retrieves 11 rows, with highlighted CDX files that contain an aliphatic six-membered ring from the test database:

```
select id, CsCartridge.Highlight(1) hl from teststructures where
CsCartridge.MoleculeContains(m, 'C1CCCCC1', '', 'highlight=yes', 1)=1;
```

Each CDX is returned as a CLOB of which you see only the first few lines in typical command-line usage. This would normally be used in a programming environment where the full CLOB can be retrieved for display or file storage. A PL/SQL example:

```
BEGIN
 FOR row IN (
  select id, CsCartridge.Highlight(1) hl from teststructures
   where CsCartridge.MoleculeContains(m, 'C1CCCCC1', '', 'highlight=yes', 1)=1
 ) LOOP
  CsCartridge.Files.WriteFile('<path on your server>' ||
   'hit-' || row.id || '.cdxml', row.hl);
 END LOOP;
END;
/
```

AUX Package

There are some applications that need to refer directly to data maintained by the Cartridge in the tables that constitute a domain index. This is typically done as an optimization. However, direct references to Cartridge tables may generate errors because the Cartridge data model is likely to change from release to release, and names or columns of index tables may change. As a result, such applications are difficult to maintain and may not work with later Cartridge versions.

To help address this problem, the package CsCartridge.AUX is made available to application programmers. The CsCartridge.AUX package consists of a set of functions that return table and column names in which certain index data are stored. Although you cannot ensure that a given column in Cartridge will continue to exist in future or that its properties and functionality will remain unchanged, use of the AUX package will help improve the functioning of applications which depend on the internals of Cartridge indexes.

The CsCartridge.AUX functions return VARCHAR2 values. Each one has either zero or two arguments: the arguments identifying an index by the name of the schema in which it was created and the name of the index. The functions are as follows:

- `MWTabName(schemaName, indexName)`: Returns the schemaName.tableName of the table containing precomputed molecular weights.
- `MWRidName`: Returns the column name of the field holding ROWID cross-references to the indexed table containing the structures for which the molecular weights were computed.
- `MWFldName`: Returns the column name of the molecular weight field.
- `FMTabName(schemaName, indexName)`: Returns the schemaName.tableName of the table containing precomputed formulae.
- `FMRidName`: Returns the column name of the field holding ROWID cross-references to the indexed table containing the structures for which the formulae were computed.

- `FMFldName`: Returns the column name of the formula field. This column is of type VARCHAR2(200). If a formula is longer than 200 characters, a NULL is stored, and the formula must be computed from the original structure with ConvertCDX.CDXToFormula().
- `CNTabName(schemaName, indexName)`: Returns the schemaName.tableName of the table containing precomputed canonical codes.
- `CNRidName`: Returns the column name of the field holding ROWID cross-references to the indexed table containing the structures for which the canonical codes were computed.
- `CNFldName`: Returns the column name of the canonical code field.

The following Visual Basic example illustrates how to use the AUX functions to retrieve the row ids of two different tables joined by their canonical codes. This `SELECT` statement will retrieve references to only those records that are chemically identical in both tables.

In this example, the index-table names do not appear; the index names are used instead to specify the tables.

```
Dim CNTab1, CNTab2, CNRidName, CNFldName As String
Set c = New ADODB.Connection
Dim r As ADODB.Recordset
c.Open ConnectionString:="Provider=OraOLEDB.Oracle.1;Persist Security
Info=False;Data Source=" + "myserver", UserId:="myschema",
Password:="mypassword"
sql = "SELECT CsCartridge.Aux.CNTabName('myschema', 'my1index') FROM DUAL"
Set r = c.Execute(sql)
If Not r.EOF Then CNTab1 = r.Fields(0)
sql = "SELECT CsCartridge.Aux.CNTabName('myschema', 'my2index') FROM DUAL"
Set r = c.Execute(sql)
If Not r.EOF Then CNTab2 = r.Fields(0)
sql = "SELECT CsCartridge.Aux.CNRidName FROM DUAL"
Set r = c.Execute(sql)
If Not r.EOF Then CNRidName = r.Fields(0)
sql = "SELECT CsCartridge.Aux.CNFldName FROM DUAL"
Set r = c.Execute(sql)
If Not r.EOF Then CNFldName = r.Fields(0)
sql = "SELECT " + CNTab1 + "." + CNRidName + ", " + CNTab2 + "." + CNRidName +
" FROM " + _
CNTab1 + ", " + CNTab2 + " WHERE " + CNTab1 + "." + CNFldName + " = " + CNTab2
+ "." + CNFldName
Set r = c.Execute(sql)
```

Disabling Advanced Chemical Intelligence Features

At times, users specifically want to disable features in the Cartridge to produce less-intelligent behavior. The Cartridge allows users to disable some of its advanced chemical intelligence.

When the Cartridge is first loaded into memory, it will look for the CS_Global_Chemical_Preferences.txt file in the same directory as the CsCartridge shared library. If the file is available, it will be scanned for keyword=value value pairs terminated either by comma, or put in separate lines.

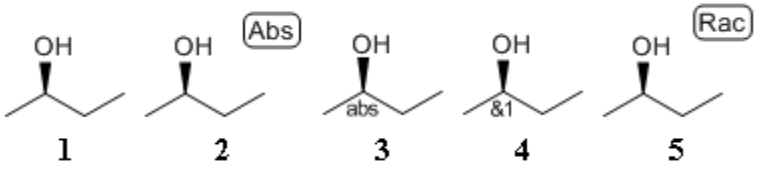
The recognized keywords are:

- `AbsoluteConfigurationRequiresChiralFlag`
- `RecognizeStereoHaworth`
- `RecognizeStereoPerspective`
- `RecognizeStereoFischer`
- `RecognizeStereoCumulene`
- `RecognizeStereoHinderedBiaryl`
- `NarrowEndStereoOnly`

The keywords are not case sensitive. Each value is either YES or NO. The CS_Global_Chemical_Preferences.txt file should contain no spaces.

Caution: If this file is changed, all existing indexes become invalid and must be dropped, and then recreated.

AbsoluteConfigurationRequiresChiralFlag

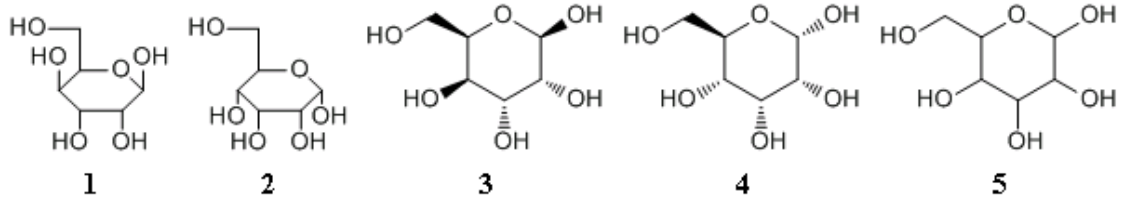


With AbsoluteConfigurationRequiresChiralFlag=NO, a structure with stereocenters is interpreted as representing only the specific stereoisomer drawn. Structures 1, 2, and 3 above will be interpreted as identical. Structures 4 and 5 above will be interpreted as identical. Structures 1 and 5 above will be interpreted as different.

With AbsoluteConfigurationRequiresChiralFlag=YES, a structure with stereocenters is interpreted as representing a mixture of the specific stereoisomer drawn and its enantiomer. The only way to represent a pure enantiomer is by adding a Chiral flag (in a Molfile) or an ABS flag (in a CDX file). Structures 1, 4, and 5 above will be interpreted as identical. Structures 2 and 3 above will be interpreted as identical. Structures 1 and 2 above will be interpreted as different.

The default is NO.

RecognizeStereoHaworth

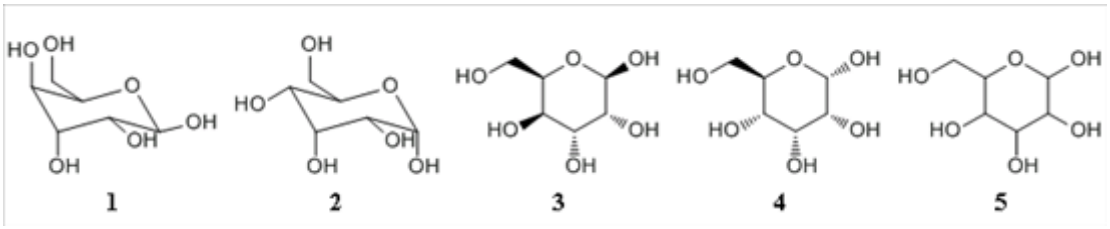


With RecognizeStereoHaworth=YES, a structure drawn as a Haworth projection is interpreted as having specified stereochemistry. Structures 1 and 3 above will be interpreted as identical. Structures 2 and 4 above will be interpreted as identical. Structures 1 and 2 above will be interpreted as different. Structure 5 above as different from any of the others.

With RecognizeStereoHaworth=NO, a structure drawn as a Haworth projection is interpreted as not having specified stereochemistry. Structures 1, 2, and 5 above will be interpreted as identical. Structures 1 and 3 above will be interpreted as different. Structures 2 and 4 above will be interpreted as different.

The default is YES.

RecognizeStereoPerspective

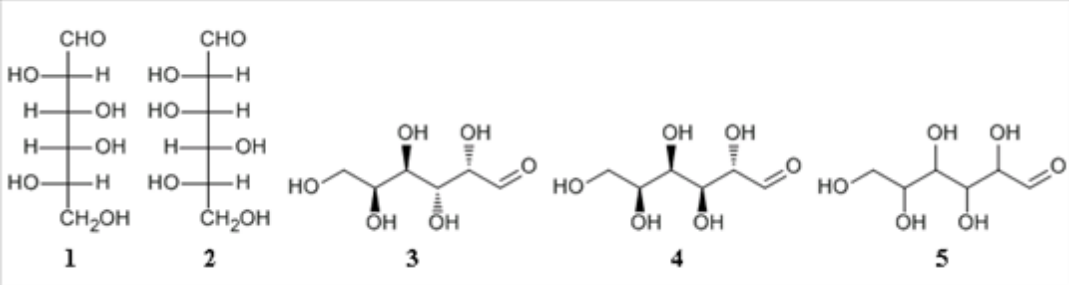


With RecognizeStereoPerspective=YES, a structure drawn as a perspective diagram is interpreted as having specified stereochemistry. Structures 1 and 3 above will be interpreted as identical. Structures 2 and 4 above will be interpreted as identical. Structures 1 and 2 above will be interpreted as different. Structure 5 above as different from any of the others.

With RecognizeStereoPerspective=NO, a structure drawn as a perspective diagram is interpreted as not having specified stereochemistry. Structures 1, 2, and 5 above will be interpreted as identical. Structures 1 and 3 above will be interpreted as different. Structures 2 and 4 above will be interpreted as different.

The default is YES.

RecognizeStereoFischer

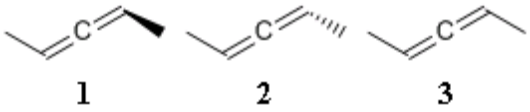


With RecognizeStereoFischer=YES, a structure drawn as a Fischer projection is interpreted as having specified stereochemistry. Structures 1 and 3 above will be interpreted as identical. Structures 2 and 4 above will be interpreted as identical. Structures 1 and 2 above will be interpreted as different. Structure 5 above as different from any of the others.

With RecognizeStereoFischer=NO, a structure drawn as a Fischer projection is interpreted as not having specified stereochemistry. Structures 1, 2, and 5 above will be interpreted as identical. Structures 1 and 3 above will be interpreted as different. Structures 2 and 4 above will be interpreted as different.

The default is YES.

RecognizeStereoCumulene

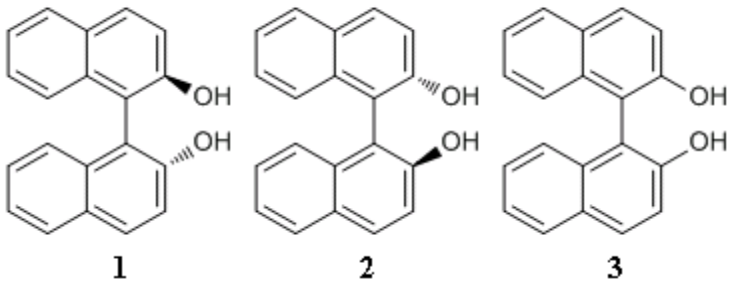


With RecognizeStereoCumulene=YES, stereochemistry is recognized in cumulenes. All three structures above will be interpreted as different.

With RecognizeStereoCumulene=NO, stereochemistry is not recognized in cumulenes. All three structures above will be interpreted as identical.

The default is YES.

RecognizeStereoHinderedBiaryl

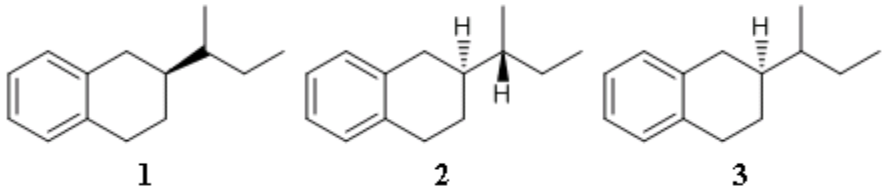


With RecognizeHinderedBiaryl=YES, stereochemistry is recognized in biaryls. All three structures above will be interpreted as different.

With RecognizeHinderedBiaryl=NO, stereochemistry is not recognized in biaryls. All three structures above will be interpreted as identical.

The default is YES.

NarrowEndStereoOnly



1 2 3

With NarrowEndStereoOnly =NO, stereochemistry is recognized at both ends of a wedged bond. A bold wedged bond interpreted as UP from atom A to atom B is also interpreted as DOWN from atom B to atom A. Structures 1 and 2 above will be interpreted as identical. Structures 1 and 3 above will be interpreted as different.

With NarrowEndStereoOnly =YES, stereochemistry is recognized only at the narrow end of a wedged bond. The atom at the wide end of a wedged bond is interpreted as having unspecified stereochemistry (unless it is also at the narrow end of some other bond). Structures 1 and 3 above will be interpreted as identical. Structures 1 and 2 above will be interpreted as different

The default is NO.

Chemical Queries

Substructure Searching

MoleculeContains Operator

Searching by chemical structure (including by substructure or chemical similarity) can be performed by using the MoleculeContains operator in a `SELECT` statement in one of the following two formats:

```
SELECT ... FROM tab WHERE CsCartridge.MoleculeContains(fld, query4000, query8000, options)
```

or

```
SELECT ... FROM tab WHERE CsCartridge.MoleculeContains(fld, query, options)=1
```

where:

- `tab` – Name of the table to be searched.
- `fld` – Name of the chemical structure field indexed by the Cartridge.
- `query4000` – First 4,000 characters of the query string. The query string can be an encoded `CDX` document, a `SMILES` string, a `MOLfile`, a `RXNfile`, a chemical name, or an InChI string. For more information about these formats, refer to DDL Considerations. The Oracle database limits the size of string literals to 4000 characters in length.
- `query8000` – Second 4,000-characters of the query string.
- `query` - A single BLOB, CLOB, or XArray holding a query of any length in any supported format.

See "Using large queries" later in this chapter for an example of queries with greater than 8,000 characters to represent the query structure.

- `options` – A string containing `keyword=value` pairs separated by commas. Keywords fall into three categories: those used for testing and troubleshooting; those that communicate with the Oracle Query Optimizer, and those used for chemical structure matching. See the sections that follow for a description of the options in each category.

MoleculeHas Operator

Searching by types or classes of chemical structures can be performed by using the MoleculeHas operator in a SELECT statement in the following format.

```
SELECT ... FROM tab WHERE CsCartridge.Moleculehas(structure field, options)=1
```

where:

- `tab`– Name of the table to be searched.
- `options` - Options should be in the following format.

  ```
  'PROPERTY=[STEREOCENTERS|MISSINGSTEREO|CHARGED|ISOTOPIC|QUERY]'
  ```

  where:

  `STEREOCENTERS`: Finds structures containing tetrahedral stereochemistry.

MISSINGSTEREO: Finds structures with atoms capable of tetrahedral or double bond stereochemistry, but drawn as “unspecified”.

CHARGED: Finds one or more atoms with a charge.

ISOTOPIC: Retrieves records containing a structure with an atom that either has a mass number, or has an Isotopic Abundance attribute.

QUERY: Locates structures with any generic variability such as atom types (R, A, M, Q, X, Alkyl, Aliphatic, EDG, EWG), bond types (single-or-double, etc.), variable charge, element lists ([C,N,O]), ring bond count, etc.

The following is an example SELECT statement with the MoleculeHas operator.

```
SELECT count* FROM SUBSTANCE WHERE CsCartridge.Moleculehas(Base64_cdx, 'PROPERTY=
QUERY')=1;
```

Explanation of Chemical Search Keywords

This section explains how to use the chemical search keywords to obtain specific search results. For example, the following SELECT statement retrieves the query structure and all tautomeric forms of that structure:

SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, 'mymol', '', 'IDENTITY=YES, TAUTOMER=YES')=1;

The following sections explain:

- Modes of Chemical Searching
- Refining Search Results
- Dependencies Among Keywords
- Useful Combinations of Keywords
- Deprecated Keywords

Modes of Chemical Searching

The Oracle Cartridge provides four modes of chemical searching:

- Identity
- Full structure
- Substructure
- Similar

Each mode corresponds to settings for the following keywords: IDENTITY, FULL, and SIMILAR.

Identity Search

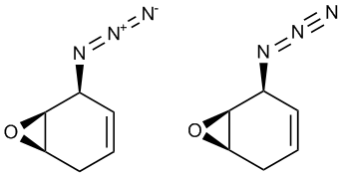
Syntax:

```
IDENTITY=YES
```

Default is NO

Identity search is intended for use in compound registration, when you must know whether a perfectly identical copy of your query compound is already present in the database. The target must be chemically identical to the

query, including stereochemistry, charges, and isotopy. For example, the following structures are chemically identical, although they use different conventions for drawing the azide functional group:



The Oracle Cartridge perceives these structures as identical because normalization algorithms enable the software to recognize different representations of a structure as the same chemical entity.

On the other hand, identity search by itself does *not* perceive the keto and enol forms of guanine as identical, because the structures are tautomers, and therefore represent different chemical entities.

Full Structure Search

Syntax:

`FULL=YES`

A full structure search retrieves structures that match the query structure and contain no additional non-hydrogen atoms directly bonded to any of the atoms in the query structure. Additional unbonded atoms, however, are allowed by default in a Full Structure search, so a Full Structure query for acetic acid would hit sodium acetate. To disallow the presence of additional fragments, see the `PERMITEXTRANEOUSFRAGMENTS` option discussed below.

Substructure Search

Syntax:

`FULL=NO`

A substructure search retrieves structures that contain the query structure embedded within them. The default is `NO`.

Similarity Search

Any other substructure search options will be ignored if similarity is specified. Similarity and other structure search options are mutually exclusive. If the similarity score (the Tanimoto coefficient) is also needed, use the Similarity() ancillary operator in the select list.

Syntax:

`SIMILAR=YES,SIMTHRESHOLD=<`*`number`*`>`

Where <*number*> is an integer from 0 to 100 that represents the degree of similarity expressed as a percentage. Default for `SIMILAR` is `NO`.

Default for `SIMTHRESHOLD` is 90 percent.

A Similar search finds structures that “look like” the query. Similarity searches are by their nature “fuzzy.” What “looks like” means is obviously subject to interpretation and depends on the application. For example, in medicinal applications the drug absorption properties are relevant, while in a toxicological context the metabolism is of interest.

Similar searching relies on the notion of molecular descriptors. Each compound can be represented by a collection of qualitative terms that describe general aspects of the structure.

For example, benzoic acid might be described as:

- organic acid
- contains 6-membered ring
- contains delocalized ring
- contains C=O

As you can see, the descriptors can be very broad, and they can overlap. The set of descriptors used by the Oracle Cartridge is very large.

During a similarity search, the Oracle Cartridge evaluates the query and target structures against all of the descriptors it knows about. Some descriptors will be present in a given compound and some will not. Since each descriptor for a given compound is either Present or Not Present, they are often stored as bits, and another name for a compound's set of descriptors is its *bitscreen*.

Consider the following query and target compounds:

Bitscreen	Desc.	#	Name
	Query	23	Q
	Target	24	T
	Q and T	17	$Q \cap T$
	Q or T	30	$Q \cup T$

Both have similar numbers of descriptors present -the query has 23, while the target has 24. But are they "close enough?"

One of the hallmarks of a good similarity algorithm is whether it is c*ommutative*. That is, two compounds should have the same similarity value no matter which you compare to which. The Tanimoto similarity test is commutative. It compares the number of descriptors they have in common (in the intersection of the query and the target) to the number of descriptors they have in total (in the union of the query or the target). The ratio of these two values is known as a *Tanimoto coefficient*, and is always a value between 0% and 100%.

Tanimoto coefficient

For the two compounds above, the Tanimoto coefficient is 17/30, or about 57%. This is not very similar. Although the Oracle Cartridge allows you to specify any Tanimoto value down to 0%, in most cases you will likely be looking for compounds that have Tanimoto coefficients of 90% or higher.

Refining Search Results

Use these keywords to fine-tune the results of a search, by specifying:

- Whether the query matches its tautomeric forms: `TAUTOMER`. Additionally, this option can be used to fine-tune the results of an `IDENTITY` search.

- How closely the stereoconfiguration at asymmetric carbons in the query matches the target: `TETRAHEDRALSTEREO`
- Whether the stereoconfiguration of E/Z geometric double bonds in the query matches the target: `DOUBLEBONDSTEREO`.
- Whether uncharged atoms in the query match a charged atoms in the target: `HITANYCHARGECARBON` and `HITANYCHARGEHETERO`
- In reaction queries, whether bonds that change in the reaction (reaction centers) must match: `REACTIONCENTER`
- How to match substances that consist of multiple fragments, such as salts, hydrates, and metal complexes: `SALTSPLITTING`, `PERMITEXTRANEOUSFRAGMENTS`, and `PERMITEXTRANEOUSFRAGMENTSIFRXN`

Note: *When IDENTITY=YES, keywords other than TAUTOMER are ignored.*

For details on each option, see the sections that follow.

ADDAROMATICPREDICATE

Syntax:

```
ADDAROMATICPREDICATE [YES|NO]
```

Default is `NO`. The `ADDAROMATICPREDICATE` flag value can be manually set to either `YES|NO` in the CsCartridge.GLOBALS table. Updating the flag value will override what is in the GLOBALS table.

When `ADDAROMATICPREDICATE=YES`, the substructure queries will be modified to have extra predicates added to single and double bonds as follows:

- single bond queries will be augmented to single or aromatic
- double bond queries will be augmented to double or aromatic

This will allow queries with exocyclic single (or double) bonds to match single (or double) bonds in aromatic rings. So, for example a toluene query would match naphthalene.

DOUBLEBONDSTEREO

Syntax:

```
DOUBLEBONDSTEREO [YES|NO]
```

Default is `YES`.

When `DOUBLEBONDSTEREO=YES`, then double bonds with E/Z geometric stereochemistry match only bonds with the same stereoconfiguration.

HITANYCHARGECARBON and HITANYCHARGEHETERO

Syntax:

```
HITANYCHARGECARBON [YES|NO]
```

Default is `YES`.

```
HITANYCHARGEHETERO [YES|NO]
```

Default is `YES`.

When HITANYCHARGECARBON=YES, uncharged carbon atoms in the query can match charged atoms in the target. When set to NO, uncharged carbons hit only uncharged carbons.

HITANYCHARGEHETERO works similarly for non-carbon atoms (heteroatoms).

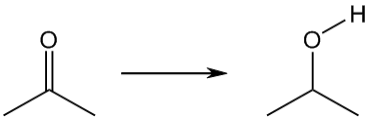
REACTIONCENTER

Syntax

REACTIONCENTER [YES | NO]. Default is YES.

A *reaction center* is a bond that participates in a reaction. When REACTIONCENTER=YES, the query retrieves only reactions in which the keto group is transformed as shown. When REACTIONCENTER=NO, this requirement is dropped and additional reactions may be retrieved.

In the following figure, the C=O bond in the reactant is a reaction center, because it is transformed into a single bond in the product. The O-H bond in the product is also a reaction center because it is created in the reaction.



PERMITEXTRANEOUSFRAGMENTS

Syntax:

PERMITEXTRANEOUSFRAGMENTS=[YES|NO]
Default is NO.

When set to YES, the query matches structures that contain additional fragments. For example, the acetate ion matches sodium acetate and potassium acetate.

PERMITEXTRANEOUSFRAGMENTSIFRXN

Syntax:

PERMITEXTRANEOUSFRAGMENTSIFRXN=[YES|NO]
Default is NO.

This option is the same as PERMITEXTRANEOUSFRAGMENTS, but applies solely to reactions.

SALTSPLITTING

Syntax:

SALTSPLITTING=[YES|NO]
Default is NO.

When SALTSPLITTING=YES, then salt counterions in the query are ignored, and both the parent compound and salts with different counterions are retrieved. For example, a sodium acetate query retrieves acetic acid, sodium acetate, and potassium acetate.

This option is intended for salts, such as sodium acetate, in which one ion is larger than the other and can be recognized as a parent compound. Results may be unexpected for queries, such as sodium chloride or tetramethylammonium hexafluorophosphate that lack an obvious parent.

TAUTOMER

Syntax

```
TAUTOMER [YES | NO]
```
Default is `NO`.

When set to `TAUTOMER=YES`, the query matches identically drawn structures and all tautomeric forms.

TETRAHEDRALSTEREO

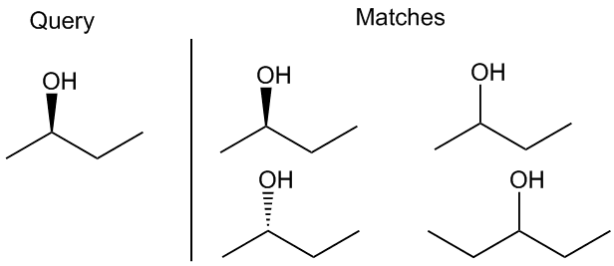
Syntax:

```
TETRAHEDRALSTEREO [SAME|EITHER|ANY|IDENTITY]
```
Default is `ANY`.

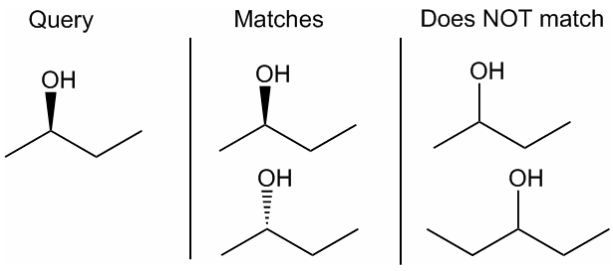
Two additional values, `YES` and `NO`, are retained for backward compatibility. These options are equivalent to `SAME` and `ANY`.

The values specify how a query with stereochemistry defined by Up and Down stereobonds will match target structures. You can ignore stereoconfiguration altogether or specify that a the query matches its stereoisomers. In substructure searching, you can specify whether a query that contains an asymmetric carbon with defined stereoconfiguration matches structures that contain it, but have a non-asymmetric carbon at that position.

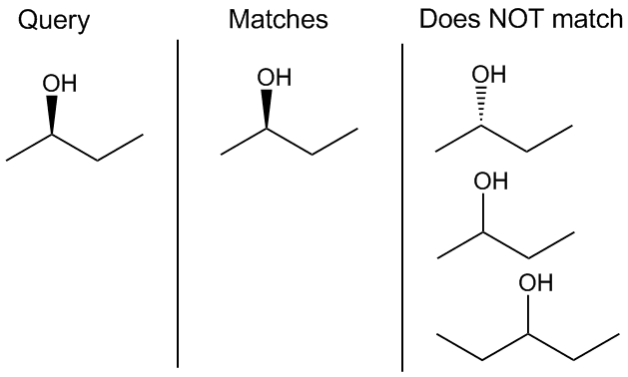
- `ANY`: Stereoconfiguration is ignored. A query with stereoconfiguration defined by Up and Down stereobonds matches structures with the same configuration, opposite configuration, unspecified configuration, and structures without tetrahedral stereochemistry:



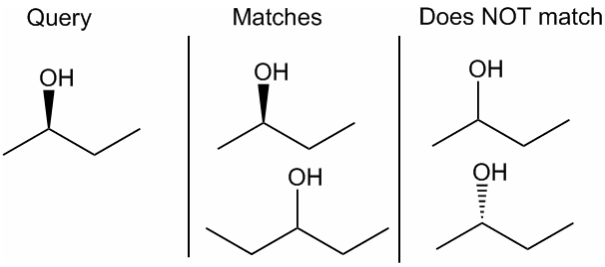
- `EITHER`: Fully defined stereoconfigurations in the query match the same or opposite stereoconfiguration in the target, but do not match structures with unspecified stereoconfiguration and structures without tetrahedral stereochemistry:



- `IDENTITY`: Stereogenic centers with explicitly defined stereoconfiguration match only those with the same stereoconfiguration, but do not match structures with opposite stereoconfiguration, unspecified stereochemistry and structures without tetrahedral stereochemistry.



- SAME: Similar to IDENTITY, except that stereogenic centers with explicitly defined stereoconfiguration match structures without tetrahedral stereochemistry:



For information on stereochemical perception, see “Stereochemistry” earlier in this chapter.

Dependencies Among Keywords

IDENTITY

When IDENTITY=YES, chemical search options other than TAUTOMER are ignored. For example, the following SELECT statement does *not* retrieve stereoisomers:

SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, 'D-glucose.cdx', '', 'IDENTITY=YES, TETRAHEDRALSTEREO=EITHER')=1;

SIMTHRESHOLD

SIMTHRESHOLD is ignored when SIMILAR=NO (the default).

Useful Combinations of Keywords

Salt and Parent Compound Search

The following SELECT statement retrieves the parent compound of the query and its salts:

SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, 'mysalt.cdx', '', 'FULL=YES,SALTSPLITTING=YES')=1;

Isomer Search

The following SELECT statement retrieves tetrahedral and geometric stereo isomers of the structure query:

SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, 'mysalt.cdx', '', 'FULL=YES, TETRAHEDRALSTEREO=ANY,DOUBLEBONDSTEREO=NO')=1;

Deprecated Keywords

These keywords have been deprecated and are retained solely for backward compatibility.

Syntax

`RELATIVETETSTEREO=[YES|NO]`
Default is `NO`.

`ABSOLUTEHITSREL=[YES|NO]`
Default is `NO`.

`RELATIVETETSTEREO` activates special treatment of unwedged bold and hashed bonds, so that they are interpreted as indicating relative stereochemistry while the wedged versions indicate absolute stereochemistry. When set to `NO`, unwedged bonds are interpreted the same as the corresponding wedged bonds.

`ABSOLUTEHITSREL` is ignored unless `RELATIVETETSTEREO=YES`. When set to `ABSOLUTEHITSREL=YES`, structures with absolute stereochemistry specified by wedged bonds also match structures with relative stereochemistry specified by unwedged bonds.

Using Large Queries

If a query exceeds 8,000 bytes, another format of the MoleculeContains operator can be used to pass queries up to a length of 4 million bytes by means of a CLOB, BLOB, or XArray.

Since a BLOB or CLOB can be used directly as a query, you can store queries of any length in a table and then use them in the MoleculeContains operator with a subselect.

For example, if you have a table in your schema named QUERIES with a QUERY column, a SELECT using MoleculeContains and a subselect could look like this:

```
SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, (SELECT query
FROM queries WHERE id = 1234), '')=1;
```

Since the query is a LOB, there is no second query argument.

Older versions of the Cartridge that did not accept LOB queries permitted a `VARCHAR2 SELECT` string to be entered as the query argument. This form of query is still accepted, although it is more limited than a subselect and should be used carefully to avoid SQL-injection security risks.

To pass a `SELECT` statement as a string:

- The first member of the `SELECT` statement's select list must be a `CLOB` or `BLOB` field, and the execution of the `SELECT` statement should result in one output record. This data will be used by the Cartridge as the query structure.
- The `SELECT` statement may not use any Cartridge operators because Oracle does not permit recursive data-cartridge operations. This select will be executed from within the Cartridge, so any tables it references must be in the Cartridge schema or must be fully qualified.
- The third parameter should be left blank.
- The application program must store the query structure before using the `MoleculeContains` operator.

- The `CsCartridge` user must have access to the table or tables referenced in its parameter. The programmer can use the Queries table in the CsCartridge schema to store query structures. The Queries table has three fields:
  - An ID - a numerical identifier (a primary key).
  - A Text field – an optional string that may be used to add some meaningful explanation to the query
  - The Query itself.

The following code example demonstrates the programming required for the use of this feature:

```
INSERT INTO CsCartridge.Queries(id, note, query) VALUES(1234, 'benzene query',
'c1ccccc1');
SELECT mol FROM moltable WHERE CsCartridge.MoleculeContains(mol, 'SELECT query
FROM CsCartridge.Queries WHERE id = 1234', '', '')=1;
```

With an XArray, a maximum of 1,000 segments each with a maximum size of 4,000 characters can be passed to the Cartridge in the place of the second parameter. However, even this use of an XArray fails when using ADO if the query structure is passed as a list of literals. The XArray strategy can be used only if the query is specified as a list of parameters and the parameters are passed separately.

The `SELECT` statement with an XArray would look like this:

```
SELECT ... FROM tab WHERE CsCartridge.MoleculeContains(fld, CsCartridge.XArray
(:xpar0, :xpar1, :xpar2,...), options)=1
```

where each parameter (`xpar`) represents a string up to 4,000 characters in size. That 4,000 characters is a theoretical value. Due to an Oracle bug, the safe value is 666. For more information about the other variables in the above statement, see the “Substructure Searching” section earlier in this chapter.

The programming required to pass the parameters to the Oracle database varies according to the environment used to develop the client application.

Using Optimizer Hints

The `MoleculeContains` operator has a dual implementation. One of those implementations can employ the Cartridge index and is used when the query optimizer decides to use index scan, and the other implementation is a simple function called when the optimizer decides against using the Cartridge index. Most commonly, the Cartridge index might not be used when there is a chance that including other conditions in a `WHERE` clause result in higher selectivity. In addition, ignoring the index can also be caused by join conditions in the `WHERE` clause. Picking the right algorithm for executing a `SELECT` statement can cause dramatic differences in speed. If the performance is poor when a complicated `SELECT` statement is executed, optimizer hints can be used to nudge the optimizer toward preferred execution methods.

The `INDEX` hint can be used to instruct the optimizer to prefer using a particular index. The syntax of the `INDEX` hint is:

```
SELECT /*+ INDEX(table_name index_name) */ .....
```

`Table_name` and `index_name` are names of the preferred table and index, respectively.

Using the `INDEX` hint is recommended when it is in doubt which implementation will be used by the optimizer.

Formula Queries

Use the FormulaContains operator to select a subset of a table by setting constraints on the formula. This operator can be used in a `SELECT` statement as follows:

```
SELECT ... FROM tab WHERE CsCartridge.FormulaContains(fld, formula, options)=1
```

where:

- `tab` – The name of the table to be retrieved
- `fld` – The name of the field indexed by the Cartridge
- `formula` – The list of atom symbols, with each symbol optionally followed by a number and optionally another number preceded by a minus sign to indicate a range.
- `options` – Can include an empty string or the string '`FULL=YES`' or '`FULL=NO`'. The default is `FULL=NO`.

By default, the `FormulaContains` operator selects records in which the counts of the specified atoms are in the prescribed limits. If the options field specified full match, then no other kinds of atom are allowed in the formula. To select structures that contain only the atoms specified in the query, set `FULL=YES`.

For example:

SELECT mol FROM moltable WHERE CsCartridge.FormulaContains(mol, 'C5-10H2-10Br2-4', 'FULL=YES')=1;

- This will match structures with formulas of C5H6D4Br2, C9H9Br3, and so on. With FULL=NO, it would also match a structure, such as C6H3Br2Cl because the presence of elements not specified in the query would be allowed.
- This will not match a structure with a formula of C2Br2 or C4H4Br2. Regardless of any other elements that may or may not be present, the query specifies that there must be at least 5 carbon and 2 hydrogen atoms, and that restriction must be satisfied.

Molecular Weight Queries

Use the MolWeightContains operator to select structures by molecular mass. The Cartridge allows quick search by molecular weight by the MolWeightContains operator. This operator can be used in a `SELECT` statement as follows:

```
SELECT ... FROM tab WHERE CsCartridge.MolWeightContains(fld, massMin, massMax,
options)=1
```

where:

- `tab` – Name of the table to be retrieved
- `fld` – Name of the field indexed by the Cartridge
- `massMin` – Numeric value setting the minimal molecular mass retrieved
- `massMax` – Numeric value setting the maximal molecular mass retrieved
- `options` – Currently always the empty string (provided for future compatibility)

For example:

```
SELECT mol FROM moltable WHERE CsCartridge.MolWeightContains(mol, 70.0, 70.1,
'')=1;
```

Query Optimization

The Cartridge currently provides limited support for Oracle's extensible optimizer.

The Oracle optimizer attempts to develop an execution plan for a query that will minimize search time. A well optimized query can return in a fraction of a second the same results that could require days with poor optimization. The goal for an optimum execution plan is to use the most selective indexes first. This minimizes the number of rows that must be retrieved and examined to determine whether the query conditions (called the predicate, the part of a `SELECT` statement following the word `WHERE`) are satisfied.

With its extensible optimizer, Oracle allows the Cartridge to supply several metrics by which it ranks Cartridge operators such as MoleculeContains against other operators in a predicate. Oracle uses these metrics to decide when during query execution to evaluate Cartridge operators, and whether to evaluate them through the Cartridge's indexes or, during row-by-row retrieval from the table, through the Cartridge's functional implementation.

Currently, the Cartridge implements the following optimizer metrics:

- Selectivity – A number between 0 and 100, an estimate of the percentage of rows that will satisfy the condition described by the operator's arguments.
- Cost – A positive number that indicates how expensive it will be (how much time or other resource is consumed, in relative terms) to execute either an index scan or a functional evaluation of a row.

If an index is available, most Cartridge operator options return a fairly small selectivity value, indicating they are highly selective. Identity searching is extremely selective, and substructure searches use index statistics to attempt to estimate the actual selectivity. When a good index is unavailable, a selectivity of 100 is returned to encourage Oracle to evaluate other conditions in the predicate first, because then the Cartridge operations are performed through the functional implementation, which examines each structure in detail and is thus relatively slow.

Cost values are empirical and are currently relatively rudimentary. Function calls have a fairly high cost of 1500. Most index scans have a cost of 100, but for identity searches they have a cost of 0. These costs help encourage Oracle to evaluate the most selective index scans first, and delay function calls until the set of candidate structures has been reduced by other conditions in the predicate.

The options:

```
SELECTIVITY=<number>

COST=<number>

FUNCTIONCOST=<number>
```

can be used in a query to override the values described above. This permits you to bias Oracle's optimizer to shift the order of execution within a predicate based on your own knowledge of your query and your database. It provides finer control than does the use of optimizer hints.

Be aware that Oracle evaluates optimizer metrics only once for a given operator argument list. For example, if you nest a `SELECT` statement that uses MoleculeContains as one of the parameters of another MoleculeContains operator, the metrics will be evaluated during only the outer `SELECT` and not during the nested one, even if the nested one is the more definitive part of your query. As a result, the inner query may be evaluated using metrics from the outer one and might be improperly optimized. You might need to rely on hints to control the optimizer in that situation; or avoid the situation altogether by coding differently.

Performance Tuning

General Considerations

As with any other Oracle-dependent application, the performance of the Cartridge depends on the availability of database cache memory. The more data that is already cached into main memory while performing a SELECT statement, the better the performance. Additionally, the Cartridge relies on merging lists retrieved from the database, so the size of the sort area is also important.

The most important factor is the size of the database cache. There is no definitive answer for the proper size for the database cache. Each computer environment is unique; it depends on the amount of available internal memory, and on the number of different applications running on the database server and competing for resources. However, tuning the Oracle server can have a profound effect on the performance of the Cartridge. Index creation speed on a populated table can double, and substructure search can be ten times as fast when cache size is configured properly. The size of the database cache can be checked with the following SQL commands while connected as a user with DBA privileges:

```
show parameter db_cache_size
show sga
```

The first command will typically show a zero value in Oracle 11g or later if Automatic Memory Management (AMM) is enabled and is automatically tuning the cache size. In any case, the “Database Buffers” value reported by the second command is the actual current size of the cache (all buffer pools combined). The default value without AMM depends on the platform and on the Oracle version, and is usually between 16MB and 64MB.

The amount of data in an Oracle Cartridge index can be estimated with the following SQL command while connected to the schema in which the Cartridge is installed (CSCARTRIDGE by default), replacing USERNAME_INDEXNAME with the actual user and index names and TABLENAME with the actual table name:

```
select 2*sum(BYTES) from user_segments
where segment_name in ((select 'USERNAME_INDEXNAME' from dual) union (select
segment_name from user_lobs where table_name='TABLENAME'));
```

The amount that must be cached for optimal performance is about 60% of this total. The cache must also accommodate other database activity on the server, if the server is not dedicated to a single chemical database. On a system that is to be devoted primarily to a chemical database, you might prefer to assign the Oracle Cartridge index tables’ cache to an alternate buffer pool, the KEEP pool, which can be sized and managed independently of AMM. When A`LTERTER INDEX … 'KEEP'` is used, some Cartridge tables are assigned to the KEEP pool and some to the `RECYCLE` pool. Therefore, space must be allocated in these pools by setting the `db_keep_cache_size` and `db_recycle_cache_size initialization` parameters. With `KEEP=YES`, the main index table is placed in the KEEP pool, which must be set to or increased by the size of this table. That size is half the value

reported by the above SQL command. With either `KEEP=YES` or `KEEP=OPT`, the inverted screen table is placed in the RECYCLE pool and the inverted-screen count and cross-reference tables are placed in the KEEP pool. For each inverted screen that is used in the index (`NORMAL=INDEX, FULLEXACT=INDEX, SKELETAL=INDEX, and/or SIMILAR=INDEX`), add about 3% more to the KEEP pool. If any inverted screens are used, the RECYCLE pool should be set to or increased by approximately 10% of the size of the main index table.

In any case, the maximum size of the cache is limited by your system memory because all buffer pools used for caching are allocated from the SGA (System Global Area), which holds Oracle buffers and runtime data structures. The SGA should fit in real (not virtual) memory in the computer, because performance would suffer severely if it were allowed to be paged out. The initialization parameter sga_max_size controls the SGA size. The maximum SGA size is typically 40-70% of real memory, depending on how much memory is required for other Oracle and system operations. The main other Oracle data structure that occupies memory is the PGA (Program Data Area), and the primary consideration is how many simultaneous sessions the server must manage.

In the PGA, the `sort_area_size` is an important parameter to check. However, the `sort_area_size` parameter is ignored if the `workarea_size_policy` parameter is set to AUTO. Check PGA parameters with the SQL command:

```
show parameter area_size
```

If the computer running Oracle has multiple processors, and it is used by only a few users, enabling parallel operations also could help. Set the `parallel_automatic_tuning parameter` to `TRUE`, and Oracle will take care of the rest.

`DB_CACHE_SIZE`

If `SGA_TARGET` is set: If the parameter is not specified, then the default is 0 (internally determined by the Oracle Database). If the parameter is specified, then the user-specified value indicates a minimum value for the memory pool.

If `SGA_TARGET` is not set, then the default is either 48 MB or 4 MB x number of CPUs, whichever is greater.

Using Oracle Enterprise Manager Console to Reconfigure the Oracle Service

To ensure that the Oracle service is permanently changed, alter the default spfile by using the Oracle Enterprise Manager Console program. The steps are as follows:

To view or modify the initialization parameters in Oracle 12c:

1. Log in as SYS, as SYSDBA by selecting “as sysdba” checkbok.
2. Select the relevant **Instance**.
3. Click on **Configuration** drop down and select **Initialization Parameters**.
4. Select `db_cache_size` and/or any other parameters and click on “Set” to change them to the desired values.
5. The scope option lets you specify when the change takes effect.

- Memory scope indicates that the change is made in memory, takes effect immediately, and persists until the database is shut down.
- SPFile scope indicates that the change is made in the server parameter file. The new setting takes effect when the database is next shut down and started up again.

6. Click the **OK** button.

To view or modify the initialization parameters in Oracle 11g:

1. At the top of the Database Home page, click Server to view the Server subpage.
2. Under Database Configuration, click Initialization Parameters.
   The Initialization Parameters page has two subpages:
   - Current—This subpage (the default) displays all initialization parameter values that are currently active (in memory) for the Oracle instance.
   - SPFile—This subpage displays initialization parameter settings in the server parameter file. This subpage is present only when the current instance started up with a server parameter file. The file location is displayed at the top of the subpage.
3. (Optional) On either subpage, reduce the number of initialization parameters displayed by doing one or both of the following, and then clicking Go:
   a. In the Name field, enter text.
   b. Select from one or more of the lists next to the Name field.

For example, to view only initialization parameters that have the text "DEST" anywhere in the parameter name, enter "dest" in the Name field, and then click Go.

4. To modify one or more initialization parameters for the currently running instance only, with the modifications being lost when the instance is restarted, complete the following steps:
   a. On the Current subpage, in the Value column, enter new values for the initialization parameters.

   Note: *If the Value column cannot be written to for a particular initialization parameter, then it means that this parameter is not dynamic—that is, it cannot be changed for the current instance.*

   b. Ensure that “Apply changes in current running instance(s) mode to SPFile” is not selected.
   c. (Optional) In the Comments column, enter text explaining the reasons for the changes.
   d. Click Apply.
   e. A confirmation message appears.
5. To modify initialization parameters for the currently running instance, and also record the modifications in the server parameter file that will persist when the database is restarted, complete the following steps:
   a. On the Current subpage, in the Value column, enter new values for the initialization parameters.
   b. Select Apply changes in current running instance(s) mode to SPFile.
   c. (Optional) In the **Comments** column, enter text explaining the reasons for the changes.
   d. Click **Apply**.
   e. A confirmation message appears.

6. To modify initialization parameters in the server parameter file only, such that the current instance is not affected and changes take effect only when the database is next restarted, complete the following steps:
   a. Click **SPFile** to view the SPFile subpage.
   b. (Optional) Reduce the number of entries in the initialization parameter list as described in Step 3.
   c. In the **Value** column, enter new values for the initialization parameters.
   d. (Optional) In the **Comments** column, enter text explaining the reasons for the changes.
   e. Click **Apply**.
   f. A confirmation message appears.

Reconfiguring the Oracle Service with a pfile

The steps below are necessary when the Oracle service is not configured with a `spfile` and relies instead on an older-style text `pfile` (parameter file) for its startup configuration. If you cannot find the `pfile` for your instance, you can generate one with Oracle Enterprise Manager Console.

1. Open your instance's `pfile` with a text editor.
   The `pfile` must be on the client machine from which you run SQL*Plus, which is not necessarily the host machine.
2. Edit an existing line if the parameter is already specified, or navigate to the very end of the text to add a new line, such as:
   ```
   db_cache_size = nnnnnnnnn
   ```
3. Save the edited text file.
4. Use SQL*Plus to stop then restart the Oracle database with the pfile saved in step 3:
   ```
   # sqlplus sys/<password> as sysdba
   SQLPLUS> shutdown
   ```

After "password" you must specify the host's instance name if your SQL*Plus client machine is different from the host machine.

Wait until the shutdown command has completed executing. The command will not finish until all user sessions are done, so the command is safe to issue at any time; but you might want to ask users to finish their sessions. Then restart the Oracle service with the following command:

```
SQLPLUS> startup pfile = filename
```

The filename parameter in step 4 must be the name of the file saved in step 3.

Platform specific details

- To start SQL*Plus in Microsoft Windows, you must open a command prompt window then issue the command sqlplus at the prompt.
- On Linux systems SQL*Plus is started with the `$ORACLE_HOME/bin/sqlplus` command. (If your environment PATH is configured to include `$ORACLE_HOME/bin`, then `sqlplus` alone will suffice.)

- A `pfile` must reside on the client computer, which is not necessarily the same machine as the server that hosts the Oracle service, so the filename must correspond to the conventions on the client where the Enterprise Manager Console and SQL*Plus are run.

Setup Problems and Error Handling

This chapter discusses how to address Oracle Cartridge configuration problems that can sometimes arise during installation, as well as how the Oracle Cartridge handles errors:

Setup Problems

When the Host machine, its Oracle instance, the Oracle client, or the files that enable communication between them are not configured properly, you might encounter problems while setting up or attempting to use the Oracle Cartridge. This section describes diagnostics you can use before installation to prevent later problems. It also tells you about problems that the installer can detect and help you to resolve, and about what to do if you encounter Oracle error messages that result from setup problems.

During an installation, log files are written in the specified "Local Directory". If installation is unsuccessful, these log files may contain information that will help you diagnose the failure:

- `Cartridge Install Trace.txt`: This file lists commands and operations performed by the installer. The last few lines shows the operation that was in progress at the time of failure.
- `CsCartridgeTrace.txt`: This file records any Oracle error messages that were generated during installation, which might show the reason for a failure.
- `expdat.log`: This file contains output from the IMP command that populates the CSCUser schema for the installation test. A failure here could indicate an improperly installed client or incompatibility between the client and server versions.

Problem in Starting the Installer

The Cartridge installer is now started by running the program SetupCartridge.exe. Only 64-bit clients are supported. If the installer does not start properly and an error message, such as the following is displayed: “This application has failed to start because OCI.dll was not found. Re-installing the application may fix this problem.” then make sure you are logged onto the client Windows machine with correct administrator privileges to perform an installation, and make sure the correct Oracle client is fully installed.

Note that Instant Client is not a suitable client; the full client or a full Oracle Database instance is required.

Checks to Perform Before Installation

tnsping

Before installation, or if you have any reason to suspect that there might be communication problems between the Oracle Client and the Host, you should issue the command:

tnsping <service-name>

from a command window on the Client machine. `<service-name>` is the name of the Host Oracle service into which you will install the PerkinElmer Oracle Cartridge.

If this command fails, you must fix your Oracle networking configuration. Diagnosing the network configuration is beyond the scope of this document. The `tnsping` report contains information about how Oracle attempted to make the connection, and that should help guide your next steps.

systemtest

The Cartridge distribution package includes a readme file and a diagnostic `SQL` script `systemtest.sql`. The readme describes in more detail how to use the script. You should be able to run the script (as sys or system user) from either the Host or from a Client SQL*Plus session that connects to the Host. The expected behavior is that the script should complete its execution successfully. If it fails with an Oracle **ORA-28575** error message, the Oracle instance is unable to run external procedures and must be reconfigured. **ORA-28575** is discussed in more detail later in this chapter.

Dependencies

Versions of the **CsCartridge** shared executable can be installed on the following operating systems. In addition to Oracle's OCI library, the Cartridge is dependent on certain system-dependent runtime libraries. The table below lists the runtime libraries it needs on each system. (extension may be `.dll`,`.so`). In addition to the OCI library, the following files are a requirement of the specified Operating Systems:

Windows
MSVCRT.DLL
MSVCP140.DLL
VCRUNTIME140.DLL
VCRUNTIME140_1.DLL

CsCartridge Runtime Library Dependencies

In addition to these file requirements, the environment must be set up so that the system can find the required files. These issues are explained in detail in the manuals for each system. For example:

Setup Help From the Installer

The Oracle Cartridge installer can help you to solve some setup problems.

After the installer has copied the Oracle Cartridge support files to the Host system and has successfully installed all of the Cartridge `PL/SQL` procedures and related objects in the Host Oracle instance, it performs diagnostics to detect two types of problems:

*Host computer is missing or has wrong versions of some runtime libraries (error message **ORA-06520**)*

*External Procedure Agent or Listener is not properly configured (error messages **ORA-28575** or **ORA-28595**).*

The above error messages are discussed in more detail later in this Chapter. When such problems are detected, the installer displays a popup box describing actions you can take to address them.

You or your system administrator can examine these files, copy them to the Host computer system, and install them as instructed. The installer does not reconfigure the system for you. That is something you must do after giving consideration to the resources it provides.

The following sections describe in more detail how the installer handles these two types of problems

Runtime library problem

The **ORA-06520** (error loading external library) message can occur when there is a system library dependency problem that can be fixed on Windows by installing a new Visual Studio runtime package, or on Linux by installing

or updating GCC to 4.4.7. This error can occur for various reasons, but it is the only error that might be resolved by updating runtime libraries. When the installer detects ORA-06520, it will offer you help in updating the runtime libraries.

On Windows, the installer will prompt you to run an MSI installer for the required Visual-Studio runtime package. On Linux, the installer will advise you of what GCC version is required.

External Procedure Agent or Listener Configuration Problem

If there is a problem, it will generally be necessary to modify one or more of the configuration files (`listener.ora`, `tnsnames.ora`, and `sqlnet.ora`) in the Host Oracle instance and restart is not necessary anymore. These files are described in more detail later in this chapter.

To enable you to test the Cartridge itself, the installer creates simplified versions of the configuration files. You should save the original files, and put these in their place on the Host. After restarting the listener, you should rerun the installer in test mode to validate the configuration.

The simplified configuration created by the installer is temporary. If it lacks services that were functional in the original configuration, you will need to merge the two configurations in a way that preserves the functionality of both and reflects the preferences of your DBA. The detailed procedure for such a merge is beyond the scope of this document.

Error Messages Indicating Configuration Problems

ORA-28575 Unable to Open RPC Connection to External Procedure Agent

The Oracle Server is not configured correctly and was unable to load the External Procedure Agent (extproc) program. This agent in turn loads the CsCartridge shared executable into memory. The extproc is Oracle's technique for allowing third-party programs to be called in `SQL` commands, and is thus the enabling technology for data cartridges.

To find the extproc program, an Oracle database uses three configuration files:

- `listener.ora`
- `tnsnames.ora`
- `sqlnet.ora`

This error is issued when the configuration files are set up incorrectly or if the extproc program is missing.

By default these files are located in the `$ORACLE_HOME/network/admin` directory, although environment and registry settings can affect the location of `tnsnames.ora`. If you make changes to `tnsnames.ora` and they have no effect, make sure you have edited the correct copy.

ORA-28595: Extproc agent: Invalid DLL Path

The CsCartridge shared executable file is missing or might have been misplaced in a manual installation. Or the path is specified incorrectly in the environment variables defined in the `ENVS` entry for the listener in listener.ora.

ORA-06520 PL/SQL: Error loading external library

The extproc agent cannot load the CsCartridge shared executable file into memory. If the file exists and is in the correct location, the problem may be that some runtime libraries on which it depends, do not exist, are of the wrong

version, do not have execute permission, or are not on the load path. It might be necessary to add or update the `LD_LIBRARY_PATH` (`LINUX`) or `PATH` (Windows) environment variable that can be set with `ENVS` in the `listener.ora` file (next section) because this environment variable controls where the extproc looks for runtime libraries.

On some systems, an ORA-06520 error can indicate that the normal Oracle `LD_LIBRARY_PATH` is not set properly when the extproc attempts to load the Cartridge. By default, within Oracle, the `LD_LIBRARY_PATH` environment variable should include at least the following directories, in addition, possibly, to some system directories such as /usr/lib:

```
LD_LIBRARY_PATH=$ORACLE_HOME/lib:$ORACLE_HOME/rdbms/lib
```

Check the value in your system's Oracle account and create or add to an ENVS entry in your extproc listener.ora path or a path that includes at least all of the same directories. When entering directories in ENVS, use the actual `$ORACLE_HOME` path and not the `$ORACLE_HOME` symbol because it is possible the symbol will not be defined in the environment when the listener evaluates the path. Below is an example of a block from a listener.ora file that defines `LD_LIBRARY_PATH` this way for the extproc:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /u01/app/oracle/product/10.2.0/db_1)
      (PROGRAM = extproc)
      (ENVS = "EXTPROC_DLLS=ANY,LD_LIBRARY_
PATH=/u01/app/oracle/product/10.2.0/db_1/lib:/u01/app/oracle/product/10.2.0/db_
1/rdbms/lib:$LD_LIBRARY_PATH")
    )
  )
```

Note that ENVS must be entered on a single line with no line-breaks, even though due to typographical constraints it appears on multiple lines above; and the line must begin with at least one space.

Please see the next section for more information on configuring the listener.ora file.

The listener.ora File

The `listener.ora` file is used whenever the listener service is restarted. On a Windows computer, it is usually started when the computer starts up, and can be stopped/started by using **Services** in the **Control Panel**.

Note: *The path to Services on Microsoft Windows varies according to which Operating System you happen to be using to Host your Oracle database server upon.*

On a `LINUX` computer, the listener service is controlled by the `$ORACLE_HOME/bin/lsnrctl <option>` command, where `<option>` is status, start, or stop to query, restart, or stop the service.

You must stop and restart the listener whenever you change `listener.ora`. Make sure `$ORACLE_HOME` is set properly for your Oracle database before you use the lsnrctl command.

For proper configuration of the `extproc`, the `listener.ora` file must include the following lines:

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
    )
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = <location of Oracle instance>)
      (PROGRAM = extproc)
    )
  )
```

The value of the KEY symbol may be different than EXTPROC, but must be the same as the value of the KEY symbol in the EXTPROC_CONNECTION_DATA entry in the tnsnames.ora file. The SID name in the listener list must be PLSExtProc.

Newer versions of Oracle automatically configure portions of the listener even without some of the entries described above. Therefore, you might not see all of the above sections in the default listener.ora file. You must add them explicitly, before you can customize them.

In some cases, it is necessary to add information about the environment in which the extproc will run. This is done by adding or modifying environment variables in an ENVS line that would be inserted following the (PROGRAM=extproc) line in the example above, such as:

(ENVS="EXTPROC_DLLS=ANY,LD_LIBRARY_PATH=/usr/ora10201/lib:$LD_LIBRARY_PATH")

In this case, two environment variables are defined. EXTPROC_DLLS specifies that any executable shared library file can be loaded by extproc. For more security, an explicit path can be placed here. The specification is not necessary if the CsCartridge shared executable is installed in its default location in the $ORACLE_HOME directory tree. LD_LIBRARY_PATH is a LINUX environment variable that adds directories to the path searched for runtime libraries. On Windows, this would be PATH, and it would be necessary to use Windows instead of LINUX path punctuation (percent, semicolon, and backslash, instead of dollar-sign, colon, and forward slash).

The tsnames.ora File

For proper configuration of extproc, the tnsnames.ora file must contain an EXTPROC_CONNECTION_DATA entry:

```
EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
```

```
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
    )
  )
```

As noted above, the value of `KEY` must be the same as that used in the `listener.ora` file, and the value of `SID` must be `PLSExtProc`.

If the `sqlnet.ora` file has an entry such as:

names.default_domain = <ourdomain>

then the domain name must be appended to the `EXTPROC_CONNECTION_DATA` entry name:

```
EXTPROC_CONNECTION_DATA.<ourdomain> =
...
```

`EXTPROC_CONNECTION_DATA` is just one of many entries in a typical `tnsnames.ora` file. Most entries define connections to Oracle services on either local or remote systems.

On an Oracle Client, there may be many, but the configuration on the Host computer (which is the one that must define the `extproc`), might include only the local service, for example:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = <hostname>)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

The name of such an entry in the Oracle client configuration, however, is the name you enter as the “Service name” when you run the installer, or when you run `tnsping`.

The sqlnet.ora File and Troubleshooting Oracle Wallet Configuration

The proper configuration of this file depends on details of your Oracle network configuration, and for the most part is beyond the scope of this document. However, there is a setting in it that can cause the Cartridge to fail.

If the parameter `SQLNET.INBOUND_CONNECT_TIMEOUT` is present with a nonzero value, Cartridge processes may be halted by Oracle before they have completed. This parameter was added in Oracle 10.2.0.3 to protect the database from denial of service attacks, but it is incompatible with connections made from external procedure agents, such as the PerkinElmer Chemistry Cartridge. When this parameter is nonzero, it causes any external procedure to terminate after the timeout. Cartridge connections must be maintained during the entire calling session

(not just for the length of one function call), an indefinite length of time, especially in the case of mid-tier servers, which cache their sessions.

Oracle advises that this parameter is incompatible with systems that require external process connections. They recommend either disabling the parameter (set it to zero) or establishing an alternate sqlnet.ora file and listener for which this setting is disabled, to use with external procedure connections.

The following issue with Oracle Wallet has a solution provided by Oracle:

SYMPTOMS

After configuring password credentials for connecting to databases stored in an Oracle Wallet using `mkstore`, `extproc` programs start to fail as follows:

```
ORA-28579: network error during callback from external procedure agent
ORA-06512: at "CSCARTRIDGE.EXTSYNCHRONIZESOME", line 1
ORA-06512: at "CSCARTRIDGE.SYNCHRONIZE_INDEX", line 18
ORA-06512: at line 1
```

CAUSE

These two features; the Oracle Wallet and the `Extproc` do not work well together and can cause the *ORA-28578* or *ORA-28579* error or a hang in the `Extproc` process.

This is because the `Extproc` process reads the server side *sqlnet.ora* file that has the `SQLNET.WALLET_OVERRIDE = TRUE` parameter.

SOLUTION

Because the `Extproc` agent is a server side functionality and the `mkstore` is typically for clients, this is not considered a bug. However, if you need to use the client side Wallet also on the server, you can use one of the following workarounds so that the `Extproc` agent no longer reads the same *sqlnet.ora* file configured in the server side.

1. When the `Extproc` agent is spawned by the database: edit the **extproc.ora**.
   Edit the file *$ORACLE_HOME/hs/admin/extproc.ora* and add the `TNS_ADMIN` variable which points to the *sqlnet.ora* without `WALLET_LOCATION`.

   For example:

   ```
   SET LD_LIBRARY_PATH=/u01/app/oracle/product/12.1.0/dbhome_1/lib:$LD_LIBRARY_PATH
   SET EXTPROC_DLLS=ANY
   SET TNS_ADMIN=/home/oracle/mytns
   ```

2. When the `Extproc` agent is spawned by the listener: use `ENVS` parameter.
   Use the ENVS parameter inside the *listener.ora* entry for your `Extproc` agent and use it to set the `TNS_ADMIN` environment variable to a different `sqlnet.ora` directory that does not have the `SQLNET.WALLET_OVERRIDE = TRUE` parameter.

In the section (`SID_DESC =` set the following:

(*ENVS="TNS_ADMIN=/<path to alternate sqlnet.ora file>*")

For example:

```
SID_LIST_LISTENER =
    (SID_LIST =
        (SID_DESC =
           (SID_NAME = PLSExtProc)
           (ORACLE_HOME = /u01/app/oracle/product/10.2.0)
           (ENVS="TNS_ADMIN=/u01/app/oracle/product/10.2.0/network/ext_env")
           (PROGRAM = extproc)
        )
     )
```

Reference to Oracle document: Ora-28578 From Extproc After Configuring Wallet (Doc ID 753530.1)

Oracle Cartridge Error Logging

From the application server on the Host computer where it operates, the Oracle Cartridge cannot communicate directly with a user. When it detects a problem, it can raise an exception or store information about the problem in a table.

If the cartridge raises an exception, then the client program will be terminated. This is the default action during normal operations other than index creation. You can change the default behavior (for any field indexed by the cartridge) with the `RAISE_ERROR` flag in the option table for that index. Use the following `SQL` statements to change the flag to prevent exceptions:

UPDATE CsCartridge.<schema-name>_<index-name>_O SET RAISE_ERROR='NO';

COMMIT;

Use this setting to ensure that a batch-mode application loading large numbers of records will run to completion.

When the Cartridge does not raise an exception, it stores a record with information about the problem in an error table associated with the index. The error table is described in more detail in Appendix B, “Data Structures." Its name is: `CsCartridge.<schema-name>_<index-name>_E`

In general, an error indicates that an indexed field contains invalid chemical data. The `rid` field of the error table stores the rowid of each record in which the cartridge encountered such a problem. The text field contains additional information.

After running a batch application, you can list erroneous records with a `SQL` query that joins the error table with the indexed table. For example, if

- `cs_notebook` is the user’s schema
- `eln_structures` is the indexed table
- `sx` is the name of the cartridge index on the structure field
- `key` is the name of a field in `eln_structures` that uniquely identifies records

then you should issue the following command in SQL*Plus to list records that were not processed because of errors:

```
SELECT eln_structures.key from eln_structures, CsCartridge.cs_notebook_sx_E
where eln_structures.rowid = CsCartridge.cs_notebook_sx_E.rid;
```

Troubleshooting Procedures

Verify the Oracle Cartridge Exists in the Oracle Database

To verify that the **Oracle Cartridge** exists in the Oracle database, perform the following:

1. Log in to the Oracle database as a DBA.
2. At the SQL command prompt, type the following:

```
SQL> SELECT username from all_users where username = 'CSCARTRIDGE'
```

If this query returns no result, then the **Oracle Cartridge** is not present in the Host Oracle database.

Verify That The Oracle Cartridge Can be Loaded

To verify that the Cartridge executable has been properly installed and the Oracle listener, extproc, and operating system are properly configured to load and run it, issue the following SQL command:

execute CsCartridge.DumpText('');

This calls a function within the Cartridge that does nothing, but if it succeeds, the Oracle extproc is able to load and transfer control to the Cartridge executable. If an Oracle error occurs, see "Error Messages Indicating Configuration Problems" on page 73.

Verify the Oracle Cartridge Version

The following statements yield a string the actual version of the **Oracle Cartridge** software.

```
select value from CsCartridge. Globals where id = 'VERSION';
```

```
select CsCartridge.VERSION FROM dual;
```

The version number stored in the `CsCartridge.Globals` table refers to the version of the install script, which installs objects into the **CsCartridge** schema. The version number returned by the `VERSION` function is the version of the `CsCartridge.(dll/so)` shared library. The two can be different. The shared library should always be as old or newer than the install script.

Identify Fields that Have Been Indexed by the Oracle Cartridge

Oracle identifies indexes created by the Oracle Cartridge as `DOMAIN` indexes with type-name `MOLECULEINDEXTYPE`. A user can query the `USER_INDEXES` view to find all such indexes on tables he owns. If you connect as a `DBA`, you can use the `DBA_INDEXES` view to list the owner schema, table name, indexed column name, index name, and index-creation parameters, for all PerkinElmer Oracle Cartridge indexes that have been created in an Oracle database, with the following `SQL` query:

```
select i.owner,i.table_name,c.column_name,i.index_name,i.parameters
  from dba_indexes i, dba_ind_columns c
  where i.index_type='DOMAIN' and i.ityp_name='MOLECULEINDEXTYPE'
  and i.owner=c.index_owner and i.index_name=c.index_name;
```

📝 ***Note:*** *That the publicly accessible table CsCartridge.all_csc_indexes (see "Data Structures") also lists all indexes created by the Oracle Cartridge. This is redundant information that should match what you retrieve from the Oracle DBA views.*

When Creating an Index Fails

When the user executes a `CREATE INDEX` command, the Oracle Cartridge reads all the rows from the table, and creates its own tables for fast substructure search.

The Oracle Cartridge creates the index in all but the most extreme situations. A `PL/SQL` exception handler is implemented to catch an abnormal termination in the processing of data in the rows of the table by the external functions.

After an exception has been encountered, the exception handler restarts indexing with extra tracing enabled, and records rows that throw errors. On the next pass, the faulty record will be bypassed by the database. Rows that fail to be indexed are recorded in the error table, as discussed in "Error Handling" chapter.

If a predefined maximum number of failures is encountered, the Oracle Cartridge assumes a serious systematic problem and stops without processing any more input records. The resulting incomplete index will have the `LOADING/FAILED` status. To continue index creation even so, the:

```
ALTER INDEX indexname REBUILD
```

command can be used to continue the processing of the records of the table from the point where it stopped. This command can be used more than once until it succeeds and the index no longer has the `LOADING/FAILED` status.

The `QUIT_INDEX_CREATE` key in the `GLOBALS` table is used to control the number of failures the Oracle Cartridge wil endure before giving up trying to create the index. The default value is 10 failures per attempt to build the index. Failures of any sort during index creation are rare, and so it would be unusual to encounter even one, much less ten.

When indexing has completed successfully or with warnings, the number of records processed by the Oracle Cartridge will match the number non-`NULL` of records in the original table. If the key field in the original table is named ID, the following SQL will print the keys of all records that were neither indexed nor listed in the index error table:

```
SELECT s.ID
FROM (MYTABLE s LEFT JOIN CsCartridge.MYSCHEMA_MYINDEX i ON s.rowid = i.rid)
     LEFT JOIN CsCartridge.MYSCHEMA_MYINDEX_E e ON s.rowid = e.rid
WHERE i.rid IS NULL AND e.rid IS NULL;
```

📝 ***Note:*** *Use the appropriate table, schema, and index names in the above query.*

If duplicates were set not to be allowed by using the `UNIQUE=YES|STEREO|RELATIVETETSTEREO` option, then duplicates may not be properly found if the `extproc` program crashed, which also can be seen in the error table.

📝 ***Note:*** *If there is a failure in the index creation when duplicates were not allowed, then the index should be dropped first, then recreated with duplicates allowed using the procedure described here, and only after all the failing records are removed should the index be recreated with duplicate filtering.*

On Windows servers a crash in the `extproc` program might not be silent. Windows might display a pop-up a dialog box asking if just-in-time debugging is planned. The index creation waits until a user answers that question. This can be controlled by deleting the Windows registry entry value Debugger.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AeDebug\Auto
```

If the Visual Studio development system is installed on the computer, then its own just-in-time debugging should be set to `OFF`. This will allow the index creation process to proceed without warning, even in the event of a crash. This can be achieved by executing the **Options** command from the **Tools** menu, and disabling the native just-in-time debugging.

Note: *PerkinElmer strongly recommends the usage of Oracle's Auto-Extend capability to extend tablespace automatically when needed. Failure to run Oracle in this mode can result in index creation failures when the tablespace reaches capacity. Customers who wish to run with Auto-Extend disabled must vigilantly monitor the percent of available tablespace (including temporary tablespace) used. If the application becomes unresponsive or appears to be hung, the Oracle error log files should be reviewed to verify that no indexes are marked as* `FAILED` *or* `LOADING`*. The tablespace will need to be extended manually if it is not marked to Auto-Extend.*

What to do When the Client Program Fails

Many different Client programs can use the features offered by the PerkinElmer Oracle Cartridge. However, they can terminate unexpectedly due to a failure in the Oracle Cartridge. In this case most likely PerkinElmer has to be contacted to investigate the problem and find a resolution. Prior to contacting PerkinElmer, users should replicate the problem after enabling Oracle Cartridge tracing. Enable tracing by issuing the following `SQL` commands:

```
UPDATE CsCartridge.Globals SET value = 'YES' WHERE id = 'TRACING';
```

and/or:

```
UPDATE CsCartridge.Globals SET value = 'YES' WHERE id = 'MONITORING';
```

and commit the changes.

If tracing is on, then the Oracle Cartridge keeps a log of all the executed `SQL` statements in a trace file. The name of the trace file is `CsCartridgeTrace.txt`, and it is located in a directory whose name is a concatenation of the schema name and the index name connected with an underscore. This directory in turn is in the system temporary directory. The system temporary directory is usually:

`C:\winnt\temp` or `C:\windows\temp`

on Windows systems, and:

`/tmp`

on `LINUX` systems. The contents of this file can greatly simplify the troubleshooting process.

If the monitoring is `ON`, then the Oracle Cartridge maintains a call stack. This file is stored in a directory called `CsCartridge_PIDS`, which is located in the `SYSTEM` temporary directory. The filename is a number that reflects the actual `PID` of the process, and the extension of the file is `.txt`. That might provide further information to allow PerkinElmer to help resolve the problem.

Both the trace file and the monitor file are readable `ASCII` text files. The contents of the monitor file can also be displayed by the **statement**:

```
SELECT CsCartridge.MONITOR from DUAL;
```

The most detailed output of a DML process can be acquired by setting the global TRACING on by using the statement:

```
UPDATE CsCartridge.Globals SET value = 'YES' WHERE id = 'TRACING';
```

This setting affects every domain index maintained by the Oracle Cartridge. If set, then the `CsCartridgeTrace.txt` file in the system temporary directory will contain a very detailed log of functions called by the Oracle Cartridge. This `ASCII` text file also may be valuable when an error is to be tracked."

If the execution of a `SQL` statement fails, or other serious problem arises the Oracle Cartridge records information into the trace file even if `TRACING` was set to `NO`.

The `TRACING` and `MONITORING` flags should be restored to the default `NO` when the experiments are finished, otherwise the Oracle Cartridge will continue to create large trace files, and there will be heavy speed penalty for writing those files.

When a Search Result is Unexpected

Sometimes the results of a structure search may be unexpected, where a query does not hit a target that it was expected to hit, or does hit a target that seems chemically unrelated. In such cases, the first course of action is to examine the query and the target structures manually. As in written text, it is easy to make “typos” in chemical structure diagrams. The search results might well be correct for the structures as they truly exist, even if those structures were not the ones that the searcher wanted to use. If the results still cannot be explained, additional “dump files” should be produced to record the exact details of the query and the target.

When the unsatisfactory search results can be reproduced by using `SQL*Plus` or a similar Client program, then the search should be repeated with the option string (the last parameter in the **MoleculeContains** operator) containing the `DUMPQUERY=YES` and if desired the `DUMPTARGET=YES` settings.

When the problem can be reproduced only by using an application that does not allow external modification of the search option string for the **MoleculeContains** operator, it is still possible to capture the desired dump files. The values of the `DUMPQUERY` and `DUMPTARGET` keys in the `GLOBALS` table should be changed to `YES`, and the experiment should be repeated. You must initiate a new Oracle session; otherwise, the old values will still be in use. Restarting the application will ensure this.

If dump is enabled with any of the methods described here, a file with the name `CsCartridgeDump.txt` will be created in the system temporary directory, which is the same location as for the `CsCartridgeTrace.txt` file described in the previous section.

When the dump file is ready, then it must be sent to PerkinElmer for analysis. When the troubleshooting is finished, the values of the `DUMPQUERY` and `DUMPTARGET` keys should be reset to NO to avoid performance degradation cause by the maintenance of the dump files.

Reinstall Oracle Cartridge and Recreate Indexes

If you have a serious problem with the Oracle Cartridge that cannot be resolved by any other means, consider reinstalling it and recreating the indexes.

You can reinstall the Oracle Cartridge to replace an existing installation by rerunning the installation program using the same settings as before, but in addition check the "Recreate existing indexes" box. The installation program will record the parameters for all existing indexes before dropping them at the start of installation, and then will create a script to recreate the indexes at the end. You can either allow the installation program to run the script, or you can run it manually later.

If you performed a manual installation, you may still have the original CsCartridge.sql script from that process. Instead of performing a complete reinstallation, you can rerun that script. If necessary, use the "Manual install" option again in the installation program to regenerate the `CsCartridge.sql` script.

If you use the CsCartridge.sql script, you must also manually drop and recreate all Cartridge indexes. See the section "Identify fields that have been indexed by the Oracle Cartridge" earlier in this chapter to list those indexes.

Note: *The index recreation can be a time-consuming process unless the clause* `parameters('skip_populating=yes')` *is used.*

No actual index data are stored if you skip populating the index, but work that uses the data tables can resume immediately (`skip_populating` is discussed in more detail in "Index Creation and Maintenance" chapter). To repopulate an index, update each record in the table with the `SQL` statements:

```
UPDATE <table> SET <field>=<field>;

COMMIT;
```

Commit after every record if you are concerned that some data may cause the Oracle Cartridge to fail.

If this does not resolve the problem, please contact PerkinElmer.

Identifying Cartridge Sessions

At times, you may need to identify Oracle sessions in which the Cartridge is currently active. For example, if the Cartridge is to be updated, you must identify Oracle sessions to ensure that no one else is using it (which would block the update). Also, when the parameters of a Cartridge domain index are changed with an `ALTER INDEX` command, the change affects only the current session and subsequent sessions, but it does not affect other concurrent sessions at the time the change is made.

While connected as the `SYSTEM` or other DBA user, the following SQL command lists all current sessions, so that you can identify those accessing the `CSCARTRIDGE` schema name and end or restart them:

```
select sid,username,schemaname,lockwait,status from v$session;
```

If it is not clear where a session that uses the Cartridge originated, the following SQL command will also show the application controlling each session:

```
SET LINESIZE 100
COLUMN spid FORMAT A10
COLUMN username FORMAT A10
COLUMN program FORMAT A45
SELECT s.inst_id, s.sid, s.serial#, p.spid, s.username, s.program
FROM   gv$session s
       JOIN gv$process p ON p.addr = s.paddr AND p.inst_id = s.inst_id
```

```
WHERE  s.type != 'BACKGROUND';
```

If you are unable to end the program or session in an orderly fashion but must do so, as DBA, kill it in order to complete Cartridge maintenance. The above command also lists the SID and serial number that you would use in an ALTER SYSTEM KILL SESSION command, or the PID for an operating-system command to kill the program. Remember that killing sessions or processes must be done with great care; it could affect the stability of your system and could result in data loss.

Profiling Cartridge Execution

When using the Cartridge in a complex application, if performance is not satisfactory it can be difficult to determine the source of the problem. Reports from Oracle's tkprof profiling tool can help to diagnose such problems. The reports that tkprof creates from trace files will show which SQL commands run slowly and will often help to determine why, and whether the problem SQL was issued by the Cartridge or by another part of the application.

Refer to Oracle documentation for details about how to use tkprof. Note that the session in which the Cartridge-calling application runs will require special Oracle privileges in order to collect the necessary information in a trace file. You must work with your Oracle DBA to obtain the necessary privileges. If the application runs in schema <user>, the following are the essential SQL grants on SYS objects:

```
connect sys/<password> as sysdba
grant alter session to <user>;
grant select on v_$process to <user>;
grant select on v_$session to <user>;
grant select on v_$parameter to <user>;
```

To start and stop trace file data collection, you must bracket the portion of the application being profiled with the following SQL commands:

```
alter session set events '10046 trace name context forever, level 12';
...
alter session set events '10046 trace name context off';
```

The location of the trace file on the server is system and Oracle-version dependent. Consult Oracle documentation and your DBA. You will probably need special rights to read the trace file from the Oracle directory where it is generated, or your DBA may arrange for sending a copy to you. In Oracle 11g and higher, the following command lists the trace file for the current session:

```
select value from v$diag_info where name = 'Default Trace File';
```

Oracle Cartridge Advanced Techniques

Using Microsoft ADO

By using Microsoft's ADO you can conveniently write Client programs that utilize the Cartridge.

To create an Oracle Service on any Client computer, the Oracle Client tools must be installed, and the Net Manager program must be used to create a service name by entering the following information:

- **Net Service Name:** any valid string would do (this string must be used in ADO)
- **Protocol:** TCP/IP
- **Host Name:** a computer running the Oracle database
- **Port Number:** 1521
- **Service Name:** the Oracle service name on the host computer.

```
DIM c as ADODB.Connection
set c = new ADODB.Connection
Serviceman = "" Rem the name of Oracle service
UserName = "" Rem user name
Password = "" Rem password
s = "Provider=OraOLEDB.Oracle.1;Persist Security Info=False;" + _
"Data Source=" + ServiceName +
s = s + "User ID=" + UserName + ";"
s = s + "Password=" + Password
c.Open s
Dim r as ADODB.Recordset
Set r = c.Execute("SELECT * FROM meltable WHERE_
CsCartridge.MoleculeContains(mol, 'c1ccccc1','', '')=1");
Rem process the recordset
c.Close
```

When using Microsoft ADO data controls to access the Cartridge, the **CursorLocation** property must be set to **adUseServer**. Otherwise data updates will fail if the structure field is larger than 4000 bytes.

Example of Passing Large Queries

```
Dim r As ADODB.Recordset
ChemDrawCtl1.DataEncoded = True ' b64 encoded cdx is required
Dim s As String
s = ChemDrawCtl1.Data("cdx") ' get the cdx document from the ChemDraw ActiveX
control
Dim com As ADODB.Command ' create a new ADODB.command
```

```
Set com = New ADODB.Command
Rem now build the sql string step by step, and create the corresponding
parameters as well
Rem the query data will be passed as an array defined in CsCartridge
Rem
sql = "Select " + idSelect ' idSelect contains the column name to be selected
sql = sql + " from " + TableName ' from table
sql = sql + " where CsCartridge.MoleculeContains(" + StructName ' first param is
column name of the chemical
structure
sql = sql + ", CsCartridge.Xarray(" ' second param is an array of string literals
n = Len(s) ' number of characters in the query structure
pos = 1 ' starting position
i = 0 ' starting index
segmentLength = 4000 ' the length of a segment of the query string
While pos < n ' there is remaining part of a query string to be passed
si = Mid(s, pos, segmentLength) ' get one 4000 character chunk of the query
Dim v As Variant
v = si ' create parameter expects the value as variant
parname = ":xpar" + CStr(i) ' parameter name as a stem and running index
Dim par As ADODB.Parameter ' create the parameter
Set par = com.CreateParameter(parname, adLongVarChar, adParamInput, Len(si), v)
com.Parameters.Append par ' append it to the parameters accompanying the command
sql = sql + parname ' the sql command will have a new parameter
pos = pos + segmentLength ' move the position
If pos < n Then ' if there is data still to be processed
sql = sql + "," ' add a comma to separate the parameters
End If
i = i + 1
Wend
sql = sql + "),'FULL=NO')=1" ' finish up the sql text
com.CommandType = adCmdText
com.ActiveConnection = c
com.CommandText = sql ' set the CommandText field of the command object
Set r = com.Execute '
And now the command is ready to be executed.
```

Optimizing Client Programs

Client programs usually use ADO to retrieve data from the database. In the case of presentation-tier programs, they present the results to the user; or in the case of middle-tier programs, the result set is passed further to the presentation layer. A typical program looks like the skeleton program below:

```
dim rs as recordset
set rs = adoConnection.Execute "SELECT ..."
while not rs.eof do
rem deposit result from the result set into a placeholder for the presentation
layer
rs.MoveNext
wend
rem Do the presentation.....
```

This is the simplest strategy, but a better user experience can be achieved with somewhat more sophisticated programming. In the Cartridge, the execution of the `adoConnection.Execute` statement is fast, and most of the calculations are postponed until the `MoveNext` statements are executed.

With the programming schema above the user cannot communicate with the program until the whole record set is deposited into the presentation layer.

A much better user experience can be created if the program executes only enough MoveNext statements to retrieve the records that will be presented in one page and data to be presented on the next page is retrieved only when the user hits the **Page Down** button, or clicks on the scroll bar. The result is presented much faster than with the simple scheme above. An even better user experience can be achieved if the data retrieval and the presentation is executed in a different thread. Then the user can manipulate the part of the record set already transmitted to the client computer, and the data processing still goes on in a background thread.

An improved program would look like this:

```
dim rs as recordset
set rs = adoConnection.Execute "SELECT ..."
if not rs.eof
Rem display the data in the recordset and store it into a buffer
end if
docontinue = true
while docontinue
rem wait for user action requesting next or previous record
or finishing process
if pagedown then
rem if record is already in the buffer, then display,
otherwise rs.MoveNext first
end if
if pageup then
```

```
rem show previous record already in the buffer
end if
if home then
rem show the first record
end if

if end then
rem display a message, that this will be slow
while not rs.eof do
rem store data into buffer
rs.MoveNext
wend
rem show the last record
end if

if cancel then
docontinue = false
end if
wend
```

Custom Screening

The Cartridge uses a set of predefined screens well suited to general chemical databases. If a database contains many similar structures, however, one can benefit by adding customized screens that will help to distinguish among them.

No more than 32 custom screens can be added to an index.

Note: *There is no recommendation on how to choose custom screens. Choosing them should be based on the particulars of the database and expected interest of the users.*

Defining Custom Screening

Use the `CUSTOM_SCREENS` index-creation parameter to include custom screening in an index. The custom screening must be defined before an index is created by adding a row for each screen to the `CUSTOM_SCREENS` table. If such a table exists in the schema of the table being indexed, it will be used for that index; otherwise the `CsCartridge.CUSTOM_SCREENS` table will be used.

See the description of this table in Appendix B.

In order to change the custom screens, you must recreate the index after updating the `CUSTOM_SCREENS` table.

The Files Package

The files package is a collection of routines to exchange data between an Oracle table and a flat file. If the filename does not contain a computer specification, it is created on or expected to exist on the Server computer, where the

Oracle service runs.

To read a file from the Client computer, the Client computer specification also must be part of the file name. These routines are intended for debugging and/or administration of the Oracle Cartridge. Their usage for other purposes is not recommended.

The routines of the files package are:

```
FUNCTION TempDirectoryName RETURN VARCHAR2;
```

This function returns the name of a directory to store temporary files. The directory name is terminated by either a backslash (\) on Windows systems or by a slash (/) character on `LINUX` systems. To access or create a file in that directory, concatenate a filename to the name of the temporary directory.

FUNCTION ReadBFile(fname VARCHAR2) RETURN BLOB;

This function reads the whole contents of the file specified by the *fname* parameter. The file is opened as a binary file, and a temporary lob is returned with the contents of the file.

```
FUNCTION ReadCFile(fname VARCHAR2) RETURN CLOB;
```

This function reads the whole contents of the file specified by the *fname* parameter. The file is opened as an ASCII file, and a temporary CLOB is returned with the contents of the file.

```
PROCEDURE WriteFile(fname VARCHAR2, data CLOB);
```

This procedure allows the creation of a file with the contents of the CLOB specified by the *data* parameter.

```
PROCEDURE WriteFile(fname VARCHAR2, data BLOB);
```

This procedure is the same as the previous one, allowing the saving of a `BLOB` into a binary file.

Examples:

```
CREATE TABLE t as
SELECT CsCartridge.Files.ReadCFile
(CsCartridge.Files.TempDirectoryName ||
'CsCartridgePlan.txt') as plan from dual;
```

The statement above will create a table with the name `t`, having one `CLOB` column named `plan` and will insert one record into that table with the contents of the `CsCartridgePlan.txt` from the user's temporary directory.

```
DECLARE
b BLOB;
BEGIN
EXECUTE IMMEDIATE 'select m from acx50k where id = 1' INTO b;
CsCartridge.Files.WriteFile
(CsCartridge.Files.TempDirectoryName ||
'docid1.cdx', b);
END
```

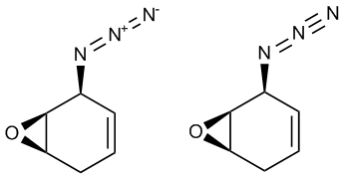
The script above reads a record from the table `acx50k` and places the results into the variable b. Then it creates the `docid1` file in the user's temp directory and writes the contents of the `BLOB` variable to that file.

Appendix A: Chemical Searching

Chemicals exist in real life. You can have a flask containing sodium azide, and either it contains that substance or it doesn't. There isn't a notion of "this form of sodium azide" or "that style of sodium azide." But when representing structures graphically, it is possible to draw the same physical substance in different ways.

Structure Normalization

Structure normalization is the process of examining various drawing styles and determining that they do, in fact, refer to the same physical substance. The goal of normalization is to ensure that different representations of molecules, provided they are drawn so that a chemist would deduce the same physical entity, are detected as being identical by the software. For example, the structures in the following figure show different conventions for drawing azides:



The PerkinElmer Oracle Cartridge perceives these structures as identical because they represent the same chemical substance. In contrast, chemical search engines from other software vendors typically do not perform normalization. Instead, those other packages require that structures in chemical databases conform to standard rules for structure drawing, and the chemist must also conform to those rules when drawing chemical search queries. Chemical structures that do not conform to these rules (for example, structures drawn and stored an electronic laboratory notebook) must be converted to the correct convention when stored in the database. None of this is necessary for the PerkinElmer Oracle Cartridge. Chemists are free to use different conventions for chemical structures that they store in electronic notebooks or use as search queries.

Note: *Although it is not necessary to normalize structure drawings to standard rules, PerkinElmer recognizes that some customers might want to do so and provides the* ChemScript *package for this purpose.* ChemScript *is an open environment that allows the enforcement of drawing conventions as submitted by the user.*

Tautomerism

Tautomerism is a phenomenon by which two or more structurally distinct molecules interconvert rapidly. Most commonly, tautomeric structures feature a mobile group, usually hydrogen, that moves from one location within the structure to another. Because so many compounds exhibit tautomerism, proper consideration of tautomerism is critical for proper inventory management. Individual tautomers are distinct structures that can be isolated individually under the proper conditions. For example, it is impossible (under normal conditions) to have two distinct flasks, one containing pyridin-2(1*H*)-one and one containing pyridin-2-ol, since the latter will rapidly convert into the former. On the other hand, someone doing low-temperature work (or varying pH), for example, may really be able to work with both tautomeric forms. Consequently, sometimes it is reasonable to treat tautomers as the same substance, and sometimes not.

PerkinElmer addresses two main issues in the consideration of tautomerism:

Identifying substances that exhibit tautomerism

Selecting the preferred tautomeric form

Identifying tautomers

The Oracle Cartridge identifies as tautomers structural fragments of the form HX-(Y=N)$_n$-Y=Z, where:

X and Z are any of C, N, O, S, Se, Te

X and Z are not both C

Y is any of B, C, Si, N, P, As, Sb, S, Se, Te, Cl, Br, I

H is any of H, D, T

n may be zero or any positive integer

It is worth emphasizing that if X and Z are both C, the structure is not tautomeric. If it were, then any double bond would freely migrate within any alkene, which is clearly not observed experimentally. The specific tautomerism HO-CR=CR$_2$ <> O=CR-CHR$_2$ is known as a *keto-enol tautomerism*. The specific tautomerism H$_2$N-CR=CR$_2$ <=> HN=CR-CHR$_2$ is known as *imine-enamine tautomerism*. The latter form of tautomerism is also responsible for hydrogen migration in some five-member aromatic systems containing nitrogen.

Note: *Because tautomerism can be defined as the migration of a group from one location to another, almost any rearrangement could be considered a type of tautomerism. In particular, [1,3]- and [1,5]-sigmatropic migrations of hydrogen and [3,3]-sigmatropic rearrangements (Cope rearrangements) could be considered as tautomeric forms. With rare exceptions, however, these rearrangements do not proceed as rapidly under standard conditions as do the other types of tautomerism discussed above. Accordingly, these cases are not considered tautomeric.*

Selecting the preferred tautomeric form

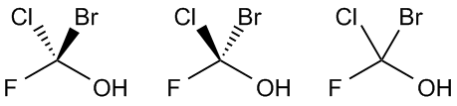
Selection of a preferred tautomeric form is fundamentally a type of normalization. As discussed above, the PerkinElmer Oracle Cartridge is designed to behave correctly without requiring changes to the structures drawn. Unlike in other software, there is no need to select a preferred tautomeric form when using the PerkinElmer Oracle Cartridge.

Stereochemistry

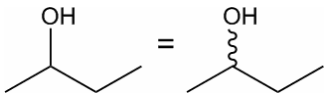
This section describes the types of stereochemistry that the Oracle Cartridge recognizes.

Tetrahedral stereochemistry at asymmetric carbon atoms

For example:

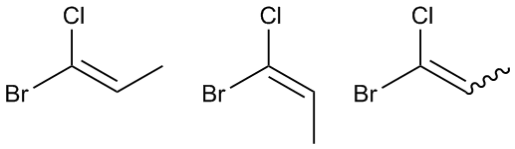


A wavy bond and a single bond have exactly the same meaning. Thus, the structures below are equivalent and behave the same in searching:



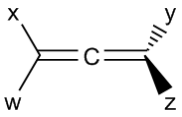
E/Z geometric stereochemistry

For example:



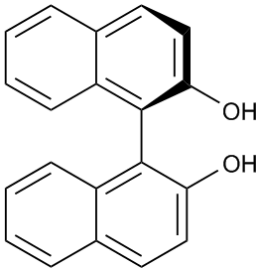
Hindered rotation in allenes

For example:



Hindered rotation in biphenyls

For example:



Appendix B: Data Structures

Schema overview

Whenever a new record is stored in a table, is deleted from a table, or an indexed field is modified, by the Oracle Cartridge, Oracle makes a call to the Cartridge passing the rowid of the affected record and the value of the indexed field. The Cartridge processes this data, and stores the results in tables in the CsCartridge schema to improve search speed.

The major steps of this process are as follows:

- Attempt to interpret the passed binary
- Extract the chemically meaningful part of the data into a private data structure
- Create screens used in substructure search for the structure
- Store the interpreted chemical structure information

All the information the Oracle Cartridge maintains is stored in the CsCartridge schema. For every index created by the Cartridge, eight (8) tables are created in that schema. These table names are generated from the name of the schema the indexed table resides in, and from the name of the index. Table names are maximized at 20 characters, so the Cartridge requires that the name of the schema and the index together cannot be longer than 16 characters.

Tables related to an index

The following tables are created:

📝 ***Note:*** *In the following list of tables,* ***sch*** *represents the schema name, and* ***ix*** *represents the index name.*

sch_ix – Stores most data, most frequently used for substructure search.

sch_ix_XY – Where X refers to the screen type, and Y to the kind of data, stores screening data.

sch_ix_E – Stores error messages for a longer process.

sch_ix_O – Stores options set by the end user.

sch_ix_T3 – Stores temporary 3d screening data until synchronization.

The sch_ix Table

The following fields are part of the sch_ix table:

RID – Rowid of the indexed record in the original table. This is returned to Oracle as a result of a search.

MOLSTGLENGTH – Size of the stored chemical structure information.

MOLSTG1 – Stored chemical structure if it is shorter than 2,001 bytes, otherwise empty.

MOLSTG – Stored chemical structure if it is longer than 2,000 bytes, otherwise empty.

CTEXT – Canonical code of the chemical structure.

FORMULA – Chemical formula of the structure. If a formula is longer than 200 characters, a NULL is stored, and the formula must be computed from the original structure with ConvertCDX.CDXToFormula(). Also, for nonlinear reactions, a gross formula instead of a reaction formula is stored.

MOLWEIGHT – Molecular weight of the structure.

NC – Number of carbon atoms in the structure.

NN – Number of nitrogen atoms in the structure.

NO – Number of oxygen atoms in the structure.

NHAL – Number of halogen atoms in the structure.

NOTH – Number of other atoms of elements in the structure.

N_LARGE_ALIPH_BLOCK – Number of carbon atoms in the largest aliphatic block.

N_LONG_ALIPH_CHAIN – Number of carbon atoms in the longest aliphatic chain.

N_QUATERN_C – Number of quaternary carbon atoms.

ISNEW – Either `Y` for records not synchronized yet, or `NULL` for synchronized records.

SN0, SN1,... – Normal screens stored in 10-digit numbers. The count of them depends on the screenwidth.

SF0, SF1,... – Fullexact tautomer screens stored in 10-digit numbers. The count of them depends on the screenwidth.

SK0, SK1,... – Skeletal tautomer screens stored in 10-digit numbers. The count of them depends on the screenwidth.

MOLSTG_REACTANTS – Stored chemical structure of reactants if the original document is a reaction.

SCREENS_REACTANTS – Screens of reactants if the original document is a reaction.

MOLSTG_PRODUCTS – Stored chemical structure of products if the original document is a reaction.

SCREENS_PRODUCTS – Screens of products if the original document is a reaction

The **RID** field stores the `rowid` of the record indexed by the Cartridge. The value for `RID` is passed to the Cartridge by Oracle. The stored chemical field contains all the chemically valuable information from the `CDX` document stored in a row referred to by the `RID` field. This field is used when the Cartridge has to do atom by atom matching during substructure search. The data in this field is stored in a highly compressed format for better retrieval speed. The `CTEXT` field is the canonical code of the chemical structure, and used for identity search. The Identity search is much faster than full structure search. The `FORMULA`, `NC`, `NN`, `NO`, `NHAL` and `NOTH` fields are used for formula search. The `MOLWEIGHT` field is used for molecular weight searches. Any one of the `SCREENS_NORMAL`, `SCREENS_SIMILAR`, `SCREENS_FULLEXACT and SCREENS_SKELETAL` columns may be missing if during index creation the particular feature was not requested.

When a program executes a `SQL` statement referring to the `MoleculeContains` operator, the Cartridge generates its own screening statement retrieving the molstg information from the *sch_ix* table. The subset of records is selected by using the screen bits of the query and one of the screens from the *sch_ix* table.

The *sch_ix*_NR or other inverted screen table also may be used if present to increase speed of the retrieval of screening data. The generated `SQL` statements can be found in the CsCartridgeHistory.txt file located in the system temporary directory.

The sch_ix_NR table

That table is the inverted screen table for regular screens. It consists of the following columns:

BIT – Screen bit number

RID – Rowid of the indexed record

This table is partitioned by the bit column. It has 256 partitions related to each of the screen bits generated by the screening algorithms.

The sch_ix_NC table

This table contains the cardinalities of the particular screen bits. It consists of the columns as follows:

BIT – Screen bit number

COUNTS – Cardinality of the screen bit

This table has 256 rows.

The sch_ix_NX table

Screen cross reference table for regular screens. It has the following columns:

BIT1 – Screen bit number

BIT2 – Screen bit number

COUNTS – Cardinality of the screen bit

This table has 64K rows for each possible combination of `BIT1` and `BIT2` values.

The sch_ix_FR, sch_ix_FC, sch_ix_FX tables

Same contents as the tables above for screens for tautomer full exact search.

The sch_ix_KR, sch_ix_KC, sch_ix_KX tables

Same contents as the tables above for screens for tautomer substructure search.

The sch_ix_E table

The following fields are part of the *sch_ix*_E table:

RID – Rowid of the indexed record

TEXT – Explanation of the error

This table is supposed to be empty. If an error is detected by the Cartridge it usually throws an exception. Depending on the content of the *sch_ix*_O table, the Cartridge might not throw an exception, but instead continues operation and stores the reason of the error into the **TEXT** field.

The sch_ix_SR table

This table contains the screen bits and their counts organized in a way that permits optimization of similarity searches.

The sch_ix_3 table

The following fields are part of the *sch_ix*_3 table:

ATOMPAIR – Is a string of two or three element symbols

RIDSVAL – Nested table containing the rowids and distance or angle information of the indexed records.

This table contains information used for fast 3D screening. The **ATOMPAIR** field contains two element symbols in a string for distances, and three element symbols for angles. The **RIDSVAL** nested table contains all the rowids and values for the particular **ATOMPAIRs**. For a two element **ATOMPAIR** the values are the distances in Angstroms, and for three element **ATOMPAIRs**, the angle value in radians is stored in the value field.

The sch_ix_O table

The following fields are part of the *sch_ix*_O table:

ID – This field is a string containing one of a few predefined keywords

VALUE – This field contains the corresponding value. In most cases the accepted values are **YES** and **NO**.

The list of accepted keywords follows:

RAISE_ERROR – To control if the Cartridge should raise an error when encountering an illegal input, or store the information in the _E table

CACHE_RESULTS – Controls whether the Cartridge saves results of substructure searches. If the same query is repeated, then the Cartridge can return the existing hit list much faster. This cache is emptied fully if the table changes, so it is mostly recommended for read-only databases.

NORMAL – To direct the Cartridge whether maintain normal screens

SIMILAR – To direct the Cartridge whether to maintain screens for similarity search.

FULLEXACT – To direct the Cartridge whether to maintain screens for full structure search considering tautomers.

SKELETAL – To direct the Cartridge to maintain screens for substructure search considering tautomers.

THREE_D – To direct the Cartridge to maintain screens for 3d screening.

For the `NORMAL, SIMILAR, FULLEXACT, SKELETAL` keywords the acceptable values are `NO`, `YES` and `INDEX`. In the case of `NO` no screens of that particular kind are stored at all. In the case of yes a bitmap field is stored for the screen, and in the case of INDEX an inverted table is maintained for that kind of screen.

SYNCHRONIZATION – Controls if the index is synchronized all the time or if it needs occasional manual synchronization.

THREE_D_RESOLUTION – Can be `LOW` or `HIGH`. If it is low, then less data is stored.

REACTION_SCREENING[1] – Controls if the Cartridge keeps screening information for the starting reactants and the final products of a reaction separately from the main screen. It can be useful in a reaction database, when queries are regularly restricted to reactants and/or products.

DETACHED – Used during transportable tablespace creation.

UNIQUE – See CREATE INDEX parameter.

SALTSPLITTING – See CREATE INDEX parameter.

NUMBER_OF_ROWS – Size of index.

COMPRESSED – Obsolete.

COUNTS – See CREATE INDEX parameter.

SCREENWIDTH –Number of screens used by the Cartridge; cannot be changed by user.

DEFERRED – Deferred maintenance mode.

LOGGING – PerkinElmer diagnostic use.

USEPARTITIONS – See CREATE INDEX parameter.

AUTOSYNC – Specify NO to exclude from automatic synchronization job.

COMPATIBILITY_VERSION – Index can be transported only to a Cartridge with the same internal code version.

There is one record in the *sch_ix*_O table for every keyword. It is not recommended to edit this table any way manually. The contents of the table should be changed using the `ALTER INDEX` command.

Administrative tables

ALL_CSC_INDEXES table

The **ALL_CSC_INDEXES** table lists all the indexes created by the Cartridge.

OWNER – Name of the owner's schema

INDEX_NAME – Name of the index

TABLE_NAME – Name of the table

---

[1]This parameter is not used by Cartridge 19.1. However, it was used in earlier versions of the Cartridge and it may be re-enabled in a future version of the Cartridge.

COLUMN_NAME – Name of the indexed column

COLUMN_TYPE – `CLOB` or `BLOB` type files

OPTIONS - Parameters with which index was created, updated when index is altered

CUSTOM_SCREENS table

The **CUSTOM_SCREENS** table contains the list of custom screens that can be applied when an index is created. After installation, this table contains the `SMILES` string for benzene and cyclohexane. The administrators can add new items to this list, or remove the ones there.

SMILES –The smiles string of the custom screen. An encoded CDX string is also permitted, but it must fi within the 2000-character width of this field.

DESCRIPTION – Optional documentation for the custom screen. The `DESCRIPTION` field is not used by the Cartridge; it is provided to help administrators.

GLOBALS table

The **GLOBALS** table contains information globally applicable to the Cartridge.

Currently the following identifiers are stored in the **GLOBALS** table:

VERSION – Current version number

LIBRARY_DIRECTORY – The directory the CsCartridge shared library is stored

USE_THREADS YES or **NO** (default **YES**) – The Cartridge generates threads in an attempt to improve the speed of substructure search. On some platforms that resulted in worse performance. The Administrator can control the behavior of the Cartridge by the **USE_THREADS** parameter.

MAX_THREADS – Maximum number of threads to permit if USE_THREADS=YES, limited by processor count.

STORE_INPUT_DATA YES or **NO** (default **NO**) – If the **STORE_INPUT_DATA** parameter is set to `YES`, then the Cartridge stores the data being processed into a file. The file's name is `CsCartridgeData.txt`, and it is stored in the system temporary directory along with other diagnostic files, as described in Troubleshooting Procedures chapter.

TRACING YES or **NO** or string (default **NO**) – If the **TRACING** parameter is set to `YES`, then the Cartridge keeps a log in the `CsCartridgeTrace.txt` file in the system temporary directory, as described in Troubleshooting Procedures chapter. These files can be used for troubleshooting. The value of this parameter must be changed directly by a statement:

UPDATE CsCartridge.Globals SET value='YES' WHERE id='TRACING'

The value of the string depends on the version and build number of the Cartridge. The string is a concatenation of different trace selector. The tracing settings select certain features for tracing. The most commonly used is likely the 'Statement:Execute' trace selector selecting trace for every `SQL` statement executed.

📝 ***Note:** If **TRACING=YES** is set, the program execution slows down dramatically.*

MONITORING YES or **NO** (default **NO**) – Once monitoring is set, then the program execution can be monitored by the **MonitorCartridge.exe** program, which is a stand-alone program running on Windows platforms, which shows the "call stack" of execution.

QUIT_INDEX_CREATE number (default 10) – The number of attempts the Cartridge makes when creating an index and a failure is detected. Normally a Cartridge index creation ends in an error message and no data structures stored if there were more than *number* failures. A "failure" may happen if a document can not interpreted any way at all. The Cartridge catches and registers those exceptions, and continues building the Cartridge tables. Those exceptional documents are rare, explaining why the small default number. This default number is also a protection against an error if an index is to be built on a column that does not contain any meaningful chemical structures.

SYNCHRONIZE_BUFFER number (default 200) – This number instructs the Cartridge about the number of rows processed in one step during synchronization. This number has a profound effect on the speed of synchronization. Though it cannot be increased without concerns to Oracle's inherent limitations, which can vary from installation to installation. With 1,000 records processed in one step, the Cartridge uses about 45MB of memory. With 200 records it uses only 11 MB.

QUERY_LOGGING YES or **NO** (default **NO**) – If **QUERY_LOGGING** is set to `YES` then the Cartridge will create and write files in the `Windows\Temp` (`Windows`) directory or the /tmp (`LINUX`) directory related to executed substructure searches. Those are the `CsCartridgeHistory.txt` and `CsCartridgeStatistics.txt` and the `CsCartridgePlan.txt` files, and the file containing the hitlists of substructure searches. The name of those files is the canonical code of the query structure.

DUMPQUERY – See usage in "When a search result is unexpected" ("Troubleshooting Procedures").

DUMPTARGET – See usage in "When a search result is unexpected" ("Troubleshooting Procedures").

ERROR_LOGGING – PerkinElmer diagnostic use.

SYNCJOB_START – Automatic synchronization job ID, or -1 if none; see usage in "Recommendations for Index Maintenance"chapter.

USE_PARTITIONS (NO or, by default, not present) – This is set to NO if the "Partition inverted screen tables when possible" check box in the main dialog was unchecked during Cartridge installation.

ADDAROMATICPREDICATE YES or **NO** (default **NO**) – When `ADDAROMATICPREDICATE=YES`, the substructure queries will be modified to have extra predicates added to single and double bonds as described in the Chemical Queries chapter.

Data stored outside of Oracle

The Oracle Cartridge stores some auxiliary information outside of the Oracle database on the Host computer. It creates files in the `/tmp` directory on `LINUX` systems, and in the directory the `GetTempDir()` Windows system call returns on Windows machines.

CsCartridgeData file – This file contains the original data being processed while storing, updating, or deleting a record from a table. In case of unexpected failure, it might be useful to open the original document to help troubleshoot the problem.

CsCartridgeRowid.txt file – This file consists of one line, and that is the `rowid` of the last processed record from a table to be indexed, or be modified.

Views to Check Data Consistency and Performance

These views are created when an index is created. They can be used to check the integrity of an index, and in a case of failure, they also can be used to repair a damaged index. The views are created in the CsCartridge schema. The name of those views is a concatenation of the schema name where the index is created and the name of the index itself, connected by an underscore character, and one or two more letters.

When a statement is executed, there should be no rows retrieved by the views that check for inconsistency. If there are rows returned, then they contain rowids of records which should not exist. To repair those indices the most likely action is to delete the rows that are identified by the listed rowids:

sch_ix_V1

This view retrieves the rowids of the records in the **sch_ix** table which point to nonexistent records in the original table.

sch_ix_V2

This view retrieves the rowids of records in the user's indexed table, which are not `NULL` and are not empty, but still have no match in the CsCartridge schema. These are probably records that have some serious defect.

sch_ix_VR

Retrieves the percentage of synchronized records. Depending on the size of the database, synchronization should be run when about 50,000 to 100,000 non-synchronized records are in a table.

sch_ix_VN, sch_ix_VS, sch_ix_VF, sch_ix_VK

These views check the inverted screen tables for normal, tautomer full exact, and tautomer skeletal indexes. They list records which point to non-existent records in the *sch_ix* table.

sch_ix_VNC, sch_ix_VSC, sch_ix_VFC, sch_ix_VKC

These views report the percentage of the screen bits set, which is a rough measurement of the selectivity of the screens. The closer the number is to 50 percent, the better. If this number is too high, that means that the chosen property set to screen the molecules is not selective; most records have the same property. If it is too low, then the screens are too selective; there is a chance that the queries will not have any useful screen bits.

*sch_ix*_VNX, *sch_ix*_VSX, *sch_ix*_VFX, *sch_ix*_VKX

The retrieved percentage is informative regading how much ortogonality can be expected from the screen bits. The lower the number is the better.

Oracle Cartridge Program Files

The Oracle Cartridge consists of the following files:

- `CsCartridge.dll` (Windows)
- `CsCartridge.so` (all Linux systems)
- `CSChemFinderCustomElements.txt`
- `NameToStructure.dat`

These files are located in the `$ORACLE_HOME\bin\instance_name` directory on Windows computers and in the `$ORACLE_HOME/lib/instance_name` for `LINUX` computers, where `instance_name` is `CsCartridge`, or the name entered in the instance name field on the installer dialog.