CX$_{Oracle}$

Release 6.0

August 15, 2017

Contents

cx_Oracle is a module that enables access to Oracle Database and conforms to the Python database API specification. This module is currently tested against Oracle Client 11.2, 12.1 and 12.2 and Python 2.7, 3.4, 3.5 and 3.6.

cx_Oracle is distributed under an open-source *license* (the BSD license).

Contents:

CHAPTER 1

cx_Oracle 6 Installation

Overview

Before cx_Oracle can be installed, an installation of Python is needed first. Python 2.7 and Python 3.4 and higher are supported.

The simplest method of installing cx_Oracle is to *Install Using Pip*. You can also *Install Using GitHub*. If you run into trouble, check out the section on *Troubleshooting*. cx_Oracle uses ODPI-C, which means that the ODPI-C installation instructions can be useful to review.

After cx_Oracle has been installed, you must also *Install Oracle Client*, if that has not been done already. Oracle Client versions 12.2, 12.1 and 11.2 are supported.

Finally, you need an *Oracle Database* for Python to connect to. Oracle's standard client-server version interoperability allows cx_Oracle to connect to both older and newer databases. Python can be local or remote to the database.

Upgrading from cx_Oracle 5

If you are upgrading from cx_Oracle 5 note these installation changes:

- When using Oracle Instant Client, you should not set `ORACLE_HOME`.
- On Linux, cx_Oracle 6 no longer uses Instant Client RPMs automatically. You must set `LD_LIBRARY_PATH` or use `ldconfig` to locate the Oracle client library.
- PyPI no longer allows Windows installers or Linux RPMs to be hosted. Use the supplied cx_Oracle Wheels instead.

Install Using Pip

Pip is the generic tool for installing Python packages. If you do not have pip, see the pip installation documentation.

The command to install cx_Oracle 6 using pip on all platforms is:

```
python -m pip install cx_Oracle --upgrade
```

This will download and install a pre-compiled binary matching your platform and architecture automatically, if one is available. Pre-compiled binaries are available for Windows and Linux. See PyPI.

If a pre-compiled binary is not available, the source will be downloaded, compiled, and the resulting binary installed. On Linux if cx_Oracle needs to be compiled for the default python package, you will need the `python-devel` package or equivalent, which provides the *Python.h* header file.

On macOS make sure you are not using the bundled Python - use Homebrew or Python.org instead.

Install Using GitHub

In order to install using the source on GitHub, use the following commands:

```
git clone https://github.com/oracle/python-cx_Oracle.git cx_Oracle
cd cx_Oracle
git submodule init
git submodule update
python setup.py install
```

Note that if you download a source zip file directly from GitHub then you will also need to download an ODPI-C source zip file and extract it inside the directory called "odpi".

Install Using Source from PyPI

The source package can be downloaded manually from PyPI and extracted, after which the following commands should be run:

python setup.py build python setup.py install

Install Oracle Client

Using cx_Oracle requires Oracle Client libraries to be installed. The libraries provide the necessary network connectivity allowing applications to access an Oracle Database instance. They also provide basic and advanced connection management and data features to cx_Oracle. cx_Oracle uses the shared library loading mechanism available on each supported platform to load the Oracle Client library at runtime. Oracle Client versions 12.2, 12.1 and 11.2 are supported.

The simplest Oracle Client is the free Oracle Instant Client. Only the "Basic" or "Basic Light" package is required. Oracle Client libraries are also available in any Oracle Database installation or full Oracle Client installation.

Make sure your library loading path, such as `PATH` on Windows, or `LD_LIBRARY_PATH` on Linux, is set to the location of the Oracle Client libraries. On macOS the libraries should be in `~/lib` or `/usr/local/lib`.

On Windows, Microsoft Windows Redistributables matching the version of the Oracle client libraries need to be installed.

See ODPI-C installation instructions for details on configuration.

Oracle Database

Oracle Client libraries allow connection to older and newer databases. In summary, Oracle Client 12.2 can connect to Oracle Database 11.2 or greater. Oracle Client 12.1 can connect to Oracle Database 10.2 or greater. Oracle Client 11.2 can connect to Oracle Database 9.2 or greater. For additional information on which Oracle Database releases are supported by which Oracle client versions, please see Doc ID 207303.1.

Since a single cx_Oracle binary can use multiple client versions and access multiple database versions, it is important your application is tested in your intended release environments. Newer Oracle clients support new features, such as the oraaccess.xml external configuration file available with 12.1 or later clients, and session pool enhancements to dead connection detection in 12.2 clients.

The function *clientversion()* can be used to determine which Oracle Client version is in use and the attribute *Connection.version* can be used to determine which Oracle Database version a connection is accessing. These can then be used to adjust application behavior accordingly. Attempts to use some Oracle features that are not supported by a particular client/server combination may result in runtime errors. These include:

- when attempting to access attributes that are not supported by the current Oracle Client library you will get the error "ORA-24315: illegal attribute type"
- when attempting to use implicit results with Oracle Client 11.2 against Oracle Database 12c you will get the error "ORA-29481: Implicit results cannot be returned to client"
- when attempting to get array DML row counts with Oracle Client 11.2 you will get the error "DPI-1013: not supported"

Troubleshooting

If installation fails:

- Use option `-v` with pip. Review your output and logs. Try to install using a different method. **Google anything that looks like an error.** Try some potential solutions.
- Was there a network connection error? Do you need to see the environment variables `http_proxy` and/or `https_proxy`?
- Do you get the error "`No module named pip`"? The pip module is builtin to Python from version 2.7.9 but is sometimes removed by the OS. Use the venv module (builtin to Python 3.x) or virtualenv module (Python 2.x) instead.
- Do you get the error "`fatal error: dpi.h: No such file or directory`" when building from source code? Ensure that your source installation has a subdirectory called "odpi" containing files. If missing, review the section on *Install Using GitHub*.

If importing cx_Oracle fails:

- Do you get the error "`DPI-1047: Oracle Client library cannot be loaded`"?
  - Check the `PATH` environment variable on Windows. Ensure that you have restarted your command prompt if you have modified environment variables.
  - Check the `LD_LIBRARY_PATH` environment variable on Linux.
  - On macOS, make sure Oracle Instant Client is in `~/lib` or `/usr/local/lib` and that you are not using the bundled Python (use Homebrew or Python.org instead).
  - Check that Python, cx_Oracle and your Oracle Client libraries are all 64-bit or all 32-bit.
  - on Windows, check that the correct Windows Redistributables have been installed.

- If you have both Python 2 and 3 installed, make sure you are using the correct python and pip (or python3 and pip3) executables.

CHAPTER 2

Module Interface

cx_Oracle.**Binary**(*string*)
Construct an object holding a binary (long) string value.

cx_Oracle.**clientversion**()
Return the version of the client library being used as a 5-tuple. The five values are the major version, minor version, update number, patch number and port update number.

Note: This method is an extension to the DB API definition.

cx_Oracle.**Connection**([*user*, *password*, *dsn*, *mode*, *handle*, *pool*, *threaded*, *events*, *cclass*, *purity*, *newpassword*, *encoding*, *nencoding*, *edition*, *appcontext*, *tag*, *matchanytag*])
cx_Oracle.**connect**([*user*, *password*, *dsn*, *mode*, *handle*, *pool*, *threaded*, *events*, *cclass*, *purity*, *newpassword*, *encoding*, *nencoding*, *edition*, *appcontext*, *tag*, *matchanytag*])
Constructor for creating a connection to the database. Return a *connection object*. All arguments are optional and can be specified as keyword parameters.

The dsn (data source name) is the TNS entry (from the Oracle names server or tnsnames.ora file) or is a string like the one returned from *makedsn()*. If only one parameter is passed, a connect string is assumed which is to be of the format `user/password@dsn`, the same format accepted by Oracle applications such as SQL*Plus.

If the mode is specified, it must be one of *SYSDBA*, *SYSASM* or *SYSOPER* which are defined at the module level; otherwise, it defaults to the normal mode of connecting.

If the handle is specified, it must be of type OCISvcCtx* and is only of use when embedding Python in an application (like PowerBuilder) which has already made the connection.

The pool argument is expected to be a *session pool object* and the use of this argument is the equivalent of calling pool.acquire().

The threaded argument is expected to be a boolean expression which indicates whether or not Oracle should wrap accesses to connections with a mutex. Doing so in single threaded applications imposes a performance penalty of about 10-15% which is why the default is False.

The events argument is expected to be a boolean expression which indicates whether or not to initialize Oracle in events mode.

The cclass argument is expected to be a string and defines the connection class for database resident connection pooling (DRCP).

The purity argument is expected to be one of *ATTR_PURITY_NEW*, *ATTR_PURITY_SELF*, or *ATTR_PURITY_DEFAULT*.

The newpassword argument is expected to be a string if specified and sets the password for the logon during the connection process.

The encoding argument is expected to be a string if specified and sets the encoding to use for regular database strings. If not specified, the environment variable NLS_LANG is used. If the environment variable NLS_LANG is not set, ASCII is used.

The nencoding argument is expected to be a string if specified and sets the encoding to use for national character set database strings. If not specified, the environment variable NLS_NCHAR is used. If the environment variable NLS_NCHAR is not used, the environment variable NLS_LANG is used instead, and if the environment variable NLS_LANG is not set, ASCII is used.

The edition argument is expected to be a string if specified and sets the edition to use for the session. It is only relevant if both the client and the server are at least Oracle Database 11.2.

The appcontext argument is expected to be a list of 3-tuples, if specified, and sets the application context for the connection. Application context is available in the database by using the sys_context() PL/SQL method and can be used within a logon trigger as well as any other PL/SQL procedures. Each entry in the list is expected to contain three strings: the namespace, the name and the value.

The tag argument, if specified, is expected to be a string and will limit the sessions that can be returned from a session pool unless the matchanytag argument is set to True. In that case sessions with the specified tag will be preferred over others, but if no such sessions are available a session with a different tag may be returned instead. In any case, untagged sessions will always be returned if no sessions with the specified tag are available. Sessions are tagged when they are *released* back to the pool.

cx_Oracle.**Cursor**(*connection*)

Constructor for creating a cursor. Return a new *cursor object* using the connection.

Note: This method is an extension to the DB API definition.

---

cx_Oracle.**Date**(*year*, *month*, *day*)

Construct an object holding a date value.

cx_Oracle.**DateFromTicks**(*ticks*)

Construct an object holding a date value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details).

cx_Oracle.**makedsn**(*host*, *port*, *sid=None*, *service_name=None*, *region=None*, *sharding_key=None*, *super_sharding_key=None*)

Return a string suitable for use as the dsn argument for *connect()*. This string is identical to the strings that are defined by the Oracle names server or defined in the tnsnames.ora file.

Note: This method is an extension to the DB API definition.

---

cx_Oracle.**SessionPool**(*user*, *password*, *database*, *min*, *max*, *increment*[, *connectiontype=cx_Oracle.Connection*, *threaded=False*, *getmode=cx_Oracle.SPOOL_ATTRVAL_NOWAIT*, *homogeneous=True*, *externalauth=False*, *encoding=None*, *nencoding=None*, *edition=None*])

Create and return a *session pool object*. This allows for very fast connections to the database and is of primary

use in a server where the same connection is being made multiple times in rapid succession (a web server, for example).

If the connection type is specified, all calls to *acquire()* will create connection objects of that type, rather than the base type defined at the module level.

The threaded attribute is expected to be a boolean expression which indicates whether Oracle should wrap accesses to connections with a mutex. Doing so in single threaded applications imposes a performance penalty of about 10-15% which is why the default is False.

The encoding argument is expected to be a string if specified and sets the encoding to use for regular database strings. If not specified, the environment variable NLS_LANG is used. If the environment variable NLS_LANG is not set, ASCII is used.

The nencoding argument is expected to be a string if specified and sets the encoding to use for national character set database strings. If not specified, the environment variable NLS_NCHAR is used. If the environment variable NLS_NCHAR is not used, the environment variable NLS_LANG is used instead, and if the environment variable NLS_LANG is not set, ASCII is used.

The edition argument is expected to be a string, if specified, and sets the edition to use for the sessions in the pool. It is only relevant if both the client and the server are at least Oracle Database 11.2.

Note: This method is an extension to the DB API definition.

---

cx_Oracle.**Time**(*hour*, *minute*, *second*)
Construct an object holding a time value.

cx_Oracle.**TimeFromTicks**(*ticks*)
Construct an object holding a time value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details).

cx_Oracle.**Timestamp**(*year*, *month*, *day*, *hour*, *minute*, *second*)
Construct an object holding a time stamp value.

cx_Oracle.**TimestampFromTicks**(*ticks*)
Construct an object holding a time stamp value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details).

Constants

General

cx_Oracle.**apilevel**
String constant stating the supported DB API level. Currently '2.0'.

cx_Oracle.**buildtime**
String constant stating the time when the binary was built.

Note: This constant is an extension to the DB API definition.

---

cx_Oracle.**paramstyle**
String constant stating the type of parameter marker formatting expected by the interface. Currently 'named' as in 'where name = :name'.

cx_Oracle.**threadsafety**
Integer constant stating the level of thread safety that the interface supports. Currently 2, which means that threads may share the module and connections, but not cursors. Sharing means that a thread may use a resource without wrapping it using a mutex semaphore to implement resource locking.

Note that in order to make use of multiple threads in a program which intends to connect and disconnect in different threads, the threaded argument to *connect()* or *SessionPool()* must be true.

cx_Oracle.**version**

cx_Oracle.**__version__**
String constant stating the version of the module. Currently '6.0'.

Note: This attribute is an extension to the DB API definition.

Advanced Queuing: Delivery Modes

These constants are extensions to the DB API definition. They are possible values for the *deliverymode* attribute of the *dequeue options object* passed as the options argument to the *Connection.deq()* method as well as the *deliverymode* attribute of the *enqueue options object* passed as the options argument to the *Connection.enq()* method. They are also possible values for the *deliverymode* attribute of the *message properties object* passed as the msgproperties argument to the *Connection.deq()* and *Connection.enq()* methods.

cx_Oracle.**MSG_BUFFERED**
This constant is used to specify that enqueue/dequeue operations should enqueue or dequeue buffered messages.

cx_Oracle.**MSG_PERSISTENT**
This constant is used to specify that enqueue/dequeue operations should enqueue or dequeue persistent messages. This is the default value.

cx_Oracle.**MSG_PERSISTENT_OR_BUFFERED**
This constant is used to specify that dequeue operations should dequeue either persistent or buffered messages.

Advanced Queuing: Dequeue Modes

These constants are extensions to the DB API definition. They are possible values for the *mode* attribute of the *dequeue options object*. This object is the options argument for the *Connection.deq()* method.

cx_Oracle.**DEQ_BROWSE**
This constant is used to specify that dequeue should read the message without acquiring any lock on the message (eqivalent to a select statement).

cx_Oracle.**DEQ_LOCKED**
This constant is used to specify that dequeue should read and obtain a write lock on the message for the duration of the transaction (equivalent to a select for update statement).

cx_Oracle.**DEQ_REMOVE**
This constant is used to specify that dequeue should read the message and update or delete it. This is the default value.

cx_Oracle.**DEQ_REMOVE_NODATA**
This constant is used to specify that dequeue should confirm receipt of the message but not deliver the actual message content.

Advanced Queuing: Dequeue Navigation Modes

These constants are extensions to the DB API definition. They are possible values for the *navigation* attribute of the *dequeue options object*. This object is the options argument for the *Connection.deq()* method.

cx_Oracle.**DEQ_FIRST_MSG**
: This constant is used to specify that dequeue should retrieve the first available message that matches the search criteria. This resets the position to the beginning of the queue.

cx_Oracle.**DEQ_NEXT_MSG**
: This constant is used to specify that dequeue should retrieve the next available message that matches the search criteria. If the previous message belongs to a message group, AQ retrieves the next available message that matches the search criteria and belongs to the message group. This is the default.

cx_Oracle.**DEQ_NEXT_TRANSACTION**
: This constant is used to specify that dequeue should skip the remainder of the transaction group and retrieve the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue.

Advanced Queuing: Dequeue Visibility Modes

These constants are extensions to the DB API definition. They are possible values for the *visibility* attribute of the *dequeue options object*. This object is the options argument for the *Connection.deq()* method.

cx_Oracle.**DEQ_IMMEDIATE**
: This constant is used to specify that dequeue should perform its work as part of an independent transaction.

cx_Oracle.**DEQ_ON_COMMIT**
: This constant is used to specify that dequeue should be part of the current transaction. This is the default value.

Advanced Queuing: Dequeue Wait Modes

These constants are extensions to the DB API definition. They are possible values for the *wait* attribute of the *dequeue options object*. This object is the options argument for the *Connection.deq()* method.

cx_Oracle.**DEQ_NO_WAIT**
: This constant is used to specify that dequeue not wait for messages to be available for dequeuing.

cx_Oracle.**DEQ_WAIT_FOREVER**
: This constant is used to specify that dequeue should wait forever for messages to be available for dequeuing. This is the default value.

Advanced Queuing: Enqueue Visibility Modes

These constants are extensions to the DB API definition. They are possible values for the *visibility* attribute of the *enqueue options object*. This object is the options argument for the *Connection.enq()* method.

cx_Oracle.**ENQ_IMMEDIATE**
: This constant is used to specify that enqueue should perform its work as part of an independent transaction.

cx_Oracle.**ENQ_ON_COMMIT**
: This constant is used to specify that enqueue should be part of the current transaction. This is the default value.

Advanced Queuing: Message States

These constants are extensions to the DB API definition. They are possible values for the *state* attribute of the *message properties object*. This object is the msgproperties argument for the *Connection.deq()* and *Connection.enq()* methods.

cx_Oracle.**MSG_EXPIRED**
: This constant is used to specify that the message has been moved to the exception queue.

cx_Oracle.**MSG_PROCESSED**
: This constant is used to specify that the message has been processed and has been retained.

cx_Oracle.**MSG_READY**
: This constant is used to specify that the message is ready to be processed.

cx_Oracle.**MSG_WAITING**
: This constant is used to specify that the message delay has not yet been reached.

Advanced Queuing: Other

These constants are extensions to the DB API definition. They are special constants used in advanced queuing.

cx_Oracle.**MSG_NO_DELAY**
: This constant is a possible value for the *delay* attribute of the *message properties object* passed as the msgproperties parameter to the *Connection.deq()* and *Connection.enq()* methods. It specifies that no delay should be imposed and the message should be immediately available for dequeuing. This is also the default value.

cx_Oracle.**MSG_NO_EXPIRATION**
: This constant is a possible value for the *expiration* attribute of the *message properties object* passed as the msgproperties parameter to the *Connection.deq()* and *Connection.enq()* methods. It specifies that the message never expires. This is also the default value.

Connection Authorization Modes

These constants are extensions to the DB API definition. They are possible values for the mode parameter of the *connect()* method.

cx_Oracle.**PRELIM_AUTH**
: This constant is used to specify that preliminary authentication is to be used. This is needed for performing database startup and shutdown.

cx_Oracle.**SYSASM**
: This constant is used to specify that SYSASM access is to be acquired.

cx_Oracle.**SYSDBA**
: This constant is used to specify that SYSDBA access is to be acquired.

cx_Oracle.**SYSOPER**
: This constant is used to specify that SYSOPER access is to be acquired.

Database Shutdown Modes

These constants are extensions to the DB API definition. They are possible values for the mode parameter of the *Connection.shutdown()* method.

cx_Oracle.**DBSHUTDOWN_ABORT**
: This constant is used to specify that the caller should not wait for current processing to complete or for users to disconnect from the database. This should only be used in unusual circumstances since database recovery may be necessary upon next startup.

cx_Oracle.**DBSHUTDOWN_FINAL**
: This constant is used to specify that the instance can be truly halted. This should only be done after the database has been shutdown with one of the other modes (except abort) and the database has been closed and dismounted using the appropriate SQL commands.

cx_Oracle.**DBSHUTDOWN_IMMEDIATE**
: This constant is used to specify that all uncommitted transactions should be rolled back and any connected users should be disconnected.

cx_Oracle.**DBSHUTDOWN_TRANSACTIONAL**
: This constant is used to specify that further connections to the database should be prohibited and no new transactions should be allowed. It then waits for all active transactions to complete.

cx_Oracle.**DBSHUTDOWN_TRANSACTIONAL_LOCAL**
: This constant is used to specify that further connections to the database should be prohibited and no new transactions should be allowed. It then waits for only local active transactions to complete.

Event Types

These constants are extensions to the DB API definition. They are possible values for the *Message.type* attribute of the messages that are sent for subscriptions created by the *Connection.subscribe()* method.

cx_Oracle.**EVENT_DEREG**
: This constant is used to specify that the subscription has been deregistered and no further notifications will be sent.

cx_Oracle.**EVENT_NONE**
: This constant is used to specify no information is available about the event.

cx_Oracle.**EVENT_OBJCHANGE**
: This constant is used to specify that a database change has taken place on a table registered with the *Subscription.registerquery()* method.

cx_Oracle.**EVENT_QUERYCHANGE**
: This constant is used to specify that the result set of a query registered with the *Subscription.registerquery()* method has been changed.

cx_Oracle.**EVENT_SHUTDOWN**
: This constant is used to specify that the instance is in the process of being shut down.

cx_Oracle.**EVENT_SHUTDOWN_ANY**
: This constant is used to specify that any instance (when running RAC) is in the process of being shut down.

cx_Oracle.**EVENT_STARTUP**
: This constant is used to specify that the instance is in the process of being started up.

Operation Codes

These constants are extensions to the DB API definition. They are possible values for the operations argument for the *Connection.subscribe()* method. One or more of these values can be OR'ed together. These values are also used by the *MessageTable.operation* or *MessageQuery.operation* attributes of the messages that are sent.

cx_Oracle.**OPCODE_ALLOPS**
This constant is used to specify that messages should be sent for all operations.

cx_Oracle.**OPCODE_ALLROWS**
This constant is used to specify that the table or query has been completely invalidated.

cx_Oracle.**OPCODE_ALTER**
This constant is used to specify that messages should be sent when a registered table has been altered in some fashion by DDL, or that the message identifies a table that has been altered.

cx_Oracle.**OPCODE_DELETE**
This constant is used to specify that messages should be sent when data is deleted, or that the message identifies a row that has been deleted.

cx_Oracle.**OPCODE_DROP**
This constant is used to specify that messages should be sent when a registered table has been dropped, or that the message identifies a table that has been dropped.

cx_Oracle.**OPCODE_INSERT**
This constant is used to specify that messages should be sent when data is inserted, or that the message identifies a row that has been inserted.

cx_Oracle.**OPCODE_UPDATE**
This constant is used to specify that messages should be sent when data is updated, or that the message identifies a row that has been updated.

Session Pool Get Modes

These constants are extensions to the DB API definition. They are possible values for the getmode parameter of the *SessionPool()* method.

cx_Oracle.**SPOOL_ATTRVAL_FORCEGET**
This constant is used to specify that a new connection will be returned if there are no free sessions available in the pool.

cx_Oracle.**SPOOL_ATTRVAL_NOWAIT**
This constant is used to specify that an exception should be raised if there are no free sessions available in the pool. This is the default value.

cx_Oracle.**SPOOL_ATTRVAL_WAIT**
This constant is used to specify that the caller should wait until a session is available if there are no free sessions available in the pool.

Session Pool Purity

These constants are extensions to the DB API definition. They are possible values for the purity parameter of the *connect()* method, which is used in database resident connection pooling (DRCP).

cx_Oracle.**ATTR_PURITY_DEFAULT**
This constant is used to specify that the purity of the session is the default value identified by Oracle (see Oracle's documentation for more information). This is the default value.

cx_Oracle.**ATTR_PURITY_NEW**
This constant is used to specify that the session acquired from the pool should be new and not have any prior session state.

cx_Oracle.**ATTR_PURITY_SELF**
: This constant is used to specify that the session acquired from the pool need not be new and may have prior session state.

Subscription Namespaces

These constants are extensions to the DB API definition. They are possible values for the namespace parameter of the *Connection.subscribe()* method.

cx_Oracle.**SUBSCR_NAMESPACE_DBCHANGE**
: This constant is used to specify that database change notification or query change notification messages are to be sent. This is the default value and currently the only value that is supported.

Subscription Protocols

These constants are extensions to the DB API definition. They are possible values for the protocol parameter of the *Connection.subscribe()* method.

cx_Oracle.**SUBSCR_PROTO_HTTP**
: This constant is used to specify that notifications will be sent to an HTTP URL when a message is generated. This value is currently not supported.

cx_Oracle.**SUBSCR_PROTO_MAIL**
: This constant is used to specify that notifications will be sent to an e-mail address when a message is generated. This value is currently not supported.

cx_Oracle.**SUBSCR_PROTO_OCI**
: This constant is used to specify that notifications will be sent to the callback routine identified when the subscription was created. It is the default value and the only value currently supported.

cx_Oracle.**SUBSCR_PROTO_SERVER**
: This constant is used to specify that notifications will be sent to a PL/SQL procedure when a message is generated. This value is currently not supported.

Subscription Quality of Service

These constants are extensions to the DB API definition. They are possible values for the qos parameter of the *Connection.subscribe()* method. One or more of these values can be OR'ed together.

cx_Oracle.**SUBSCR_CQ_QOS_BEST_EFFORT**
: This constant is used to specify that best effort filtering for query result set changes is acceptable. False positive notifications may be received. This behaviour may be suitable for caching applications.

  Deprecated since version 5.3: Use *SUBSCR_QOS_BEST_EFFORT* instead.

cx_Oracle.**SUBSCR_CQ_QOS_QUERY**
: This constant is used to specify that notifications should be sent if the result set of the registered query changes. By default no false positive notifications will be generated.

  Deprecated since version 5.3: Use *SUBSCR_QOS_QUERY* instead.

cx_Oracle.**SUBSCR_QOS_BEST_EFFORT**
: This constant is used to specify that best effort filtering for query result set changes is acceptable. False positive notifications may be received. This behaviour may be suitable for caching applications.

cx_Oracle.**SUBSCR_QOS_DEREG_NFY**
This constant is used to specify that the subscription should be automatically unregistered after the first notification is received.

cx_Oracle.**SUBSCR_QOS_PURGE_ON_NTFN**
This constant is used to specify that the subscription should be automatically unregistered after the first notification is received.

Deprecated since version 5.3: Use *SUBSCR_QOS_DEREG_NFY* instead.

cx_Oracle.**SUBSCR_QOS_QUERY**
This constant is used to specify that notifications should be sent if the result set of the registered query changes. By default no false positive notifications will be generated.

cx_Oracle.**SUBSCR_QOS_RELIABLE**
This constant is used to specify that notifications should not be lost in the event of database failure.

cx_Oracle.**SUBSCR_QOS_ROWIDS**
This constant is used to specify that the rowids of the inserted, updated or deleted rows should be included in the message objects that are sent.

Types

cx_Oracle.**BINARY**
This type object is used to describe columns in a database that contain binary data. In Oracle this is RAW columns.

cx_Oracle.**BFILE**
This type object is used to describe columns in a database that are BFILEs.

Note: This type is an extension to the DB API definition.

cx_Oracle.**BLOB**
This type object is used to describe columns in a database that are BLOBs.

Note: This type is an extension to the DB API definition.

cx_Oracle.**BOOLEAN**
This type object is used to represent PL/SQL booleans.

New in version 5.2.1.

Note: This type is an extension to the DB API definition. It is only available in Oracle 12.1 and higher and only within PL/SQL. It cannot be used in columns.

cx_Oracle.**CLOB**
This type object is used to describe columns in a database that are CLOBs.

Note: This type is an extension to the DB API definition.

cx_Oracle.**CURSOR**
This type object is used to describe columns in a database that are cursors (in PL/SQL these are known as ref cursors).

Note: This type is an extension to the DB API definition.

cx_Oracle.**DATETIME**
This type object is used to describe columns in a database that are dates.

cx_Oracle.**FIXED_CHAR**
This type object is used to describe columns in a database that are fixed length strings (in Oracle these is CHAR columns); these behave differently in Oracle than varchar2 so they are differentiated here even though the DB API does not differentiate them.

Note: This attribute is an extension to the DB API definition.

cx_Oracle.**FIXED_NCHAR**
This type object is used to describe columns in a database that are NCHAR columns in Oracle; these behave differently in Oracle than nvarchar2 so they are differentiated here even though the DB API does not differentiate them.

Note: This type is an extension to the DB API definition.

cx_Oracle.**INTERVAL**
This type object is used to describe columns in a database that are of type interval day to second.

Note: This type is an extension to the DB API definition.

cx_Oracle.**LOB**
This type object is the Python type of *BLOB* and *CLOB* data that is returned from cursors.

Note: This type is an extension to the DB API definition.

cx_Oracle.**LONG_BINARY**
This type object is used to describe columns in a database that are long binary (in Oracle these are LONG RAW columns).

Note: This type is an extension to the DB API definition.

cx_Oracle.**LONG_NCHAR**
This type object is used to describe columns in a database that are long NCHAR columns. There is no direct support for this in Oracle but long NCHAR strings are bound this way in order to avoid the "unimplemented or unreasonable conversion requested" error.

Deprecated since version 5.3.

Note: This type is an extension to the DB API definition.

cx_Oracle.**LONG_STRING**
This type object is used to describe columns in a database that are long strings (in Oracle these are LONG columns).

Note: This type is an extension to the DB API definition.

cx_Oracle.**NATIVE_FLOAT**
This type object is used to describe columns in a database that are of type binary_double or binary_float.

Note: This type is an extension to the DB API definition.

cx_Oracle.**NATIVE_INT**
This type object is used to bind integers using Oracle's native integer support, rather than the standard number support, which improves performance.

New in version 5.3.

Note: This type is an extension to the DB API definition.

cx_Oracle.**NCHAR**
This type object is used to describe national character strings (NVARCHAR2) in Oracle.

Note: This type is an extension to the DB API definition.

cx_Oracle.**NCLOB**
This type object is used to describe columns in a database that are NCLOBs.

Note: This type is an extension to the DB API definition.

cx_Oracle.**NUMBER**
This type object is used to describe columns in a database that are numbers.

cx_Oracle.**OBJECT**
This type object is used to describe columns in a database that are objects.

Note: This type is an extension to the DB API definition.

cx_Oracle.**ROWID**
This type object is used to describe the pseudo column "rowid".

cx_Oracle.**STRING**
This type object is used to describe columns in a database that are strings (in Oracle this is VARCHAR2 columns).

cx_Oracle.**TIMESTAMP**
This type object is used to describe columns in a database that are timestamps.

Note: This attribute is an extension to the DB API definition.

Exceptions

exception cx_Oracle.**Warning**
Exception raised for important warnings and defined by the DB API but not actually used by cx_Oracle.

exception cx_Oracle.**Error**
Exception that is the base class of all other exceptions defined by cx_Oracle and is a subclass of the Python StandardError exception (defined in the module exceptions).

exception cx_Oracle.**InterfaceError**
Exception raised for errors that are related to the database interface rather than the database itself. It is a subclass of Error.

exception cx_Oracle.**DatabaseError**
Exception raised for errors that are related to the database. It is a subclass of Error.

exception cx_Oracle.**DataError**
Exception raised for errors that are due to problems with the processed data. It is a subclass of DatabaseError.

exception cx_Oracle.**OperationalError**
Exception raised for errors that are related to the operation of the database but are not necessarily under the control of the progammer. It is a subclass of DatabaseError.

exception cx_Oracle.**IntegrityError**
Exception raised when the relational integrity of the database is affected. It is a subclass of DatabaseError.

exception cx_Oracle.**InternalError**
Exception raised when the database encounters an internal error. It is a subclass of DatabaseError.

exception cx_Oracle.**ProgrammingError**
Exception raised for programming errors. It is a subclass of DatabaseError.

exception cx_Oracle.**NotSupportedError**
Exception raised when a method or database API was used which is not supported by the database. It is a subclass of DatabaseError.

Exception handling

Note: PEP 249 (Python Database API Specification v2.0) says the following about exception values:

> [...] The values of these exceptions are not defined. They should give the user a fairly good idea of what went wrong, though. [...]

With cx_Oracle every exception object has exactly one argument in the `args` tuple. This argument is a `cx_Oracle._Error` object which has the following five read-only attributes.

`_Error.`**code**
Integer attribute representing the Oracle error number (ORA-XXXXX).

`_Error.`**offset**
Integer attribute representing the error offset when applicable.

`_Error.`**message**
String attribute representing the Oracle message of the error. This message is localized by the environment of the Oracle connection.

_Error.**context**
: String attribute representing the context in which the exception was raised.

_Error.**isrecoverable**
: Boolean attribute representing whether the error is recoverable or not. This is False in all cases unless Oracle Database 12.1 is being used on both the server and the client.

  New in version 5.3.

This allows you to use the exceptions for example in the following way:

```
from __future__ import print_function

import cx_Oracle

connection = cx_Oracle.Connection("cx_Oracle/dev@localhost/orclpdb")
cursor = connection.cursor()

try:
    cursor.execute("select 1 / 0 from dual")
except cx_Oracle.DatabaseError as exc:
    error, = exc.args
    print("Oracle-Error-Code:", error.code)
    print("Oracle-Error-Message:", error.message)
```

CHAPTER 3

Connection Object

Note: Any outstanding changes will be rolled back when the connection object is destroyed or closed.

`Connection.__enter__()`
The entry point for the connection as a context manager, a feature available in Python 2.5 and higher. It returns itself.

Note: This method is an extension to the DB API definition.

`Connection.__exit__()`
The exit point for the connection as a context manager, a feature available in Python 2.5 and higher. In the event of an exception, the transaction is rolled back; otherwise, the transaction is committed.

Note: This method is an extension to the DB API definition.

`Connection.action`
This write-only attribute sets the action column in the v$session table. It is a string attribute and cannot be set to None – use the empty string instead.

Note: This attribute is an extension to the DB API definition.

`Connection.autocommit`
This read-write attribute determines whether autocommit mode is on or off. When autocommit mode is on, all statements are committed as soon as they have completed executing.

Note: This attribute is an extension to the DB API definition.

Connection.**begin**([*formatId*, *transactionId*, *branchId*])
Explicitly begin a new transaction. Without parameters, this explicitly begins a local transaction; otherwise, this explicitly begins a distributed (global) transaction with the given parameters. See the Oracle documentation for more details.

Note that in order to make use of global (distributed) transactions, the *internal_name* and *external_name* attributes must be set.

Note: This method is an extension to the DB API definition.

Connection.**cancel**()
Cancel a long-running transaction.

Note: This method is an extension to the DB API definition.

Connection.**changepassword**(*oldpassword*, *newpassword*)
Change the password of the logon.

Note: This method is an extension to the DB API definition.

Connection.**client_identifier**
This write-only attribute sets the client_identifier column in the v$session table.

Note: This attribute is an extension to the DB API definition.

Connection.**clientinfo**
This write-only attribute sets the client_info column in the v$session table.

Note: This attribute is an extension to the DB API definition.

Connection.**close**()
Close the connection now, rather than whenever __del__ is called. The connection will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the connection. The same applies to any cursor objects trying to use the connection.

Connection.**commit**()
Commit any pending transactions to the database.

Connection.**current_schema**
This read-write attribute sets the current schema attribute for the session.

Note: This attribute is an extension to the DB API definition.

Connection.**cursor**()
Return a new *cursor object* using the connection.

Connection.**dbop**
This write-only attribute sets the database operation that is to be monitored. This can be viewed in the DBOP_NAME column of the V$SQL_MONITOR table.

Note: This attribute is an extension to the DB API definition.

Connection.**deq**(*name*, *options*, *msgproperties*, *payload*)

Returns a message id after successfully dequeuing a message. The options object can be created using *deqoptions()* and the msgproperties object can be created using *msgproperties()*. The payload must be an object created using *ObjectType.newobject()*.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**deqoptions**()

Returns an object specifying the options to use when dequeuing messages. See *Dequeue Options* for more information.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**dsn**

This read-only attribute returns the TNS entry of the database to which a connection has been established.

Note: This attribute is an extension to the DB API definition.

Connection.**edition**

This read-only attribute gets the session edition and is only available in Oracle Database 11.2 (both client and server must be at this level or higher for this to work).

New in version 5.3.

Note: This attribute is an extension to the DB API definition.

Connection.**encoding**

This read-only attribute returns the IANA character set name of the character set in use by the Oracle client for regular strings.

Note: This attribute is an extension to the DB API definition.

Connection.**enq**(*name*, *options*, *msgproperties*, *payload*)

Returns a message id after successfully enqueuing a message. The options object can be created using *enqoptions()* and the msgproperties object can be created using *msgproperties()*. The payload must be an object created using *ObjectType.newobject()*.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**enqoptions**()
Returns an object specifying the options to use when enqueuing messages. See *Enqueue Options* for more information.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**external_name**
This read-write attribute specifies the external name that is used by the connection when logging distributed transactions.

New in version 5.3.

Note: This attribute is an extension to the DB API definition.

Connection.**gettype**(*name*)
Return a *type object* given its name. This can then be used to create objects which can be bound to cursors created by this connection.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**handle**
This read-only attribute returns the OCI service context handle for the connection. It is primarily provided to facilitate testing the creation of a connection using the OCI service context handle.

Note: This attribute is an extension to the DB API definition.

Connection.**inputtypehandler**
This read-write attribute specifies a method called for each value that is bound to a statement executed on any cursor associated with this connection. The method signature is handler(cursor, value, arraysize) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the default behavior will take place for all values bound to statements.

Note: This attribute is an extension to the DB API definition.

Connection.**internal_name**
This read-write attribute specifies the internal name that is used by the connection when logging distributed transactions.

New in version 5.3.

Note: This attribute is an extension to the DB API definition.

Connection.**ltxid**
This read-only attribute returns the logical transaction id for the connection. It is used within Oracle Transaction Guard as a means of ensuring that transactions are not duplicated. See the Oracle documentation and the provided sample for more information.

New in version 5.3.

Connection.**maxBytesPerCharacter**
This read-only attribute returns the maximum number of bytes each character can use for the client character set.

Note: This attribute is an extension to the DB API definition.

Connection.**module**
This write-only attribute sets the module column in the v$session table. The maximum length for this string is 48 and if you exceed this length you will get ORA-24960.

Connection.**msgproperties**()
Returns an object specifying the properties of messages used in advanced queuing. See *Message Properties* for more information.

New in version 5.3.

Note: This method is an extension to the DB API definition.

Connection.**nencoding**
This read-only attribute returns the IANA character set name of the national character set in use by the Oracle client.

Note: This attribute is an extension to the DB API definition.

Connection.**outputtypehandler**
This read-write attribute specifies a method called for each column that is going to be fetched from any cursor associated with this connection. The method signature is handler(cursor, name, defaultType, length, precision, scale) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the default behavior will take place for all columns fetched from cursors.

Note: This attribute is an extension to the DB API definition.

Connection.**ping**()
Ping the server which can be used to test if the connection is still active.

Note: This method is an extension to the DB API definition.

Connection.**prepare**()
Prepare the distributed (global) transaction for commit. Return a boolean indicating if a transaction was actually prepared in order to avoid the error ORA-24756 (transaction does not exist).

Note: This method is an extension to the DB API definition.

Connection.**rollback**()
Rollback any pending transactions.

Connection.**shutdown**([*mode*])

Shutdown the database. In order to do this the connection must be connected as *SYSDBA* or *SYSOPER*. Two calls must be made unless the mode specified is *DBSHUTDOWN_ABORT*. An example is shown below:

```
import cx_Oracle

connection = cx_Oracle.Connection(mode = cx_Oracle.SYSDBA)
connection.shutdown(mode = cx_Oracle.DBSHUTDOWN_IMMEDIATE)
cursor = connection.cursor()
cursor.execute("alter database close normal")
cursor.execute("alter database dismount")
connection.shutdown(mode = cx_Oracle.DBSHUTDOWN_FINAL)
```

Note: This method is an extension to the DB API definition.

Connection.**startup**(*force=False*, *restrict=False*)

Startup the database. This is equivalent to the SQL*Plus command "startup nomount". The connection must be connected as *SYSDBA* or *SYSOPER* with the *PRELIM_AUTH* option specified for this to work. An example is shown below:

```
import cx_Oracle

connection = cx_Oracle.Connection(
        mode = cx_Oracle.SYSDBA | cx_Oracle.PRELIM_AUTH)
connection.startup()
connection = cx_Oracle.connect(mode = cx_Oracle.SYSDBA)
cursor = connection.cursor()
cursor.execute("alter database mount")
cursor.execute("alter database open")
```

Note: This method is an extension to the DB API definition.

Connection.**stmtcachesize**

This read-write attribute specifies the size of the statement cache. This value can make a significant difference in performance (up to 100x) if you have a small number of statements that you execute repeatedly.

Note: This attribute is an extension to the DB API definition.

Connection.**subscribe**(*namespace=cx_Oracle.SUBSCR_NAMESPACE_DBCHANGE*, *protocol=cx_Oracle.SUBSCR_PROTO_OCI*, *callback=None*, *timeout=0*, *operations=OPCODE_ALLOPS*, *port=0*, *qos=0*)

Return a new *subscription object* using the connection. Currently the namespace and protocol arguments cannot have any other meaningful values.

The callback is expected to be a callable that accepts a single argument. A *message object* is passed to this callback whenever a notification is received.

The timeout value specifies that the subscription expires after the given time in seconds. The default value of 0 indicates that the subscription never expires.

The operations argument enables filtering of the messages that are sent (insert, update, delete). The default value will send notifications for all operations.

The port specifies the listening port for callback notifications from the database server. If not specified, an unused port will be selected by the database.

The qos argument specifies quality of service options. It should be one or more of the following flags, OR'ed together: *cx_Oracle.SUBSCR_QOS_RELIABLE*, *cx_Oracle.SUBSCR_QOS_DEREG_NFY*, *cx_Oracle.SUBSCR_QOS_ROWIDS*, *cx_Oracle.SUBSCR_QOS_QUERY*, *cx_Oracle.SUBSCR_QOS_BEST_EFFORT*.

Note: This method is an extension to the DB API definition.

Note: Do not close the connection before the subscription object is deleted or the subscription object will not be deregistered in the database. This is done automatically if connection.close() is never called.

Connection.**tnsentry**

This read-only attribute returns the TNS entry of the database to which a connection has been established.

Note: This attribute is an extension to the DB API definition.

Connection.**username**

This read-only attribute returns the name of the user which established the connection to the database.

Note: This attribute is an extension to the DB API definition.

Connection.**version**

This read-only attribute returns the version of the database to which a connection has been established.

Note: This attribute is an extension to the DB API definition.

CHAPTER 4

Cursor Object

Cursor.**arraysize**
: This read-write attribute specifies the number of rows to fetch at a time internally and is the default number of rows to fetch with the *fetchmany()* call. It defaults to 100 meaning to fetch 100 rows at a time. Note that this attribute can drastically affect the performance of a query since it directly affects the number of network round trips that need to be performed. This is the reason for setting it to 100 instead of the 1 that the DB API recommends.

Cursor.**bindarraysize**
: This read-write attribute specifies the number of rows to bind at a time and is used when creating variables via *setinputsizes()* or *var()*. It defaults to 1 meaning to bind a single row at a time.

Note: The DB API definition does not define this attribute.

Cursor.**arrayvar**(*dataType*, *value*[, *size*])
: Create an array variable associated with the cursor of the given type and size and return a *variable object*. The value is either an integer specifying the number of elements to allocate or it is a list and the number of elements allocated is drawn from the size of the list. If the value is a list, the variable is also set with the contents of the list. If the size is not specified and the type is a string or binary, 4000 bytes is allocated. This is needed for passing arrays to PL/SQL (in cases where the list might be empty and the type cannot be determined automatically) or returning arrays from PL/SQL.

Note: The DB API definition does not define this method.

Cursor.**bindnames**()
: Return the list of bind variable names bound to the statement. Note that a statement must have been prepared first.

Note: The DB API definition does not define this method.

Cursor.**bindvars**
This read-only attribute provides the bind variables used for the last execute. The value will be either a list or a dictionary depending on whether binding was done by position or name. Care should be taken when referencing this attribute. In particular, elements should not be removed.

Note: The DB API definition does not define this attribute.

Cursor.**callfunc**(*name*, *returnType*, *parameters=[]*, *keywordParameters={}*)
Call a function with the given name. The return type is specified in the same notation as is required by *setinputsizes()*. The sequence of parameters must contain one entry for each argument that the function expects. Any keyword parameters will be included after the positional parameters. The result of the call is the return value of the function.

Note: The DB API definition does not define this method.

Note: If you intend to call *Cursor.setinputsizes()* on the cursor prior to making this call, then note that the first item in the argument list refers to the return value of the function.

Cursor.**callproc**(*name*, *parameters=[]*, *keywordParameters={}*)
Call a procedure with the given name. The sequence of parameters must contain one entry for each argument that the procedure expects. The result of the call is a modified copy of the input sequence. Input parameters are left untouched; output and input/output parameters are replaced with possibly new values. Keyword parameters will be included after the positional parameters and are not returned as part of the output sequence.

Note: The DB API definition does not allow for keyword parameters.

Cursor.**close**()
Close the cursor now, rather than whenever __del__ is called. The cursor will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the cursor.

Cursor.**connection**
This read-only attribute returns a reference to the connection object on which the cursor was created.

Note: This attribute is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**description**
This read-only attribute is a sequence of 7-item sequences. Each of these sequences contains information describing one result column: (name, type, display_size, internal_size, precision, scale, null_ok). This attribute will be None for operations that do not return rows or if the cursor has not had an operation invoked via the *execute()* method yet.

The type will be one of the type objects defined at the module level.

Cursor.**execute**(*statement*[, *parameters*], ***keywordParameters*)
Execute a statement against the database. Parameters may be passed as a dictionary or sequence or as keyword arguments. If the arguments are a dictionary, the values will be bound by name and if the arguments are a sequence the values will be bound by position. Note that if the values are bound by position, the order of the variables is from left to right as they are encountered in the statement and SQL statements are processed

differently than PL/SQL statements. For this reason, it is generally recommended to bind parameters by name instead of by position.

A reference to the statement will be retained by the cursor. If None or the same string object is passed in again, the cursor will execute that statement again without performing a prepare or rebinding and redefining. This is most effective for algorithms where the same statement is used, but different parameters are bound to it (many times). Note that parameters that are not passed in during subsequent executions will retain the value passed in during the last execution that contained them.

For maximum efficiency when reusing an statement, it is best to use the *setinputsizes()* method to specify the parameter types and sizes ahead of time; in particular, None is assumed to be a string of length 1 so any values that are later bound as numbers or dates will raise a TypeError exception.

If the statement is a query, the cursor is returned as a convenience to the caller (so it can be used directly as an iterator over the rows in the cursor); otherwise, None is returned.

Cursor.**executemany**(*statement*, *parameters*, *batcherrors=False*, *arraydmlrowcounts=False*)

Prepare a statement for execution against a database and then execute it against all parameter mappings or sequences found in the sequence parameters. The statement is managed in the same way as the *execute()* method manages it.

When true, the batcherrors parameter enables batch error support within Oracle and ensures that the call succeeds even if an exception takes place in one or more of the sequence of parameters. The errors can then be retrieved using *getbatcherrors()*.

When true, the arraydmlrowcounts parameter enables DML row counts to be retrieved from Oracle after the method has completed. The row counts can then be retrieved using *getarraydmlrowcounts()*.

Cursor.**executemanyprepared**(*numIters*)

Execute the previously prepared and bound statement the given number of times. The variables that are bound must have already been set to their desired value before this call is made. This method was designed for the case where optimal performance is required as it comes at the expense of compatibility with the DB API.

Note: The DB API definition does not define this method.

Cursor.**fetchall**()

Fetch all (remaining) rows of a query result, returning them as a list of tuples. An empty list is returned if no more rows are available. Note that the cursor's arraysize attribute can affect the performance of this operation, as internally reads from the database are done in batches corresponding to the arraysize.

An exception is raised if the previous call to *execute()* did not produce any result set or no call was issued yet.

Cursor.**fetchmany**([*numRows=cursor.arraysize*])

Fetch the next set of rows of a query result, returning a list of tuples. An empty list is returned if no more rows are available. Note that the cursor's arraysize attribute can affect the performance of this operation.

The number of rows to fetch is specified by the parameter. If it is not given, the cursor's arrysize attribute determines the number of rows to be fetched. If the number of rows available to be fetched is fewer than the amount requested, fewer rows will be returned.

An exception is raised if the previous call to *execute()* did not produce any result set or no call was issued yet.

Cursor.**fetchone**()

Fetch the next row of a query result set, returning a single tuple or None when no more data is available.

An exception is raised if the previous call to *execute()* did not produce any result set or no call was issued yet.

Cursor.**fetchraw**([*numRows=cursor.arraysize*])

Fetch the next set of rows of a query result into the internal buffers of the defined variables for the cursor. The number of rows actually fetched is returned. This method was designed for the case where optimal performance is required as it comes at the expense of compatibility with the DB API.

An exception is raised if the previous call to *execute()* did not produce any result set or no call was issued yet.

Note: The DB API definition does not define this method.

Cursor.**fetchvars**

This read-only attribute specifies the list of variables created for the last query that was executed on the cursor. Care should be taken when referencing this attribute. In particular, elements should not be removed.

Note: The DB API definition does not define this attribute.

Cursor.**getarraydmlrowcounts**()

Retrieve the DML row counts after a call to *executemany()* with arraydmlrowcounts enabled. This will return a list of integers corresponding to the number of rows affected by the DML statement for each element of the array passed to *executemany()*.

Note: The DB API definition does not define this method and it is only available for Oracle 12.1 and higher.

Cursor.**getbatcherrors**()

Retrieve the exceptions that took place after a call to *executemany()* with batcherors enabled. This will return a list of Error objects, one error for each iteration that failed. The offset can be determined by looking at the offset attribute of the error object.

Note: The DB API definition does not define this method.

Cursor.**getimplicitresults**()

Return a list of cursors which correspond to implicit results made available from a PL/SQL block or procedure without the use of OUT ref cursor parameters. The PL/SQL block or procedure opens the cursors and marks them for return to the client using the procedure dbms_sql.return_result. Cursors returned in this fashion should not be closed. They will be closed automatically by the parent cursor when it is closed. Closing the parent cursor will invalidate the cursors returned by this method.

New in version 5.3.

Note: The DB API definition does not define this method and it is only available for Oracle Database 12.1 (both client and server must be at this level or higher). It is most like the DB API method nextset(), but unlike that method (which requires that the next result set overwrite the current result set), this method returns cursors which can be fetched independently of each other.

Cursor.**inputtypehandler**

This read-write attribute specifies a method called for each value that is bound to a statement executed on the cursor and overrides the attribute with the same name on the connection if specified. The method signature is handler(cursor, value, arraysize) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the value of the attribute with the same name on the connection is used.

Note: This attribute is an extension to the DB API definition.

Cursor.**__iter__**()
Returns the cursor itself to be used as an iterator.

Note: This method is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**next**()
Fetch the next row of a query result set, using the same semantics as the method fetchone().

Note: This method is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**outputtypehandler**
This read-write attribute specifies a method called for each column that is to be fetched from this cursor. The method signature is handler(cursor, name, defaultType, length, precision, scale) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the value of the attribute with the same name on the connection is used instead.

Note: This attribute is an extension to the DB API definition.

Cursor.**parse**(*statement*)
This can be used to parse a statement without actually executing it (this step is done automatically by Oracle when a statement is executed).

Note: The DB API definition does not define this method.

Note: You can parse any DML or DDL statement. DDL statements are executed immediately and an implied commit takes place.

Cursor.**prepare**(*statement*[, *tag*])
This can be used before a call to *execute()* to define the statement that will be executed. When this is done, the prepare phase will not be performed when the call to *execute()* is made with None or the same string object as the statement. If specified the statement will be returned to the statement cache with the given tag. See the Oracle documentation for more information about the statement cache.

Note: The DB API definition does not define this method.

Cursor.**rowcount**
This read-only attribute specifies the number of rows that have currently been fetched from the cursor (for select statements) or that have been affected by the operation (for insert, update and delete statements).

Cursor.**rowfactory**
This read-write attribute specifies a method to call for each row that is retrieved from the database. Ordinarily a

tuple is returned for each row but if this attribute is set, the method is called with the argument tuple that would normally be returned and the result of the method is returned instead.

Note: The DB API definition does not define this attribute.

Cursor.**scroll**(*value=0*, *mode="relative"*)

Scroll the cursor in the result set to a new position according to the mode.

If mode is "relative" (the default value), the value is taken as an offset to the current position in the result set. If set to "absolute", value states an absolute target position. If set to "first", the cursor is positioned at the first row and if set to "last", the cursor is set to the last row in the result set.

An error is raised if the mode is "relative" or "absolute" and the scroll operation would position the cursor outside of the result set.

New in version 5.3.

Note: This method is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**scrollable**

This read-write boolean attribute specifies whether the cursor can be scrolled or not. By default, cursors are not scrollable, as the server resources and response times are greater than nonscrollable cursors. This attribute is checked and the corresponding mode set in Oracle when calling the method *execute()*.

New in version 5.3.

Note: The DB API definition does not define this attribute.

Cursor.**setinputsizes**(**args*, ***keywordArgs*)

This can be used before a call to *execute()*, *callfunc()* or *callproc()* to predefine memory areas for the operation's parameters. Each parameter should be a type object corresponding to the input that will be used or it should be an integer specifying the maximum length of a string parameter. Use keyword arguments when binding by name and positional arguments when binding by position. The singleton None can be used as a parameter when using positional arguments to indicate that no space should be reserved for that position.

Note: If you plan to use *callfunc()* then be aware that the first argument in the list refers to the return value of the function.

Cursor.**setoutputsize**(*size*[, *column*])

This method does nothing and is retained solely for compatibility with the DB API. The module automatically allocates as much space as needed to fetch LONG and LONG RAW columns (or CLOB as string and BLOB as bytes).

Cursor.**statement**

This read-only attribute provides the string object that was previously prepared with *prepare()* or executed with *execute()*.

Note: The DB API definition does not define this attribute.

`Cursor.`**`var`**(*dataType*[, *size*, *arraysize*, *inconverter*, *outconverter*, *typename*])

Create a variable associated with the cursor of the given type and characteristics and return a *variable object*. If the size is not specified and the type is a string or binary, 4000 bytes is allocated; if the size is not specified and the type is a long string or long binary, 128KB is allocated. If the arraysize is not specified, the bind array size (usually 1) is used. The inconverter and outconverter specify methods used for converting values to/from the database. More information can be found in the section on variable objects.

To create an empty SQL object variable, specify the typename parameter.

This method was designed for use with PL/SQL in/out variables where the length or type cannot be determined automatically from the Python object passed in or for use in input and output type handlers defined on cursors or connections.

Note: The DB API definition does not define this method.

CHAPTER 5

Variable Objects

Note: The DB API definition does not define this object.

Variable.**actualElements**
: This read-only attribute returns the actual number of elements in the variable. This corresponds to the number of elements in a PL/SQL index-by table for variables that are created using the method *Cursor.arrayvar()*. For output variables in a DML returning statement this value corresponds to the number of rows returned. For all other variables this value will be identical to the attribute *numElements*.

Variable.**bufferSize**
: This read-only attribute returns the size of the buffer allocated for each element in bytes.

Variable.**getvalue**([*pos=0*])
: Return the value at the given position in the variable.

Variable.**inconverter**
: This read-write attribute specifies the method used to convert data from Python to the Oracle database. The method signature is converter(value) and the expected return value is the value to bind to the database. If this attribute is None, the value is bound directly without any conversion.

Variable.**numElements**
: This read-only attribute returns the number of elements allocated in an array, or the number of scalar items that can be fetched into the variable or bound to the variable.

Variable.**outconverter**
: This read-write attribute specifies the method used to convert data from from the Oracle to Python. The method signature is converter(value) and the expected return value is the value to return to Python. If this attribute is None, the value is returned directly without any conversion.

Variable.**setvalue**(*pos*, *value*)
: Set the value at the given position in the variable.

Variable.**size**
: This read-only attribute returns the size of the variable. For strings this value is the size in characters. For all others, this is same value as the attribute bufferSize.

`Variable.`**`type`**
: This read-only attribute returns the type of the variable for those variables that bind Oracle objects (it is not present for any other type of variable).

`Variable.`**`values`**
: This read-only attribute returns a copy of the value of all actual positions in the variable as a list. The number of items in the list will correspond to the value of the *`actualElements`* attribute.

CHAPTER 6

SessionPool Object

Note: This object is an extension to the DB API.

SessionPool.**acquire**(*user=None*, *password=None*, *cclass=None*, *purity=cx_Oracle.ATTR_PURITY_DEFAULT*, *tag=None*, *matchanytag=False*)

Acquire a connection from the session pool and return a *connection object*.

The user and password arguments may not be specified if the pool is homogeneous. In that case an exception will be raised.

The cclass argument, if specified, should be a string corresponding to the connection class for database resident connection pooling (DRCP).

The purity argument is expected to be one of *ATTR_PURITY_NEW*, *ATTR_PURITY_SELF*, or *ATTR_PURITY_DEFAULT*.

The tag argument, if specified, is expected to be a string and will limit the sessions that can be returned from a session pool unless the matchanytag argument is set to True. In that case sessions with the specified tag will be preferred over others, but if no such sessions are available a session with a different tag may be returned instead. In any case, untagged sessions will always be returned if no sessions with the specified tag are available. Sessions are tagged when they are *released* back to the pool.

SessionPool.**busy**

This read-only attribute returns the number of sessions currently acquired.

SessionPool.**drop**(*connection*)

Drop the connection from the pool which is useful if the connection is no longer usable (such as when the session is killed).

SessionPool.**dsn**

This read-only attribute returns the TNS entry of the database to which a connection has been established.

SessionPool.**homogeneous**

This read-write boolean attribute indicates whether the pool is considered homogeneous or not. If the pool is not homogeneous different authentication can be used for each connection acquired from the pool.

SessionPool.**increment**
: This read-only attribute returns the number of sessions that will be established when additional sessions need to be created.

SessionPool.**max**
: This read-only attribute returns the maximum number of sessions that the session pool can control.

SessionPool.**max_lifetime_session**
: This read-write attribute returns the lifetime (in seconds) for all of the sessions in the pool. Sessions in the pool are terminated when they have reached their lifetime. If timeout is also set, the session will be terminated if either the idle timeout happens or the max lifetime setting is exceeded. This attribute is only available in Oracle Database 12.1.

  New in version 5.3.

SessionPool.**min**
: This read-only attribute returns the number of sessions with which the session pool was created and the minimum number of sessions that will be controlled by the session pool.

SessionPool.**name**
: This read-only attribute returns the name assigned to the session pool by Oracle.

SessionPool.**opened**
: This read-only attribute returns the number of sessions currently opened by the session pool.

SessionPool.**release**(*connection*, *tag=None*)
: Release the connection back to the pool. This will be done automatically as well if the connection object is garbage collected.

  If a tag is specified, the session will be tagged (or retagged) with the specified value before being returned to the pool.

SessionPool.**stmtcachesize**
: This read-write attribute specifies the size of the statement cache that will be used as the starting point for any connections that are created by the session pool. Once created, the connection's statement cache size can only be changed by setting the stmtcachesize attribute on the connection itself.

  New in version 6.0.

SessionPool.**timeout**
: This read-write attribute indicates the time (in seconds) after which idle sessions will be terminated in order to maintain an optimum number of open sessions.

SessionPool.**tnsentry**
: This read-only attribute returns the TNS entry of the database to which a connection has been established.

SessionPool.**username**
: This read-only attribute returns the name of the user which established the connection to the database.

CHAPTER 7

Subscription Object

Note: This object is an extension the DB API.

`Subscription.`**`callback`**
: This read-only attribute returns the callback that was registered when the subscription was created.

`Subscription.`**`connection`**
: This read-only attribute returns the connection that was used to register the subscription when it was created.

`Subscription.`**`id`**
: This read-only attribute returns the registration ID returned by the database when the subscription was created.

`Subscription.`**`namespace`**
: This read-only attribute returns the namespace used to register the subscription when it was created.

`Subscription.`**`operations`**
: This read-only attribute returns the operations that will send notifications for each table or query that is registered using this subscription.

`Subscription.`**`port`**
: This read-only attribute returns the port used for callback notifications from the database server. If not set during construction, this value is zero.

`Subscription.`**`protocol`**
: This read-only attribute returns the protocol used to register the subscription when it was created.

`Subscription.`**`qos`**
: This read-only attribute returns the quality of service flags used to register the subscription when it was created.

`Subscription.`**`registerquery`**(*statement*[, *args*])
: Register the query for subsequent notification when tables referenced by the query are changed. This behaves similarly to cursor.execute() but only queries are permitted and the arguments must be a sequence or dictionary. If the qos parameter included the flag cx_Oracle.SUBSCR_QOS_QUERY when the subscription was created, then the ID for the registered query is returned; otherwise, None is returned.

Subscription.**timeout**
: This read-only attribute returns the timeout (in seconds) that was specified when the subscription was created. A value of 0 indicates that there is no timeout.

Message Objects

Note: This object is created internally when notification is received and passed to the callback procedure specified when a subscription is created.

Message.**dbname**
: This read-only attribute returns the name of the database that generated the notification.

Message.**queries**
: This read-only attribute returns a list of message query objects that give information about query result sets changed for this notification. This attribute will be None if the qos parameter did not include the flag *SUBSCR_QOS_QUERY* when the subscription was created.

Message.**subscription**
: This read-only attribute returns the subscription object for which this notification was generated.

Message.**tables**
: This read-only attribute returns a list of message table objects that give information about the tables changed for this notification. This attribute will be None if the qos parameter included the flag *SUBSCR_QOS_QUERY* when the subscription was created.

Message.**type**
: This read-only attribute returns the type of message that has been sent. See the constants section on event types for additional information.

Message Table Objects

Note: This object is created internally for each table changed when notification is received and is found in the tables attribute of message objects, and the tables attribute of message query objects.

MessageTable.**name**
: This read-only attribute returns the name of the table that was changed.

MessageTable.**operation**
: This read-only attribute returns the operation that took place on the table that was changed.

MessageTable.**rows**
: This read-only attribute returns a list of message row objects that give information about the rows changed on the table. This value is only filled in if the qos parameter to the *Connection.subscribe()* method included the flag *SUBSCR_QOS_ROWIDS*.

Message Row Objects

Note: This object is created internally for each row changed on a table when notification is received and is found in the rows attribute of message table objects.

`MessageRow.`**`operation`**
This read-only attribute returns the operation that took place on the row that was changed.

`MessageRow.`**`rowid`**
This read-only attribute returns the rowid of the row that was changed.

Message Query Objects

Note: This object is created internally for each query result set changed when notification is received and is found in the queries attribute of message objects.

`MessageQuery.`**`id`**
This read-only attribute returns the query id of the query for which the result set changed. The value will match the value returned by Subscription.registerquery when the related query was registered.

`MessageQuery.`**`operation`**
This read-only attribute returns the operation that took place on the query result set that was changed. Valid values for this attribute are *EVENT_DEREG* and *EVENT_QUERYCHANGE*.

`MessageQuery.`**`tables`**
This read-only attribute returns a list of message table objects that give information about the table changes that caused the query result set to change for this notification.

CHAPTER 8

LOB Objects

Note: This object is an extension the DB API. It is returned whenever Oracle `CLOB`, `BLOB` and `BFILE` columns are fetched.

`LOB.`**`close`**`()`
Close the LOB. Call this when writing is completed so that the indexes associated with the LOB can be updated – but only if *`open()`* was called first.

`LOB.`**`fileexists`**`()`
Return a boolean indicating if the file referenced by the BFILE type LOB exists.

`LOB.`**`getchunksize`**`()`
Return the chunk size for the internal LOB. Reading and writing to the LOB in chunks of multiples of this size will improve performance.

`LOB.`**`getfilename`**`()`
Return a two-tuple consisting of the directory alias and file name for a BFILE type LOB.

`LOB.`**`isopen`**`()`
Return a boolean indicating if the LOB has been opened using the method *`open()`*.

`LOB.`**`open`**`()`
Open the LOB for writing. This will improve performance when writing to a LOB in chunks and there are functional or extensible indexes associated with the LOB. If this method is not called, each write will perform an open internally followed by a close after the write has been completed.

`LOB.`**`read`**`(`[*offset=1*[, *amount*]]`)`
Return a portion (or all) of the data in the LOB object. Note that the amount and offset are in bytes for BLOB and BFILE type LOBs and in UCS-2 code points for CLOB and NCLOB type LOBs. UCS-2 code points are equivalent to characters for all but supplemental characters. If supplemental characters are in the LOB, the offset and amount will have to be chosen carefully to avoid splitting a character.

`LOB.`**`setfilename`**`(`*dirAlias*, *name*`)`
Set the directory alias and name of the BFILE type LOB.

LOB.**size**()
: Returns the size of the data in the LOB object. For BLOB and BFILE type LOBs this is the number of bytes. For CLOB and NCLOB type LOBs this is the number of UCS-2 code points. UCS-2 code points are equivalent to characters for all but supplemental characters.

LOB.**trim**([*newSize=0*])
: Trim the LOB to the new size.

LOB.**write**(*data*[, *offset=1*])
: Write the data to the LOB object at the given offset. The offset is in bytes for BLOB type LOBs and in UCS-2 code points for CLOB and NCLOB type LOBs. UCS-2 code points are equivalent to characters for all but supplemental characters. If supplemental characters are in the LOB, the offset will have to be chosen carefully to avoid splitting a character. Note that if you want to make the LOB value smaller, you must use the *trim()* function.

CHAPTER 9

Object Type Objects

Note: This object is an extension to the DB API. It is returned by the *Connection.gettype()* call and is available as the *Variable.type* for variables containing Oracle objects.

ObjectType([*sequence*])
The object type may be called directly and serves as an alternative way of calling *newobject()*.

ObjectType.**attributes**
This read-only attribute returns a list of the attributes that make up the object type. Each attribute has a name attribute on it.

ObjectType.**iscollection**
This read-only attribute returns a boolean indicating if the object type refers to a collection or not.

ObjectType.**name**
This read-only attribute returns the name of the type.

ObjectType.**newobject**([*sequence*])
Return a new Oracle object of the given type. This object can then be modified by setting its attributes and then bound to a cursor for interaction with Oracle. If the object type refers to a collection, a sequence may be passed and the collection will be initialized with the items in that sequence.

ObjectType.**schema**
This read-only attribute returns the name of the schema that owns the type.

Object Objects

Note: This object is an extension to the DB API. It is returned by the *ObjectType.newobject()* call and can be bound to variables of type *OBJECT*. Attributes can be retrieved and set directly.

Object.**append**(*element*)
: Append an element to the collection object. If no elements exist in the collection, this creates an element at index 0; otherwise, it creates an element immediately following the highest index available in the collection.

Object.**aslist**()
: Return a list of each of the collection's elements in index order.

Object.**copy**()
: Create a copy of the object and return it.

Object.**delete**(*index*)
: Delete the element at the specified index of the collection. If the element does not exist or is otherwise invalid, an error is raised. Note that the indices of the remaining elements in the collection are not changed. In other words, the delete operation creates holes in the collection.

Object.**exists**(*index*)
: Return True or False indicating if an element exists in the collection at the specified index.

Object.**extend**(*sequence*)
: Append all of the elements in the sequence to the collection. This is the equivalent of performing *append()* for each element found in the sequence.

Object.**first**()
: Return the index of the first element in the collection. If the collection is empty, None is returned.

Object.**getelement**(*index*)
: Return the element at the specified index of the collection. If no element exists at that index, an exception is raised.

Object.**last**()
: Return the index of the last element in the collection. If the collection is empty, None is returned.

Object.**next**(*index*)
: Return the index of the next element in the collection following the specified index. If there are no elements in the collection following the specified index, None is returned.

Object.**prev**(*index*)
: Return the index of the element in the collection preceding the specified index. If there are no elements in the collection preceding the specified index, None is returned.

Object.**setelement**(*index*, *value*)
: Set the value in the collection at the specified index to the given value.

Object.**size**()
: Return the number of elements in the collection.

Object.**trim**(*num*)
: Remove the specified number of elements from the end of the collection.

CHAPTER 10

Advanced Queuing

Dequeue Options

Note: This object is an extension to the DB API. It is returned by the *Connection.deqoptions()* call and is used in calls to *Connection.deq()*.

DeqOptions.**condition**
: This attribute specifies a boolean expression similar to the where clause of a SQL query. The boolean expression can include conditions on message properties, user data properties and PL/SQL or SQL functions. The default is to have no condition specified.

DeqOptions.**consumername**
: This attribute specifies the name of the consumer. Only messages matching the consumer name will be accessed. If the queue is not set up for multiple consumers this attribute should not be set. The default is to have no consumer name specified.

DeqOptions.**correlation**
: This attribute specifies the correlation identifier of the message to be dequeued. Special pattern-matching characters, such as the percent sign (%) and the underscore (_), can be used. If multiple messages satisfy the pattern, the order of dequeuing is indeterminate. The default is to have no correlation specified.

DeqOptions.**deliverymode**
: This write-only attribute specifies what types of messages should be dequeued. It should be one of the values *MSG_PERSISTENT* (default), *MSG_BUFFERED* or *MSG_PERSISTENT_OR_BUFFERED*.

DeqOptions.**mode**
: This attribute specifies the locking behaviour associated with the dequeue operation. It should be one of the values *DEQ_BROWSE*, *DEQ_LOCKED*, *DEQ_REMOVE* (default), or *DEQ_REMOVE_NODATA*.

DeqOptions.**msgid**
: This attribute specifies the identifier of the message to be dequeued. The default is to have no message identifier specified.

DeqOptions.**navigation**
: This attribute specifies the position of the message that is retrieved. It should be one of the values *DEQ_FIRST_MSG*, *DEQ_NEXT_MSG* (default), or *DEQ_NEXT_TRANSACTION*.

DeqOptions.**transformation**
: This attribute specifies the name of the transformation that must be applied after the message is dequeued from the database but before it is returned to the calling application. The transformation must be created using dbms_transform. The default is to have no transformation specified.

DeqOptions.**visibility**
: This attribute specifies the transactional behavior of the enqueue request. It should be one of the values *ENQ_ON_COMMIT* (default) or *ENQ_IMMEDIATE*. This attribute is ignored when using the *DEQ_BROWSE* mode.

DeqOptions.**wait**
: This attribute specifies the time to wait, in seconds, for a message matching the search criteria to become available for dequeuing. One of the values *DEQ_NO_WAIT* or *DEQ_WAIT_FOREVER* can also be used. The default is *DEQ_WAIT_FOREVER*.

Enqueue Options

Note: This object is an extension to the DB API. It is returned by the *Connection.enqoptions()* call and is used in calls to *Connection.enq()*.

EnqOptions.**deliverymode**
: This write-only attribute specifies what type of messages should be enqueued. It should be one of the values *MSG_PERSISTENT* (default) or *MSG_BUFFERED*.

EnqOptions.**transformation**
: This attribute specifies the name of the transformation that must be applied before the message is enqueued into the database. The transformation must be created using dbms_transform. The default is to have no transformation specified.

EnqOptions.**visibility**
: This attribute specifies the transactional behavior of the enqueue request. It should be one of the values *ENQ_ON_COMMIT* (default) or *ENQ_IMMEDIATE*.

Message Properties

Note: This object is an extension to the DB API. It is returned by the *Connection.msgproperties()* call and is used in calls to *Connection.deq()* and *Connection.enq()*.

MessageProperties.**attempts**
: This read-only attribute specifies the number of attempts that have been made to dequeue the message.

MessageProperties.**correlation**
: This attribute specifies the correlation used when the message was enqueued.

MessageProperties.**delay**
: This attribute specifies the number of seconds to delay an enqueued message. Any integer is acceptable but the constant *MSG_NO_DELAY* can also be used indicating that the message is available for immediate dequeuing.

MessageProperties.**deliverymode**
: This read-only attribute specifies the type of message that was dequeued. It will be one of the values *MSG_PERSISTENT* or *MSG_BUFFERED*.

MessageProperties.**enqtime**
: This read-only attribute specifies the time that the message was enqueued.

MessageProperties.**exceptionq**
: This attribute specifies the name of the queue to which the message is moved if it cannot be processed successfully. Messages are moved if the number of unsuccessful dequeue attempts has exceeded the maximum number of retries or if the message has expired. All messages in the exception queue are in the *MSG_EXPIRED* state. The default value is the name of the exception queue associated with the queue table.

MessageProperties.**expiration**
: This attribute specifies, in seconds, how long the message is available for dequeuing. This attribute is an offset from the delay attribute. Expiration processing requires the queue monitor to be running. Any integer is accepted but the constant *MSG_NO_EXPIRATION* can also be used indicating that the message never expires.

MessageProperties.**msgid**
: This attribute specifies the id of the message in the last queue that generated this message.

MessageProperties.**priority**
: This attribute specifies the priority of the message. A smaller number indicates a higher priority. The priority can be any integer, including negative numbers. The default value is zero.

MessageProperties.**state**
: This read-only attribute specifies the state of the message at the time of the dequeue. It will be one of the values *MSG_WAITING*, *MSG_READY*, *MSG_PROCESSED* or *MSG_EXPIRED*.

CHAPTER 11

cx_Oracle Release Notes

6.x releases

Version 6.0 (August 2017)

1. Update to ODPI-C 2.0.
    - Prevent closing the connection when there are any open statements or LOBs and add new error "DPI-1054: connection cannot be closed when open statements or LOBs exist" when this situation is detected; this is needed to prevent crashes under certain conditions when statements or LOBs are being acted upon while at the same time (in another thread) a connection is being closed; it also prevents leaks of statements and LOBs when a connection is returned to a session pool.
    - On platforms other than Windows, if the regular method for loading the Oracle Client libraries fails, try using $ORACLE_HOME/lib/libclntsh.so (ODPI-C issue 20).
    - Use the environment variable DPI_DEBUG_LEVEL at runtime, not compile time.
    - Added support for DPI_DEBUG_LEVEL_ERRORS (reports errors and has the value 8) and DPI_DEBUG_LEVEL_SQL (reports prepared SQL statement text and has the value 16) in order to further improve the ability to debug issues.
    - Correct processing of `Cursor.scroll()` in some circumstances.
2. Delay initialization of the ODPI-C library until the first standalone connection or session pool is created so that manipulation of the environment variable NLS_LANG can be performed after the module has been imported; this also has the added benefit of reducing the number of errors that can take place when the module is imported.
3. Prevent binding of null values from generating the exception "ORA-24816: Expanded non LONG bind data supplied after actual LONG or LOB column" in certain circumstances (issue 50).
4. Added information on how to run the test suite (issue 33).
5. Documentation improvements.

Version 6.0 rc 2 (July 2017)

1. Update to ODPI-C rc 2.
   - Provide improved error message when OCI environment cannot be created, such as when the oraaccess.xml file cannot be processed properly.
   - On Windows, convert system message to Unicode first, then to UTF-8; otherwise, the error message returned could be in a mix of encodings (issue 40).
   - Corrected support for binding decimal values in object attribute values and collection element values.
   - Corrected support for binding PL/SQL boolean values to PL/SQL procedures with Oracle client 11.2.
2. Define exception classes on the connection object in addition to at module scope in order to simplify error handling in multi-connection environments, as specified in the Python DB API.
3. Ensure the correct encoding is used for setting variable values.
4. Corrected handling of CLOB/NCLOB when using different encodings.
5. Corrected handling of TIMESTAMP WITH TIME ZONE attributes on objects.
6. Ensure that the array position passed to var.getvalue() does not exceed the number of elements allocated in the array.
7. Reworked test suite and samples so that they are independent of each other and so that the SQL scripts used to create/drop schemas are easily adjusted to use different schema names, if desired.
8. Updated DB API test suite stub to support Python 3.
9. Added additional test cases and samples.
10. Documentation improvements.

Version 6.0 rc 1 (June 2017)

1. Update to ODPI-C rc 1.
2. The method *Cursor.setoutputsize()* no longer needs to do anything, since ODPI-C automatically manages buffer sizes of LONG and LONG RAW columns.
3. Handle case when both precision and scale are zero, as occurs when retrieving numeric expressions (issue 34).
4. OCI requires that both encoding and nencoding have values or that both encoding and encoding do not have values. These parameters are used in functions *cx_Oracle.connect()* and *cx_Oracle.SessionPool()*. The missing value is set to its default value if one of the values is set and the other is not (issue 36).
5. Permit use of both string and unicode for Python 2.7 for creating session pools and for changing passwords (issue 23).
6. Corrected handling of BFILE LOBs.
7. Add script for dropping test schemas.
8. Documentation improvements.

Version 6.0 beta 2 (May 2017)

1. Added support for getting/setting attributes of objects or element values in collections that contain LOBs, BINARY_FLOAT values, BINARY_DOUBLE values and NCHAR and NVARCHAR2 values. The error message for any types that are not supported has been improved as well.
2. Enable temporary LOB caching in order to avoid disk I/O as suggested.
3. Added support for setting the debug level in ODPI-C, if desirable, by setting environment variable DPI_DEBUG_LEVEL prior to building cx_Oracle.
4. Correct processing of strings in *Cursor.executemany()* when a larger string is found after a shorter string in the list of data bound to the statement.
5. Correct handling of long Python integers that cannot fit inside a 64-bit C integer (issue 18).
6. Correct creation of pool using external authentication.
7. Handle edge case when an odd number of zeroes trail the decimal point in a value that is effectively zero (issue 22).
8. Prevent segfault under load when the attempt to create an error fails.
9. Eliminate resource leak when a standalone connection or pool is freed.
10. Correct typo.
11. Correct handling of REF cursors when the array size is manipulated.
12. Prevent attempts from binding the cursor being executed to itself.
13. Correct reference count handling of parameters when creating a cursor.
14. Correct determination of the names of the bind variables in prepared SQL statements (which behaves a little differently from PL/SQL statements).

Version 6.0 beta 1 (April 2017)

1. Simplify building cx_Oracle considerably by use of ODPI-C. This means that cx_Oracle can now be built without Oracle Client header files or libraries and that at runtime cx_Oracle can adapt to Oracle Client 11.2, 12.1 or 12.2 libraries without needing to be rebuilt. This also means that wheels can now be produced and installed via pip.
2. Added attribute *SessionPool.stmtcachesize* to support getting and setting the default statement cache size for connections in the pool.
3. Added attribute *Connection.dbop* to support setting the database operation that is to be monitored.
4. Added attribute *Connection.handle* to facilitate testing the creation of a connection using a OCI service context handle.
5. Added parameters tag and matchanytag to the *cx_Oracle.connect()* and *SessionPool.acquire()* methods and added parameters tag and retag to the *SessionPool.release()* method in order to support session tagging.
6. Added parameter edition to the *cx_Oracle.SessionPool()* method.
7. Added support for universal rowids.
8. Added support for DML returning of multiple rows.
9. Added attributes *Variable.actualElements* and *Variable.values* to variables.

10. Added parameters region, sharding_key and super_sharding_key to the *cx_Oracle.makedsn()* method to support connecting to a sharded database (new in Oracle Database 12.2).
11. Added support for smallint and float data types in Oracle objects, as requested.
12. An exception is no longer raised when a collection is empty for methods *Object.first()* and *Object.last()*. Instead, the value None is returned to be consistent with the methods *Object.next()* and *Object.prev()*.
13. If the environment variables NLS_LANG and NLS_NCHAR are being used, they must be set before the module is imported. Using the encoding and nencoding parameters to the *cx_Oracle.connect()* and *cx_Oracle.SessionPool()* methods is a simpler alternative to setting these environment variables.
14. Removed restriction on fetching LOBs across round trips to the database (eliminates error "LOB variable no longer valid after subsequent fetch").
15. Removed requirement for specifying a maximum size when fetching LONG or LONG raw columns. This also allows CLOB, NCLOB, BLOB and BFILE columns to be fetched as strings or bytes without needing to specify a maximum size.
16. Dropped deprecated parameter twophase from the *cx_Oracle.connect()* method. Applications should set the *Connection.internal_name* and *Connection.external_name* attributes instead to a value appropriate to the application.
17. Dropped deprecated parameters action, module and clientinfo from the *cx_Oracle.connect()* method. The appcontext parameter should be used instead as shown in this sample.
18. Dropped deprecated attribute numbersAsString from *cursor objects*. Use an output type handler instead as shown in this sample.
19. Dropped deprecated attributes cqqos and rowids from *subscription objects*. Use the qos attribute instead as shown in this sample.
20. Dropped deprecated parameters cqqos and rowids from the *Connection.subscribe()* method. Use the qos parameter instead as shown in this sample.

5.x releases

Version 5.3 (March 2017)

1. Added support for Python 3.6.
2. Dropped support for Python versions earlier than 2.6.
3. Dropped support for Oracle clients earlier than 11.2.
4. Added support for *fetching implicit results* (available in Oracle 12.1)
5. Added support for *Transaction Guard* (available in Oracle 12.1).
6. Added support for setting the *maximum lifetime* of pool connections (available in Oracle 12.1).
7. Added support for large row counts (larger than 2 ** 32, available in Oracle 12.1)
8. Added support for *advanced queuing*.
9. Added support for *scrollable cursors*.
10. Added support for *edition based redefinition*.
11. Added support for *creating*, modifying and binding user defined types and collections.

12. Added support for creating, modifying and binding PL/SQL records and collections (available in Oracle 12.1).
13. Added support for binding *native integers*.
14. Enabled statement caching.
15. Removed deprecated variable attributes maxlength and allocelems.
16. Corrected support for setting the encoding and nencoding parameters when *creating a connection* and added support for setting these when creating a session pool. These can now be used instead of setting the environment variables NLS_LANG and NLS_NCHAR.
17. Use None instead of 0 for items in the *Cursor.description* attribute that do not have any validity.
18. Changed driver name to match informal driver name standard used by Oracle for other drivers.
19. Add check for maximum of 10,000 arguments when calling a stored procedure or function in order to prevent a possible improper memory access from taking place.
20. Removed -mno-cygwin compile flag since it is no longer used in newer versions of the gcc compiler for Cygwin.
21. Simplified test suite by combining Python 2 and 3 scripts into one script and separated out 12.1 features into a single script.
22. Updated samples to use code that works on both Python 2 and 3
23. Added support for pickling/unpickling error objects (Issue #23)
24. Dropped support for callbacks on OCI functions.
25. Removed deprecated types UNICODE, FIXED_UNICODE and LONG_UNICODE (use NCHAR, FIXED_NCHAR and LONG_NCHAR instead).
26. Increased default array size to 100 (from 50) to match other drivers.
27. Added support for setting the *internal_name* and *external_name* on the connection directly. The use of the twophase argument is now deprecated. Applications should set the internal_name and external_name attributes directly to a value appropriate to the application.
28. Added support for using application context when *creating a connection*. This should be used in preference to the module, action and clientinfo arguments which are now deprecated.
29. Reworked database change notification and continuous query notification to more closely align with the PL/SQL implementation and prepare for sending notifications for AQ messages. The following changes were made:
    - added constant *SUBSCR_QOS_BEST_EFFORT* to replace deprecated constant SUBSCR_CQ_QOS_BEST_EFFORT
    - added constant *SUBSCR_QOS_QUERY* to replace deprecated constant SUBSCR_CQ_QOS_QUERY
    - added constant *SUBSCR_QOS_DEREG_NFY* to replace deprecated constant SUBSCR_QOS_PURGE_ON_NTFN
    - added constant *SUBSCR_QOS_ROWIDS* to replace argument rowids for method *Connection.subscribe()*
    - deprecated argument cqqos for method *Connection.subscribe()*. The qos argument should be used instead.
    - dropped constants SUBSCR_CQ_QOS_CLQRYCACHE, SUBSCR_QOS_HAREG, SUBSCR_QOS_MULTICBK, SUBSCR_QOS_PAYLOAD, SUBSCR_QOS_REPLICATE, and SUBSCR_QOS_SECURE since they were never actually used
30. Deprecated use of the numbersAsStrings attribute on cursors. An output type handler should be used instead.

Version 5.2.1 (January 2016)

1. Added support for Python 3.5.
2. Removed password attribute from connection and session pool objects in order to promote best security practices (if stored in RAM in cleartext it can be read in process dumps, for example). For those who would like to retain this feature, a subclass of Connection could be used to store the password.
3. Added optional parameter externalauth to SessionPool() which enables wallet based or other external authentication mechanisms to be used.
4. Use the national character set encoding when required (when char set form is SQLCS_NCHAR); otherwise, the wrong encoding would be used if the environment variable NLS_NCHAR is set.
5. Added support for binding boolean values to PL/SQL blocks and stored procedures (available in Oracle 12.1).

Version 5.2 (June 2015)

1. Added support for strings up to 32k characters (new in Oracle 12c).
2. Added support for getting array DML row counts (new in Oracle 12c).
3. Added support for fetching batch errors.
4. Added support for LOB values larger than 4 GB.
5. Added support for connections as SYSASM.
6. Added support for building without any configuration changes to the machine when using instant client RPMs on Linux.
7. Added types NCHAR, FIXED_NCHAR and LONG_NCHAR to replace the types UNICODE, FIXED_UNICODE and LONG_UNICODE (which are now deprecated). These types are available in Python 3 as well so they can be used to specify the use of NCHAR type fields when binding or using setinputsizes().
8. Fixed binding of booleans in Python 3.x.
9. Test suite now sets NLS_LANG if not already set.
10. Enhanced documentation for connection.action attribute and added note on cursor.parse() method to make clear that DDL statements are executed when parsed.
11. Removed remaining remnants of support Oracle 9i.
12. Added __version__ attribute to conform with PEP 396.
13. Ensure that sessions are released to the pool when calling connection.close() (Issue #2)
14. Fixed handling of datetime intervals (Issue #7)

Version 5.1.3 (May 2014)

1. Added support for Oracle 12c.
2. Added support for Python 3.4.
3. Added support for query result set change notification. Thanks to Glen Walker for the patch.
4. Ensure that in Python 3.x that NCHAR and NVARCHAR2 and NCLOB columns are retrieved properly without conversion issues. Thanks to Joakim Andersson for pointing out the issue and the possible solution.

5. Fix bug when an exception is caught and then another exception is raised while handling that exception in Python 3.x. Thanks to Boris Dzuba for pointing out the issue and providing a test case.
6. Enhance performance returning integers between 10 and 18 digits on 64-bit platforms that support it. Thanks for Shai Berger for the initial patch.
7. Fixed two memory leaks.
8. Fix to stop current_schema from throwing a MemoryError on 64-bit platforms on occasion. Thanks to Andrew Horton for the fix.
9. Class name of cursors changed to real name cx_Oracle.Cursor.

Version 5.1.2 (July 2012)

1. Added support for LONG_UNICODE which is a type used to handle long unicode strings. These are not explicitly supported in Oracle but can be used to bind to NCLOB, for example, without getting the error "unimplemented or unreasonable conversion requested".
2. Set the row number in a cursor when executing PL/SQL blocks as requested by Robert Ritchie.
3. Added support for setting the module, action and client_info attributes during connection so that logon triggers will see the supplied values, as requested by Rodney Barnett.

Version 5.1.1 (October 2011)

1. Simplify management of threads for callbacks performed by database change notification and eliminate a crash that occurred under high load in certain situations. Thanks to Calvin S. for noting the issue and suggesting a solution and testing the patch.
2. Force server detach on close so that the connection is completely closed and not just the session as before.
3. Force use of OCI_UTF16ID for NCLOBs as using the default character set would result in ORA-03127 with Oracle 11.2.0.2 and UTF8 character set.
4. Avoid attempting to clear temporary LOBs a second time when destroying the variable as in certain situations this results in spurious errors.
5. Added additional parameter service_name to makedsn() which can be used to use the service_name rather than the SID in the DSN string that is generated.
6. Fix cursor description in test suite to take into account the number of bytes per character.
7. Added tests for NCLOBS to the test suite.
8. Removed redundant code in setup.py for calculating the library path.

Version 5.1 (March 2011)

1. Remove support for UNICODE mode and permit Unicode to be passed through in everywhere a string may be passed in. This means that strings will be passed through to Oracle using the value of the NLS_LANG environment variable in Python 3.x as well. Doing this eliminated a bunch of problems that were discovered by using UNICODE mode and also removed an unnecessary restriction in Python 2.x that Unicode could not be used in connect strings or SQL statements, for example.
2. Added support for creating an empty object variable via a named type, the first step to adding full object support.
3. Added support for Python 3.2.

4. Account for lib64 used on x86_64 systems. Thanks to Alex Wood for supplying the patch.
5. Clear up potential problems when calling cursor.close() ahead of the cursor being freed by going out of scope.
6. Avoid compilation difficulties on AIX5 as OCIPing does not appear to be available on that platform under Oracle 10g Release 2. Thanks to Pierre-Yves Fontaniere for the patch.
7. Free temporary LOBs prior to each fetch in order to avoid leaking them. Thanks to Uwe Hoffmann for the initial patch.

Version 5.0.4 (July 2010)

1. Added support for Python 2.7.
2. Added support for new parameter (port) for subscription() call which allows the client to specify the listening port for callback notifications from the database server. Thanks to Geoffrey Weber for the initial patch.
3. Fixed compilation under Oracle 9i.
4. Fixed a few error messages.

Version 5.0.3 (February 2010)

1. Added support for 64-bit Windows.
2. Added support for Python 3.1 and dropped support for Python 3.0.
3. Added support for keyword arguments in cursor.callproc() and cursor.callfunc().
4. Added documentation for the UNICODE and FIXED_UNICODE variable types.
5. Added extra link arguments required for Mac OS X as suggested by Jason Woodward.
6. Added additional error codes to the list of error codes that raise OperationalError rather than DatabaseError.
7. Fixed calculation of display size for strings with national database character sets that are not the default AL16UTF16.
8. Moved the resetting of the setinputsizes flag before the binding takes place so that if an error takes place and a new statement is prepared subsequently, spurious errors will not occur.
9. Fixed compilation with Oracle 10g Release 1.
10. Tweaked documentation based on feedback from a number of people.
11. Added support for running the test suite using "python setup.py test"
12. Added support for setting the CLIENT_IDENTIFIER value in the v$session table for connections.
13. Added exception when attempting to call executemany() with arrays which is not supported by the OCI.
14. Fixed bug when converting from decimal would result in OCI-22062 because the locale decimal point was not a period. Thanks to Amaury Forgeot d'Arc for the solution to this problem.

Version 5.0.2 (May 2009)

1. Fix creation of temporary NCLOB values and the writing of NCLOB values in non Unicode mode.
2. Re-enabled parsing of non select statements as requested by Roy Terrill.
3. Implemented a parse error offset as requested by Catherine Devlin.

4. Removed lib subdirectory when forcing RPATH now that the library directory is being calculated exactly in setup.py.
5. Added an additional cast in order to support compiling by Microsoft Visual C++ 2008 as requested by Marco de Paoli.
6. Added additional include directory to setup.py in order to support compiling by Microsoft Visual Studio was requested by Jason Coombs.
7. Fixed a few documentation issues.

Version 5.0.1 (February 2009)

1. Added support for database change notification available in Oracle 10g Release 2 and higher.
2. Fix bug where NCLOB data would be corrupted upon retrieval (non Unicode mode) or would generate exception ORA-24806 (LOB form mismatch). Oracle insists upon differentiating between CLOB and NCLOB no matter which character set is being used for retrieval.
3. Add new attributes size, bufferSize and numElements to variable objects, deprecating allocelems (replaced by numElements) and maxlength (replaced by bufferSize)
4. Avoid increasing memory allocation for strings when using variable width character sets and increasing the number of elements in a variable during executemany().
5. Tweaked code in order to ensure that cx_Oracle can compile with Python 3.0.1.

Version 5.0 (December 2008)

1. Added support for Python 3.0 with much help from Amaury Forgeot d'Arc.
2. Removed support for Python 2.3 and Oracle 8i.
3. Added support for full unicode mode in Python 2.x where all strings are passed in and returned as unicode (module must be built in this mode) rather than encoded strings
4. nchar and nvarchar columns now return unicode instead of encoded strings
5. Added support for an output type handler and/or an input type handler to be specified at the connection and cursor levels.
6. Added support for specifying both input and output converters for variables
7. Added support for specifying the array size of variables that are created using the cursor.var() method
8. Added support for events mode and database resident connection pooling (DRCP) in Oracle 11g.
9. Added support for changing the password during construction of a new connection object as well as after the connection object has been created
10. Added support for the interval day to second data type in Oracle, represented as datetime.timedelta objects in Python.
11. Added support for getting and setting the current_schema attribute for a session
12. Added support for proxy authentication in session pools as requested by Michael Wegrzynek (and thanks for the initial patch as well).
13. Modified connection.prepare() to return a boolean indicating if a transaction was actually prepared in order to avoid the error ORA-24756 (transaction does not exist).
14. Raise a cx_Oracle.Error instance rather than a string for column truncation errors as requested by Helge Tesdal.

15. Fixed handling of environment handles in session pools in order to allow session pools to fetch objects without exceptions taking place.

Older releases

Version 4.4.1 (October 2008)

1. Make the bind variables and fetch variables accessible although they need to be treated carefully since they are used internally; support added for forward compatibility with version 5.x.
2. Include the "cannot insert null value" in the list of errors that are treated as integrity errors as requested by Matt Boersma.
3. Use a cx_Oracle.Error instance rather than a string to hold the error when truncation (ORA-1406) takes place as requested by Helge Tesdal.
4. Added support for fixed char, old style varchar and timestamp attribute values in objects.
5. Tweaked setup.py to check for the Oracle version up front rather than during the build in order to produce more meaningful errors and simplify the code.
6. In setup.py added proper detection for the instant client on Mac OS X as recommended by Martijn Pieters.
7. In setup.py, avoided resetting the extraLinkArgs on Mac OS X as doing so prevents simple modification where desired as expressed by Christian Zagrodnick.
8. Added documentation on exception handling as requested by Andreas Mock, who also graciously provided an initial patch.
9. Modified documentation indicating that the password attribute on connection objects can be written.
10. Added documentation warning that parameters not passed in during subsequent executions of a statement will retain their original values as requested by Harald Armin Massa.
11. Added comments indicating that an Oracle client is required since so many people find this surprising.
12. Removed all references to Oracle 8i from the documentation and version 5.x will eliminate all vestiges of support for this version of the Oracle client.
13. Added additional link arguments for Cygwin as requested by Rob Gillen.

Version 4.4 (June 2008)

1. Fix setup.py to handle the Oracle instant client and Oracle XE on both Linux and Windows as pointed out by many. Thanks also to the many people who also provided patches.
2. Set the default array size to 50 instead of 1 as the DB API suggests because the performance difference is so drastic and many people have recommended that the default be changed.
3. Added Py_BEGIN_ALLOW_THREADS and Py_END_ALLOW_THREADS around each blocking call for LOBs as requested by Jason Conroy who also provided an initial patch and performed a number of tests that demonstrate the new code is much more responsive.
4. Add support for acquiring cursor.description after a parse.
5. Defer type assignment when performing executemany() until the last possible moment if the value being bound in is null as suggested by Dragos Dociu.

6. When dropping a connection from the pool, ignore any errors that occur during the rollback; unfortunately, Oracle decides to commit data even when dropping a connection from the pool instead of rolling it back so the attempt still has to be made.
7. Added support for setting CLIENT_DRIVER in V$SESSION_CONNECT_INFO in Oracle 11g and higher.
8. Use cx_Oracle.InterfaceError rather than the builtin RuntimeError when unable to create the Oracle environment object as requested by Luke Mewburn since the error is specific to Oracle and someone attempting to catch any exception cannot simply use cx_Oracle.Error.
9. Translated some error codes to OperationalError as requested by Matthew Harriger; translated if/elseif/else logic to switch statement to make it more readable and to allow for additional translation if desired.
10. Transformed documentation to new format using restructured text. Thanks to Waldemar Osuch for contributing the initial draft of the new documentation.
11. Allow the password to be overwritten by a new value as requested by Alex VanderWoude; this value is retained as a convenience to the user and not used by anything in the module; if changed externally it may be convenient to keep this copy up to date.
12. Cygwin is on Windows so should be treated in the same way as noted by Matthew Cahn.
13. Add support for using setuptools if so desired as requested by Shreya Bhatt.
14. Specify that the version of Oracle 10 that is now primarily used is 10.2, not 10.1.

Version 4.3.3 (October 2007)

1. Added method ping() on connections which can be used to test whether or not a connection is still active (available in Oracle 10g R2).
2. Added method cx_Oracle.clientversion() which returns a 5-tuple giving the version of the client that is in use (available in Oracle 10g R2).
3. Added methods startup() and shutdown() on connections which can be used to startup and shutdown databases (available in Oracle 10g R2).
4. Added support for Oracle 11g.
5. Added samples directory which contains a handful of scripts containing sample code for more advanced techniques. More will follow in future releases.
6. Prevent error "ORA-24333: zero iteration count" when calling executemany() with zero rows as requested by Andreas Mock.
7. Added methods __enter__() and __exit__() on connections to support using connections as context managers in Python 2.5 and higher. The context managed is the transaction state. Upon exit the transaction is either rolled back or committed depending on whether an exception took place or not.
8. Make the search for the lib32 and lib64 directories automatic for all platforms.
9. Tweak the setup configuration script to include all of the metadata and allow for building the module within another setup configuration script
10. Include the Oracle version in addition to the Python version in the build directories that are created and in the names of the binary packages that are created.
11. Remove unnecessary dependency on win32api to build module on Windows.

Version 4.3.2 (August 2007)

1. Added methods open(), close(), isopen() and getchunksize() in order to improve performance of reading/writing LOB values in chunks.
2. Fixed support for native doubles and floats in Oracle 10g; added new type NATIVE_FLOAT to allow specification of a variable of that specific type where desired. Thanks to D.R. Boxhoorn for pointing out the fact that this was not working properly when the arraysize was anything other than 1.
3. When calling connection.begin(), only create a new tranasction handle if one is not already associated with the connection. Thanks to Andreas Mock for discovering this and for Amaury Forgeot d'Arc for diagnosing the problem and pointing the way to a solution.
4. Added attribute cursor.rowfactory which allows a method to be called for each row that is returned; this is about 20% faster than calling the method in Python using the idiom [method(*r) for r in cursor].
5. Attempt to locate an Oracle installation by looking at the PATH if the environment variable ORACLE_HOME is not set; this is of primary use on Windows where this variable should not normally be set.
6. Added support for autocommit mode as requested by Ian Kelly.
7. Added support for connection.stmtcachesize which allows for both reading and writing the size of the statement cache size. This parameter can make a huge difference with the length of time taken to prepare statements. Added support for setting the statement tag when preparing a statement. Both of these were requested by Bjorn Sandberg who also provided an initial patch.
8. When copying the value of a variable, copy the return code as well.

Version 4.3.1 (April 2007)

1. Ensure that if the client buffer size exceeds 4000 bytes that the server buffer size does not as strings may only contain 4000 bytes; this allows handling of multibyte character sets on the server as well as the client.
2. Added support for using buffer objects to populate binary data and made the Binary() constructor the buffer type as requested by Ken Mason.
3. Fix potential crash when using full optimization with some compilers. Thanks to Aris Motas for noticing this and providing the initial patch and to Amaury Forgeot d'Arc for providing an even simpler solution.
4. Pass the correct charset form in to the write call in order to support writing to national character set LOB values properly. Thanks to Ian Kelly for noticing this discrepancy.

Version 4.3 (March 2007)

1. Added preliminary support for fetching Oracle objects (SQL types) as requested by Kristof Beyls (who kindly provided an initial patch). Additional work needs to be done to support binding and updating objects but the basic structure is now in place.
2. Added connection.maxBytesPerCharacter which indicates the maximum number of bytes each character can use; use this value to also determine the size of local buffers in order to handle discrepancies between the client character set and the server character set. Thanks to Andreas Mock for providing the initial patch and working with me to resolve this issue.
3. Added support for querying native floats in Oracle 10g as requested by Danny Boxhoorn.
4. Add support for temporary LOB variables created via PL/SQL instead of only directly by cx_Oracle; thanks to Henning von Bargen for discovering this problem.

5. Added support for specifying variable types using the builtin types int, float, str and datetime.date which allows for finer control of what type of Python object is returned from cursor.callfunc() for example.
6. Added support for passing booleans to callproc() and callfunc() as requested by Anana Aiyer.
7. Fixed support for 64-bit environments in Python 2.5.
8. Thanks to Filip Ballegeer and a number of his co-workers, an intermittent crash was tracked down; specifically, if a connection is closed, then the call to OCIStmtRelease() will free memory twice. Preventing the call when the connection is closed solves the problem.

Version 4.2.1 (September 2006)

1. Added additional type (NCLOB) to handle CLOBs that use the national character set as requested by Chris Dunscombe.
2. Added support for returning cursors from functions as requested by Daniel Steinmann.
3. Added support for getting/setting the "get" mode on session pools as requested by Anand Aiyer.
4. Added support for binding subclassed cursors.
5. Fixed binding of decimal objects with absolute values less than 0.1.

Version 4.2 (July 2006)

1. Added support for parsing an Oracle statement as requested by Patrick Blackwill.
2. Added support for BFILEs at the request of Matthew Cahn.
3. Added support for binding decimal.Decimal objects to cursors.
4. Added support for reading from NCLOBs as requested by Chris Dunscombe.
5. Added connection attributes encoding and nencoding which return the IANA character set name for the character set and national character set in use by the client.
6. Rework module initialization to use the techniques recommended by the Python documentation as one user was experiencing random segfaults due to the use of the module dictionary after the initialization was complete.
7. Removed support for the OPT_Threading attribute. Use the threaded keyword when creating connections and session pools instead.
8. Removed support for the OPT_NumbersAsStrings attribute. Use the numbersAsStrings attribute on cursors instead.
9. Use type long rather than type int in order to support long integers on 64-bit machines as reported by Uwe Hoffmann.
10. Add cursor attribute "bindarraysize" which is defaulted to 1 and is used to determine the size of the arrays created for bind variables.
11. Added repr() methods to provide something a little more useful than the standard type name and memory address.
12. Added keyword argument support to the functions that imply such in the documentation as requested by Harald Armin Massa.
13. Treat an empty dictionary passed through to cursor.execute() as keyword arguments the same as if no keyword arguments were specified at all, as requested by Fabien Grumelard.
14. Fixed memory leak when a LOB read would fail.

15. Set the LDFLAGS value in the environment rather than directly in the setup.py file in order to satisfy those who wish to enable the use of debugging symbols.
16. Use __DATE__ and __TIME__ to determine the date and time of the build rather than passing it through directly.
17. Use Oracle types and add casts to reduce warnings as requested by Amaury Forgeot d'Arc.
18. Fixed typo in error message.

Version 4.1.2 (December 2005)

1. Restore support of Oracle 9i features when using the Oracle 10g client.

Version 4.1.1 (December 2005)

1. Add support for dropping a connection from a session pool.
2. Add support for write only attributes "module", "action" and "clientinfo" which work only in Oracle 10g as requested by Egor Starostin.
3. Add support for pickling database errors.
4. Use the previously created bind variable as a template if available when creating a new variable of a larger size. Thanks to Ted Skolnick for the initial patch.
5. Fixed tests to work properly in the Python 2.4 environment where dates and timestamps are different Python types. Thanks to Henning von Bargen for pointing this out.
6. Added additional directories to search for include files and libraries in order to better support the Oracle 10g instant client.
7. Set the internal fetch number to 0 in order to satisfy very picky source analysis tools as requested by Amaury Fogeot d'Arc.
8. Improve the documentation for building and installing the module from source as some people are unaware of the standard methods for building Python modules using distutils.
9. Added note in the documentation indicating that the arraysize attribute can drastically affect performance of queries since this seems to be a common misunderstanding of first time users of cx_Oracle.
10. Add a comment indicating that on HP-UX Itanium with Oracle 10g the library ttsh10 must alos be linked against. Thanks to Bernard Delmee for the information.

Version 4.1 (January 2005)

1. Fixed bug where subclasses of Cursor do not pass the connection in the constructor causing a segfault.
2. DDL statements must be reparsed before execution as noted by Mihai Ibanescu.
3. Add support for setting input sizes by position.
4. Fixed problem with catching an exception during execute and then still attempting to perform a fetch afterwards as noted by Leith Parkin.
5. Rename the types so that they can be pickled and unpickled. Thanks to Harri Pasanen for pointing out the problem.
6. Handle invalid NLS_LANG setting properly (Oracle seems to like to provide a handle back even though it is invalid) and determine the number of bytes per character in order to allow for proper support in the future of multibyte and variable width character sets.

7. Remove date checking from the native case since Python already checks that dates are valid; enhance error message when invalid dates are encountered so that additional processing can be done.
8. Fix bug executing SQL using numeric parameter names with predefined variables (such as what takes place when calling stored procedures with out parameters).
9. Add support for reading CLOB values using multibyte or variable length character sets.

Version 4.1 beta 1 (September 2004)

1. Added support for Python 2.4. In Python 2.4, the datetime module is used for both binding and fetching of date and timestamp data. In Python 2.3, objects from the datetime module can be bound but the internal datetime objects will be returned from queries.
2. Added pickling support for LOB and datetime data.
3. Fully qualified the table name that was missing in an alter table statement in the setup test script as noted by Marc Gehling.
4. Added a section allowing for the setting of the RPATH linker directive in setup.py as requested by Iustin Pop.
5. Added code to raise a programming error exception when an attempt is made to access a LOB locator variable in a subsequent fetch.
6. The username, password and dsn (tnsentry) are stored on the connection object when specified, regardless of whether or not a standard connection takes place.
7. Added additional module level constant called "LOB" as requested by Joseph Canedo.
8. Changed exception type to IntegrityError for constraint violations as requested by Joseph Canedo.
9. If scale and precision are not specified, an attempt is made to return a long integer as requested by Joseph Canedo.
10. Added workaround for Oracle bug which returns an invalid handle when the prepare call fails. Thanks to alantam@hsbc.com for providing the code that demonstrated the problem.
11. The cursor method arrayvar() will now accept the actual list so that it is not necessary to call cursor.arrayvar() followed immediately by var.setvalue().
12. Fixed bug where attempts to execute the statement "None" with bind variables would cause a segmentation fault.
13. Added support for binding by position (paramstyle = "numeric").
14. Removed memory leak created by calls to OCIParamGet() which were not mirrored by calls to OCIDescriptorFree(). Thanks to Mihai Ibanescu for pointing this out and providing a patch.
15. Added support for calling cursor.executemany() with statement None implying that the previously prepared statement ought to be executed. Thanks to Mihai Ibanescu for providing a patch.
16. Added support for rebinding variables when a subsequent call to cursor.executemany() uses a different number of rows. Thanks to Mihai Ibanescu for supplying a patch.
17. The microseconds are now displayed in datetime variables when nonzero similar to method used in the datetime module.
18. Added support for binary_float and binary_double columns in Oracle 10g.

Version 4.0.1 (February 2004)

1. Fixed bugs on 64-bit platforms that caused segmentation faults and bus errors in session pooling and determining the bind variables associated with a statement.

2. Modified test suite so that 64-bit platforms are tested properly.
3. Added missing commit statements in the test setup scripts. Thanks to Keith Lyon for pointing this out.
4. Fix setup.py for Cygwin environments. Thanks to Doug Henderson for providing the necessary fix.
5. Added support for compiling cx_Oracle without thread support. Thanks to Andre Reitz for pointing this out.
6. Added support for a new keyword parameter called threaded on connections and session pools. This parameter defaults to False and indicates whether threaded mode ought to be used. It replaces the module level attribute OPT_Threading although examining the attribute will be retained until the next release at least.
7. Added support for a new keyword parameter called twophase on connections. This parameter defaults to False and indicates whether support for two phase (distributed or global) transactions ought to be present. Note that support for distributed transactions is buggy when crossing major version boundaries (Oracle 8i to Oracle 9i for example).
8. Ensure that the rowcount attribute is set properly when an exception is raised during execution. Thanks to Gary Aviv for pointing out this problem and its solution.

Version 4.0 (December 2003)

1. Added support for subclassing connections, cursors and session pools. The changes involved made it necessary to drop support for Python 2.1 and earlier although a branch exists in CVS to allow for support of Python 2.1 and earlier if needed.
2. Connections and session pools can now be created with keyword parameters, not just sequential parameters.
3. Queries now return integers whenever possible and long integers if the number will overflow a simple integer. Floats are only returned when it is known that the number is a floating point number or the integer conversion fails.
4. Added initial support for user callbacks on OCI functions. See the documentation for more details.
5. Add support for retrieving the bind variable names associated with a cursor with a new method bindnames().
6. Add support for temporary LOB variables. This means that setinputsizes() can be used with the values CLOB and BLOB to create these temporary LOB variables and allow for the equivalent of empty_clob() and empty_blob() since otherwise Oracle will treat empty strings as NULL values.
7. Automatically switch to long strings when the data size exceeds the maximum string size that Oracle allows (4000 characters) and raise an error if an attempt is made to set a string variable to a size that it does not support. This avoids truncation errors as reported by Jon Franz.
8. Add support for global (distributed) transactions and two phase commit.
9. Force the NLS settings for the session so that test tables are populated correctly in all circumstances; problems were noted by Ralf Braun and Allan Poulsen.
10. Display error messages using the environment handle when the error handle has not yet been created; this provides better error messages during this rather rare situation.
11. Removed memory leak in callproc() that was reported by Todd Whiteman.
12. Make consistent the calls to manipulate memory; otherwise segfaults can occur when the pymalloc option is used, as reported by Matt Hoskins.
13. Force a rollback when a session is released back to the session pool. Apparently the connections are not as stateless as Oracle's documentation suggests and this makes the logic consistent with normal connections.
14. Removed module method attach(). This can be replaced with a call to Connection(handle=) if needed.

Version 3.1 (August 2003)

1. Added support for connecting with SYSDBA and SYSOPER access which is needed for connecting as sys in Oracle 9i.
2. Only check the dictionary size if the variable is not NULL; otherwise, an error takes place which is not caught or cleared; this eliminates a spurious "Objects/dictobject.c:1258: bad argument to internal function" in Python 2.3.
3. Add support for session pooling. This is only support for Oracle 9i but is amazingly fast – about 100 times faster than connecting.
4. Add support for statement caching when pooling sessions, this reduces the parse time considerably. Unfortunately, the Oracle OCI does not allow this to be easily turned on for normal sessions.
5. Add method trim() on CLOB and BLOB variables for trimming the size.
6. Add support for externally identified users; to use this feature leave the username and password fields empty when connecting.
7. Add method cancel() on connection objects to cancel long running queries. Note that this only works on non-Windows platforms.
8. Add method callfunc() on cursor objects to allow calling a function without using an anonymous PL/SQL block.
9. Added documentation on objects that were not documented. At this point all objects, methods and constants in cx_Oracle have been documented.
10. Added support for timestamp columns in Oracle 9i.
11. Added module level method makedsn() which creates a data source name given the host, port and SID.
12. Added constant "buildtime" which is the time when the module was built as an additional means of identifying the build that is in use.
13. Binding a value that is incompatible to the previous value that was bound (data types do not match or array size is larger) will now result in a new bind taking place. This is more consistent with the DB API although it does imply a performance penalty when used.

Version 3.0a (June 2003)

1. Fixed bug where zero length PL/SQL arrays were being mishandled
2. Fixed support for the data type "float" in Oracle; added one to the display size to allow for the sign of the number, if necessary; changed the display size of unconstrained numbers to 127, which is the largest number that Oracle can handle
3. Added support for retrieving the description of a bound cursor before fetching it
4. Fixed a couple of build issues on Mac OS X, AIX and Solaris (64-bit)
5. Modified documentation slightly based on comments from several people
6. Included files in MANIFEST that are needed to generate the binaries
7. Modified test suite to work within the test environment at Computronix as well as within the packages that are distributed

Version 3.0 (March 2003)

1. Removed support for connection to Oracle7 databases; it is entirely possible that it will still work but I no longer have any way of testing and Oracle has dropped any meaningful support for Oracle7 anyway
2. Fetching of strings is now done with predefined memory areas rather than dynamic memory areas; dynamic fetching of strings was causing problems with Oracle 9i in some instances and databases using a different character set other than US ASCII
3. Fixed bug where segfault would occur if the '/' character preceded the '@' character in a connect string
4. Added two new cursor methods var() and arrayvar() in order to eliminate the need for setinputsizes() when defining PL/SQL arrays and as a generic method of acquiring bind variables directly when needed
5. Fixed support for binding cursors and added support for fetching cursors (these are known as ref cursors in PL/SQL).
6. Eliminated discrepancy between the array size used internally and the array size specified by the interface user; this was done earlier to avoid bus errors on 64-bit platforms but another way has been found to get around that issue and a number of people were getting confused because of the discrepancy
7. Added support for the attribute "connection" on cursors, an optional DB API extension
8. Added support for passing a dictionary as the second parameter for the cursor.execute() method in order to comply with the DB API more closely; the method of passing parameters with keyword arguments is still supported and is in fact preferred
9. Added support for the attribute "statement" on cursors which is a reference to the last SQL statement prepared or executed
10. Added support for passing any sequence to callproc() rather than just lists as before
11. Fixed bug where segfault would occur if the array size was changed after the cursor was executed but before it was fetched
12. Ignore array size when performing executemany() and use the length of the list of arguments instead
13. Rollback when connection is closed or destroyed to follow DB API rather than use the Oracle default (which is commit)
14. Added check for array size too large causing an integer overflow
15. Added support for iterators for Python 2.2 and above
16. Added test suite based on PyUnitTest
17. Added documentation in HTML format similar to the documentation for the core Python library

Version 2.5a (August 2002)

1. Fix problem with Oracle 9i and retrieving strings; it seems that Oracle 9i uses the correct method for dynamic callback but Oracle 8i will not work with that method so an #ifdef was added to check for the existence of an Oracle 9i feature; thanks to Paul Denize for discovering this problem

Version 2.5 (July 2002)

1. Added flag OPT_NoOracle7 which, if set, assumes that connections are being made to Oracle8 or higher databases; this allows for eliminating the overhead in performing this check at connect time
2. Added flag OPT_NumbersAsStrings which, if set, returns all numbers as strings rather than integers or floats; this flag is used when defined variables are created (during select statements only)

3. Added flag OPT_Threading which, if set, uses OCI threading mode; there is a significant performance degradation in this mode (about 15-20%) but it does allow threads to share connections (threadsafety level 2 according to the Python Database API 2.0); note that in order to support this, Oracle 8i or higher is now required
4. Added Py_BEGIN_ALLOW_THREADS and Py_END_ALLOW_THREADS pairs where applicable to support threading during blocking OCI calls
5. Added global method attach() to cx_Oracle to support attaching to an existing database handle (as provided by PowerBuilder, for example)
6. Eliminated the cursor method fetchbinds() which was used for returning the list of bind variables after execution to get the values of out variables; the cursor method setinputsizes() was modified to return the list of bind variables and the cursor method execute() was modified to return the list of defined variables in the case of a select statement being executed; these variables have three methods available to them: getvalue([<pos>]) to get the value of a variable, setvalue(<pos>, <value>) to set its value and copy(<var>, <src_pos>, <targ_pos>) to copy the value from a variable in a more efficient manner than setvalue(getvalue())
7. Implemented cursor method executemany() which expects a list of dictionaries for the arguments
8. Implemented cursor method callproc()
9. Added cursor method prepare() which parses (prepares) the statement for execution; subsequent execute() or executemany() calls can pass None as the statement which will imply use of the previously prepared statement; used for high performance only
10. Added cursor method fetchraw() which will perform a raw fetch of the cursor returning the number of rows thus fetched; this is used to avoid the overhead of generating result sets; used for high performance only
11. Added cursor method executemanyprepared() which is identical to the method executemany() except that it takes a single argument which is the number of times to execute a previously prepared statement and it assumes that the bind variables already have their values set; used for high performance only
12. Added support for rowid being returned in a select statement
13. Added support for comparing dates returned by cx_Oracle
14. Integrated patch from Andre Reitz to set the null ok flag in the description attribute of the cursor
15. Integrated patch from Andre Reitz to setup.py to support compilation with Python 1.5
16. Integrated patch from Benjamin Kearns to setup.py to support compilation on Cygwin

Version 2.4 (January 2002)

1. String variables can now be made any length (previously restricted to the 64K limit imposed by Oracle for default binding); use the type cx_Oracle.LONG_STRING as the argument to setinputsizes() for binding in string values larger than 4000 bytes.
2. Raw and long raw columns are now supported; use the types cx_Oracle.BINARY and cx_Oracle.LONG_BINARY as the argument to setinputsizes() for binding in values of these types.
3. Functions DateFromTicks(), TimeFromTicks() and TimestampFromTicks() are now implemented.
4. Function cursor.setoutputsize() implemented
5. Added the ability to bind arrays as out parameters to procedures; use the format [cx_Oracle.<DataType>, <NumElems>] as the input to the function setinputsizes() for binding arrays
6. Discovered from the Oracle 8.1.6 version of the documentation of the OCI libraries, that the size of the memory location required for the precision variable is larger than the printed documentation says; this was causing a problem with the code on the Sun platform.
7. Now support building RPMs for Linux.

Version 2.3 (October 2001)

1. Incremental performance enhancements (dealing with reusing cursors and bind handles)
2. Ensured that arrays of integers with a single float in them are all treated as floats, as suggested by Martin Koch.
3. Fixed code dealing with scale and precision for both defining a numeric variable and for providing the cursor description; this eliminates the problem of an underflow error (OCI-22054) when retrieving data with non-zero scale.

Version 2.2 (July 2001)

1. Upgraded thread safety to level 1 (according to the Python DB API 2.0) as an internal project required the ability to share the module between threads.
2. Added ability to bind ref cursors to PL/SQL blocks as requested by Brad Powell.
3. Added function write(Value, [Offset]) to LOB variables as requested by Matthias Kirst.
4. Procedure execute() on Cursor objects now permits a value None for the statement which means that the previously prepared statement will be executed and any input sizes set earlier will be retained. This was done to improve the performance of scripts that execute one statement many times.
5. Modified module global constants BINARY and DATETIME to point to the external representations of those types so that the expression type(var) == cx_Oracle.DATETIME will work as expected.
6. Added global constant version to provide means of determining the current version of the module.
7. Modified error checking routine to distinguish between an Oracle error and invalid handles.
8. Added error checking to avoid setting the value of a bind variable to a value that it cannot support and raised an exception to indicate this fact.
9. Added extra compile arguments for the AIX platform as suggested by Jehwan Ryu.
10. Added section to the README to indicate the method for a binary installation as suggested by Steve Holden.
11. Added simple usage example as requested by many people.
12. Added HISTORY file to the distribution.

CHAPTER 12

License

LICENSE AGREEMENT FOR CX_ORACLE

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the disclaimer that follows.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the names of the copyright holders nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS *AS IS* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Computronix ® is a registered trademark of Computronix (Canada) Ltd.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W