cx$_Oracle$

Release 5.1.3

September 22, 2014

Contents

cx_Oracle is a module that enables access to Oracle databases and conforms to the Python database API specification. This module is currently built against Oracle 11.2 and 12.1 and works for both Python 2.x and 3.x.

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

Contents:

CHAPTER 1

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 and is only available in Oracle 10g Release 2 and higher.

`cx_Oracle.`**`Connection`**`(`[*user*, *password*, *dsn*, *mode*, *handle*, *pool*, *threaded*, *twophase*, *events*, *cclass*, *purity*, *newpassword*]`)`

`cx_Oracle.`**`connect`**`(`[*user*, *password*, *dsn*, *mode*, *handle*, *pool*, *threaded*, *twophase*, *events*, *cclass*, *purity*, *newpassword*]`)`

Constructor for creating a connection to the database. Return a Connection object (*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 (*SessionPool 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 use the mode OCI_THREADED to 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 twophase argument is expected to be a boolean expression which indicates whether or not the attributes should be set on the connection object to allow for two phase commit. The default for this value is also False because of bugs in Oracle prior to Oracle 10g.

The events argument is expected to be a boolean expression which indicates whether or not to initialize Oracle in events mode (only available in Oracle 11g and higher).

The cclass argument is expected to be a string and defines the connection class for database resident connection pooling (DRCP) in Oracle 11g and higher.

The purity argument is expected to be one of `ATTR_PURITY_NEW` (the session must be new without any prior session state), `ATTR_PURITY_SELF` (the session may have been used before) or `ATTR_PURITY_DEFAULT` (the default behavior which is defined by Oracle in its documentation). This argument is only relevant in Oracle 11g and higher.

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

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

Constructor for creating a cursor. Return a new Cursor object (*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*[, *service_name*])

Return a string suitable for use as the dsn for the connect() method. This string is identical to the strings that are defined by the Oracle names server or defined in the tnsnames.ora file. If you wish to use the service name instead of the sid, do not include a value for the parameter sid and use the keyword parameter service_name instead.

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

`cx_Oracle.`**`SessionPool`**(*user*, *password*, *database*, *min*, *max*, *increment*[, *connectiontype*, *threaded*, *getmode=cx_Oracle.SPOOL_ATTRVAL_NOWAIT*, *homogeneous=True*])

Create a session pool (see Oracle 9i documentation for more information) and return a session pool object (*SessionPool 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 or not Oracle should use the mode OCI_THREADED to 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.

Note: This method is an extension to the DB API definition and is only available in Oracle 9i.

`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).

1.1 Constants

1.1.1 Global

`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.`**`SYSDBA`**

Value to be passed to the connect() method which indicates that SYSDBA access is to be acquired. See the Oracle documentation for more details.

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

`cx_Oracle.`**`SYSASM`**

Value to be passed to the connect() method which indicates that SYSASM access is to be acquired. See the Oracle documentation for more details.

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

`cx_Oracle.`**`SYSOPER`**

Value to be passed to the connect() method which indicates that SYSOPER access is to be acquired. See the Oracle documentation for more details.

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

`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 the Connection constructor must be a true value. See the comments on the Connection constructor for more information (*Module Interface*).

`cx_Oracle.`**`version`**

String constant stating the version of the module. Currently '5.1.3'.

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

1.1.2 Database Callbacks

Note: These constants are extensions to the DB API definition.

`cx_Oracle.`**`FNCODE_BINDBYNAME`**
This constant is used to register callbacks on the OCIBindByName() function of the OCI.

`cx_Oracle.`**`FNCODE_BINDBYPOS`**
This constant is used to register callbacks on the OCIBindByPos() function of the OCI.

`cx_Oracle.`**`FNCODE_DEFINEBYPOS`**
This constant is used to register callbacks on the OCIDefineByPos() function of the OCI.

`cx_Oracle.`**`FNCODE_STMTEXECUTE`**
This constant is used to register callbacks on the OCIStmtExecute() function of the OCI.

`cx_Oracle.`**`FNCODE_STMTFETCH`**
This constant is used to register callbacks on the OCIStmtFetch() function of the OCI.

`cx_Oracle.`**`FNCODE_STMTPREPARE`**
This constant is used to register callbacks on the OCIStmtPrepare() function of the OCI.

`cx_Oracle.`**`UCBTYPE_ENTRY`**
This constant is used to register callbacks on entry to the function of the OCI. In other words, the callback will be called prior to the execution of the OCI function.

`cx_Oracle.`**`UCBTYPE_EXIT`**
This constant is used to register callbacks on exit from the function of the OCI. In other words, the callback will be called after the execution of the OCI function.

`cx_Oracle.`**`UCBTYPE_REPLACE`**
This constant is used to register callbacks that completely replace the call to the OCI function.

1.1.3 Database Change Notification

Note: These constants are extensions to the DB API definition.

`cx_Oracle.`**`EVENT_DEREG`**
This constant is a possible value for the type of a message and indicates that the subscription object has been deregistered.

`cx_Oracle.`**`EVENT_NONE`**
This constant is a possible value for the type of a message and provides no additional information about the event.

`cx_Oracle.`**`EVENT_OBJCHANGE`**
This constant is a possible value for the type of a message and indicates that an object change of some sort has taken place.

`cx_Oracle.`**`EVENT_QUERYCHANGE`**
This constant is a possible value for the type of a message and indicates that the result set of a registered query has changed.

`cx_Oracle.`**`EVENT_SHUTDOWN`**
This constant is a possible value for the type of a message and indicates that the instance is in the process of being shut down.

`cx_Oracle.`**`EVENT_SHUTDOWN_ANY`**
This constant is a possible value for the type of a message and indicates that any instance (when running RAC) is in the process of being shut down.

cx_Oracle.**EVENT_STARTUP**
This constant is a possible value for the type of a message and indicates that the instance is in the process of being started up.

cx_Oracle.**OPCODE_ALLOPS**
This constant is the default value when creating a subscription and specifies that messages are to be sent for all operations.

cx_Oracle.**OPCODE_ALLROWS**
This constant is a possible value for the operation attribute of one of the table objects that are part of a message. It specifies that the table has been completely invalidated.

cx_Oracle.**OPCODE_ALTER**
This constant is a possible value for the operation attribute of one of the table objects that are part of a message. It specifies that the table has been altered in some fashion using DDL.

cx_Oracle.**OPCODE_DELETE**
This constant can be used when creating a subscription and specifies that messages are to be sent only when data is deleted. It is also a possible value for the operation attribute of one of the table objects that are part of a message.

cx_Oracle.**OPCODE_DROP**
This constant is a possible value for the operation attribute of one of the table objects that are part of a message. It specifies that the table has been dropped.

cx_Oracle.**OPCODE_INSERT**
This constant can be used when creating a subscription and specifies that messages are to be sent only when data is inserted. It is also a possible value for the operation attribute of one of the table objects that are part of a message.

cx_Oracle.**OPCODE_UPDATE**
This constant can be used when creating a subscription and specifies that messages are to be sent only when data is updated. It is also a possible value for the operation attribute of one of the table objects that are part of a message.

cx_Oracle.**SUBSCR_CQ_QOS_QUERY**
This constant can be used when creating a subscription and specifies that notifications should only be sent if the result set of the registered query changes. By default no false positive notifictions will be generated.

cx_Oracle.**SUBSCR_CQ_QOS_BEST_EFFORT**
This constant can be used when creating a subscription and specifies 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_CQ_QOS_CLQRYCACHE**
This constant is a future possible value for the cqqos argument when creating a subscription. It specifies that client query caching be enabled.

cx_Oracle.**SUBSCR_NAMESPACE_DBCHANGE**
This constant is the default (and currently only) value for the namespace argument when creating a subscription.

cx_Oracle.**SUBSCR_PROTO_HTTP**
This constant is a future possible value for the protocol argument when creating a subscription. It specifies that notification will be sent to the HTTP URL when a message is generated.

cx_Oracle.**SUBSCR_PROTO_MAIL**
This constant is a future possible value for the protocol argument when creating a subscription. It specifies that an e-mail message should be sent to the target when a message is generated.

`cx_Oracle.`**`SUBSCR_PROTO_OCI`**

This constant is the default (and currently only valid) value for the protocol argument when creating a subscription.

`cx_Oracle.`**`SUBSCR_PROTO_SERVER`**

This constant is a future possible value for the protocol argument when creating a subscription. It specifies that the database procedure will be invoked when a message is generated.

`cx_Oracle.`**`SUBSCR_QOS_HAREG`**

This constant is a future possible value for the qos argument when creating a subscription.

`cx_Oracle.`**`SUBSCR_QOS_MULTICBK`**

This constant is a future possible value for the qos argument when creating a subscription.

`cx_Oracle.`**`SUBSCR_QOS_PAYLOAD`**

This constant is a future possible value for the qos argument when creating a subscription. It specifies that a payload be delivered with the message.

`cx_Oracle.`**`SUBSCR_QOS_PURGE_ON_NTFN`**

This constant can be used when creating a subscription and specifies that the subscription should be automatically unregistered after the first notification.

`cx_Oracle.`**`SUBSCR_QOS_RELIABLE`**

This constant is a future possible value for the qos argument when creating a subscription. It specifies that notifications should not be lost in the event of database failure.

`cx_Oracle.`**`SUBSCR_QOS_REPLICATE`**

This constant is a future possible value for the qos argument when creating a subscription.

`cx_Oracle.`**`SUBSCR_QOS_SECURE`**

This constant is a future possible value for the qos argument when creating a subscription.

1.1.4 Database Resident Connection Pooling

Note: These constants are extensions to the DB API definition.

`cx_Oracle.`**`ATTR_PURITY_DEFAULT`**

This constant is used when using database resident connection pooling (DRCP) and specifies that the purity of the session is the default value used by Oracle (see Oracle's documentation for more information).

`cx_Oracle.`**`ATTR_PURITY_NEW`**

This constant is used when using database resident connection pooling (DRCP) and specifies 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 when using database resident connection pooling (DRCP) and specifies that the session acquired from the pool need not be new and may have prior session state.

1.1.5 Database Startup/Shutdown

Note: These constants are extensions to the DB API definition.

`cx_Oracle.`**`PRELIM_AUTH`**

This constant is used to define the preliminary authentication mode required for performing database startup and shutdown.

`cx_Oracle.`**`DBSHUTDOWN_ABORT`**
This constant is used in database shutdown to indicate that the program should not wait for current calls to complete or for users to disconnect from the database. Use only in unusual circumstances since database recovery may be necessary upon next startup.

`cx_Oracle.`**`DBSHUTDOWN_FINAL`**
This constant is used in database shutdown to indicate that the instance can be truly halted. This should only be done after the database has been shut down in one of the other modes (except abort) and the database has been closed and dismounted using the appropriate SQL commands. See the method `shutdown()` in the section on connections (*Connection Object*).

`cx_Oracle.`**`DBSHUTDOWN_IMMEDIATE`**
This constant is used in database shutdown to indicate that all uncommitted transactions should be rolled back and any connected users should be disconnected.

`cx_Oracle.`**`DBSHUTDOWN_TRANSACTIONAL`**
This constant is used in database shutdown to indicate that further connections should be prohibited and no new transactions should be allowed. It then waits for active transactions to complete.

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

1.1.6 Session Pooling

Note: These constants are extensions to the DB API definition.

`cx_Oracle.`**`SPOOL_ATTRVAL_FORCEGET`**
This constant is used to define the "get" mode on session pools and indicates 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 define the "get" mode on session pools and indicates that an exception is raised if there are no free sessions available in the pool.

`cx_Oracle.`**`SPOOL_ATTRVAL_WAIT`**
This constant is used to define the "get" mode on session pools and indicates that the acquisition of a connection waits until a session is freed if there are no free sessions available in the pool.

1.2 Types

`cx_Oracle.`**`BINARY`**
This type object is used to describe columns in a database that are binary (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.`**`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 this 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_UNICODE`**

This type object is used to describe columns in a database that are fixed length unicode strings (in Oracle this is NCHAR columns); 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 and is only available in Python 2.x. In Python 3.x these types of columns are returned as FIXED_CHAR.

`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_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.`**`LONG_UNICODE`**

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

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 and is only available in Oracle 10g.

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 and is only available in Oracle 9i.

`cx_Oracle.`**`UNICODE`**
This type object is used to describe columns in a database that are unicode (in Oracle this is NVARCHAR2 columns).

Note: This type is an extension to the DB API definition and is only available in Python 2.x. In Python 3.x these types of columns are returned as STRING.

1.3 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.

1.4 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 three 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..

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

```
import sys
import cx_Oracle

connection = cx_Oracle.Connection("user/pw@tns")
cursor = connection.cursor()

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

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 and is only available in Oracle 10g. 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.

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

`Connection.`**`begin`**`()`
`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 twophase argument to the Connection constructor must be a true value. See the comments on the Connection constructor for more information (*Module Interface*).

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. This method also modifies the attribute `Connection.password` upon successful completion.

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 and is only available in Oracle 10g.

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 (*Cursor Object*) using the connection.

`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.`**`encoding`**
This read-only attribute returns the IANA character set name of the character set in use by the Oracle client.

Note: This attribute is an extension to the DB API definition and is only available in Python 2.x when not built in unicode mode.

`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.`**`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 and is only available in Oracle 10g. The maximum length for this string is 48 and if you exceed this length you will get ORA-24960.

`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 and is only available in Python 2.x when not built in unicode mode.

`Connection.`**`outputtypehandler`**

This read-write attribute specifies a method called for each value that is 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 values fetched from cursors.

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

`Connection.`**`password`**

This read-write attribute initially contains the password of the user which established the connection to the database.

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 and is only available in Oracle 10g R2 and higher.

`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.`**`register`**`(`*code*, *when*, *function*`)`

Register the function as an OCI callback. The code is one of the function codes defined in the Oracle documentation of which the most common ones are defined as constants in this module. The when parameter is one of `UCBTYPE_ENTRY`, `UCBTYPE_EXIT` or `UCBTYPE_REPLACE`. The function is a Python function which will accept the parameters that the OCI function accepts, modified as needed to return Python objects that are of some use. Note that this is a highly experimental method and can cause cx_Oracle to crash if not used properly. In particular, the OCI does not provide sizing information to the callback so attempts to access a variable beyond the allocated size will crash cx_Oracle. Use with caution.

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 connected as `SYSDBA` or `SYSOPER`. First shutdown using one of the DBSHUTDOWN constants defined in the constants (*Constants*) section. Next issue the SQL statements required to close the database ("alter database close normal") and dismount the database ("alter database dismount") followed by a second call to this method with the `DBSHUTDOWN_FINAL` mode.

Note: This method is an extension to the DB API definition and is only available in Oracle 10g R2 and higher.

`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. Once this method has completed, connect again without the `PRELIM_AUTH` option and issue the statements required to mount ("alter database mount") and open ("alter database open") the database.

Note: This method is an extension to the DB API definition and is only available in Oracle 10g R2 and higher.

`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*, *rowids=False*, *port=0*, *qos=0*, *cqqos=0*`)`

Return a new Subscription object (*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 which is a message object. The timeout value specifies that the subscription expires after the given time in seconds. The default value of 0 indicates that the subscription does not expire. The operations argument enables filtering of the messages that are sent (insert, update, delete). The rowids flag specifies whether the rowids of affected rows should be included in the messages that are sent. The port specifies the listening port for callback notifications from the database server. The qos argument specifies quality of service options common to all subscriptions. Currently the only meaningful option is cx_Oracle.SUBSCR_QOS_PURGE_ON_NTFN which indicates that the subscription should be automatically unregistered after the first notification. The default value of 0 indicates that the subscription should persist after notification. The cqqos argument specifies quality of service options specific to continuous query notification. The default value of 0 indicates that change notification is required at database object granularity. The flag cx_Oracle.SUBSCR_CQ_QOS_QUERY specifies that change notification is required at query result set granularity, and the flag SUBSCR_CQ_QOS_BEST_EFFORT specifies that best effort filtering is accpetable, and false positive notifications may be received.

Note: This method is an extension to the DB API definition and is only available in Oracle 10g R2 and higher.

Note: Query result set change notification is only available in Oracle 11g and higher.

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.`**`unregister`**`(`*code*, *when*`)`

Unregister the function as an OCI callback. The code is one of the function codes defined in the Oracle documentation of which the most common ones are defined as constants in this module. The when parameter is one of `UCBTYPE_ENTRY`, `UCBTYPE_EXIT` or `UCBTYPE_REPLACE`.

Note: This method 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.

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 50 meaning to fetch 50 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 50 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 (*Variable Objects*). 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 (maximum allowable by Oracle) 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 the statement must have been prepared first.

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

`Cursor.`**`bindvars`**

This read-only attribute specifies 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 setinput-sizes(). 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 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*=[ ], *keyewordParameters = {}*)
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.

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, a list of variable objects (*Variable Objects*) will be returned corresponding to the list of variables into which data will be fetched with the `fetchone()`, `fetchmany()` and `fetchall()` methods; otherwise, `None` will be returned.

`Cursor.`**`executemany`**(*statement*, *parameters*)
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.

`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.`**`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.`**`numbersAsStrings`**

This integer attribute defines whether or not numbers should be returned as strings rather than integers or floating point numbers. This is useful to get around the fact that Oracle floating point numbers have considerably greater precision than C floating point numbers and not require a change to the SQL being executed.

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

`Cursor.`**`outputtypehandler`**

This read-write attribute specifies a method called for each value 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.

`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 (Oracle 9i and higher) 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.`**`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 can be used before a call to execute() to predefine memory areas for the long columns that will be fetched. The column is specified as an index into the result sequence. Not specifying the column will set the default size for all large columns in the cursor.

`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 (*Variable Objects*). If the size is not specified and the type is a string or binary, 4000 bytes (maximum allowable by Oracle) 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. Additional support for editing the attributes of this object is not yet available but will be forthcoming in a future release.

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 4

Variable Objects

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

`Variable.`**`allocelems`**

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.

Note: This attribute is deprecated. Use attribute numElements instead.

`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.`**`maxlength`**

This read-only attribute returns the maximum length of the variable.

Note: This attribute is deprecated. Use attribute bufferSize instead.

`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.

CHAPTER 5

SessionPool Object

Note: This object is an extension the DB API and is only available in Oracle 9i.

`SessionPool.`**`acquire`**`()`
: Acquire a connection from the session pool and return a connection object (*Connection Object*).

`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.`**`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.`**`password`**
: This read-only attribute returns the password of the user which established the connection to the database.

`SessionPool.`**`release`**`(`*connection*`)`
: Release the connection back to the pool. This will be done automatically as well if the connection object is garbage collected.

`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 6

Subscription Object

Note: This object is an extension the DB API and is only available in Oracle 10g Release 2 and higher.

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

`Subscription.`**`id`**
This read-only attribute returns the registration ID returned by the database when this subscription was created. Support for this attribute requires Oracle 11g or higher.

`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 qos 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 cqqos parameter included the flag cx_Oracle.SUBSCR_CQ_QOS_QUERY when the subscription was created then the queryid for the registeredquery is returned.

Note: Query result set change notification is only available in Oracle 11g and higher.

`Subscription.`**`rowids`**
This read-only attribute returns True or False specifying if rowids will be included in notifications sent using this subscription.

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

6.1 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 cqqos parameter did not include the flag cx_Oracle.SUBSCR_CQ_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 cqqos parameter included the flag cx_Oracle.SUBSCR_CQ_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 database change notification for additional information.

6.2 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 rowids argument to the Connection.subscribe() method is True.

6.3 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.

6.4 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.

Note: Query result set change notification is only available in Oracle 11g and higher.

`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 cx_Oracle.EVENT_DEREG and cx_Oracle.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 7

LOB Objects

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

Note: Internally, Oracle uses LOB locators which are allocated based on the cursor array size. Thus, it is important that the data in the LOB object be manipulated before another internal fetch takes place. The safest way to do this is to use the cursor as an iterator. In particular, do not use the fetchall() method. The exception "LOB variable no longer valid after subsequent fetch" will be raised if an attempt to access a LOB variable after a subsequent fetch is detected.

`LOB.`**`close`**`()`
Close the `LOB`. Call this when writing is completed so that the indexes associated with the LOB can be updated.

`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 is opened.

`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.

`LOB.`**`read`**`(`[*offset=1*[, *amount*]]`)`
Return a portion (or all) of the data in the LOB object.

`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.

`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. Note that if you want to make the LOB value smaller, you must use the trim() function.

CHAPTER 8

Release notes

8.1 5.x releases

8.1.1 Version 5.1.3

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.

8.1.2 Version 5.1.2

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.

8.1.3 Version 5.1.1

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.

8.1.4 Version 5.1

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.

8.1.5 Version 5.0.4

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.

8.1.6 Version 5.0.3

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.

8.1.7 Version 5.0.2

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.

8.1.8 Version 5.0.1

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.

8.1.9 Version 5.0

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.

8.2 Older releases

8.2.1 Version 4.4.1

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.

8.2.2 Version 4.4

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.

8.2.3 Version 4.3.3

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.

8.2.4 Version 4.3.2

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.

8.2.5 Version 4.3.1

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.

8.2.6 Version 4.3

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.

8.2.7 Version 4.2.1

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.

8.2.8 Version 4.2

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.

8.2.9 Version 4.1.2

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

8.2.10 Version 4.1.1

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.

8.2.11 Version 4.1

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.

8.2.12 Version 4.1 beta 1

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 cusor method arravar() 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 OCIDescriptor-Free(). 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.

8.2.13 Version 4.0.1

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.

8.2.14 Version 4.0

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.

8.2.15 Version 3.1

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.

8.2.16 Version 3.0a

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

8.2.17 Version 3.0

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

8.2.18 Version 2.5a

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

8.2.19 Version 2.5

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

8.2.20 Version 2.4

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.

8.2.21 Version 2.3

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.

8.2.22 Version 2.2

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 9

License

CHAPTER 10

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index