:mod:`sqlite3` --- DB-API 2.0 interface for SQLite databases

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 4)

Unknown directive type "module".

```
.. module:: sqlite3
   :synopsis: A DB-API 2.0 implementation using SQLite 3.x.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 7)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Gerhard HÃ¤ring <gh@ghaering.de>
```

Source code: :source:`Lib/sqlite3/`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 9); *backlink*

Unknown interpreted text role "source".

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard HÃ¤ring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249, and requires SQLite 3.7.15 or newer.

To use the module, start by creating a :class:`Connection` object that represents the database. Here the data will be stored in the :file:`example.db` file:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 24); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 24); *backlink*

Unknown interpreted text role "file".

```
import sqlite3
con = sqlite3.connect('example.db')
```

The special path name `:memory:` can be provided to create a temporary database in RAM.

Once a :class:`Connection` has been established, create a :class:`Cursor` object and call its :meth:`~Cursor.execute` method to perform SQL commands:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 34); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 34); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 34); *backlink*

Unknown interpreted text role "meth".

```
cur = con.cursor()

# Create table
cur.execute('''CREATE TABLE stocks
               (date text, trans text, symbol text, qty real, price real)''')

# Insert a row of data
cur.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")

# Save (commit) the changes
con.commit()

# We can also close the connection if we are done with it.
# Just be sure any changes have been committed or they will be lost.
con.close()
```

The saved data is persistent: it can be reloaded in a subsequent session even after restarting the Python interpreter:

```
import sqlite3
con = sqlite3.connect('example.db')
cur = con.cursor()
```

To retrieve data after executing a SELECT statement, either treat the cursor as an :term:`iterator`, call the cursor's :meth:`~Cursor.fetchone` method to retrieve a single matching row, or call :meth:`~Cursor.fetchall` to get a list of the matching rows.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 60); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 60); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 60); *backlink*

Unknown interpreted text role "meth".

This example uses the iterator form:

```
>>> for row in cur.execute('SELECT * FROM stocks ORDER BY price'):
        print(row)

('2006-01-05', 'BUY', 'RHAT', 100, 35.14)
('2006-03-28', 'BUY', 'IBM', 1000, 45.0)
('2006-04-06', 'SELL', 'IBM', 500, 53.0)
('2006-04-05', 'BUY', 'MSFT', 1000, 72.0)
```

SQL operations usually need to use values from Python variables. However, beware of using Python's string operations to assemble queries, as they are vulnerable to SQL injection attacks (see the xkcd webcomic for a humorous example of what can go wrong):

```
# Never do this -- insecure!
symbol = 'RHAT'
cur.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)
```

Instead, use the DB-API's parameter substitution. To insert a variable into a query string, use a placeholder in the string, and substitute the actual values into the query by providing them as a :class:`tuple` of values to the second argument of the cursor's :meth:`~Cursor.execute` method. An SQL statement may use one of two kinds of placeholders: question marks (qmark style) or named placeholders (named style). For the qmark style, `parameters` must be a :term:`sequence <sequence>`. For the named style, it can be either a :term:`sequence <sequence>` or :class:`dict` instance. The length of the :term:`sequence <sequence>` must match the number of placeholders, or a :exc:`ProgrammingError` is raised. If a :class:`dict` is given, it must contain keys for all named parameters. Any extra items are ignored. Here's an example of both styles:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 87); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 100)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/execute_1.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 103)

Unknown directive type "seealso".

```
.. seealso::

   https://www.sqlite.org
      The SQLite web page; the documentation describes the syntax and the
      available data types for the supported SQL dialect.

   https://www.w3schools.com/sql/
      Tutorial, reference and examples for learning SQL syntax.

   :pep:`249` - Database API Specification 2.0
      PEP written by Marc-AndrÃ© Lemburg.
```

Module functions and constants

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 122)

Unknown directive type "data".

```
.. data:: apilevel

   String constant stating the supported DB-API level. Required by the DB-API.
   Hard-coded to ``"2.0"``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 127)

Unknown directive type "data".

```
.. data:: paramstyle

   String constant stating the type of parameter marker formatting expected by
   the :mod:`sqlite3` module. Required by the DB-API. Hard-coded to
   ``"qmark"``.

   .. note::

      The :mod:`sqlite3` module supports both ``qmark`` and ``numeric`` DB-API
      parameter styles, because that is what the underlying SQLite library
      supports. However, the DB-API does not allow multiple values for
      the ``paramstyle`` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 140)

Unknown directive type "data".

```
.. data:: version

   The version number of this module, as a string. This is not the version of
   the SQLite library.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 146)

Unknown directive type "data".

```
.. data:: version_info

   The version number of this module, as a tuple of integers. This is not the
   version of the SQLite library.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 152)

Unknown directive type "data".

```
.. data:: sqlite_version

   The version number of the run-time SQLite library, as a string.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 157)

Unknown directive type "data".

```
.. data:: sqlite_version_info

   The version number of the run-time SQLite library, as a tuple of integers.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 162)

Unknown directive type "data".

```
.. data:: threadsafety

   Integer constant required by the DB-API 2.0, stating the level of thread
   safety the :mod:`sqlite3` module supports. This attribute is set based on
   the default `threading mode <https://sqlite.org/threadsafe.html>`_ the
   underlying SQLite library is compiled with. The SQLite threading modes are:

     1. **Single-thread**: In this mode, all mutexes are disabled and SQLite is
        unsafe to use in more than a single thread at once.
     2. **Multi-thread**: In this mode, SQLite can be safely used by multiple
        threads provided that no single database connection is used
        simultaneously in two or more threads.
     3. **Serialized**: In serialized mode, SQLite can be safely used by
        multiple threads with no restriction.

   The mappings from SQLite threading modes to DB-API 2.0 threadsafety levels
   are as follows:

   +------------------+-----------------+----------------------+-------------------------------+
   | SQLite threading | `threadsafety`_ | `SQLITE_THREADSAFE`_ | DB-API 2.0 meaning            |
   | mode             |                 |                      |                               |
   +==================+=================+======================+===============================+
   | single-thread    | 0               | 0                    | Threads may not share the     |
   |                  |                 |                      | module                        |
   +------------------+-----------------+----------------------+-------------------------------+
   | multi-thread     | 1               | 2                    | Threads may share the module, |
   |                  |                 |                      | but not connections           |
   +------------------+-----------------+----------------------+-------------------------------+
   | serialized       | 3               | 1                    | Threads may share the module, |
   |                  |                 |                      | connections and cursors       |
   +------------------+-----------------+----------------------+-------------------------------+

   .. _threadsafety: https://peps.python.org/pep-0249/#threadsafety
   .. _SQLITE_THREADSAFE: https://sqlite.org/compile.html#threadsafe

   .. versionchanged:: 3.11
      Set *threadsafety* dynamically instead of hard-coding it to ``1``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 200)

Unknown directive type "data".

```
.. data:: PARSE_DECLTYPES

   This constant is meant to be used with the *detect_types* parameter of the
   :func:`connect` function.

   Setting it makes the :mod:`sqlite3` module parse the declared type for each
   column it returns.  It will parse out the first word of the declared type,
   i. e.  for "integer primary key", it will parse out "integer", or for
   "number(10)" it will parse out "number". Then for that column, it will look
   into the converters dictionary and use the converter function registered for
```

```
   that type there.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 213)

Unknown directive type "data".

```
.. data:: PARSE_COLNAMES

   This constant is meant to be used with the *detect_types* parameter of the
   :func:`connect` function.

   Setting this makes the SQLite interface parse the column name for each column it
   returns.  It will look for a string formed [mytype] in there, and then decide
   that 'mytype' is the type of the column. It will try to find an entry of
   'mytype' in the converters dictionary and then use the converter function found
   there to return the value. The column name found in :attr:`Cursor.description`
   does not include the type, i. e. if you use something like
   ``'as "Expiration date [datetime]"'`` in your SQL, then we will parse out
   everything until the first ``'['`` for the column name and strip
   the preceding space: the column name would simply be "Expiration date".
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 229)

Unknown directive type "function".

```
.. function:: connect(database[, timeout, detect_types, isolation_level, check_same_thread, factory, cached_statements, uri])

   Opens a connection to the SQLite database file *database*. By default returns a
   :class:`Connection` object, unless a custom *factory* is given.

   *database* is a :term:`path-like object` giving the pathname (absolute or
   relative to the current  working directory) of the database file to be opened.
   You can use ``":memory:"`` to open a database connection to a database that
   resides in RAM instead of on disk.

   When a database is accessed by multiple connections, and one of the processes
   modifies the database, the SQLite database is locked until that transaction is
   committed. The *timeout* parameter specifies how long the connection should wait
   for the lock to go away until raising an exception. The default for the timeout
   parameter is 5.0 (five seconds).

   For the *isolation_level* parameter, please see the
   :attr:`~Connection.isolation_level` property of :class:`Connection` objects.

   SQLite natively supports only the types TEXT, INTEGER, REAL, BLOB and NULL. If
   you want to use other types you must add support for them yourself. The
   *detect_types* parameter and the using custom **converters** registered with the
   module-level :func:`register_converter` function allow you to easily do that.

   *detect_types* defaults to 0 (i. e. off, no type detection), you can set it to
   any combination of :const:`PARSE_DECLTYPES` and :const:`PARSE_COLNAMES` to turn
   type detection on. Due to SQLite behaviour, types can't be detected for generated
   fields (for example ``max(data)``), even when *detect_types* parameter is set. In
   such case, the returned type is :class:`str`.

   By default, *check_same_thread* is :const:`True` and only the creating thread may
   use the connection. If set :const:`False`, the returned connection may be shared
   across multiple threads. When using multiple threads with the same connection
   writing operations should be serialized by the user to avoid data corruption.

   By default, the :mod:`sqlite3` module uses its :class:`Connection` class for the
   connect call.  You can, however, subclass the :class:`Connection` class and make
   :func:`connect` use your class instead by providing your class for the *factory*
   parameter.

   Consult the section :ref:`sqlite3-types` of this manual for details.

   The :mod:`sqlite3` module internally uses a statement cache to avoid SQL parsing
   overhead. If you want to explicitly set the number of statements that are cached
   for the connection, you can set the *cached_statements* parameter. The currently
   implemented default is to cache 128 statements.

   If *uri* is :const:`True`, *database* is interpreted as a
   :abbr:`URI (Uniform Resource Identifier)` with a file path and an optional
   query string.  The scheme part *must* be ``"file:"``.  The path can be a
   relative or absolute file path.  The query string allows us to pass
   parameters to SQLite. Some useful URI tricks include::

       # Open a database in read-only mode.
       con = sqlite3.connect("file:template.db?mode=ro", uri=True)

       # Don't implicitly create a new database file if it does not already exist.
       # Will raise sqlite3.OperationalError if unable to open a database file.
       con = sqlite3.connect("file:nosuchdb.db?mode=rw", uri=True)

       # Create a shared named in-memory database.
       con1 = sqlite3.connect("file:mem1?mode=memory&cache=shared", uri=True)
       con2 = sqlite3.connect("file:mem1?mode=memory&cache=shared", uri=True)
       con1.executescript("create table t(t); insert into t values(28);")
       rows = con2.execute("select * from t").fetchall()

   More information about this feature, including a list of recognized
   parameters, can be found in the
   `SQLite URI documentation <https://www.sqlite.org/uri.html>`_.

   .. audit-event:: sqlite3.connect database sqlite3.connect
   .. audit-event:: sqlite3.connect/handle connection_handle sqlite3.connect

   .. versionchanged:: 3.4
      Added the *uri* parameter.

   .. versionchanged:: 3.7
      *database* can now also be a :term:`path-like object`, not only a string.

   .. versionchanged:: 3.10
      Added the ``sqlite3.connect/handle`` auditing event.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 312)

Unknown directive type "function".

```
.. function:: register_converter(typename, callable)

   Registers a callable to convert a bytestring from the database into a custom
   Python type. The callable will be invoked for all database values that are of
   the type *typename*. Confer the parameter *detect_types* of the :func:`connect`
   function for how the type detection works. Note that *typename* and the name of
   the type in your query are matched in case-insensitive manner.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 321)

Unknown directive type "function".

```
.. function:: register_adapter(type, callable)

   Registers a callable to convert the custom Python type *type* into one of
   SQLite's supported types. The callable *callable* accepts as single parameter
   the Python value, and must return a value of the following types: int,
   float, str or bytes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 329)

Unknown directive type "function".

```
.. function:: complete_statement(sql)

   Returns :const:`True` if the string *sql* contains one or more complete SQL
   statements terminated by semicolons. It does not verify that the SQL is
   syntactically correct, only that there are no unclosed string literals and the
   statement is terminated by a semicolon.

   This can be used to build a shell for SQLite, as in the following example:


   .. literalinclude:: ../includes/sqlite3/complete_statement.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 342)

Unknown directive type "function".

```
.. function:: enable_callback_tracebacks(flag)

   By default you will not get any tracebacks in user-defined functions,
   aggregates, converters, authorizer callbacks etc. If you want to debug them,
   you can call this function with *flag* set to :const:`True`. Afterwards, you
   will get tracebacks from callbacks on :data:`sys.stderr`. Use :const:`False`
   to disable the feature again.

   Register an :func:`unraisable hook handler <sys.unraisablehook>` for an
   improved debug experience::

      >>> import sqlite3
      >>> sqlite3.enable_callback_tracebacks(True)
      >>> cx = sqlite3.connect(":memory:")
      >>> cx.set_trace_callback(lambda stmt: 5/0)
      >>> cx.execute("select 1")
      Exception ignored in: <function <lambda> at 0x10b4e3ee0>
      Traceback (most recent call last):
        File "<stdin>", line 1, in <lambda>
      ZeroDivisionError: division by zero
      >>> import sys
      >>> sys.unraisablehook = lambda unraisable: print(unraisable)
      >>> cx.execute("select 1")
      UnraisableHookArgs(exc_type=<class 'ZeroDivisionError'>, exc_value=ZeroDivisionError('division by zero'), exc_traceback=<tra
      <sqlite3.Cursor object at 0x10b1fe840>
```

Connection Objects

An SQLite database connection has the following attributes and methods:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 378)

Unknown directive type "attribute".

```
.. attribute:: isolation_level

   Get or set the current default isolation level. :const:`None` for autocommit mode or
   one of "DEFERRED", "IMMEDIATE" or "EXCLUSIVE". See section
   :ref:`sqlite3-controlling-transactions` for a more detailed explanation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 384)

Unknown directive type "attribute".

```
.. attribute:: in_transaction

   :const:`True` if a transaction is active (there are uncommitted changes),
   :const:`False` otherwise.  Read-only attribute.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 391)

Unknown directive type "method".

```
.. method:: cursor(factory=Cursor)

   The cursor method accepts a single optional parameter *factory*. If
   supplied, this must be a callable returning an instance of :class:`Cursor`
   or its subclasses.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 397)

Unknown directive type "method".

```
.. method:: commit()

   This method commits the current transaction. If you don't call this method,
   anything you did since the last call to ``commit()`` is not visible from
   other database connections. If you wonder why you don't see the data you've
   written to the database, please check you didn't forget to call this method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 404)

Unknown directive type "method".

```
.. method:: rollback()

   This method rolls back any changes to the database since the last call to
   :meth:`commit`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 409)

Unknown directive type "method".

```
.. method:: close()

   This closes the database connection. Note that this does not automatically
   call :meth:`commit`. If you just close your database connection without
   calling :meth:`commit` first, your changes will be lost!
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 415)

Unknown directive type "method".

```
.. method:: execute(sql[, parameters])

   This is a nonstandard shortcut that creates a cursor object by calling
   the :meth:`~Connection.cursor` method, calls the cursor's
   :meth:`~Cursor.execute` method with the *parameters* given, and returns
   the cursor.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 422)

Unknown directive type "method".

```
.. method:: executemany(sql[, parameters])

   This is a nonstandard shortcut that creates a cursor object by
   calling the :meth:`~Connection.cursor` method, calls the cursor's
   :meth:`~Cursor.executemany` method with the *parameters* given, and
   returns the cursor.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 429)

Unknown directive type "method".

```
.. method:: executescript(sql_script)

   This is a nonstandard shortcut that creates a cursor object by
   calling the :meth:`~Connection.cursor` method, calls the cursor's
   :meth:`~Cursor.executescript` method with the given *sql_script*, and
   returns the cursor.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 436)

Unknown directive type "method".

```
.. method:: create_function(name, num_params, func, *, deterministic=False)

   Creates a user-defined function that you can later use from within SQL
   statements under the function name *name*. *num_params* is the number of
   parameters the function accepts (if *num_params* is -1, the function may
   take any number of arguments), and *func* is a Python callable that is
   called as the SQL function. If *deterministic* is true, the created function
   is marked as `deterministic <https://sqlite.org/deterministic.html>`_, which
   allows SQLite to perform additional optimizations. This flag is supported by
   SQLite 3.8.3 or higher, :exc:`NotSupportedError` will be raised if used
   with older versions.

   The function can return any of the types supported by SQLite: bytes, str, int,
   float and ``None``.

   .. versionchanged:: 3.8
      The *deterministic* parameter was added.

   Example:

   .. literalinclude:: ../includes/sqlite3/md5func.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 459)

Unknown directive type "method".

```
.. method:: create_aggregate(name, num_params, aggregate_class)

   Creates a user-defined aggregate function.

   The aggregate class must implement a ``step`` method, which accepts the number
   of parameters *num_params* (if *num_params* is -1, the function may take
   any number of arguments), and a ``finalize`` method which will return the
   final result of the aggregate.

   The ``finalize`` method can return any of the types supported by SQLite:
   bytes, str, int, float and ``None``.

   Example:

   .. literalinclude:: ../includes/sqlite3/mysumaggr.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 476)

Unknown directive type "method".

```
.. method:: create_collation(name, callable)

   Creates a collation with the specified *name* and *callable*. The callable will
   be passed two string arguments. It should return -1 if the first is ordered
```

```
   lower than the second, 0 if they are ordered equal and 1 if the first is ordered
   higher than the second.  Note that this controls sorting (ORDER BY in SQL) so
   your comparisons don't affect other SQL operations.

   Note that the callable will get its parameters as Python bytestrings, which will
   normally be encoded in UTF-8.

   The following example shows a custom collation that sorts "the wrong way":

   .. literalinclude:: ../includes/sqlite3/collation_reverse.py

   To remove a collation, call ``create_collation`` with ``None`` as callable::

      con.create_collation("reverse", None)

   .. versionchanged:: 3.11
      The collation name can contain any Unicode character.  Earlier, only
      ASCII characters were allowed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 500)

Unknown directive type "method".

```
.. method:: interrupt()

   You can call this method from a different thread to abort any queries that might
   be executing on the connection. The query will then abort and the caller will
   get an exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 507)

Unknown directive type "method".

```
.. method:: set_authorizer(authorizer_callback)

   This routine registers a callback. The callback is invoked for each attempt to
   access a column of a table in the database. The callback should return
   :const:`SQLITE_OK` if access is allowed, :const:`SQLITE_DENY` if the entire SQL
   statement should be aborted with an error and :const:`SQLITE_IGNORE` if the
   column should be treated as a NULL value. These constants are available in the
   :mod:`sqlite3` module.

   The first argument to the callback signifies what kind of operation is to be
   authorized. The second and third argument will be arguments or :const:`None`
   depending on the first argument. The 4th argument is the name of the database
   ("main", "temp", etc.) if applicable. The 5th argument is the name of the
   inner-most trigger or view that is responsible for the access attempt or
   :const:`None` if this access attempt is directly from input SQL code.

   Please consult the SQLite documentation about the possible values for the first
   argument and the meaning of the second and third argument depending on the first
   one. All necessary constants are available in the :mod:`sqlite3` module.

   Passing :const:`None` as *authorizer_callback* will disable the authorizer.

   .. versionchanged:: 3.11
      Added support for disabling the authorizer using :const:`None`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 533)

Unknown directive type "method".

```
.. method:: set_progress_handler(handler, n)

   This routine registers a callback. The callback is invoked for every *n*
   instructions of the SQLite virtual machine. This is useful if you want to
   get called from SQLite during long-running operations, for example to update
   a GUI.

   If you want to clear any previously installed progress handler, call the
   method with :const:`None` for *handler*.

   Returning a non-zero value from the handler function will terminate the
   currently executing query and cause it to raise an :exc:`OperationalError`
   exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 548)

Unknown directive type "method".

```
.. method:: set_trace_callback(trace_callback)

   Registers *trace_callback* to be called for each SQL statement that is
   actually executed by the SQLite backend.

   The only argument passed to the callback is the statement (as
   :class:`str`) that is being executed. The return value of the callback is
   ignored. Note that the backend does not only run statements passed to the
   :meth:`Cursor.execute` methods.  Other sources include the
   :ref:`transaction management <sqlite3-controlling-transactions>` of the
   sqlite3 module and the execution of triggers defined in the current
   database.

   Passing :const:`None` as *trace_callback* will disable the trace callback.

   .. note::
      Exceptions raised in the trace callback are not propagated. As a
      development and debugging aid, use
      :meth:`~sqlite3.enable_callback_tracebacks` to enable printing
      tracebacks from exceptions raised in the trace callback.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 572)

Unknown directive type "method".

```
.. method:: enable_load_extension(enabled)

   This routine allows/disallows the SQLite engine to load SQLite extensions
```

from shared libraries. SQLite extensions can define new functions,
aggregates or whole new virtual table implementations. One well-known
extension is the fulltext-search extension distributed with SQLite.

Loadable extensions are disabled by default. See [#f1]_.

.. audit-event:: sqlite3.enable_load_extension connection,enabled sqlite3.enable_load_extension

.. versionadded:: 3.2

.. versionchanged:: 3.10
   Added the ``sqlite3.enable_load_extension`` auditing event.

.. literalinclude:: ../includes/sqlite3/load_extension.py

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 590)

Unknown directive type "method".

```
.. method:: load_extension(path)

   This routine loads an SQLite extension from a shared library.  You have to
   enable extension loading with :meth:`enable_load_extension` before you can
   use this routine.

   Loadable extensions are disabled by default. See [#f1]_.

   .. audit-event:: sqlite3.load_extension connection,path sqlite3.load_extension

   .. versionadded:: 3.2

   .. versionchanged:: 3.10
      Added the ``sqlite3.load_extension`` auditing event.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 605)

Unknown directive type "attribute".

```
.. attribute:: row_factory

   You can change this attribute to a callable that accepts the cursor and the
   original row as a tuple and will return the real result row.  This way, you can
   implement more advanced ways of returning results, such  as returning an object
   that can also access columns by name.

   Example:

   .. literalinclude:: ../includes/sqlite3/row_factory.py

   If returning a tuple doesn't suffice and you want name-based access to
   columns, you should consider setting :attr:`row_factory` to the
   highly-optimized :class:`sqlite3.Row` type. :class:`Row` provides both
   index-based and case-insensitive name-based access to columns with almost no
   memory overhead. It will probably be better than your own custom
   dictionary-based approach or even a db_row based solution.

   .. XXX what's a db_row-based solution?
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 626)

Unknown directive type "attribute".

```
.. attribute:: text_factory

   Using this attribute you can control what objects are returned for the ``TEXT``
   data type. By default, this attribute is set to :class:`str` and the
   :mod:`sqlite3` module will return :class:`str` objects for ``TEXT``.
   If you want to return :class:`bytes` instead, you can set it to :class:`bytes`.

   You can also set it to any other callable that accepts a single bytestring
   parameter and returns the resulting object.

   See the following example code for illustration:

   .. literalinclude:: ../includes/sqlite3/text_factory.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 641)

Unknown directive type "attribute".

```
.. attribute:: total_changes

   Returns the total number of database rows that have been modified, inserted, or
   deleted since the database connection was opened.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 647)

Unknown directive type "method".

```
.. method:: iterdump

   Returns an iterator to dump the database in an SQL text format.  Useful when
   saving an in-memory database for later restoration.  This function provides
   the same capabilities as the :kbd:`.dump` command in the :program:`sqlite3`
   shell.

   Example::

      # Convert file existing_db.db to SQL dump file dump.sql
      import sqlite3

      con = sqlite3.connect('existing_db.db')
      with open('dump.sql', 'w') as f:
          for line in con.iterdump():
              f.write('%s\n' % line)
      con.close()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 666)

Unknown directive type "method".

```
.. method:: backup(target, *, pages=-1, progress=None, name="main", sleep=0.250)

   This method makes a backup of an SQLite database even while it's being accessed
   by other clients, or concurrently by the same connection.  The copy will be
   written into the mandatory argument *target*, that must be another
   :class:`Connection` instance.

   By default, or when *pages* is either ``0`` or a negative integer, the entire
   database is copied in a single step; otherwise the method performs a loop
   copying up to *pages* pages at a time.

   If *progress* is specified, it must either be ``None`` or a callable object that
   will be executed at each iteration with three integer arguments, respectively
   the *status* of the last iteration, the *remaining* number of pages still to be
   copied and the *total* number of pages.

   The *name* argument specifies the database name that will be copied: it must be
   a string containing either ``"main"``, the default, to indicate the main
   database, ``"temp"`` to indicate the temporary database or the name specified
   after the ``AS`` keyword in an ``ATTACH DATABASE`` statement for an attached
   database.

   The *sleep* argument specifies the number of seconds to sleep by between
   successive attempts to backup remaining pages, can be specified either as an
   integer or a floating point value.

   Example 1, copy an existing database into another::

      import sqlite3

      def progress(status, remaining, total):
          print(f'Copied {total-remaining} of {total} pages...')

      con = sqlite3.connect('existing_db.db')
      bck = sqlite3.connect('backup.db')
      with bck:
          con.backup(bck, pages=1, progress=progress)
      bck.close()
      con.close()

   Example 2, copy an existing database into a transient copy::

      import sqlite3

      source = sqlite3.connect('existing_db.db')
      dest = sqlite3.connect(':memory:')
      source.backup(dest)

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 717)

Unknown directive type "method".

```
.. method:: getlimit(category, /)

   Get a connection run-time limit. *category* is the limit category to be
   queried.

   Example, query the maximum length of an SQL statement::

      import sqlite3
      con = sqlite3.connect(":memory:")
      lim = con.getlimit(sqlite3.SQLITE_LIMIT_SQL_LENGTH)
      print(f"SQLITE_LIMIT_SQL_LENGTH={lim}")

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 732)

Unknown directive type "method".

```
.. method:: setlimit(category, limit, /)

   Set a connection run-time limit. *category* is the limit category to be
   set. *limit* is the new limit. If the new limit is a negative number, the
   limit is unchanged.

   Attempts to increase a limit above its hard upper bound are silently
   truncated to the hard upper bound. Regardless of whether or not the limit
   was changed, the prior value of the limit is returned.

   Example, limit the number of attached databases to 1::

      import sqlite3
      con = sqlite3.connect(":memory:")
      con.setlimit(sqlite3.SQLITE_LIMIT_ATTACHED, 1)

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)sqlite3.rst, line 751)

Unknown directive type "method".

```
.. method:: serialize(*, name="main")

   This method serializes a database into a :class:`bytes` object.  For an
   ordinary on-disk database file, the serialization is just a copy of the
   disk file.  For an in-memory database or a "temp" database, the
   serialization is the same sequence of bytes which would be written to
   disk if that database were backed up to disk.

   *name* is the database to be serialized, and defaults to the main
   database.

   .. note::

      This method is only available if the underlying SQLite library has the
      serialize API.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 770)

Unknown directive type "method".

```
.. method:: deserialize(data, /, *, name="main")

   This method causes the database connection to disconnect from database
   *name*, and reopen *name* as an in-memory database based on the
   serialization contained in *data*.  Deserialization will raise
   :exc:`OperationalError` if the database connection is currently involved
   in a read transaction or a backup operation.  :exc:`DataError` will be
   raised if ``len(data)`` is larger than ``2**63 - 1``, and
   :exc:`DatabaseError` will be raised if *data* does not contain a valid
   SQLite database.

   .. note::

      This method is only available if the underlying SQLite library has the
      deserialize API.

   .. versionadded:: 3.11
```

Cursor Objects

A :class:`Cursor` instance has the following attributes and methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 796); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 798)

Unknown directive type "index".

```
.. index:: single: ? (question mark); in SQL statements
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 799)

Unknown directive type "index".

```
.. index:: single: : (colon); in SQL statements
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 801)

Unknown directive type "method".

```
.. method:: execute(sql[, parameters])

   Executes an SQL statement. Values may be bound to the statement using
   :ref:`placeholders <sqlite3-placeholders>`.

   :meth:`execute` will only execute a single SQL statement. If you try to execute
   more than one statement with it, it will raise a :exc:`.Warning`. Use
   :meth:`executescript` if you want to execute multiple SQL statements with one
   call.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 812)

Unknown directive type "method".

```
.. method:: executemany(sql, seq_of_parameters)

   Executes a :ref:`parameterized <sqlite3-placeholders>` SQL command
   against all parameter sequences or mappings found in the sequence
   *seq_of_parameters*. The :mod:`sqlite3` module also allows using an
   :term:`iterator` yielding parameters instead of a sequence.

   .. literalinclude:: ../includes/sqlite3/executemany_1.py

   Here's a shorter example using a :term:`generator`:

   .. literalinclude:: ../includes/sqlite3/executemany_2.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 826)

Unknown directive type "method".

```
.. method:: executescript(sql_script)

   This is a nonstandard convenience method for executing multiple SQL statements
   at once. It issues a ``COMMIT`` statement first, then executes the SQL script it
   gets as a parameter.  This method disregards :attr:`isolation_level`; any
   transaction control must be added to *sql_script*.

   *sql_script* can be an instance of :class:`str`.

   Example:

   .. literalinclude:: ../includes/sqlite3/executescript.py
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 840)

Unknown directive type "method".

```
.. method:: fetchone()

   Fetches the next row of a query result set, returning a single sequence,
   or :const:`None` when no more data is available.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 846)

Unknown directive type "method".

```
.. method:: fetchmany(size=cursor.arraysize)

   Fetches the next set of rows of a query result, returning a list.  An empty
   list is returned when no more rows are available.

   The number of rows to fetch per call is specified by the *size* parameter.
   If it is not given, the cursor's arraysize determines the number of rows
   to be fetched. The method should try to fetch as many rows as indicated by
   the size parameter. If this is not possible due to the specified number of
   rows not being available, fewer rows may be returned.

   Note there are performance considerations involved with the *size* parameter.
   For optimal performance, it is usually best to use the arraysize attribute.
   If the *size* parameter is used, then it is best for it to retain the same
   value from one :meth:`fetchmany` call to the next.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 862)

Unknown directive type "method".

```
.. method:: fetchall()

   Fetches all (remaining) rows of a query result, returning a list.  Note that
   the cursor's arraysize attribute can affect the performance of this operation.
   An empty list is returned when no rows are available.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 868)

Unknown directive type "method".

```
.. method:: close()

   Close the cursor now (rather than whenever ``__del__`` is called).

   The cursor will be unusable from this point forward; a :exc:`ProgrammingError`
   exception will be raised if any operation is attempted with the cursor.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 875)

Unknown directive type "method".

```
.. method:: setinputsizes(sizes)

   Required by the DB-API. Does nothing in :mod:`sqlite3`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 879)

Unknown directive type "method".

```
.. method:: setoutputsize(size [, column])

   Required by the DB-API. Does nothing in :mod:`sqlite3`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 883)

Unknown directive type "attribute".

```
.. attribute:: rowcount

   Although the :class:`Cursor` class of the :mod:`sqlite3` module implements this
   attribute, the database engine's own support for the determination of "rows
   affected"/"rows selected" is quirky.

   For :meth:`executemany` statements, the number of modifications are summed up
   into :attr:`rowcount`.

   As required by the Python DB API Spec, the :attr:`rowcount` attribute "is -1 in
   case no ``executeXX()`` has been performed on the cursor or the rowcount of the
   last operation is not determinable by the interface". This includes ``SELECT``
   statements because we cannot determine the number of rows a query produced
   until all rows were fetched.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 898)

Unknown directive type "attribute".

```
.. attribute:: lastrowid

   This read-only attribute provides the row id of the last inserted row. It
   is only updated after successful ``INSERT`` or ``REPLACE`` statements
   using the :meth:`execute` method.  For other statements, after
   :meth:`executemany` or :meth:`executescript`, or if the insertion failed,
   the value of ``lastrowid`` is left unchanged.  The initial value of
   ``lastrowid`` is :const:`None`.

   .. note::
      Inserts into ``WITHOUT ROWID`` tables are not recorded.

   .. versionchanged:: 3.6
      Added support for the ``REPLACE`` statement.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 913)

Unknown directive type "attribute".

```
.. attribute:: arraysize

   Read/write attribute that controls the number of rows returned by :meth:`fetchmany`.
   The default value is 1 which means a single row would be fetched per call.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 918)

Unknown directive type "attribute".

```
.. attribute:: description
```

```
   This read-only attribute provides the column names of the last query. To
   remain compatible with the Python DB API, it returns a 7-tuple for each
   column where the last six items of each tuple are :const:`None`.

   It is set for ``SELECT`` statements without any matching rows as well.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 926)

Unknown directive type "attribute".

```
.. attribute:: connection

   This read-only attribute provides the SQLite database :class:`Connection`
   used by the :class:`Cursor` object.  A :class:`Cursor` object created by
   calling :meth:`con.cursor() <Connection.cursor>` will have a
   :attr:`connection` attribute that refers to *con*::

      >>> con = sqlite3.connect(":memory:")
      >>> cur = con.cursor()
      >>> cur.connection == con
      True
```

Row Objects

A :class:`Row` instance serves as a highly optimized :attr:`~Connection.row_factory` for :class:`Connection` objects. It tries to mimic a tuple in most of its features.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 945); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 945); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 945); *backlink*

Unknown interpreted text role "class".

It supports mapping access by column name and index, iteration, representation, equality testing and :func:`len`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 949); *backlink*

Unknown interpreted text role "func".

If two :class:`Row` objects have exactly the same columns and their members are equal, they compare equal.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 952); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 955)

Unknown directive type "method".

```
.. method:: keys

   This method returns a list of column names. Immediately after a query,
   it is the first member of each tuple in :attr:`Cursor.description`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 960)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
   Added support of slicing.
```

Let's assume we initialize a table as in the example given above:

```
con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.execute('''create table stocks
(date text, trans text, symbol text,
 qty real, price real)''')
cur.execute("""insert into stocks
            values ('2006-01-05','BUY','RHAT',100,35.14)""")
con.commit()
cur.close()
```

Now we plug :class:`Row` in:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 975); *backlink*

Unknown interpreted text role "class".

```
>>> con.row_factory = sqlite3.Row
>>> cur = con.cursor()
>>> cur.execute('select * from stocks')
<sqlite3.Cursor object at 0x7f4e7dd8fa80>
>>> r = cur.fetchone()
>>> type(r)
<class 'sqlite3.Row'>
>>> tuple(r)
('2006-01-05', 'BUY', 'RHAT', 100.0, 35.14)
>>> len(r)
5
>>> r[2]
'RHAT'
>>> r.keys()
['date', 'trans', 'symbol', 'qty', 'price']
>>> r['qty']
100.0
>>> for member in r:
...     print(member)
```

```
...
2006-01-05
BUY
RHAT
100.0
35.14
```

Exceptions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1009)

Unknown directive type "exception".

```
.. exception:: Warning

   A subclass of :exc:`Exception`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1013)

Unknown directive type "exception".

```
.. exception:: Error

   The base class of the other exceptions in this module.  It is a subclass
   of :exc:`Exception`.

   .. attribute:: sqlite_errorcode

      The numeric error code from the
      `SQLite API <https://sqlite.org/rescode.html>`_

      .. versionadded:: 3.11

   .. attribute:: sqlite_errorname

      The symbolic name of the numeric error code
      from the `SQLite API <https://sqlite.org/rescode.html>`_

      .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1032)

Unknown directive type "exception".

```
.. exception:: DatabaseError

   Exception raised for errors that are related to the database.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1036)

Unknown directive type "exception".

```
.. exception:: IntegrityError

   Exception raised when the relational integrity of the database is affected,
   e.g. a foreign key check fails.  It is a subclass of :exc:`DatabaseError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1041)

Unknown directive type "exception".

```
.. exception:: ProgrammingError

   Exception raised for programming errors, e.g. table not found or already
   exists, syntax error in the SQL statement, wrong number of parameters
   specified, etc.  It is a subclass of :exc:`DatabaseError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1047)

Unknown directive type "exception".

```
.. exception:: OperationalError

   Exception raised for errors that are related to the database's operation
   and not necessarily under the control of the programmer, e.g. an unexpected
   disconnect occurs, the data source name is not found, a transaction could
   not be processed, etc.  It is a subclass of :exc:`DatabaseError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1054)

Unknown directive type "exception".

```
.. exception:: NotSupportedError

   Exception raised in case a method or database API was used which is not
   supported by the database, e.g. calling the :meth:`~Connection.rollback`
   method on a connection that does not support transaction or has
   transactions turned off.  It is a subclass of :exc:`DatabaseError`.
```

SQLite and Python types

Introduction

SQLite natively supports the following types: `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`.

The following Python types can thus be sent to SQLite without any problem:

Python type	SQLite type

Python type	SQLite type
:const:`None` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1080); backlink Unknown interpreted text role "const".	NULL
:class:`int` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1082); backlink Unknown interpreted text role "class".	INTEGER
:class:`float` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1084); backlink Unknown interpreted text role "class".	REAL
:class:`str` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1086); backlink Unknown interpreted text role "class".	TEXT
:class:`bytes` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1088); backlink Unknown interpreted text role "class".	BLOB

This is how SQLite types are converted to Python types by default:

SQLite type	Python type
NULL	:const:`None` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1097); backlink Unknown interpreted text role "const".
INTEGER	:class:`int` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1099); backlink Unknown interpreted text role "class".
REAL	:class:`float` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1101); backlink Unknown interpreted text role "class".
TEXT	depends on :attr:`~Connection.text_factory`, :class:`str` by default System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1103); backlink Unknown interpreted text role "attr". System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1103); backlink Unknown interpreted text role "class".
BLOB	:class:`bytes` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1106); backlink Unknown interpreted text role "class".

The type system of the :mod:`sqlite3` module is extensible in two ways: you can store additional Python types in an SQLite database via object adaptation, and you can let the :mod:`sqlite3` module convert SQLite types to different Python types via converters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1108); backlink
Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1108); backlink
Unknown interpreted text role "mod".

Using adapters to store additional Python types in SQLite databases

As described before, SQLite supports only a limited set of types natively. To use other Python types with SQLite, you must **adapt** them to one of the sqlite3 module's supported types for SQLite: one of NoneType, int, float, str, bytes.

There are two ways to enable the :mod:`sqlite3` module to adapt a custom Python type to one of the supported ones.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1122); *backlink*

Unknown interpreted text role "mod".

Letting your object adapt itself

This is a good approach if you write the class yourself. Let's suppose you have a class like this:

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y
```

Now you want to store the point in a single SQLite column. First you'll have to choose one of the supported types to be used for representing the point. Let's just use str and separate the coordinates using a semicolon. Then you need to give your class a method `__conform__(self, protocol)` which must return the converted value. The parameter *protocol* will be :class:`PrepareProtocol`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1136); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1142)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/adapter_point_1.py
```

Registering an adapter callable

The other possibility is to create a function that converts the type to the string representation and register the function with :meth:`register_adapter`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1148); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1151)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/adapter_point_2.py
```

The :mod:`sqlite3` module has two default adapters for Python's built-in :class:`datetime.date` and :class:`datetime.datetime` types. Now let's suppose we want to store :class:`datetime.datetime` objects not in ISO representation, but as a Unix timestamp.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1153); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1153); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1153); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1153); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1158)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/adapter_datetime.py
```

Converting SQLite values to custom Python types

Writing an adapter lets you send custom Python types to SQLite. But to make it really useful we need to make the Python to SQLite to Python roundtrip work.

Enter converters.

Let's go back to the :class:`Point` class. We stored the x and y coordinates separated via semicolons as strings in SQLite.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1169); *backlink*

Unknown interpreted text role "class".

First, we'll define a converter function that accepts the string as a parameter and constructs a :class:`Point` object from it.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1172); *backlink*

Unknown interpreted text role "class".

Note

Converter functions **always** get called with a :class:`bytes` object, no matter under which data type you sent the value to SQLite.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1177); *backlink*

Unknown interpreted text role "class".

```
def convert_point(s):
    x, y = map(float, s.split(b";"))
    return Point(x, y)
```

Now you need to make the :mod:`sqlite3` module know that what you select from the database is actually a point. There are two ways of doing this:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1186); *backlink*

Unknown interpreted text role "mod".

- Implicitly via the declared type
- Explicitly via the column name

Both ways are described in section :ref:`sqlite3-module-contents`, in the entries for the constants :const:`PARSE_DECLTYPES` and :const:`PARSE_COLNAMES`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1193); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1193); *backlink*

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1193); *backlink*

Unknown interpreted text role "const".

The following example illustrates both approaches.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1198)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/converter_point.py
```

Default adapters and converters

There are default adapters for the date and datetime types in the datetime module. They will be sent as ISO dates/ISO timestamps to SQLite.

The default converters are registered under the name "date" for :class:`datetime.date` and under the name "timestamp" for :class:`datetime.datetime`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1207); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1207); *backlink*

Unknown interpreted text role "class".

This way, you can use date/timestamps from Python without any additional fiddling in most cases. The format of the adapters is also compatible with the experimental SQLite date/time functions.

The following example demonstrates this.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1217)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/pysqlite_datetime.py
```

If a timestamp stored in SQLite has a fractional part longer than 6 numbers, its value will be truncated to microsecond precision by the timestamp converter.

Note

The default "timestamp" converter ignores UTC offsets in the database and always returns a naive :class:`datetime.datetime` object. To preserve UTC offsets in timestamps, either leave converters disabled, or register an offset-aware converter with :func:`register_converter`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1225); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1225); *backlink*

Unknown interpreted text role "func".

Controlling Transactions

The underlying `sqlite3` library operates in `autocommit` mode by default, but the Python :mod:`sqlite3` module by default does not.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1235); backlink

Unknown interpreted text role "mod".

`autocommit` mode means that statements that modify the database take effect immediately. A `BEGIN` or `SAVEPOINT` statement disables `autocommit` mode, and a `COMMIT`, a `ROLLBACK`, or a `RELEASE` that ends the outermost transaction, turns `autocommit` mode back on.

The Python :mod:`sqlite3` module by default issues a `BEGIN` statement implicitly before a Data Modification Language (DML) statement (i.e. `INSERT`/`UPDATE`/`DELETE`/`REPLACE`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1243); backlink

Unknown interpreted text role "mod".

You can control which kind of `BEGIN` statements :mod:`sqlite3` implicitly executes via the *isolation_level* parameter to the :func:`connect` call, or via the :attr:`isolation_level` property of connections. If you specify no *isolation_level*, a plain `BEGIN` is used, which is equivalent to specifying `DEFERRED`. Other possible values are `IMMEDIATE` and `EXCLUSIVE`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1247); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1247); backlink

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1247); backlink

Unknown interpreted text role "attr".

You can disable the :mod:`sqlite3` module's implicit transaction management by setting :attr:`isolation_level` to `None`. This will leave the underlying `sqlite3` library operating in `autocommit` mode. You can then completely control the transaction state by explicitly issuing `BEGIN`, `ROLLBACK`, `SAVEPOINT`, and `RELEASE` statements in your code.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1254); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1254); backlink

Unknown interpreted text role "attr".

Note that :meth:`~Cursor.executescript` disregards :attr:`isolation_level`; any transaction control must be added explicitly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1260); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1260); backlink

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1263)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   :mod:`sqlite3` used to implicitly commit an open transaction before DDL
   statements.  This is no longer the case.
```

Using :mod:`sqlite3` efficiently

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1268); backlink

Unknown interpreted text role "mod".

Using shortcut methods

Using the nonstandard :meth:`execute`, :meth:`executemany` and :meth:`executescript` methods of the :class:`Connection` object, your code can be written more concisely because you don't have to create the (often superfluous) :class:`Cursor` objects explicitly. Instead, the :class:`Cursor` objects are created implicitly and these shortcut methods return the cursor objects. This way, you can execute a `SELECT` statement and iterate over it directly using only a single call on the :class:`Connection` object.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1275); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1283)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/shortcut_methods.py
```

Accessing columns by name instead of by index

One useful feature of the :mod:`sqlite3` module is the built-in :class:`sqlite3.Row` class designed to be used as a row factory.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1289); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1289); *backlink*

Unknown interpreted text role "class".

Rows wrapped with this class can be accessed both by index (like tuples) and case-insensitively by name:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1295)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/rowclass.py
```

Using the connection as a context manager

Connection objects can be used as context managers that automatically commit or rollback transactions. In the event of an exception, the transaction is rolled back; otherwise, the transaction is committed:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1306)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sqlite3/ctx_manager.py
```

Footnotes

[1] The sqlite3 module is not built with loadable extension support by default, because some platforms (notably macOS) have SQLite libraries which are compiled without this feature. To get loadable extension support, you must pass the :option:`--enable-loadable-sqlite-extensions` option to configure.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)sqlite3.rst, line 1311); *backlink*

Unknown interpreted text role "option".