pymssql Documentation

Release 2.1.1

pymssql developers

May 06, 2015

Introduction

1.1 Architecture

The pymssql package consists of two modules:

- *pymssql* – use it if you care about DB-API compliance, or if you are accustomed to DB-API syntax,

- *_mssql* – use it if you care about performance and ease of use (_mssql module is easier to use than pymssql).

1.2 Supported related software

Python Python 2.x: 2.6 or newer. Python 3.x: 3.3 or newer.

FreeTDS 0.82 or newer.

Cython 0.15 or newer.

Microsoft SQL Server 2005 or newer.

1.3 Project Discussion

Discussions and support take place on pymssql mailing list here: http://groups.google.com/group/pymssql, you can participate via web, e-mail or read-only subscribing to the mailing list feeds.

This is the best place to get help, please feel free to drop by and ask a question.

1.4 Project Status

Current release: 2.x is the branch under current development. It is a complete rewrite using Cython and the latest FreeTDS libraries (which remove many of the limitations of previous versions of FreeTDS).

Legacy release: 1.0.3 is the legacy version and is no longer under active development.

Note: This documentation is for pymssql 2.x.

The document set you are reading describes exclusively the code base of pymssql 2.x and newer. All description of functionality, workarounds, limitations, dependencies, etc. of older revisions has been removed.

If you need help for building/using pymssql 1.x please refer to the old Google Code documentation Wiki.

1.5 Current Development

Official development repositories and issue trackers have been moved to GitHub at https://github.com/pymssql/pymssql.

We would be happy to have:

- A couple more developers
- Help from the community with maintenance of this documentation.

If interested, please connect with us on the mailing list.

pymssql examples

Example scripts using pymssql module.

2.1 Basic features (strict DB-API compliance)

```python
from os import getenv
import pymssql

server = getenv("PYMSSQL_TEST_SERVER")
user = getenv("PYMSSQL_TEST_USERNAME")
password = getenv("PYMSSQL_TEST_PASSWORD")

conn = pymssql.connect(server, user, password, "tempdb")
cursor = conn.cursor()
cursor.execute("""
IF OBJECT_ID('persons', 'U') IS NOT NULL
    DROP TABLE persons
CREATE TABLE persons (
    id INT NOT NULL,
    name VARCHAR(100),
    salesrep VARCHAR(100),
    PRIMARY KEY(id)
)
""")
cursor.executemany(
    "INSERT INTO persons VALUES (%d, %s, %s)",
    [(1, 'John Smith', 'John Doe'),
     (2, 'Jane Doe', 'Joe Dog'),
     (3, 'Mike T.', 'Sarah H.')])
# you must call commit() to persist your data if you don't set autocommit to True
conn.commit()

cursor.execute('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')
row = cursor.fetchone()
while row:
    print("ID=%d, Name=%s" % (row[0], row[1]))
    row = cursor.fetchone()

conn.close()
```

2.2 Iterating through results

You can also use iterators instead of while loop.

```
conn = pymssql.connect(server, user, password, "tempdb")
cursor = conn.cursor()
cursor.execute('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')

for row in cursor:
    print('row = %r' % (row,))

conn.close()
```

Note: Iterators are a pymssql extension to the DB-API.

2.3 Important note about Cursors

A connection can have only one cursor with an active query at any time. If you have used other Python DBAPI databases, this can lead to surprising results:

```
c1 = conn.cursor()
c1.execute('SELECT * FROM persons')

c2 = conn.cursor()
c2.execute('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')

print( "all persons" )
print( c1.fetchall() )   # shows result from c2 query!

print( "John Doe" )
print( c2.fetchall() )   # shows no results at all!
```

In this example, the result printed after `"all persons"` will be the result of the *second* query (the list where `salesrep='John Doe'`) and the result printed after "John Doe" will be empty. This happens because the underlying TDS protocol does not have client side cursors. The protocol requires that the client flush the results from the first query before it can begin another query.

(Of course, this is a contrived example, intended to demonstrate the failure mode. Actual use cases that follow this pattern are usually much more complicated.)

Here are two reasonable workarounds to this:

- Create a second connection. Each connection can have a query in progress, so multiple connections can execute multiple conccurent queries.

- use the fetchall() method of the cursor to recover all the results before beginning another query:

```
c1.execute('SELECT ...')
c1_list = c1.fetchall()

c2.execute('SELECT ...')
c2_list = c2.fetchall()

# use c1_list and c2_list here instead of fetching individually from
# c1 and c2
```

2.4 Rows as dictionaries

Rows can be fetched as dictionaries instead of tuples. This allows for accessing columns by name instead of index. Note the `as_dict` argument.

```
conn = pymssql.connect(server, user, password, "tempdb")
cursor = conn.cursor(as_dict=True)

cursor.execute('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')
for row in cursor:
    print("ID=%d, Name=%s" % (row['id'], row['name']))

conn.close()
```

Note: The `as_dict` parameter to `cursor()` is a pymssql extension to the DB-API.

2.5 Using the `with` statement (context managers)

You can use Python's `with` statement with connections and cursors. This frees you from having to explicitly close cursors and connections.

```
with pymssql.connect(server, user, password, "tempdb") as conn:
    with conn.cursor(as_dict=True) as cursor:
        cursor.execute('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')
        for row in cursor:
            print("ID=%d, Name=%s" % (row['id'], row['name']))
```

Note: The context manager personality of connections and cursor is a pymssql extension to the DB-API.

2.6 Calling stored procedures

As of pymssql 2.0.0 stored procedures can be called using the rpc interface of db-lib.

```
with pymssql.connect(server, user, password, "tempdb") as conn:
    with conn.cursor(as_dict=True) as cursor:
        cursor.execute("""
        CREATE PROCEDURE FindPerson
            @name VARCHAR(100)
        AS BEGIN
            SELECT * FROM persons WHERE name = @name
        END
        """)
        cursor.callproc('FindPerson', ('Jane Doe',))
        for row in cursor:
            print("ID=%d, Name=%s" % (row['id'], row['name']))
```

2.7 Using pymssql with cooperative multi-tasking systems

New in version 2.1.0.

You can use the *pymssql.set_wait_callback()* function to install a callback function you should write yourself.

This callback can yield to another greenlet, coroutine, etc. For example, for gevent, you could use its gevent.socket.wait_read() function:

```python
import gevent.socket
import pymssql

def wait_callback(read_fileno):
    gevent.socket.wait_read(read_fileno)

pymssql.set_wait_callback(wait_callback)
```

The above is useful if you're say, running a Gunicorn server with the gevent worker. With this callback in place, when you send a query to SQL server and are waiting for a response, you can yield to other greenlets and process other requests. This is super useful when you have high concurrency and/or slow database queries and lets you use less Gunicorn worker processes and still handle high concurrency.

Note: set_wait_callback() is a pymssql extension to the DB-API 2.0.

_mssql examples

Example scripts using _mssql module.

3.1 Quickstart usage of various features

```python
import _mssql
conn = _mssql.connect(server='SQL01', user='user', password='password', \
    database='mydatabase')
conn.execute_non_query('CREATE TABLE persons(id INT, name VARCHAR(100))')
conn.execute_non_query("INSERT INTO persons VALUES(1, 'John Doe')")
conn.execute_non_query("INSERT INTO persons VALUES(2, 'Jane Doe')")
```

```python
# how to fetch rows from a table
conn.execute_query('SELECT * FROM persons WHERE salesrep=%s', 'John Doe')
for row in conn:
    print "ID=%d, Name=%s" % (row['id'], row['name'])
```

New in version 2.1.0: Iterating over query results by iterating over the connection object just like it's already possible with pymssql connections is new in 2.1.0.

```python
# examples of other query functions
numemployees = conn.execute_scalar("SELECT COUNT(*) FROM employees")
numemployees = conn.execute_scalar("SELECT COUNT(*) FROM employees WHERE name LIKE 'J%'")    # note
employeedata = conn.execute_row("SELECT * FROM employees WHERE id=%d", 13)
```

```python
# how to fetch rows from a stored procedure
conn.execute_query('sp_spaceused')    # sp_spaceused without arguments returns 2 result sets
res1 = [ row for row in conn ]        # 1st result
res2 = [ row for row in conn ]        # 2nd result
```

```python
# how to get an output parameter from a stored procedure
sqlcmd = """
DECLARE @res INT
EXEC usp_mystoredproc @res OUT
SELECT @res
"""
res = conn.execute_scalar(sqlcmd)
```

```python
# how to get more output parameters from a stored procedure
sqlcmd = """
DECLARE @res1 INT, @res2 TEXT, @res3 DATETIME
```

```
EXEC usp_getEmpData %d, %s, @res1 OUT, @res2 OUT, @res3 OUT
SELECT @res1, @res2, @res3
"""
res = conn.execute_row(sqlcmd, (13, 'John Doe'))
```

```
# examples of queries with parameters
conn.execute_query('SELECT * FROM empl WHERE id=%d', 13)
conn.execute_query('SELECT * FROM empl WHERE name=%s', 'John Doe')
conn.execute_query('SELECT * FROM empl WHERE id IN (%s)', ((5, 6),))
conn.execute_query('SELECT * FROM empl WHERE name LIKE %s', 'J%')
conn.execute_query('SELECT * FROM empl WHERE name=%(name)s AND city=%(city)s', \
    { 'name': 'John Doe', 'city': 'Nowhere' } )
conn.execute_query('SELECT * FROM cust WHERE salesrep=%s AND id IN (%s)', \
    ('John Doe', (1, 2, 3)))
conn.execute_query('SELECT * FROM empl WHERE id IN (%s)', (tuple(xrange(4)),))
conn.execute_query('SELECT * FROM empl WHERE id IN (%s)', \
    (tuple([3, 5, 7, 11]),))
```

```
conn.close()
```

Please note the usage of iterators and ability to access results by column name. Also please note that parameters to connect method have different names than in `pymssql` module.

3.2 An example of exception handling

```python
import _mssql

conn = _mssql.connect(server='SQL01', user='user', password='password',
                      database='mydatabase')
try:
    conn.execute_non_query('CREATE TABLE t1(id INT, name VARCHAR(50))')
except _mssql.MssqlDatabaseException as e:
    if e.number == 2714 and e.severity == 16:
        # table already existed, so quieten the error
    else:
        raise # re-raise real error
finally:
    conn.close()
```

3.3 Custom message handlers

New in version 2.1.1.

You can provide your own message handler callback function that will be invoked by the stack with informative messages sent by the server. Set it on a per _mssql *connection* basis by using the *_mssql.MSSQLConnection.set_msghandler()* method:

```python
import _mssql

def my_msg_handler(msgstate, severity, srvname, procname, line, msgtext):
    """
    Our custom handler -- It simpy prints a string to stdout assembled from
    the pieces of information sent by the server.
    """
```

```
    print("my_msg_handler: msgstate = %d, severity = %d, procname = '%s', "
          "line = %d, msgtext = '%s'" % (msgstate, severity, procname,
                                         line, msgtext))

conn = _mssql.connect(server='SQL01', user='user', password='password')
try:
    conn.set_msghandler(my_msg_handler)  # Install our custom handler
    cnx.execute_non_query("USE mydatabase")  # It gets called at this point
finally:
    conn.close()
```

Something similar to this would be printed to the standard output:

```
my_msg_handler: msgstate = x, severity = y, procname = '', line = 1, msgtext = 'Changed database cont
```

Todo

Add an example of invoking a Stored Procedure using _mssql.

Release notes

Release notes – All breaking changes and other noteworthy things.

4.1 pymssql 2.0.0

This is a new major version of pymssql. It is totally rewritten from scratch in Cython. Our goals for this version were to:

- Provide support for Python 3.0 and newer,
- Implement support for stored procedures,
- Rewrite DB-API compilant pymssql module in C (actually in Cython) for increased performance,
- Clean up the module API and the code.

That's why we decided to bump major version number. Unfortunately new version introduces incompatible changes in API. Existing scripts may not work with it, and you'll have to audit them. If you care about compatibility, just continue using pymssql 1.0.x and slowly move to 2.0.

Project hosting has also changed. Now pymssql is developed on GitHub: http://github.com/pymssql/pymssql.

Credits for the release go to:

- Marc Abramowitz <msabramo_at_gmail_com> who joined the project in Jan 2013 and is responsible for the actual release of the 2.0 version by fixing many old tickets, coding the port to Python 3 and driving the migration to Git and GitHub.
- Randy Syring who converted the repository to Mercurial, extended tests and ported them to nose, enhanced the code in several fronts like multi-platform (compilers, OSes) compatibility, error handling, support of new data types, SQLAlchemy compatibility and expanded the documentation.
- Damien Churchill <damoxc_at_gmail_com> who set the foundations of the new Cython-based code base, release engineering, new site features like Sphinx, SimpleJSON and others,
- Andrzej Kukuła <akukula_at_gmail_com> who did all the docs, site migration, and other boring but necessary stuff.
- Jooncheol Park <jooncheol_at_gmail_com> who did develop the initial version of pymssql (until 0.5.2). Now just doing boring translation docs for Korean.

4.1.1 `pymssql` module

- Rewritten from scratch in C, you should observe even better performance than before

- dsn parameter to *pymssql.connect()* has been removed

- host parameter to *pymssql.connect()* has been renamed to server to be consistent with _mssql module

- max_conn parameter to *pymssql.connect()* has been removed

Connection class

- autocommit() function has been changed to *pymssql.Connection.autocommit* property that you can set or get its current state.

Cursor class

- fetchone_asdict() method has been removed. Just use *pymssql.connect()* with as_dict=True, then use regular *fetchone()*

- fetchmany_asdict() method has been removed.   Just use *pymssql.connect()* with as_dict=True, then use regular *fetchmany()*

- fetchall_asdict() method has been removed. Just use *pymssql.connect()* with as_dict=True, then use regular *fetchall()*

4.1.2 _mssql module

- Added native support for stored procedures (*MSSQLStoredProcedure* class)

- maxconn parameter to _mssql.connect() has been removed

- timeout and login_timeout parameter to _mssql.connect() has been added

- *get_max_connections()* and *set_max_connections()* module-level methods have been added

- Class names have changed:

Old Name	New name
MssqlException	MSSQLException
MssqlDriverException	MSSQLDriverException
MssqlDatabaseException	MSSQLDatabaseException
MssqlRowIterator	MSSQLRowIterator
MssqlConnection	MSSQLConnection

MSSQLConnection class

- Added *tds_version* property.

FreeTDS configuration

pymssql uses FreeTDS package to connect to SQL Server instances. You have to tell it how to find your database servers. The most basic info is host name, port number, and protocol version to use.

The system-wide FreeTDS configuration file is `/etc/freetds.conf` or `C:\freetds.conf`, depending upon your system. It is also possible to use a user specific configuration file, which is `$HOME/.freetds.conf` on Linux and `%APPDATA%\.freetds.conf` on Windows. Suggested contents to start with is at least:

```
[global]
    port = 1433
    tds version = 7.0
```

With this config you will be able to enter just the hostname to *pymssql.connect()* and `_mssql.connect()`:

```python
import pymssql
connection = pymssql.connect(server='mydbserver', ...)
```

Otherwise you will have to enter the portname as in:

```
connection = pymssql.connect(server='mydbserver:1433', ...)
```

To connect to instance other than the default, you have to know either the instance name or port number on which the instance listens:

```
connection = pymssql.connect(server='mydbserver\\myinstancename', ...)
# or by port number (suppose you confirmed that this instance is on port 1237)
connection = pymssql.connect(server='mydbserver:1237', ...)
```

Please see also the pymssql module reference, _mssql module reference, and FAQ pages.

For more information on configuring FreeTDS please go to http://www.freetds.org/userguide/freetdsconf.htm

5.1 Testing the connection

If you're sure that your server is reachable, but pymssql for some reason don't let you connect, you can check the connection with `tsql` utility which is part of FreeTDS package:

```
$ tsql
Usage:  tsql [-S <server> | -H <hostname> -p <port>] -U <username> [-P <password>] [-I <config file>]
(...)
$ tsql -S mydbserver -U user
```

Note: Use the above form if and only if you specified server alias for mydbserver in freetds.conf. Otherwise use the host/port notation:

```
$ tsql -H mydbserver -p 1433 -U user
```

You'll be prompted for a password and if the connection succeeds, you'll see the SQL prompt:

```
1>
```

You can then enter queries and terminate the session with `exit`.

If the connection fails, `tsql` utility will display appropriate message.

pymssql module reference

Complete documentation of `pymssql` module classes, methods and properties.

6.1 Module-level symbols

Constants, required by the DB-API 2.0 ([PEP 249](#)) specification.

pymssql.**apilevel**
    '2.0' – `pymssql` strives for compliance with DB-API 2.0.

pymssql.**paramstyle**
    'pyformat' – `pymssql` uses extended python format codes.

pymssql.**threadsafety**
    1 – Module may be shared, but not connections.

6.2 Functions

pymssql.**connect**(*server='.'*, *user=''*, *password=''*, *database=''*, *timeout=0*, *login_timeout=60*, *charset='UTF-8'*, *as_dict=False*, *host=''*, *appname=None*, *port='1433'*, *conn_properties*)
    Constructor for creating a connection to the database. Returns a *Connection* object.

        Parameters

                • **server** (*string*) – database host

                • **user** (*string*) – database user to connect as

                • **password** (*string*) – user's password

                • **database** (*string*) – the database to initially connect to

                • **timeout** (*int*) – query timeout in seconds, default 0 (no timeout)

                • **login_timeout** (*int*) – timeout for connection and login in seconds, default 60

                • **charset** (*string*) – character set with which to connect to the database

                • **as_dict** (*boolean*) – whether rows should be returned as dictionaries instead of tuples

                • **appname** (*string*) – Set the application name to use for the connection

                • **port** (*string*) – the TCP port to use to connect to the server

- **conn_properties** – SQL queries to send to the server upon connection establishment. Can be a string or another kind of iterable of strings. Default value: See `_mssql.connect`

Note: If you need to connect to Azure make sure you: * Use FreeTDS 0.91 or newer. * Specify the database name you are connecting to in the `connect()` call.

New in version 2.1.1: The ability to connect to Azure.

New in version 2.1.1: The *conn_properties* argument.

pymssql.**get_dbversion**()
    TBD

    A pymssql extension to the DB-API 2.0.

Todo

Document `pymssql` connection `get_dbversion()`.

pymssql.**set_max_connections**(*number*)
    Sets maximum number of simultaneous database connections allowed to be open at any given time. Default is 25.

    A pymssql extension to the DB-API 2.0.

pymssql.**get_max_connections**()
    Gets current maximum number of simultaneous database connections allowed to be open at any given time.

    A pymssql extension to the DB-API 2.0.

pymssql.**set_wait_callback**(*wait_callback_callable*)
    New in version 2.1.0.

    Allows pymssql to be used along cooperative multi-tasking systems and have it call a callback when it's waiting for a response from the server.

    The passed callback callable should receive one argument: The file descriptor/handle of the network socket connected to the server, so its signature must be:

```python
def wait_callback_callable(read_fileno):
    #...
    pass
```

    Its body should invoke the appropiate API of the multi-tasking framework you are using use that results in the current greenlet yielding the CPU to its siblings whilst there isn't incoming data in the socket.

    See the pymssql examples document for a more concrete example.

    A pymssql extension to the DB-API 2.0.

6.3 Connection class

class pymssql.**Connection**(*user*, *password*, *host*, *database*, *timeout*, *login_timeout*, *charset*, *as_dict*)
    This class represents an MS SQL database connection. You can create an instance of this class by calling constructor `pymssql.connect()`. It accepts the following arguments. Note that in most cases you will want to use keyword arguments, instead of positional arguments.

        Parameters

- **user** (*str*) – Database user to connect as
- **password** (*str*) – User's password
- **host** (*str*) – Database host and instance you want to connect to. Valid examples are:
    - `r'.\SQLEXPRESS'` – SQLEXPRESS instance on local machine (Windows only)
    - `r'(local)\SQLEXPRESS'` – same as above (Windows only)
    - `'SQLHOST'` – default instance at default port (Windows only)
    - `'SQLHOST'` – specific instance at specific port set up in freetds.conf (Linux/*nix only)
    - `'SQLHOST,1433'` – specified TCP port at specified host
    - `'SQLHOST:1433'` – the same as above
    - `'SQLHOST,5000'` – if you have set up an instance to listen on port 5000
    - `'SQLHOST:5000'` – the same as above

    `'.'` (the local host) is assumed if host is not provided.
- **database** (*str*) – The database you want initially to connect to, by default *SQL Server* selects the database which is set as default for specific user
- **timeout** (*int*) – Query timeout in seconds, default is 0 (wait indefinitely)
- **login_timeout** (*int*) – Timeout for connection and login in seconds, default 60
- **charset** (*str*) – Character set with which to connect to the database
- **as_dict** (*bool*) – Whether rows should be returned as dictionaries instead of tuples. You can access columns by 0-based index or by name. Please see examples

6.3.1 Connection object properties

This class has no useful properties and data members.

6.3.2 Connection object methods

Connection.**autocommit**(*status*)

Where *status* is a boolean value. This method turns autocommit mode on or off.

By default, autocommit mode is off, what means every transaction must be explicitly committed if changed data is to be persisted in the database.

You can turn autocommit mode on, what means every single operation commits itself as soon as it succeeds.

A pymssql extension to the DB-API 2.0.

Connection.**close**()

Close the connection.

Connection.**cursor**()

Return a cursor object, that can be used to make queries and fetch results from the database.

Connection.**commit**()

Commit current transaction. You must call this method to persist your data if you leave autocommit at its default value, which is `False`.

See also pymssql examples.

Connection.**rollback**()
    Roll back current transaction.

6.4 `Cursor` class

class pymssql.**Cursor**

This class represents a Cursor (in terms of Python DB-API specs) that is used to make queries against the database and obtaining results. You create Cursor instances by calling *cursor()* method on an open *Connection* connection object.

6.4.1 Cusor object properties

Cursor.**rowcount**
    Returns number of rows affected by last operation. In case of SELECT statements it returns meaningful information only after all rows have been fetched.

Cursor.**connection**
    This is the extension of the DB-API specification. Returns a reference to the connection object on which the cursor was created.

Cursor.**lastrowid**
    This is the extension of the DB-API specification. Returns identity value of last inserted row. If previous operation did not involve inserting a row into a table with identity column, None is returned.

Cursor.**rownumber**
    This is the extension of the DB-API specification. Returns current 0-based index of the cursor in the result set.

6.4.2 Cusor object methods

Cursor.**close**()
    Close the cursor. The cursor is unusable from this point.

Cursor.**execute**(*operation*)
Cursor.**execute**(*operation*, *params*)
    operation is a string and *params*, if specified, is a simple value, a tuple, or None.

    Performs the operation against the database, possibly replacing parameter placeholders with provided values. This should be preferred method of creating SQL commands, instead of concatenating strings manually, what makes a potential of SQL Injection attacks. This method accepts the same formatting as Python's builtin string interpolation operator.

    If you call execute() with one argument, the % sign loses its special meaning, so you can use it as usual in your query string, for example in LIKE operator. See the examples.

    You must call *Connection.commit()* after execute() or your data will not be persisted in the database. You can also set connection.autocommit if you want it to be done automatically. This behaviour is required by DB-API, if you don't like it, just use the *_mssql* module instead.

Cursor.**executemany**(*operation*, *params_seq*)
    operation is a string and *params_seq* is a sequence of tuples (e.g. a list). Execute a database operation repeatedly for each element in parameter sequence.

Cursor.**fetchone**()
    Fetch the next row of a query result, returning a tuple, or a dictionary if as_dict was passed to

pymssql.connect(), or None if no more data is available. Raises OperationalError (**PEP 249#operationalerror**) if previous call to execute*() did not produce any result set or no call was issued yet.

Cursor.**fetchmany**(*size=None*)
Fetch the next batch of rows of a query result, returning a list of tuples, or a list of dictionaries if *as_dict* was passed to *pymssql.connect()*, or an empty list if no more data is available. You can adjust the batch size using the *size* parameter, which is preserved across many calls to this method. Raises OperationalError (**PEP 249#operationalerror**) if previous call to execute*() did not produce any result set or no call was issued yet.

Cursor.**fetchall**()
Fetch all remaining rows of a query result, returning a list of tuples, or a list of dictionaries if as_dict was passed to pymssql.connect(), or an empty list if no more data is available. Raises OperationalError (**PEP 249#operationalerror**) if previous call to execute*() did not produce any result set or no call was issued yet.

Cursor.**nextset**()
This method makes the cursor skip to the next available result set, discarding any remaining rows from the current set. Returns True value if next result is available, None if not.

Cursor.**__iter__**()
Cursor.**next**()
These methods facilitate Python iterator protocol. You most likely will not call them directly, but indirectly by using iterators.

A pymssql extension to the DB-API 2.0.

Cursor.**setinputsizes**()
Cursor.**setoutputsize**()
These methods do nothing, as permitted by DB-API specs.

Todo

Document all pymssql PEP 249-mandated exceptions.

_mssql module reference

Complete documentation of _mssql module classes, methods and properties.

7.1 Module-level symbols

Variables whose values you can change to alter behavior on a global basis.

_mssql.**login_timeout**
> Timeout for connection and login in seconds, default 60.

_mssql.**min_error_severity**
> Minimum severity of errors at which to begin raising exceptions. The default value of 6 should be appropriate in most cases.

7.2 Functions

_mssql.**set_max_connections**(*number*)
> Sets maximum number of simultaneous connections allowed to be open at any given time. Default is 25.

_mssql.**get_max_connections**()
> Gets current maximum number of simultaneous connections allowed to be open at any given time.

7.3 **MSSQLConnection class**

class _mssql.**MSSQLConnection**
> This class represents an MS SQL database connection. You can make queries and obtain results through a database connection.
>
> You can create an instance of this class by calling _mssql.connect(). It accepts the following arguments. Note that you can use keyword arguments, instead of positional arguments.
>
> > **Parameters**
> >
> > - **server** (*str*) – Database server and instance you want to connect to. Valid examples are:
> >
> >   - r'.\SQLEXPRESS' – SQLEXPRESS instance on local machine (Windows only)
> >
> >   - r'(local)\SQLEXPRESS' – Same as above (Windows only)
> >
> >   - 'SQLHOST' – Default instance at default port (Windows only)

- – 'SQLHOST' – Specific instance at specific port set up in freetds.conf (Linux/*nix only)

- – 'SQLHOST,1433' – Specified TCP port at specified host

- – 'SQLHOST:1433' – The same as above

- – 'SQLHOST,5000' – If you have set up an instance to listen on port 5000

- – 'SQLHOST:5000' – The same as above

- **user** (*str*) – Database user to connect as

- **password** (*str*) – User's password

- **charset** (*str*) – Character set name to set for the connection.

- **database** (*str*) – The database you want to initially to connect to; by default, *SQL Server* selects the database which is set as the default for the specific user

- **appname** (*str*) – Set the application name to use for the connection

- **port** (*str*) – the TCP port to use to connect to the server

- **tds_version** (*str*) – TDS protocol version to ask for. Default value: '7.1'

- **conn_properties** – SQL queries to send to the server upon connection establishment. Can be a string or another kind of iterable of strings. Default value:

```
SET ARITHABORT ON;
SET CONCAT_NULL_YIELDS_NULL ON;
SET ANSI_NULLS ON;
SET ANSI_NULL_DFLT_ON ON;
SET ANSI_PADDING ON;
SET ANSI_WARNINGS ON;
SET ANSI_NULL_DFLT_ON ON;
SET CURSOR_CLOSE_ON_COMMIT ON;
SET QUOTED_IDENTIFIER ON;
SET TEXTSIZE 2147483647; -- http://msdn.microsoft.com/en-us/library/aa259190%28v=sql.80%29.aspx
```

New in version 2.1.1: The *conn_properties* argument.

Changed in version 2.1.1: Before 2.1.1, the initialization queries now specified by *conn_properties* wasn't customizable and its value was hard-coded to the literal shown above.

Note: If you need to connect to Azure make sure you use FreeTDS 0.91 or newer.

New in version 2.1.1: The ability to connect to Azure.

7.3.1 `MSSQLConnection` object properties

MSSQLConnection.**connected**
    True if the connection object has an open connection to a database, False otherwise.

MSSQLConnection.**charset**
    Character set name that was passed to _mssql.connect().

MSSQLConnection.**identity**
    Returns identity value of last inserted row. If previous operation did not involve inserting a row into a table with identity column, None is returned. Example usage – assume that persons table contains an identity column in addition to name column:

```
        conn.execute_non_query("INSERT INTO persons (name) VALUES('John Doe')")
        print "Last inserted row has id = " + conn.identity
```

MSSQLConnection.**query_timeout**
> Query timeout in seconds, default is 0, which means to wait indefinitely for results. Due to the way DB-Library for C works, setting this property affects all connections opened from the current Python script (or, very technically, all connections made from this instance of dbinit()).

MSSQLConnection.**rows_affected**
> Number of rows affected by last query. For SELECT statements this value is only meaningful after reading all rows.

MSSQLConnection.**debug_queries**
> If set to true, all queries are printed to stderr after formatting and quoting, just before being sent to *SQL Server*. It may be helpful if you suspect problems with formatting or quoting.

MSSQLConnection.**tds_version**
> The TDS version used by this connection. Can be one of 4.2, 5.0 7.0, 8.0 and 7.2.

> **Warning:** For historical and backward compatibility reasons, the value used to represent TDS 7.1 is 8.0. This will change with pymssql 2.2.0 when it will be fixed to be 7.1 for correctness and consistency.

7.3.2 `MSSQLConnection` object methods

MSSQLConnection.**cancel**()
> Cancel all pending results from the last SQL operation. It can be called more than one time in a row. No exception is raised in this case.

MSSQLConnection.**close**()
> Close the connection and free all memory used. It can be called more than one time in a row. No exception is raised in this case.

MSSQLConnection.**execute_query**(*query_string*)
MSSQLConnection.**execute_query**(*query_string*, *params*)
> This method sends a query to the *MS SQL Server* to which this object instance is connected. An exception is raised on failure. If there are pending results or rows prior to executing this command, they are silently discarded.

> After calling this method you may iterate over the connection object to get rows returned by the query.

> You can use Python formatting and all values get properly quoted. Please see examples for details.

> This method is intented to be used on queries that return results, i.e. SELECT.

MSSQLConnection.**execute_non_query**(*query_string*)
MSSQLConnection.**execute_non_query**(*query_string*, *params*)
> This method sends a query to the *MS SQL Server* to which this object instance is connected. After completion, its results (if any) are discarded. An exception is raised on failure. If there are pending results or rows prior to executing this command, they are silently discarded.

> You can use Python formatting and all values get properly quoted. Please see examples for details.

> This method is useful for INSERT, UPDATE, DELETE, and for Data Definition Language commands, i.e. when you need to alter your database schema.

MSSQLConnection.**execute_scalar**(*query_string*)
MSSQLConnection.**execute_scalar**(*query_string*, *params*)
> This method sends a query to the *MS SQL Server* to which this object instance is connected, then returns first

column of first row from result. An exception is raised on failure. If there are pending results or rows prior to executing this command, they are silently discarded.

You can use Python formatting and all values get properly quoted. Please see examples for details.

This method is useful if you want just a single value from a query, as in the example below. This method works in the same way as `iter(conn).next()[0]`. Remaining rows, if any, can still be iterated after calling this method.

Example usage:

```
count = conn.execute_scalar("SELECT COUNT(*) FROM employees")
```

MSSQLConnection.**execute_row**(*query_string*)
MSSQLConnection.**execute_row**(*query_string*, *params*)

This method sends a query to the *MS SQL Server* to which this object instance is connected, then returns first row of data from result. An exception is raised on failure. If there are pending results or rows prior to executing this command, they are silently discarded.

You can use Python formatting and all values get properly quoted. Please see examples for details.

This method is useful if you want just a single row and don't want or don't need to iterate over the connection object. This method works in the same way as `iter(conn).next()` to obtain single row. Remaining rows, if any, can still be iterated after calling this method.

Example usage:

```
empinfo = conn.execute_row("SELECT * FROM employees WHERE empid=10")
```

MSSQLConnection.**get_header**()

This method is infrastructure and doesn't need to be called by your code. It gets the Python DB-API compliant header information. Returns a list of 7-element tuples describing current result header. Only name and DB-API compliant type is filled, rest of the data is `None`, as permitted by the specs.

MSSQLConnection.**init_procedure**(*name*)

Create an MSSQLStoredProcedure object that will be used to invoke thestored procedure with the given name.

MSSQLConnection.**nextresult**()

Move to the next result, skipping all pending rows. This method fetches and discards any rows remaining from current operation, then it advances to next result (if any). Returns `True` value if next set is available, `None` otherwise. An exception is raised on failure.

MSSQLConnection.**select_db**(*dbname*)

This function makes the given database the current one. An exception is raised on failure.

MSSQLConnection.**__iter__**()
MSSQLConnection.**next**()

New in version 2.1.0.

These methods implement the Python iterator protocol. You most likely will not call them directly, but indirectly by using iterators.

MSSQLConnection.**set_msghandler**(*handler*)

New in version 2.1.1.

This method allows setting a message handler function for the connection to allow a client to gain access to the messages returned from the server.

The signature of the message handler function *handler* passed to this method must be:

```
def my_msg_handler(msgstate, severity, srvname, procname, line, msgtext):
    # The body of the message handler.
```

msgstate, *severity* and *line* will be integers, *srvname*, *procname* and *msgtext* will be strings.

7.4 `MSSQLStoredProcedure` class

class _mssql.**MSSQLStoredProcedure**

This class represents a stored procedure. You create an object of this class by calling the *init_procedure()* method on *MSSQLConnection* object.

7.4.1 `MSSQLStoredProcedure` object properties

MSSQLStoredProcedure.**connection**

An underlying MSSQLConnection object.

MSSQLStoredProcedure.**name**

The name of the procedure that this object represents.

MSSQLStoredProcedure.**parameters**

The parameters that have been bound to this procedure.

7.4.2 `MSSQLStoredProcedure` object methods

MSSQLStoredProcedure.**bind**(*value*, *dbtype*, *name=None*, *output=False*, *null=False*, *max_length=-1*)

This method binds a parameter to the stored procedure. *value* and *dbtype* are mandatory arguments, the rest is optional.

Parameters

- **value** – Is the value to store in the parameter

- **dbtype** – Is one of: SQLBINARY, SQLBIT, SQLBITN, SQLCHAR, SQLDATETIME, SQLDATETIM4, SQLDATETIMN, SQLDECIMAL, SQLFLT4, SQLFLT8, SQLFLTN, SQLIMAGE, SQLINT1, SQLINT2, SQLINT4, SQLINT8, SQLINTN, SQLMONEY, SQLMONEY4, SQLMONEYN, SQLNUMERIC, SQLREAL, SQLTEXT, SQLVARBINARY, SQLVARCHAR, SQLUUID

- **name** – Is the name of the parameter

- **output** – Is the direction of the parameter: True indicates that it is also an output parameter that returns value after procedure execution

- **null** – TBD

Todo

Document null MSSQLStoredProcedure.bind() argument.

Parameters **max_length** – Is the maximum data length for this parameter to be returned from the stored procedure.

MSSQLStoredProcedure.**execute**()

Execute the stored procedure.

7.5 Module-level exceptions

Exception hierarchy:

```
MSSQLException
|
+-- MSSQLDriverException
|
+-- MSSQLDatabaseException
```

exception _mssql.**MSSQLDriverException**

MSSQLDriverException is raised whenever there is a problem within _mssql – e.g. insufficient memory for data structures, and so on.

exception _mssql.**MSSQLDatabaseException**

MSSQLDatabaseException is raised whenever there is a problem with the database – e.g. query syntax error, invalid object name and so on. In this case you can use the following properties to access details of the error:

number

The error code, as returned by *SQL Server*.

severity

The so-called severity level, as returned by *SQL Server*. If value of this property is less than the value of _mssql.min_error_severity, such errors are ignored and exceptions are not raised.

state

The third error code, as returned by *SQL Server*.

message

The error message, as returned by *SQL Server*.

You can find an example of how to use this data at the bottom of _mssql examples page.

Migrating from 1.x to 2.x

Because of the DB-API standard and because effort was made to make the interface of pymssql 2.x similar to that of pymssql 1.x, there are only a few differences and usually upgrading is pretty easy.

There are a few differences though...

8.1 `str` vs. `unicode`

Note that we are talking about Python 2, because pymssql 1.x doesn't work on Python 3.

pymssql 1.x will return `str` instances:

```
>>> pymssql.__version__
'1.0.3'
>>> conn.as_dict = True
>>> cursor = conn.cursor()
>>> cursor.execute("SELECT 'hello' AS str FROM foo")
>>> cursor.fetchall()
[{0: 'hello', 'str': 'hello'}]
```

whereas pymssql 2.x will return `unicode` instances:

```
>>> pymssql.__version__
u'2.0.1.2'
>>> conn.as_dict = True
>>> cursor = conn.cursor()
>>> cursor.execute("SELECT 'hello' AS str FROM foo")
>>> cursor.fetchall()
[{u'str': u'hello'}]
```

If your application has code that deals with `str` and `unicode` differently, then you may run into issues.

You can always convert a `unicode` to a `str` by encoding:

```
>>> cursor.execute("SELECT 'hello' AS str FROM foo")
>>> s = cursor.fetchone()['str']
>>> s
u'hello'
>>> s.encode('utf-8')
'hello'
```

8.2 Handling of `uniqueidentifier` columns

SQL Server has a data type called uniqueidentifier.

In pymssql 1.x, `uniqueidentifier` columns are returned in results as byte strings with 16 bytes; if you want a `uuid.UUID` instance, then you have to construct it yourself from the byte string:

```
>>> cursor.execute("SELECT * FROM foo")
>>> id_value = cursor.fetchone()['uniqueidentifier']
>>> id_value
'j!\xcf\x14D\xce\xe6B\xab\xe0\xd9\xbey\x0cMK'
>>> type(id_value)
<type 'str'>
>>> len(id_value)
16
>>> import uuid
>>> id_uuid = uuid.UUID(bytes_le=id_value)
>>> id_uuid
UUID('14cf216a-ce44-42e6-abe0-d9be790c4d4b')
```

In pymssql 2.x, `uniqueidentifier` columns are returned in results as instances of `uuid.UUID` and if you want the bytes, like in pymssql 1.x, you have to use `uuid.UUID.bytes_le` to get them:

```
>>> cursor.execute("SELECT * FROM foo")
>>> id_value = cursor.fetchone()['uniqueidentifier']
>>> id_value
UUID('14cf216a-ce44-42e6-abe0-d9be790c4d4b')
>>> type(id_value)
<class 'uuid.UUID'>
>>> id_value.bytes_le
'j!\xcf\x14D\xce\xe6B\xab\xe0\xd9\xbey\x0cMK'
```

8.3 Arguments to `pymssql.connect`

The arguments are a little bit different. Some notable differences:

In pymssql 1.x, the parameter to specify the host is called `host` and it can contain a host and port – e.g.:

```
conn = pymssql.connect(host='SQLHOST:1433')   # specified TCP port at a host
```

There are some other syntaxes for the `host` parameter that allow using a comma instead of a colon to delimit host and port, to specify Windows hosts, to specify a specific SQL Server instance, etc.

```
conn = pymssql.connect(host=r'SQLHOST,5000')   # specified TCP port at a host
conn = pymssql.connect(host=r'(local)\SQLEXPRESS')   # named instance on local machine [Win]
```

In pymssql 2.x, the `host` parameter is supported (I am unsure if it has all of the functionality of pymssql 1.x). There is also a parameter to specify the host that is called `server`. There is a separate parameter called `port`.

```
conn = pymssql.connect(server='SQLHOST', port=1500)
```

8.4 Parameter substitution

For parameter substitution, pymssql 2.x supports the `format` and `pyformat` PEP 249 paramstyles.

Note that for `pyformat`, PEP 249 only shows the example of a string substitution – e.g.:

```
%(name)s
```

It is not clear from PEP 249 whether other types should be supported, like:

```
%(name)d
%(name)f
```

However, in this mailing list thread, the general consensus is that the string format should be the only one required.

Note that pymssql 2.x does not support `%(name)d`, whereas pymssql 1.x did. So you may have to change code that uses this notation:

```
>>> pymssql.__version__
u'2.0.1.2'
>>> pymssql.paramstyle
'pyformat'

>>> cursor.execute("select 'hello' where 1 = %(name)d", dict(name=1))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pymssql.pyx", line 430, in pymssql.Cursor.execute (pymssql.c:5900)
    if not self._source._conn.nextresult():
pymssql.ProgrammingError: (102, "Incorrect syntax near '('.
DB-Lib error message 20018, severity 15:\n
General SQL Server error: Check messages from the SQL Server\n")
```

to:

```
>>> cursor.execute("select 'hello' where '1' = %(name)s", dict(name='1'))
>>> cursor.fetchall()
[(u'hello',)]
```

or:

```
>>> cursor.execute("select 'hello' where 1 = %d", 1)
>>> cursor.fetchall()
[(u'hello',)]
```

Examples of this problem:

- Google Group post: paramstyle changed?
- GitHub issue #155: pymssql 2.x does not support "%(foo)d" parameter substitution style; pymssql 1.x did

Frequently asked questions

9.1 Cannot connect to SQL Server

If your script can't connect to a *SQL Server* instance, try the following:

- Check that you can connect with another tool.

    If you are using FreeTDS, then you can use the included `tsql` command to try to connect – it looks
    like this:

```
$ tsql -H sqlserverhost -p 1433 -U user -P password -D tempdb
locale is "en_US.UTF-8"
locale charset is "UTF-8"
using default charset "UTF-8"
Setting tempdb as default database in login packet
1> SELECT @@VERSION
2> GO

Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.1 <X64> (Build 7601: Service Pack 1)

(1 row affected)
```

    Note: Note that I use the `-H` option rather than the `-S` option to `tsql`. This is because with `-H`, it
    will bypass reading settings from the `freetds.conf` file like `port` and `tds version`, and so
    this is more similar to what happens with pymssql.

    If you **can't** connect with `tsql` or other tools, then the problem is probably not pymssql; you prob-
    ably have a problem with your server configuration (see below), FreeTDS Configuration, network,
    etc.

    If you **can** connect with `tsql`, then you should be able to connect with pymssql with something like
    this:

```
>>> import pymssql
>>> conn = pymssql.connect(
...     server="sqlserverhost",
...     port=1433,
...     user="user",
...     password="password",
...     database="tempdb")
```

```
>>> conn
<pymssql.Connection object at 0x10107a3f8>
>>> cursor = conn.cursor()
>>> cursor.execute("SELECT @@VERSION")
>>> print(cursor.fetchone()[0])
Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
  Feb 10 2012 19:39:15
  Copyright (c) Microsoft Corporation
  Developer Edition (64-bit) on Windows NT 6.1 <X64> (Build 7601: Service Pack 1)
```

If something like the above doesn't work, then you can try to diagnose by setting one or both of the following FreeTDS environment variables that control logging:

– TDSDUMP

– TDSDUMPCONFIG

Either or both of these can be set. They can be set to a filename or to stdout or stderr.

These will cause FreeTDS to output a ton of information about what it's doing and you may very well spot that it's not using the port that you expected or something similar. For example:

```
>>> import os
>>> os.environ['TDSDUMP'] = 'stdout'
>>>
>>> import pymssql
>>> conn = pymssql.connect(server="sqlserverhost")
log.c:194:Starting log file for FreeTDS 0.92.dev.20140102
  on 2014-01-09 14:05:32 with debug flags 0x4fff.
config.c:731:Setting 'dump_file' to 'stdout' from $TDSDUMP.
...
dblib.c:7934:20013: "Unknown host machine name"
dblib.c:7955:"Unknown host machine name", client returns 2 (INT_CANCEL)
util.c:347:tdserror: client library returned TDS_INT_CANCEL(2)
util.c:370:tdserror: returning TDS_INT_CANCEL(2)
login.c:418:IP address pointer is empty
login.c:420:Server sqlserverhost:1433 not found!
...
```

Note: Note that pymssql will use a default port of 1433, despite any ports you may have specified in your freetds.conf file. So if you have SQL Server running on a port other than 1433, you must explicitly specify the port in your call to pymssql.connect. You cannot rely on it to pick up the port in your freetds.conf, even though tsql -S might do this. This is why I recommend using tsql -H instead for diagnosing connection problems.

It is also useful to know that tsql -C will output a lot of information about FreeTDS, that can be useful for diagnosing problems:

```
$ tsql -C
Compile-time settings (established with the "configure" script)
                            Version: freetds v0.92.dev.20140102
             freetds.conf directory: /usr/local/etc
     MS db-lib source compatibility: no
         Sybase binary compatibility: no
                       Thread safety: yes
                       iconv library: yes
                         TDS version: 5.0
                               iODBC: yes
                            unixodbc: no
```

```
                    SSPI "trusted" logins: no
                              Kerberos: no
                               OpenSSL: no
                                GnuTLS: no
```

- By default *SQL Server* 2005 and newer don't accept remote connections, you have to use *SQL Server Surface Area Configuration* and/or *SQL Server Configuration Manager* to enable specific protocols and network adapters; don't forget to restart *SQL Server* after making these changes,

- If *SQL Server* is on a remote machine, check whether connections are not blocked by any intermediate firewall device, firewall software, antivirus software, or other security facility,

- If you use pymssql on Linux/Unix with FreeTDS, check that FreeTDS's configuration is ok and that it can be found by pymssql. The easiest way is to test connection using `tsql` utility which can be found in FreeTDS package. See FreeTDS Configuration for more info,

- If you use pymssql on Windows and the server is on local machine, you can try the following command from the command prompt:

```
REG ADD HKLM\Software\Microsoft\MSSQLServer\Client /v SharedMemoryOn /t REG_DWORD /d 1 /f
```

9.2 Returned dates are not correct

If you use pymssql on Linux/*nix and you suspect that returned dates are not correct, please read the FreeTDS and dates page.

9.3 Queries return no rows

There is a known issue where some versions of pymssql 1.x (pymssql 1.0.2 is where I've seen this) work well with FreeTDS 0.82, but return no rows when used with newer versions of FreeTDS, such as FreeTDS 0.91. At SurveyMonkey, we ran into this problem when we were using pymssql 1.0.2 and then upgraded servers from Ubuntu 10 (which includes FreeTDS 0.82) to Ubuntu 12 (which includes FreeTDS 0.91).

E.g.:

```
>>> import pymssql
>>> pymssql.__version__
'1.0.2'
>>> conn = pymssql.connect(host='127.0.0.1:1433', user=user,
...                        password=password, database='tempdb')
>>> cursor = conn.cursor()
>>> cursor.execute('SELECT 1')
>>> cursor.fetchall()
[]
```

See GitHub issue 137: pymssql 1.0.2: No result rows are returned from queries with newer versions of FreeTDS.

There are two way to fix this problem:

1. (Preferred) Upgrade to pymssql 2.x. pymssql 1.x is not actively being worked on. pymssql 2.x is rewritten in Cython, is actively maintained, and offers better performance, Python 3 support, etc. E.g.:

```
>>> import pymssql
>>> pymssql.__version__
u'2.0.1.2'
>>> conn = pymssql.connect(host='127.0.0.1:1433', user=user,
```

```
...                                  password=password, database='tempdb')
>>> cursor = conn.cursor()
>>> cursor.execute('SELECT 1')
>>> cursor.fetchall()
[(1,)]
```

2. Upgrade to pymssql 1.0.3. This is identical to pymssql 1.0.2 except that it has a very small change that makes it so that it works with newer versions of FreeTDS as well as older versions.

   E.g.:

```
>>> import pymssql
>>> pymssql.__version__
'1.0.3'
>>> conn = pymssql.connect(host='127.0.0.1:1433', user=user,
...                          password=password, database='tempdb')
>>> cursor = conn.cursor()
>>> cursor.execute('SELECT 1')
>>> cursor.fetchall()
[(1,)]
```

9.4 Results are missing columns

One possible cause of your result rows missing columns is if you are using a connection or cursor with `as_dict=True` and your query has columns without names – for example:

```
>>> cursor = conn.cursor(as_dict=True)
>>> cursor.execute("SELECT MAX(x) FROM (VALUES (1), (2), (3)) AS foo(x)")
>>> cursor.fetchall()
[{}]
```

Whoa, what happened to `MAX(x)` ?!?!

In this case, pymssql does not know what name to use for the dict key, so it omits the column.

The solution is to supply a name for all columns – e.g.:

```
>>> cursor.execute("SELECT MAX(x) AS [MAX(x)] FROM (VALUES (1), (2), (3)) AS foo(x)")
>>> cursor.fetchall()
[{u'MAX(x)': 3}]
```

This behavior was changed in https://github.com/pymssql/pymssql/pull/160 – with this change, if you specify `as_dict=True` and omit column names, an exception will be raised:

```
>>> cursor.execute("SELECT MAX(x) FROM (VALUES (1), (2), (3)) AS foo(x)")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pymssql.pyx", line 426, in pymssql.Cursor.execute (pymssql.c:5828)
    raise ColumnsWithoutNamesError(columns_without_names)
pymssql.ColumnsWithoutNamesError: Specified as_dict=True and there are columns with no names: [0]
```

Examples of this problem:

- Google Group post: pymssql with MAX(values) function does not appear to work

9.5 pymssql does not unserialize `DATE` and `TIME` columns to `datetime.date` and `datetime.time` instances

You may notice that pymssql will unserialize a `DATETIME` column to a `datetime.datetime` instance, but it will unserialize `DATE` and `TIME` columns as simple strings. For example:

```
>>> cursor.execute("""
... CREATE TABLE dates_and_times (
...     datetime DATETIME,
...     date DATE,
...     time TIME,
... )
... """)
>>> cursor.execute("INSERT INTO dates_and_times VALUES (GETDATE(), '20140109', '6:17')")
>>> cursor.execute("SELECT * FROM dates_and_times")
>>> cursor.fetchall()
[{u'date': u'2014-01-09', u'time': u'06:17:00.0000000',
  u'datetime': datetime.datetime(2014, 1, 9, 12, 41, 59, 403000)}]
>>> cursor.execute("DROP TABLE dates_and_times")
```

Yep, so the problem here is that `DATETIME` has been supported by FreeTDS for a long time, but `DATE` and `TIME` are newer types in SQL Server and Microsoft never added support for them to db-lib and FreeTDS never added support for them either.

There was some discussion of adding it to FreeTDS, but I think that stalled. See this thread:

http://lists.ibiblio.org/pipermail/freetds/2013q2/thread.html#28348

So we would need to get FreeTDS to support it and then the user would have to make sure to use a very recent FreeTDS (unless pymssql links in said version of FreeTDS).

Links:

- https://github.com/pymssql/pymssql/issues/156

- Discussion of adding support for DATE and TIME to FreeTDS

9.6 Shared object "libsybdb.so.3" not found

On Linux/*nix you may encounter the following behaviour:

```
>>> import _mssql
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ImportError: Shared object "libsybdb.so.3" not found
```

It may mean that the FreeTDS library is unavailable, or that the dynamic linker is unable to find it. Check that it is installed and that the path to `libsybdb.so` is in `/etc/ld.so.conf` file. Then do `ldconfig` as root to refresh linker database. On Solaris, I just set the `LD_LIBRARY_PATH` environment variable to the directory with the library just before launching Python.

pymssql 2.x bundles the FreeTDS `sybdb` library for supported platforms. This error may show up in 2.x versions if you are trying to build with your own FreeTDS.

9.7 "DB-Lib error message 20004, severity 9: Read from SQL server failed" error appears

On Linux/*nix you may encounter the following behaviour:

```
>>> import _mssql
>>> c=_mssql.connect('hostname:portnumber','user','pass')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
_mssql.DatabaseException: DB-Lib error message 20004, severity 9:
Read from SQL server failed.
DB-Lib error message 20014, severity 9:
Login incorrect.
```

It may happen when one of the following is true:

- `freetds.conf` file cannot be found,

- `tds version` in `freetds.conf` file is not `7.0` or `4.2`,

- any character set is specified in `freetds.conf`,

- an unrecognized character set is passed to `_mssql.connect()` or `pymssql.connect()` method.

`"Login incorrect"` following this error is spurious, real `"Login incorrect"` messages has code=18456 and severity=14.

9.8 More troubleshooting

If the above hasn't covered the problem you can send a message describing it to the pymssql mailing list. You can also consult FreeTDS troubleshooting page for issues related to the TDS protocol.

Building and developing pymssql

10.1 Building

To build pymssql you should have:

- python >= 2.6 including development files. Please research your OS usual software distribution channels, e.g, `python-dev` or `python-devel` packages.

- Cython >= 0.15

- FreeTDS >= 0.91 including development files. Please research your OS usual software distribution channels, e.g, `freetds-dev` or `freetds-devel` packages.

10.1.1 Windows

MinGW

Add to the above requirements:

- MinGW

then you can run:

```
python setup.py build -c mingw32
```

which will build pymssql in the normal python fashion.

MS Visual C++

Environment Setup:

The commands below should be ran inside a Visual Studio command prompt or a command prompt window where the `vcsvars*.bat` file has been previously run so it can set the needed environment vars.

Building FreeTDS:

Build FreeTDS from the current or stable tarball.

Use `nmake` (included with VS C++) to build FreeTDS. To do that,

Define in the environment or on the command line:

---

1. CONFIGURATION = debug/release

2. PLATFORM = win32/x64

These will determine what is built and where outputs are placed.

Example invocations:

```
nmake.exe -f Nmakefile -nologo PLATFORM=win32 CONFIGURATION=debug
nmake.exe -f Nmakefile -nologo build-win32d
```

Fixing build errors: I ran into a couple build errors when using VS 2008, see the following links for resolutions:

- http://www.freetds.org/userguide/osissues.htm

- http://lists.ibiblio.org/pipermail/freetds/2010q4/026343.html

When this is done, the following files should be available (depending on CONFIGURATION and PLATFORM used above):

```
src\dblib\<PLATFORM>\<CONFIGURATION>\db-lib.lib
src\tds\<PLATFORM>\<CONFIGURATION>\tds.lib
```

for example:

```
src\dblib\win32\release\db-lib.lib
src\tds\win32\release\tds.lib
```

Those files should then be copied to:

```
<pymssql root>\freetds\vs2008_<bitness>\lib\
```

for example:

```
<pymssql root>\freetds\vs2008_32\lib\
<pymssql root>\freetds\vs2008_64\lib\
```

The location obviously depends on whether you are performing a 32 or 64 bit build.

Note: This process is currently only tested with Visual Studio 2008 targeting a 32-bit build. If you run into problems, please post to the mailing list.

Then you can simply run:

```
python setup.py build
```

or other setup.py commands as needed.

10.1.2 Unix

To build on Unix you must also have:

- gcc

Then you can simply run:

```
python setup.py build
```

or other setup.py commands as needed.

10.2 Testing

> **Danger:** ALL DATA IN TESTING DBS WILL BE DELETED !!!!

You will need to install two additional packages for testing:

```
easy_install pytest SQLAlchemy
```

You should build the package with:

```
python setup.py develop
```

You need to setup a `tests.cfg` file in `tests/` with the correct DB connection information for your environment:

```
cp tests/tests.cfg.tpl tests/tests.cfg
vim|emacs|notepad tests/tests.cfg
```

To run the tests:

```
cd tests # optional
py.test
```

Which will go through and run all the tests with the settings from the `DEFAULT` section of `tests.cfg`.

To run with a different `tests.cfg` section:

```
py.test --pymssql-section=<secname>
```

example:

```
py.test --pymssql-section=AllTestsWillRun
```

to avoid slow tests:

```
py.test -m "not slow"
```

to select specific tests to run:

```
py.test tests/test_types.py
py.test tests/test_types.py tests/test_sprocs.py
py.test tests/test_types.py::TestTypes
py.test tests/test_types.py::TestTypes::test_image
```

FreeTDS and dates

Explanation of how pymssql and FreeTDS can break dates.

11.1 Summary

Make sure that FreeTDS is compiled with `--enable-msdblib configure` option, or your queries will return wrong dates – `"2010-00-01"` instead of `"2010-01-01"`.

11.2 Details

There's an obscure problem on Linux/*nix that results in dates shifted back by 1 month. This behaviour is caused by different `dbdatecrack()` prototypes in *Sybase Open Client DB-Library/C* and the *Microsoft SQL DB Library for C*. The first one returns month as 0..11 whereas the second gives month as 1..12. See this FreeTDS mailing list post, Microsoft manual for dbdatecrack(), and Sybase manual for dbdatecrack() for details.

FreeTDS, which is used on Linux/*nix to connect to *Sybase* and *MS SQL* servers, tries to imitate both modes:

- Default behaviour, when compiled without `--enable-msdblib`, gives `dbdatecrack()` which is Sybase-compatible,

- When configured with `--enable-msdblib`, the `dbdatecrack()` function is compatible with *MS SQL* specs.

pymssql requires *MS SQL* mode, evidently. Unfortunately at runtime we can't reliably detect which mode FreeTDS was compiled in (as of FreeTDS 0.63). Thus at runtime it may turn out that dates are not correct. If there was a way to detect the setting, pymssql would be able to correct dates on the fly.

If you can do nothing about FreeTDS, there's a workaround. You can redesign your queries to return string instead of bare date:

```
SELECT datecolumn FROM tablename
```

can be rewritten into:

```
SELECT CONVERT(CHAR(10),datecolumn,120) AS datecolumn FROM tablename
```

This way SQL will send you string representing the date instead of binary date in datetime or smalldatetime format, which has to be processed by FreeTDS and pymssql.

Change log

```
Change Log
==========

Version 2.1.1 - 2014-11-25 - Ramiro Morales
============================================

Features
--------

- Custom message handlers (GH-139)

  The DB-Library API includes a callback mechanism so applications can provide
  functions known as *message handlers* that get passed informative messages
  sent by the server which then can be logged, shown to the user, etc.

  ``_mssql`` now allows you to install your own *message handlers* written in
  Python. See the ``_msssql`` examples and reference sections of the
  documentation for more details.

  Thanks Marc Abramowitz.

- Compatibility with Azure

  It is now possible to transparently connect to `SQL Server instances`_
  accessible as part of the Azure_ cloud services.

  .. note:: If you need to connect to Azure make sure you use FreeTDS 0.91 or
            newer.

- Customizable per-connection initialization SQL clauses (both in ``pymssql``
  and ``_mssql``) (GH-97)

  It is now possible to customize the SQL statements sent right after the
  connection is established (e.g. ``'SET ANSI_NULLS ON;'``). Previously
  it was a hard-coded list of queries. See the ``_mssql.MSSQLConnection``
  documentation for more details.

  Thanks Marc Abramowitz.

- Added ability to handle instances of ``uuid.UUID`` passed as parameters for
  SQL queries both in ``pymssql`` and ``_mssql``. (GH-209)

  Thanks Marat Mavlyutov.
```

```
- Allow using `SQL Server autocommit mode`_ from ``pymssql`` at connection
  opening time. This allows e.g. DDL statements like ``DROP DATABASE`` to be
  executed. (GH-210)

  Thanks Marat Mavlyutov.

- Documentation: Explicitly mention minimum versions supported of Python (2.6)
  and SQL Server (2005).

- Incremental enhancements to the documentation.

.. _SQL Server instances: http://www.windowsazure.com/en-us/services/sql-database/
.. _Azure: https://www.windowsazure.com/
.. _SQL Server autocommit mode: http://msdn.microsoft.com/en-us/library/ms187878%28v=sql.105%29.aspx

Bug fixes
---------

- Handle errors when calling Stored Procedures via the ``.callproc()`` pymssql
  cursor method. Now it will raise a DB-API ``DatabaseException``; previously
  it allowed a ``_mssql.MSSQLDatabaseException`` exception to surface.

- Fixes in ``tds_version`` ``_mssql`` connections property value

  Made it work with TDS protocol version 7.2. (GH-211)

  The value returned for TDS version 7.1 is still 8.0 for backward
  compatibility (this is because such feature got added in times when
  Microsoft documentation labeled the two protocol versions that followed 7.0
  as 8.0 and 9.0; later it changed them to 7.1 and 7.2 respectively) and will
  be corrected in a future release (2.2).

- PEP 249 compliance (GH-251)

  Added type constructors to increase compatibility with other libraries.

  Thanks Aymeric Augustin.

- pymssql: Made handling of integer SP params more robust (GH-237)

- Check lower bound value when convering integer values from to Python to SQL
  (GH-238)

Internals
---------

- Completed migration of the test suite from nose to py.test.

- Added a few more test cases to our suite.

- Tests: Modified a couple of test cases so the full suite can be run against
  SQL Server 2005.

- Added testing of successful build of documentation to Travis CI script.

- Build process: Cleanup intermediate and ad-hoc anciliary files (GH-231,
  GH-273)
```

```
- setup.py: Fixed handling of release tarballs contents so no extraneous files
  are shipped and the documentation tree is actually included. Also, removed
  unused code.

Version 2.1.0 - 2014-02-25 - `Marc Abramowitz <http://marc-abramowitz.com/>`_
=============================================================================


Features
--------

- Sphinx-based documentation (GH-149)

  Read it online at http://pymssql.org/

  Thanks, Ramiro Morales!

  See:

  * https://github.com/pymssql/pymssql/pull/149
  * https://github.com/pymssql/pymssql/pull/162
  * https://github.com/pymssql/pymssql/pull/164
  * https://github.com/pymssql/pymssql/pull/165
  * https://github.com/pymssql/pymssql/pull/166
  * https://github.com/pymssql/pymssql/pull/167
  * https://github.com/pymssql/pymssql/pull/169
  * https://github.com/pymssql/pymssql/pull/174
  * https://github.com/pymssql/pymssql/pull/175

- "Green" support (GH-135)

  Lets you use pymssql with cooperative multi-tasking systems like
  gevent and have pymssql call a callback when it is waiting for a
  response from the server. You can set this callback to yield to
  another greenlet, coroutine, etc. For example, for gevent, you could
  do::

      def wait_callback(read_fileno):
          gevent.socket.wait_read(read_fileno)

      pymssql.set_wait_callback(wait_callback)

  The above is useful if you're say, running a gunicorn server with the
  gevent worker. With this callback in place, when you send a query to
  SQL server and are waiting for a response, you can yield to other
  greenlets and process other requests. This is super useful when you
  have high concurrency and/or slow database queries and lets you use
  less gunicorn worker processes and still handle high concurrency.

  See https://github.com/pymssql/pymssql/pull/135

- Better error messages.

  E.g.: For a connection failure, instead of:

      pymssql.OperationalError: (20009, 'Net-Lib error during Connection
      refused')

  the dberrstr is also included, resulting in:
```

```
        pymssql.OperationalError: (20009, 'DB-Lib error message 20009,
        severity 9:\nUnable to connect: Adaptive Server is unavailable or
        does not exist\nNet-Lib error during Connection refused\n')

  See:
  * https://github.com/pymssql/pymssql/pull/151

  In the area of error messages, we also made this change:

  execute: Raise ColumnsWithoutNamesError when as_dict=True and missing
  column names (GH-160)

  because the previous behavior was very confusing; instead of raising
  an exception, we would just return row dicts with those columns
  missing. This prompted at least one question on the mailing list
  (https://groups.google.com/forum/?fromgroups#!topic/pymssql/JoZpmNZFtxM),
  so we thought it was better to handle this explicitly by raising an
  exception, so the user would understand what went wrong.

  See:
  * https://github.com/pymssql/pymssql/pull/160
  * https://github.com/pymssql/pymssql/pull/168

- Performance improvements

  You are most likely to notice a difference from these when you are
  fetching a large number of rows.

  * Reworked row fetching (GH-159)

    There was a rather large amount of type conversion occuring when
    fetching a row from pymssql. The number of conversions required have
    been cut down significantly with these changes.
    Thanks Damien, Churchill!

    See:
    * https://github.com/pymssql/pymssql/pull/158
    * https://github.com/pymssql/pymssql/pull/159

  * Modify get_row() to use the CPython tuple API (GH-178)

    This drops the previous method of building up a row tuple and switches
    to using the CPython API, which allows you to create a correctly sized
    tuple at the beginning and simply fill it in. This appears to offer
    around a 10% boost when fetching rows from a table where the data is
    already in memory.
    Thanks Damien, Churchill!

    See:
    * https://github.com/pymssql/pymssql/pull/178

- MSSQLConnection: Add `with` (context manager) support (GH-171)

  This adds `with` statement support for MSSQLConnection in the `_mssql`
  module -- e.g.::

      with mssqlconn() as conn:
          conn.execute_query("SELECT @@version AS version")
```

```
  We already have `with` statement support for the `pymssql` module.
  See:

  * https://github.com/pymssql/pymssql/pull/171

- Allow passing in binary data (GH-179)

  Use the bytesarray type added in Python 2.6 to signify that this is
  binary data and to quote it accordingly. Also modify the handling of
  str/bytes types checking the first 2 characters for b'0x' and insert
  that as binary data.
  See:

  * https://github.com/pymssql/pymssql/pull/179

- Add support for binding uuid.UUID instances to stored procedures input
  params (GH-143)
  Thanks, Ramiro Morales!

  See:
  * https://github.com/pymssql/pymssql/pull/143
  * https://github.com/pymssql/pymssql/commit/1689c83878304f735eb38b1c63c31e210b028ea7

- The version number is now stored in one place, in pymssql_version.h
  This makes it easier to update the version number and not forget any
  places, like I did with pymssql 2.0.1

  * See https://github.com/pymssql/pymssql/commit/fd317df65fa62691c2af377e4661defb721b2699

- Improved support for using py.test as test runner (GH-183)

  * See: https://github.com/pymssql/pymssql/pull/183

- Improved PEP-8 and pylint compliance

Bug Fixes
---------

- GH-142 ("Change how ``*.pyx`` files are included in package") - this
  should prevent pymssql.pyx and _mssql.pyx from getting copied into the
  root of your virtualenv. Thanks, @Arfrever!

  * See: https://github.com/pymssql/pymssql/issues/142

- GH-145 ("Prevent error string growing with repeated failed connection
  attempts.")

  See:

  * https://github.com/pymssql/pymssql/issues/145
  * https://github.com/pymssql/pymssql/pull/146

- GH-151 ("err_handler: Don't clobber dberrstr with oserrstr")

  * https://github.com/pymssql/pymssql/pull/151

- GH-152 ("_mssql.pyx: Zero init global last_msg_* vars")
  See: https://github.com/pymssql/pymssql/pull/152
```

```
- GH-177 ("binary columns sometimes are processed as varchar")
  Better mechanism for pymssql to detect that user is passing binary
  data.

  See: https://github.com/pymssql/pymssql/issues/177

- buffer overflow fix (GH-182)

  * See: https://github.com/pymssql/pymssql/pull/181
  * See: https://github.com/pymssql/pymssql/pull/182

- Return uniqueidentifer columns as uuid.UUID objects on Python 3

Version 2.0.1 - 2013-10-27 - `Marc Abramowitz <http://marc-abramowitz.com/>`_
-----------------------------------------------------------------------------

* MANIFEST.in: Add "\*.rst" to prevent install error: "IOError: [Errno 2] No
  such file or directory: 'ChangeLog_highlights.rst'"

Version 2.0.0 - 2013-10-25 - `Marc Abramowitz <http://marc-abramowitz.com/>`_
-----------------------------------------------------------------------------

* First official release of pymssql 2.X (`Cython`_-based code) to `PyPI`_!
* Compared to pymssql 1.X, this version offers:

  * Better performance
  * Thread safety
  * Fuller test suite
  * Support for Python 3
  * Continuous integration via `Travis CI`_
  * Easier to understand code, due to `Cython`_

.. _PyPI: https://pypi.python.org/pypi/pymssql/2.0.0
.. _Travis CI: https://travis-ci.org/pymssql/pymssql
.. _Cython: http://cython.org/
.. _ChangeLog: https://github.com/pymssql/pymssql/blob/master/ChangeLog

Version 2.0.0b1-dev-20130403 - 2013-04-03 - Marc Abramowitz <msabramo@gmail.com>
--------------------------------------------------------------------------------
        * Added tag 2.0.0b1-dev-20130403 for changeset 5d0c980ef8b8
        (b2b2748f7f88)
        * Fix issue 118 ("datetime conversion to sql is not converting
        sub-seconds correctly") - Pad microseconds to 3 digits so it gets
        converted correctly. Thanks, Ken Robbins (kenneth.robbins at gmail)!
        (5d0c980ef8b8)
        * Make tests/test_queries.py actually run tests. It looked like it was
        half-finished and not working. This fills it out and makes it work and
        actually test a few things. (5373541eb899)
        * setup.py: Make it possible to use `python setup.py test`
        (3c32acb41251)
        * Bunch of fixes to eliminate build/install warnings (adb0fc75bfd0,
        fe6cb9aa5120, 446f0005e638, e8d4b19d87b1, 90b2aa2ea01f, 7bb29af4b22c)
        * Add `pymssql.get_dbversion` function that wraps the dbversion
        function in FreeTDS. (1158a5d2be9c)
        * Add a `get_freetds_version` function (a4286224dcf2)
        * Fix issue 109 ("Failure to pass Unicode characters to callproc;
        failing test:
        tests.test_sprocs.TestCallProcFancy.testCallProcWithUnicodeStringWithRussianCharacters"):
```

```
       Skip test because it fails with some versions of FreeTDS but passes
       with others. (d05341273673)
       * Fix issue 116 ("A few tests fail if running on a system that has SQL
       Server available on port 1433") (0fc4086447fe)
       * Modify tests/test_config.py to use server='dontnameyourserverthis'
       when doing various tests so it doesn't try to connect to a SQL
       Server listening on localhost:1433 (0fc4086447fe)
       * tox.ini: Add {posargs:-w tests -v} to nosetests invocation so that
       we can pass arguments to tox -- e.g.: to run only specific tests
       (a105878d500d)
       * tox.ini: Add "ipdb" to deps, because the IPython debugger is very
       nice for debugging why tests are failing (be9ee40156cb)
       * Fix issue 114 ("Fix SP name handling in threaded test so we can
       actually run it.") (6ac2b75747ad)
       * Fix issue 100 (Error when executing setup.py {build,develop} on a
       system with no setuptools: "name 'STDevelopCmd' is not defined")
       (5222ee37b2ab)
       * Issue 45 ("Make SQLAlchemy tests part of our testing process"): Add
       tests/run_sqlalchemy_tests.py for running the SQLAlchemy test suite
       with pymssql using the server configured in tests/tests.cfg
       (999d9dbe791b)
       * Fix issue 92 ("Cursor fetch* methods return redundant keys for
       column names and column numbers.") (08ae783880dd)
       * tests/test_connection_as_dict.py:
       bug 18 ("FetchAll fails to return rows when running against a
       connection instantiated with as_dict=True."): Add a test which
       illustrates that the issue is resolved. (058d761cc761)
       * Fix issue 60 ("cursor.execute raise UnicodeDecodeError if query and
       params in unicode"): Add patch from tonal.promsoft and add tests.
       (49210c03a6cf)
       * Add *.c to MANIFEST.in so they get included in sdist and end-users
       don't need to run Cython. (25c1a84aac0c)
       * Fix issue 56 ("callproc do not accept None and unicode string in
       parameters"): Add patch from tonal.promsoft and add tests
       (939eb7939136)
   * version 2.0.0b1-dev-20130403

Version 2.0.0b1-dev-20130108 - 2013-01-08 - Marc Abramowitz <msabramo@gmail.com>
--------------------------------------------------------------------------------
     * change: put compiled FreeTDS for Windows in pymmsql source, add build
         instructions to README, better Visual Studio support (#61)
   + feature: support hostname, port, tds_version connect params without freetds.config
   + feature: make pymssql.Cursor.rownumber give accurate results for executemany()
   + feature: bundle FreeTDS libraries & use static libary includes to avoid
       most end-user-developers needing to mess with FreeTDS
   * change: speed up handling of tuples/lists when quoting (dieterv77)
   - bug #46: better handling for byte strings that don't represent ascii data
   - bug: custom param handling avoids bugs when '%' is used in the SQL (modulus operator)
   - bug: fix pymssql.DBAPIType so that comparisons work as expected
   - bug: fetch*() functions would erroneously raise OperationalError when rows exhausted
   - bug #47: fix threaded tests crashing
   - bug #79: fix prevision problem with floats (dieterv77)
               - bug #14: Add setup.py voodoo that undoes setuptools monkeypatching
               that causes `pip install` to not work with setuptools unless pyrex
               is installed. (86a73a19d5bd)
               - bug #106 (OS X: "Symbol not found: _dbadata" error when importing
               pymssql): Fix OS X build by modifying setup.py so that on OS X we
               don't attempt to link with the bundled FreeTDS *Linux* library.
```

```
                (88d15d125586)
                + feature: Add support for running tox (http://tox.testrun.org/) to
                test across multiple Python versions. (5fa7a6548b31)
                - bug #44: Remove test_long_identifiers from test_sqlalchemy.py
                because SQLAlchemy removed the 30 character limit on identifiers.
                (6585d44eea33)
                - feature: Add setup.py voodoo so that Cython can automatially be
                downloaded if it's not installed instead of an ImportError
                (bb459dd7fd7e)
                - bug #105: Link with librt on Unix platforms that have it (like
                Linux, but not FreeBSD or OS X) to prevent 'undefined symbol:
                clock_gettime' error when importing pymssql. (2b255b1c035f)

Tue Nov 02 09:33:00 2010 Damien Churchill <damoxc@gmail.com>
        * _mssql.pyx:
                + feature: add support for nullable ints and nullable bits in
                          stored procedure parameters.
                + feature: add support for positional parameters in stored
                          procedures.
                + bugfix:  add support for using type subclasses as parameters
                + bugfix:  correctly report incorrect logins.
                + feature: add support for setting the application name
                + bugfix:  accept more than just the decimal.Decimal type for
                          money and decimal parameters.
                + bugfix:  fix raising exceptions from convert_python_value()
                + bugfix:  fix binding parameters of int type when larger than
                          2^31 - 1 (raise exception).
                + bugfix:  use sprintf rather than python strings in the msg_handler
                + bugfix:  use sprintf rather than python strings in the err_handler
                + bugfix:  make compatible with Cython 0.13
                + feature: remove the trusted parameter to connect()
                + bugfix:  fix issue 15, not setting implicit_transactions on connect
                + bugfix:  fix issue 32, setting the wrong hostname on login
        * pymssql.pyx:
                + feature: add initial support for callproc()
                + feature: add support for setting the application name
                + bugfix:  fix issue #7, thanks has.temp3
                + bugfix:  fix issue #10, rowcount property being incorrect
                + bugfix:  make compatible with Cython 0.13
                + feature: remove the trusted parameter to connect()
                + feature: add returnvalue property with the result of a callproc()
                          call.
                + feature: fix raising exceptions when args[0] is not a string
        * MANIFEST.in:
                + feature: include the tests
                + bugfix:  include ez_setup.py
        * setup.py:
                + bugfix:  fix issue #8, ZipFile don't has the attribute
                          'extractall' error for python2.5
        * version 1.9.909

Wed Apr 28 11:10:00 2010 Damien Churchill <damoxc@gmail.com>
        * MANIFEST.in:
                + bugfix: fix recursive-include for .pyrex
        * version 1.9.908

Wed Apr 21 16:02:00 2010 Damien Churchill <damoxc@gmail.com>
        * MANIFEST.in:
```

```
                    + bugfix: include missing .pyrex folder
          * version 1.9.907


Fri Apr 09 13:16:00 2010 Damien Churchill <damoxc@gmail.com>
          * setup.py:
                    + bugfix: include hack faking that pyrex is installed to workaround
                              a bug in setuptools.
          * _mssql.pyx:
                    + bugfix: add support for connecting using "." and "(local)"
          * pymssql.pyx:
                    + feature: add the output type to be used with callproc()
                    + depreciate: the dsn keyword param to pymssql.connect()
                    + feature: add the get/set_max_connections to pymssql
          * sqlfront.pxd:
                    + feature: tidy up and remove all unused methods.
          * version 1.9.906


Mon Nov 23 13:37:00 2009 Damien Churchill <damoxc@gmail.com>
          * _mssql.pyx:
                    + feature: add support for varbinary types
                    + feature: add support for passing in charset to _quote_data
                    + bugfix: rename MSSQLConnection.next_result to
                              MSSQLConnection.nextresult as before
                    + bugfix: set the charset upon login
                    + feature: rewrite _remove_locale using C types instead, 20x faster
                    + feature: add a charset param to quote_data and relating funcs that
                               allows the charset to be specified for unicode encodes.
          * pymssql.pyx:
                    + feature: add DSN support that was missing
                    + bugfix: fix rowcount property
          * sqlfront.pxd:
                    add DBSETLCHARSET
          * tests:
                    + feature: add test for multiple results
          * setup.py:
                    + feature: fix building on windows
                    + feature: clean generated C files in the clean command
                    + feature: automatically extract freetds.zip on windows when
                               building
          * version 1.9.903


Fri Nov 20 13:03:00 2009 Damien Churchill <damoxc@gmail.com>
          * mssqldbmodule.c: deprecated in favour of _mssql.pyx
          * pymssql.py: deprecated in favour of pymssql.py
          + feature: added support for uniqueidentifier types
          + feature: added support for calling remote procedures programmatically
          * version 1.9.901


Tue May 12 15:43:00 2009  Andrzej Kukula <akukula@gmail.com>
          * mssqldbmodule.c:
                    + bugfix: pymssql didn't return second, third etc. result set
                              in case of multi-result statements, e.g. 'SELECT 1; SELECT 2',
                              thanks Damien Churchill <damoxc@users.sourceforge.net>


Wed Apr 29 19:31:00 2009  Andrzej Kukula <akukula@gmail.com>
          * mssqldbmodule.c:
                  + fixed possible memory leak, thanks Evgeny Cherkashin
                        <eugene.adm@gmail.com>
```

```
Tue Apr 23 23:00:00 2009  Andrzej Kukula <akukula@gmail.com>
        + bugfix: fixed rare quoting bug in select_db()
        + feature: added 'max_conn' parameter to pymssql.connect() and
          _mssql.connect() which defaults to 25, thanks Daniel Watrous
          <dwmaillist@gmail.com>
        * nagios-plugin update - thanks Josselin Mouette <joss@debian.org>:
          + Include a -P port option, to avoid having to passing it with the
                host name
          + Fix the encoding of the comments; utf-8 is the declared encoding
                of the file and must be followed
          + Fix a typo in the SQL syntax
          + Connect explicitly to the "master" database (required since 1.0.0)
          + Improve perfdata output.
        * version 1.0.2

Tue Apr 21 22:56:00 2009  Andrzej Kukula <akukula@gmail.com>
        * mssqldbmodule.c:
          + bugfix in format_and_run_query(): query strings were sometimes
                overwritten with garbage due to DECREF in wrong place; thanks
                Igor Nazarenko <igor.n.nazarenko@gmail.com>
          + bugfix in get_result(): if a query batch contained DECLARE or
                possibly other T-SQL statements, no results were returned
                thanks Kay Schluehr <schluehrk@users.sourceforge.net>
          + bugfix in execute_scalar(): check if there are any columns in result
          + bugfix: check for FAIL after each dbnextrow()
          + feature: Add support for bigint - #2660972; thanks Alexandr
                Zamaraev <shura_zam@users.sourceforge.net>
        * pymssql.c:
          + bugfix in execute(): if execute is called without second argument,
                don't treat '%' in query string as formatting character; restored
                compatibility with common sense and with pymssql < 1.0.0; thanks
                Corey Bertram <corey.bertram@monitoredsecurity.com>,
                Wes McKinney <wesmckinn@gmail.com>
          + feature: it is possible to specify 'as_dict' to pymssql.connect
                and rows will be returned as dictionaries instead of tuples;
                thanks Daniel Watrous <dwmaillist@gmail.com>

Thu Jan 30 18:36:00 2009  Andrzej Kukula <akukula@gmail.com>
        * mssqldbmodule.c:
          + Pyssize_t error on x64 - thanks Josselin Mouette <joss@debian.org>
          + critical charset updates, thanks Josselin Mouette <joss@debian.org>
          + more Py_ssize_t updates, further code cleanups
          + fixed some compiler warnings
        * pymssql.py:
          + execute() failed, thanks Josselin Mouette <joss@debian.org>
          + critical charset updates, thanks Josselin Mouette <joss@debian.org>
          + removed warnings, users don't want them and they are not 'MUST'
                priority in DB-API spec
        * nagios-plugin: introduced Nagios plugin, thanks Julien Blache
          and Josselin Mouette
        * version 1.0.1

Thu Jan 29 19:23:00 2009  Andrzej Kukula <akukula@gmail.com>
        * version 1.0.0
        * so many changes I'll not put them here, I'll document
          changes from now on.

Mon Sep 25 20:18:00 2006  Andrzej Kukula <akukula@gmail.com>
```

```
               * setup.py: fix for Fink (http://Fink.SF.Net) under OS X (thanks
                 Terrence Brannon <metaperl@gmail.com>)

Sun Sep 24 10:44:00 2006  Andrzej Kukula <akukula@gmail.com>
               * setup.py:
                 + it can now dynamically determine the path to SQL 2000 Developer
                       Tools, if win32api and win32con modules are available
                 + simple Python version check to prevent most frequently asked
                       question
                 + version 0.8.0

Wed Sep 13 01:20:00 2006  Andrzej Kukula <akukula@gmail.com>
               * mssqldbmodule.c:
                 + corrected misspellings in docstrings
                 + fixed segfault on connection close with Python 2.5; thanks
                       Justin Francis <jfrancis@pivotalpayments.com>
               * pymssql.py:
                 + fixed two minor DB-API incompatibilities (thanks Matthew Good
                       <matt@matt-good.net>)
                 + fixed datetime quoting (thanks Jan Finell <jfinell@regionline.fi>)
               * pymssql should be able to build on cygwin (thanks
                 rob@robnet.com)
               * docstring fixes, webpage doc updates

Tue May 15 03:18:00 2006  Jooncheol Park <exman@users.sourceforge.net>
               * setup.py, PKG-INFO, README: license change to LGPL

Wed Mar 15 08:18:00 2006  Andrzej Kukula <akukula@users.sourceforge.net>
               * pymssql.py: fixed datetime issue (thanks Jan Finell
                 <jfinell@regionline.fi>)

Fri Feb 24 16:11:00 2006  Andrzej Kukula <akukula@users.sourceforge.net>
               * mssqldbmodule.c: fixed typos in docstrings (thanks Konstantin
                 Veretennicov)

Tue Dec 27 15:14:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>
               * mssqldbmodule.c: bug fixes, improvements and cleanups:
                 + implemented set_login_timeout() and set_query_timeout() functions;
                 + eliminated unnecessary ODBC code
                 + cleaned up exception code and improved exception handling,
                       SF bug #1335560
                 + web page now correctly mentions FreeTDS 0.63 as the minimal
                       required version
                 + stdmsg() method is now deprecated; all errors are concatenated
                       in errmsg()
                 + implemented min_error_severity: all errors at or above that
                       level will raise the exception; if the severity is lower, they
                       will just accumulate in errmsg()
                 + added setting coltype to NUMBER for float types (found by
                       Jakub Labath)
               * setup.py:
                 + reincarnated ntwdblib.dll which turned out to be redistributable
                       after all; pymssql includes the latest version that allows
                       connecting to SQL 2005; eliminated some stupid notes from the
                       web page and will ease set up process for users
               * apitest_mssql.py: new file
                 + provided by Jakub Labath, this file performs some basic DB-API
                       compliance tests; it immediately triggered the unicode bug
```

```
             * version 0.7.4

Sat Oct 22 19:41:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>
             * mssqldbmodule.c: multithreading improvements - from now on pymssql
               is thread-safe, it releases GIL in proper places; idea and initial
               patch by John-Peter Lee (thanks very much!)

Mon Sep 5 23:29:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>
             * setup.py: fixed an installation issue regarding importing pymssql
               that imports _mssql which isn't installed, and blows up with
               AttributeError... (thanks Vsevolod Stakhov)
             * version 0.7.3

Mon Sep 5 00:32:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>
             * version 0.7.2

Sun Sep 4 23:12:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

             * mssqldbmodule.c: improvements and cleanups:
               + improved error handling: if the db function fails, the exception
                     is thrown automatically and immediately; no need to check
                     return value of conn.query(), just catch _mssql.error
               + improved error handling: it is possible that MS SQL calls message
                     handler twice; now _mssql catches and reports both of them at once
               + improved error handling: in some cases _mssql.query() returns
                     success but the results are invalid; now it is handled properly
                     (example "SELECT CAST(1234.5678 AS NUMERIC(4,2))")
               + added proper connection initialization: a number of SET statements
                     are executed upon connection setup to set sensible SQL behaviour;
                     see source for details; one needs to unset them if needed
               + implemented min_{message|error}_severity as it is in php_mssql
                     to ignore unimportant errors; it's work in progress
               + new function rmv_lcl() initially by Mark Pettit, to strip locale
                     crap from MONEY values converted to SQLCHAR while generating
                     Decimal object
               + other small fixes, improvements and janitorial work

Tue Aug 30 00:16:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

             * mssqldbmodule.c: new features:
               + large numbers (DECIMAL, NUMERIC, MONEY, SMALLMONEY) are returned
                     as Decimal object -- this helps maintain accuracy; thanks to
                     Mark Pettit for help
               + COMPUTE clauses are supported (it wouldn't fetch data for those
                     columns before)
               + ROWID type has been removed from _mssql module
               + new type DECIMAL to denote Decimal objects in result set

Mon Aug 29 21:59:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

             * mssqldbmodule.c: some improvements:
               + BIT values are returned as Python bool objects, suggested by
                     Mark Pettit
               + close() method returns None on success (not to be used at all)
                     and throws exception on error
               + fixed use of uninitialized value when parsing SMALLDATETIME
               + another round of performance improvements in GetRow() - eliminated
                     unnecessary data conversions and unneeded DB-Lib calls
```

```
       + janitorial fixes

Mon Aug 22 04:35:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

       * mssqldbmodule.c: massive diff:
         + fixed bug with fetching query results of some data types;
               found by Mark Pettit
         + fixed IndexError when query returns no rows; patch by Jakub Labath
         + rewritten function GetRow() that fetches query results: performance
               improvements, better handling of result data types; datetime
               is returned as datetime object instead of string (it's more
               consistent with other values -- and more pythonic :)
         + eliminated DetermineRowSize()
         + cleanups: _mssql_init() further improvements w.r.t. Python API
         + janitorial fixes
         + added licensing information
       * pymssql.py: docstring changed to look nicer with help()
       * version 0.7.2

Thu Aug 11 02:12:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

       * mssqldbmodule.c: improved module init function: added doc string,
         made compliant with Python 2.0+ module interface (there are no more
         coredumps on help())
       * mssqldbmodule.c: documented that _mssql.connect() is not portable
         between FreeTDS-dependent platforms and Windows platforms; documented
         host:port usage

Sat Jul 23 14:20:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

       * mssqldbmodule.c: eliminated problems with Python exiting upon
         invalid login credentials with FreeTDS - the culprit was INT_EXIT
         and FreeTDS setting DBDEAD
       * mssqldbmodule.c: added better error messages (esp. on Windows)
       * mssqldbmodule.c: added msg_handler and err_handler debugging
       * 0.7.1 packages re-released

Fri Jul 22 03:19:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

       * mssqldbmodule.c: major change; module revamped to support some
         more builtin Python features; some redundant code removed; memset()
         removed as there were no benefits but performance decrease
       * mssqldbmodule.c: help(_mssql) works; help for conn object works too
       * pymssql.py: _quote: removed escaping backslash -- with MSSQL it is
         only needed to escape single quotes by duplicating them
       * pymssql.py: pymssqlCnx class: added a few checks to properly support
         DB-API 2.0 (see .close() in PEP 249)
       * version 0.7.1

Wed Jul 20 22:12:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

       * mssqldbmodule.c: removed the workaround for date issue; there were
         more problems than benefits
       * mssqldbmodule_tds.c: removed
       * some more cleanups and corrections

Tue Jul 19 14:23:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>
```

```
        * mssqldbmodule.c: major change; many portability problems fixed
        * mssqldbmodule.c: eliminated port setting; this is job for freetds.conf
        * mssqldbmodule_tds.c: module to get FreeTDS compile-time settings
        * build fixes; now it builds cleanly on FreeBSD, Linux and Windows
        * version 0.7.0

Mon Jul 18 15:21:00 2005  Andrzej Kukula <akukula@users.sourceforge.net>

        * mssqldbmodule.c: fix build on Windows: changed MS_WIN32 to MS_WINDOWS
          reported by Mirek Rusin <mirek.rusin@onet.pl>
        * mssqldbmodule.c: many small fixes and cleanups; janitorial fixes;
          indentation using indent(1L)
        * ChangeLog fix! 'mysql' was mentioned instead of 'mssql'...

Fri Feb 25 02:15:01 2005  Andrzej Kukula <akukula@users.sourceforge.net>

        * Fix build on Windows with Visual Studio .NET 2003
          and MS SQL Server 2000 SP3a
        * mssqldbmodule.c: Fix compile error with Visual Studio .NET 2003
        * mssqldbmodule.c: Add detection/workaround for date issue caused by
          different dbdatecrack() prototypes
        * README.freetds: describe dbdatecrack()-related issue

Thu Feb 24 02:03:14 2005  Alejandro Dubrovsky  <alito@organicrobot.com>

        * Export column type names
        * mssqldbmodule.c:  Return column type information for headers
        * Use type information to make cursor.description conform to API 2

2005-02-17  Alejandro Dubrovsky  <alito@organicrobot.com>

        * Apply patch by Rob Nichols to get cursor.description closer to API 2 compliance

2005-02-08  Alejandro Dubrovsky  <alito@organicrobot.com>

        * Message changes in mssqldbmodule.c (typos, grammar, etc)

2005-02-07  Alejandro Dubrovsky  <alito@organicrobot.com>

        * Added ChangeLog
        * API Change:  add 6th parameter 'port' to connect
        * Don't close connection on cursor close (noted by Alberto Pastore on the sourceforge project
        * Make cursor.fetchone comply with DB-SIG return a tuple, not a list of tuples (report and pa
```

TODO

13.1 Documentation

Todo

Add an example of invoking a Stored Procedure using `_mssql`.

(The original entry is located in /var/build/user_builds/pymssql/checkouts/v2.1.1/docs/_mssql_examples.rst, line 141.)

Todo

Document `null MSSQLStoredProcedure.bind()` argument.

(The original entry is located in /var/build/user_builds/pymssql/checkouts/v2.1.1/docs/ref/_mssql.rst, line 341.)

Todo

Document `pymssql` connection `get_dbversion()`.

(The original entry is located in /var/build/user_builds/pymssql/checkouts/v2.1.1/docs/ref/pymssql.rst, line 79.)

Todo

Document all `pymssql` PEP 249-mandated exceptions.

(The original entry is located in /var/build/user_builds/pymssql/checkouts/v2.1.1/docs/ref/pymssql.rst, line 321.)

—

p

Symbols