Welcome to PyMySQL's documentation!

User Guide

The PyMySQL user guide explains how to install PyMySQL and how to contribute to the library as a developer.

Installation

The last stable release is available on PyPI and can be installed with `pip`:

```
$ python3 -m pip install PyMySQL
```

To use "sha256_password" or "caching_sha2_password" for authenticate, you need to install additional dependency:

```
$ python3 -m pip install PyMySQL[rsa]
```

Requirements

- Python – one of the following:
  - CPython >= 3.6
  - Latest PyPy 3
- MySQL Server – one of the following:
  - MySQL >= 5.6
  - MariaDB >= 10.0

Examples

CRUD

The following examples make use of a simple table

```
CREATE TABLE `users` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `email` varchar(255) COLLATE utf8_bin NOT NULL,
    `password` varchar(255) COLLATE utf8_bin NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
AUTO_INCREMENT=1 ;
```

```
import pymysql.cursors

# Connect to the database
connection = pymysql.connect(host='localhost',
                             user='user',
                             password='passwd',
                             database='db',
                             charset='utf8mb4',
                             cursorclass=pymysql.cursors.DictCursor)

with connection:
    with connection.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO `users` (`email`, `password`) VALUES (%s, %s)"
        cursor.execute(sql, ('webmaster@python.org', 'very-secret'))

    # connection is not autocommit by default. So you must commit to save
    # your changes.
    connection.commit()

    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT `id`, `password` FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org',))
        result = cursor.fetchone()
        print(result)
```

This example will print:

```
{'id': 1, 'password': 'very-secret'}
```

Resources

DB-API 2.0: http://www.python.org/dev/peps/pep-0249

MySQL Reference Manuals: http://dev.mysql.com/doc/

MySQL client/server protocol: http://dev.mysql.com/doc/internals/en/client-server-protocol.html

PyMySQL mailing list: https://groups.google.com/forum/#!forum/pymysql-users

Development

You can help developing PyMySQL by contributing on GitHub.

Building the documentation

Go to the `docs` directory and run `make html`.

Test Suite

If you would like to run the test suite, create a database for testing like this:

```
mysql -e 'create database test_pymysql  DEFAULT CHARACTER SET utf8 DEFAULT COLLATE
utf8_general_ci;'
mysql -e 'create database test_pymysql2 DEFAULT CHARACTER SET utf8 DEFAULT COLLATE
utf8_general_ci;'
```

Then, copy the file `ci/database.json` to `pymysql/tests/databases.json` and edit the new file to match your MySQL configuration:

```
$ cp ci/database.json pymysql/tests/databases.json
$ $EDITOR pymysql/tests/databases.json
```

To run all the tests, execute the script `runtests.py`:

```
$ pip install pytest
$ pytest -v pymysql
```

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

For more information, please read the Python Database API specification.

Connection Object

class **pymysql.connections.Connection**(**, user=None, password='', host=None, database=None, unix_socket=None, port=0, charset='', sql_mode=None, read_default_file=None, conv=None, use_unicode=True, client_flag=0, cursorclass=<class 'pymysql.cursors.Cursor'>, init_command=None, connect_timeout=10, read_default_group=None, autocommit=False, local_infile=False, max_allowed_packet=16777216, defer_connect=False, auth_plugin_map=None, read_timeout=None, write_timeout=None, bind_address=None, binary_prefix=False, program_name=None, server_public_key=None, ssl=None, ssl_ca=None, ssl_cert=None, ssl_disabled=None, ssl_key=None, ssl_verify_cert=None, ssl_verify_identity=None, compress=None, named_pipe=None, passwd=None, db=None*)

Representation of a socket with a mysql server.

The proper way to get an instance of this class is to call connect().

Establish a connection to the MySQL database. Accepts several arguments:

Parameters:

- **host** – Host where the database server is located.
- **user** – Username to log in as.
- **password** – Password to use.
- **database** – Database to use, None to not use a particular one.
- **port** – MySQL port to use, default is usually OK. (default: 3306)
- **bind_address** – When the client has multiple network interfaces, specify the interface from which to connect to the host. Argument can be a hostname or an IP address.
- **unix_socket** – Use a unix socket rather than TCP/IP.
- **read_timeout** – The timeout for reading from the connection in seconds (default: None - no timeout)
- **write_timeout** – The timeout for writing to the connection in seconds (default: None - no timeout)
- **charset** – Charset to use.
- **sql_mode** – Default SQL_MODE to use.
- **read_default_file** – Specifies my.cnf file to read these parameters from under the [client] section.
- **conv** – Conversion dictionary to use instead of the default one. This is used to provide custom marshalling and unmarshalling of types. See converters.
- **use_unicode** – Whether or not to default to unicode strings. This option defaults to true.
- **client_flag** – Custom flags to send to MySQL. Find potential values in constants.CLIENT.
- **cursorclass** – Custom cursor class to use.
- **init_command** – Initial SQL statement to run when connection is established.
- **connect_timeout** – The timeout for connecting to the database in seconds. (default: 10, min: 1, max: 31536000)
- **ssl** – A dict of arguments similar to mysql_ssl_set()'s parameters.
- **ssl_ca** – Path to the file that contains a PEM-formatted CA certificate.
- **ssl_cert** – Path to the file that contains a PEM-formatted client certificate.
- **ssl_disabled** – A boolean value that disables usage of TLS.
- **ssl_key** – Path to the file that contains a PEM-formatted private key for the client certificate.
- **ssl_verify_cert** – Set to true to check the server certificate's validity.
- **ssl_verify_identity** – Set to true to check the server's identity.
- **read_default_group** – Group to read from in the configuration file.
- **autocommit** – Autocommit mode. None means use server default. (default: False)
- **local_infile** – Boolean to enable the use of LOAD DATA LOCAL command. (default: False)
- **max_allowed_packet** – Max size of packet sent to server in bytes. (default: 16MB) Only used to limit size of "LOAD LOCAL INFILE" data packet

smaller than default (16KB).

- **defer_connect** – Don't explicitly connect on construction - wait for connect call. (default: False)
- **auth_plugin_map** – A dict of plugin names to a class that processes that plugin. The class will take the Connection object as the argument to the constructor. The class needs an authenticate method taking an authentication packet as an argument. For the dialog plugin, a prompt(echo, prompt) method can be used (if no authenticate method) for returning a string from the user. (experimental)
- **server_public_key** – SHA256 authentication plugin public key value. (default: None)
- **binary_prefix** – Add _binary prefix on bytes and bytearray. (default: False)
- **compress** – Not supported.
- **named_pipe** – Not supported.
- **db** – **DEPRECATED** Alias for database.
- **passwd** – **DEPRECATED** Alias for password.

See Connection in the specification.

> **begin**()
>
> Begin transaction.

> **close**()
>
> Send the quit message and close the socket.
>
> See Connection.close() in the specification.
>
> **Raises:** **Error** – If the connection is already closed.

> **commit**()
>
> Commit changes to stable storage.
>
> See Connection.commit() in the specification.

> **cursor**(*cursor=None*)
>
> Create a new cursor to execute queries with.
>
> **Parameters:** **cursor** (`Cursor` , `SSCursor` , `DictCursor` , or `SSDictCursor` .) – The type of cursor to create. None means use Cursor.

> **open**
>
> Return True if the connection is open.

ping(*reconnect=True*)

Check if the server is alive.

Parameters: **reconnect** (*boolean*) – If the connection is closed, reconnect.

Raises: **Error** – If the connection is closed and reconnect=False.

rollback()

Roll back the current transaction.

See Connection.rollback() in the specification.

select_db(*db*)

Set current db.

Parameters: **db** – The name of the db.

show_warnings()

Send the "SHOW WARNINGS" SQL command.

Cursor Objects

class **pymysql.cursors.Cursor**(*connection*)

This is the object used to interact with the database.

Do not create an instance of a Cursor yourself. Call connections.Connection.cursor().

See Cursor in the specification.

callproc(*procname*, *args=()*)

Execute stored procedure procname with args.

Parameters:
- **procname** (*str*) – Name of procedure to execute on server.
- **args** (*tuple or list*) – Sequence of parameters to use with procedure.

Returns the original args.

Compatibility warning: PEP-249 specifies that any modified parameters must be returned. This is currently impossible as they are only available by storing them in a server variable and then retrieved by a query. Since stored procedures return zero or more result sets, there is no reliable way to get at OUT or INOUT parameters via callproc. The server variables are named @_procname_n, where procname is the parameter above and n is the position of the parameter (from zero). Once all result sets generated by the procedure have been fetched, you can issue a SELECT @_procname_0, ... query using .execute() to get any OUT or INOUT values.

Compatibility warning: The act of calling a stored procedure itself creates an empty result set. This appears after any result sets generated by the procedure. This is non-standard behavior with respect to the DB-API. Be sure to use nextset() to advance through all result sets; otherwise you may get disconnected.

close()

Closing a cursor just exhausts all remaining data.

execute(*query*, *args=None*)

Execute a query.

Parameters:
- **query** (*str*) – Query to execute.
- **args** (*tuple*, *list* or *dict*) – Parameters used with query. (optional)

Returns: Number of affected rows.

Return type: int

If args is a list or tuple, %s can be used as a placeholder in the query. If args is a dict, %(name)s can be used as a placeholder in the query.

executemany(*query*, *args*)

Run several data against one query.

Parameters:
- **query** (*str*) – Query to execute.
- **args** (*tuple* or *list*) – Sequence of sequences or mappings. It is used as parameter.

Returns: Number of rows affected, if any.

Return type: int or None

This method improves performance on multiple-row INSERT and REPLACE. Otherwise it is equivalent to looping over args with execute().

fetchall()

Fetch all the rows.

fetchmany(*size=None*)

Fetch several rows.

fetchone()

Fetch the next row.

max_stmt_length= *1024000*

Max statement size which `executemany()` generates.

Max size of allowed statement is max_allowed_packet - packet_header_size. Default value of max_allowed_packet is 1048576.

mogrify(*query*, *args=None*)

Returns the exact string that would be sent to the database by calling the execute() method.

Parameters:
- **query** (*str*) – Query to mogrify.
- **args** (*tuple*, *list* or *dict*) – Parameters used with query. (optional)

Returns: The query with argument binding applied.

Return type: str

This method follows the extension to the DB API 2.0 followed by Psycopg.

setinputsizes(**args*)

Does nothing, required by DB API.

setoutputsizes(**args*)

Does nothing, required by DB API.

class **pymysql.cursors.SSCursor**(*connection*)

Unbuffered Cursor, mainly useful for queries that return a lot of data, or for connections to remote servers over a slow network.

Instead of copying every row of data into a buffer, this will fetch rows as needed. The upside of this is the client uses much less memory, and rows are returned much faster when traveling over a slow network or if the result set is very big.

There are limitations, though. The MySQL protocol doesn't support returning the total number of rows, so the only way to tell how many rows there are is to iterate over every row returned. Also, it currently isn't possible to scroll backwards, as only the current row is held in memory.

close()

Closing a cursor just exhausts all remaining data.

fetchall()

Fetch all, as per MySQLdb. Pretty useless for large queries, as it is buffered. See fetchall_unbuffered(), if you want an unbuffered generator version of this method.

fetchall_unbuffered()

Fetch all, implemented as a generator, which isn't to standard, however, it doesn't make sense to return everything in a list, as that would use ridiculous memory for large result sets.

fetchmany(*size=None*)

Fetch many.

fetchone()

Fetch next row.

read_next()

Read next row.

class **pymysql.cursors.DictCursor**(*connection*)

A cursor which returns results as a dictionary

class **pymysql.cursors.SSDictCursor**(*connection*)

An unbuffered cursor, which returns results as a dictionary

Indices and tables

- Index
- Module Index
- Search Page