pyhs Documentation

Release 0.2.4

Artem Gluvchynsky

Jun 26, 2017

Contents

pyhs is a pure Python client (with optional C speedups) for HandlerSocket plugin to MySQL database. In short, it provides access to the data omitting the SQL engine in a NoSQL-like interface. It allows all simple operations (get, insert, update, delete) over indexed data to perform considerably faster than by usual means.

See this article for more details about HandlerSocket.

This client supports both read and write operations but no batching at the moment.

Go to *Installation* and *Usage* sections for quick start. There's also a *reference* for all public interfaces.

Project is open-source and always available on the bitbucket: http://bitbucket.org/excieve/pyhs/

Contents:

CHAPTER 1

Installation

HandlerSocket plugin

First, you'll have to get this working. At the moment of writing the only way to do this, was getting the source code, compiling it and loading into the MySQL instance. Keep the HandlerSocket up to date as the client gets updated from time to time as new features or changes appear in the plugin.

See also:

Installation guide HandlerSocket installation guide at the official repository.

The Client

At the moment you can install pyhs by either using pip, easy_install, downloading from PyPI or getting source directly from bitbucket.

Pip way

This is very simple, just run:

```
pip install python-handler-socket
```

Or this to get the latest (not yet released on PyPI):

```
pip install hg+http://bitbucket.org/excieve/pyhs#egg=python-handler-socket
```

This command will install the package into your site-packages or dist-packages.

Source

Clone the source from the repository and install it:

```
hg clone http://bitbucket.org/excieve/pyhs
cd pyhs
python setup.py install
```

By default additional C speedups are also built and installed (if possible). However, if they are not needed, please use `--without-speedups` option.

Testing installation

Check your installation by running this in Python interpreter:

```
from pyhs import __version__
print __version__
```

This should show currently installed version of pyhs. You're all set now.

CHAPTER 2

Usage

Overview

Once the package is correctly installed and HandlerSocket plugin is loaded in your MySQL instance, you're ready to do some code.

The client consists of two parts: *high level* and *low level*.

In most cases you'll only need the high level part which is handled by *manager.Manager* class. It saves developer from index id allocation and reader, writer server pools management - just provides a simple interface for all supported operations.

One might want to use the low level interface in case more control over mentioned things is needed. This part is handled by *sockets.ReadSocket* and *sockets.WriteSocket* for read and write server pools/operations correspondingly. They both subclass *sockets.HandlerSocket* which defines the pool and common operations like opening an index. There's also the *sockets.Connection* which controls low-level socket operations and is managed by the pool.

Usage examples

A few simple snippets of both low and high level usage to get started.

High level

This one initialises HandlerSocket connection and inserts a row in a table:

```
from pyhs import Manager

# This will initialise both reader and writer connections to the default hosts
hs = Manager()

try:
```

```
    # Insert a row into 'cars.trucks' table using default (primary) index
    hs.insert('cars', 'trucks', [('id', '1'), ('company', 'Scania'), ('model', 'G400
↪')])
except OperationalError, e:
    print 'Could not insert because of "%s" error' % str(e)
except ConnectionError, e:
    print 'Unable to perform operation due to a connection error. Original error: "%s"
↪' % str(e)
```

Note: Look how the data is passed - it is a list of field-value pairs. Make sure that all values are strings.

Now let's get that data back:

```
from pyhs import Manager

hs = Manager()

try:
    data = hs.get('cars', 'trucks', ['id', 'company', 'model'], '1')
    print dict(data)
except OperationalError, e:
    print 'Could not get because of "%s" error' % str(e)
except ConnectionError, e:
    print 'Unable to perform operation due to a connection error. Original error: "%s"
↪' % str(e)
```

Note: *get()* is a wrapper over *find()*. It only fetches one row searched for by a single comparison value and uses only primary index for this. For more complex operations please use `find`. Make sure that the first field in the fields list is the one that is searched by and that the list is ordered in the same way fields are present in the index.

`find` and `get` return list of field-value pairs as result.

A more complex `find` request with composite index and custom servers:

```
from pyhs import Manager

# When several hosts are available, client code will try to use both of them
# to balance the load and will retry requests in case of failure on one of them.
read_servers = [('inet', '1.1.1.1', 9998), ('inet', '2.2.2.2', 9998)]
write_servers = [[('inet', '1.1.1.1', 9999), ('inet', '2.2.2.2', 9999)]]
hs = Manager(read_servers, write_servers)

try:
    # This will fetch maximum of 10 rows with 'id' >= 1 and company >= 'Scania'.
    # Unfortunately, HandlerSocket doesn't support multiple condition operations
    # on a single request.
    data = hs.find('cars', 'trucks', '>=', ['id', 'company', 'model'], ['1', 'Scania
↪'], 'custom_index_name', 10)
    # Return value is a list of rows, each of them is a list of (field, value) tuples.
    print [dict(row) for row in data]
except OperationalError, e:
    print 'Could not find because of "%s" error' % str(e)
except ConnectionError, e:
    print 'Unable to perform operation due to a connection error. Original error: "%s"
↪' % str(e)
```

Note: Fields and condition values must be ordered in the same way as present in the index (in case it's composite). All fields that aren't in the index may be ordered randomly.

Another important thing is the `limit` parameter. In case multiple results are expected to be returned by the database, this must be set explicitly. HandlerSocket will **not** return all of them by default.

A sample of increment operation with original value returned as result. Similar one exists for decrement.:

```
from pyhs import Manager

hs = Manager()

try:
    # "incr" increments a numeric value by defined step parameter. By default it is '1
↪'.
    original = hs.incr('cars', 'trucks', '=', ['id'], ['1'], return_original=True)
    print original
    # This will return ['1'] but the new value would be ['2']
except OperationalError, e:
    print 'Could not find because of "%s" error' % str(e)
except ConnectionError, e:
    print 'Unable to perform operation due to a connection error. Original error: "%s"
↪' % str(e)
```

Low level

A small overview of how to operate HandlerSocket. An opened index is required to perform any operation. To do this, use *sockets.HandlerSocket.get_index_id()* which will open the index and return its `id`.

Note: Id's are cached internally by the client and it will return existing id (without opening a new index) in case same `db`, `table` and list of `columns` is passed.

This `id` will must used in all further operations that operate over the same index and columns. There are two classes that must be used to perform actual operations: *sockets.ReadSocket* for reads and `socket.WriteSocket` for writes.

An example:

```
from pyhs.sockets import ReadSocket

hs = ReadSocket([('inet', '127.0.0.1', 9998)])

try:
    index_id = hs.get_index_id('cars', 'trucks', ['id', 'company', 'model'])
    data = hs.find(index_id, '=', ['1'])
    # Data will contain a list of results. Each result is a list of row's values.
    print data
except OperationalError, e:
    print 'Could not find because of "%s" error' % str(e)
except ConnectionError, e:
    print 'Unable to perform operation due to a connection error. Original error: "%s"
↪' % str(e)
```

Exception handling

There are three exceptions that client may raise:

exceptions.ConnectionError Something bad happened to HandlerSocket connection. Data could not be sent or received. Actual reason will be present in the first exception instance's argument. Note that the client may retry operations in case several hosts are defined.

exceptions.OperationalError Raised when HandlerSocket returned an error. Error code is present in the exception instance.

exceptions.IndexedConnectionError `ConnectionError` happened when performing an operation with already opened index. High level client uses this to retry whole operation in case something correctable failed. Developer might want to use it if low level client is used.

See also:

API reference Description of all public interfaces provided by both parts of the client

CHAPTER 3

API

This is the pyhs reference documentation, autogenerated from the source code.

sockets

class pyhs.sockets.**Connection**(*protocol*, *host*, *port=None*, *timeout=None*)

Single HandlerSocket connection.

Maintains a streamed socket connection and defines methods to send and read data from it. In case of failure `retry_time` will be set to the exact time after which the connection may be retried to deal with temporary connection issues.

Parameters

- **protocol** (*string*) – socket protocol (*'unix'* and *'inet'* are supported).
- **host** (*string*) – server host for *'inet'* protocol or socket file path for *'unix'*.
- **port** (*integer or None*) – server port for *'inet'* protocol connection.
- **timeout** (*integer or None*) – timeout value for socket, default is defined in `DEFAULT_TIMEOUT`.

connect()

Establishes connection with a new socket. If some socket is associated with the instance - no new socket will be created.

disconnect()

Closes a socket and disassociates it from the connection instance.

Note: It ignores any socket exceptions that might happen in process.

is_ready()

Checks if connection instance is ready to be used.

Return type bool

readline()
Reads one line from the socket stream and returns it. Lines are expected to be delimited with LF. Throws *ConnectionError* in case of failure.

Return type string

Note: Currently Connection class supports only one line per request/response. All data in the stream after first LF will be ignored.

---

send(*data*)
Sends all given data into the socket stream. Throws *ConnectionError* in case of failure.

Parameters **data** (*string*) – data to send

set_debug_mode(*mode*)
Changes debugging mode of the connection. If enabled, some debugging info will be printed to stdout.

Parameters **mode** (*bool*) – mode value

class pyhs.sockets.**HandlerSocket**(*servers*, *debug=False*)
Pool of HandlerSocket connections.

Manages connections and defines common HandlerSocket operations. Uses internal index id cache. Subclasses `threading.local` to put connection pool and indexes data in thread-local storage as they're not safe to share between threads.

Warning: Shouldn't be used directly in most cases. Use *ReadSocket* for read operations and *WriteSocket* for writes.

Pool constructor initializes connections for all given HandlerSocket servers.

Parameters

- **servers** (*iterable*) – a list of lists that define server data, *format*: `(protocol, host, port, timeout)`. See *Connection* for details.
- **debug** (*bool*) – enable or disable debug mode, default is `False`.

get_index_id(*db*, *table*, *fields*, *index_name=None*)
Returns index id for given index data. This id must be used in all operations that use given data.

Uses internal index cache that keys index ids on a combination of: `db:table:index_name:fields`. In case no index was found in the cache, a new index will be opened.

Note: `fields` is position-dependent, so change of fields order will open a new index with another index id.

Parameters

- **db** (*string*) – database name.
- **table** (*string*) – table name.
- **fields** (*iterable*) – list of table's fields that would be used in further operations. See `_open_index()` for more info on fields order.

- **index_name** (*string or None*) – name of the index, default is PRIMARY.

Return type integer or None

purge()
Closes all connections, cleans caches, zeroes index id counter.

purge_index(*index_id*)
Clear single index connection and cache.

Parameters **index_id** (*integer*) – id of the index to purge.

purge_indexes()
Closes all indexed connections, cleans caches, zeroes index id counter.

class pyhs.sockets.**ReadSocket**(*servers*, *debug=False*)
HandlerSocket client for read operations.

Pool constructor initializes connections for all given HandlerSocket servers.

Parameters

- **servers** (*iterable*) – a list of lists that define server data, *format*: (protocol, host, port, timeout). See *Connection* for details.
- **debug** (*bool*) – enable or disable debug mode, default is False.

find(*index_id*, *operation*, *columns*, *limit=0*, *offset=0*)
Finds row(s) via opened index.

Raises ValueError if given data doesn't validate.

Parameters

- **index_id** (*integer*) – id of opened index.
- **operation** (*string*) – logical comparison operation to use over columns. Currently allowed operations are defined in FIND_OPERATIONS. Only one operation is allowed per call.
- **columns** (*iterable*) – list of column values for comparison operation. List must be ordered in the same way as columns are defined in opened index.
- **limit** (*integer*) – optional limit of results to return. Default is one row. In case multiple results are expected, limit must be set explicitly, HS wont return all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.

Return type list

class pyhs.sockets.**WriteSocket**(*servers*, *debug=False*)
HandlerSocket client for write operations.

Pool constructor initializes connections for all given HandlerSocket servers.

Parameters

- **servers** (*iterable*) – a list of lists that define server data, *format*: (protocol, host, port, timeout). See *Connection* for details.
- **debug** (*bool*) – enable or disable debug mode, default is False.

find_modify(*index_id*, *operation*, *columns*, *modify_operation*, *modify_columns=[]*, *limit=0*, *offset=0*)
Updates/deletes row(s) using opened index.

Returns number of modified rows or a list of original values in case `modify_operation` ends with `?`.

Raises `ValueError` if given data doesn't validate.

Parameters

- **index_id** (*integer*) – id of opened index.
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **columns** (*iterable*) – list of column values for comparison operation. List must be ordered in the same way as columns are defined in opened index.
- **modify_operation** (*string*) – modification operation (update or delete). Currently allowed operations are defined in `MODIFY_OPERATIONS`.
- **modify_columns** (*iterable*) – list of column values for update operation. List must be ordered in the same way as columns are defined in opened index. Only usable for *update* operation,
- **limit** (*integer*) – optional limit of results to change. Default is one row. In case multiple rows are expected to be changed, `limit` must be set explicitly, HS wont change all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.

Return type list

insert (*index_id*, *columns*)

Inserts single row using opened index.

Raises `ValueError` if given data doesn't validate.

Parameters

- **index_id** (*integer*) – id of opened index.
- **columns** (*list*) – list of column values for insertion. List must be ordered in the same way as columns are defined in opened index.

Return type bool

manager

class `pyhs.manager.`**Manager** (*read_servers=None*, *write_servers=None*, *debug=False*)

High-level client for HandlerSocket.

This should be used in most cases except ones that you need fine-grained control over index management, low-level operations, etc. For such cases *ReadSocket* and *WriteSocket* can be used.

Constructor initializes both read and write sockets.

Parameters

- **read_servers** (*list of tuples or None*) – list of tuples that define HandlerSocket read instances. See format in *HandlerSocket* constructor.
- **write_servers** (*list of tuples or None*) – list of tuples that define HandlerSocket write instances. Format is the same as in `read_servers`.
- **debug** (*bool*) – enable debug mode by passing `True`.

find(*db*, *table*, *operation*, *fields*, *values*, *index_name=None*, *limit=0*, *offset=0*)
Finds rows that meet `values` with comparison `operation` in given `db` and `table`.

Returns a list of lists of pairs. First item in pair is field name, second is its value. For example, if two rows with two columns each are returned:

```
[[('field', 'first_row_value'), ('otherfield', 'first_row_othervalue')],
 [('field', 'second_row_value'), ('otherfield', 'second_row_othervalue')]]
```

Parameters

- **db** (*string*) – database name
- **table** (*string*) – table name
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **fields** (*list*) – list of table's fields to get, ordered by inclusion into the index.
- **values** (*list*) – values to compare to, ordered the same way as items in `fields`.
- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.
- **limit** (*integer*) – optional limit of results. Default is one row. In case multiple rows are expected to be returned, `limit` must be set explicitly, HS wont get all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.

Return type list of lists of tuples

insert(*db*, *table*, *fields*, *index_name=None*)
Inserts a single row into given `table`.

Parameters

- **db** (*string*) – database name.
- **table** (*string*) – table name.
- **fields** (*list of lists*) – list of (column, value) pairs to insert into the `table`.
- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.

Return type bool

update(*db*, *table*, *operation*, *fields*, *values*, *update_values*, *index_name=None*, *limit=0*, *offset=0*, *return_original=False*)
Update row(s) that meet conditions defined by `operation`, `fields` `values` in a given `table`.

Parameters

- **db** (*string*) – database name
- **table** (*string*) – table name
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **fields** (*list*) – list of table's fields to use, ordered by inclusion into the index.
- **values** (*list*) – values to compare to, ordered the same way as items in `fields`.

- **update_values** (*list*) – values to update, ordered the same way as items in `fields`.
- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.
- **limit** (*integer*) – optional limit of rows. Default is one row. In case multiple rows are expected to be updated, `limit` must be set explicitly, HS wont update all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.
- **return_original** (*bool*) – if set to `True`, method will return a list of original values in affected rows. Otherwise - number of affected rows (this is default behaviour).

Return type int or list

incr(*db, table, operation, fields, values, step=['1'], index_name=None, limit=0, offset=0, return_original=False*)

Increments row(s) that meet conditions defined by `operation`, `fields` `values` in a given `table`.

Parameters

- **db** (*string*) – database name
- **table** (*string*) – table name
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **fields** (*list*) – list of table's fields to use, ordered by inclusion into the index.
- **values** (*list*) – values to compare to, ordered the same way as items in `fields`.
- **step** (*list*) – list of increment steps, ordered the same way as items in `fields`.
- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.
- **limit** (*integer*) – optional limit of rows. Default is one row. In case multiple rows are expected to be updated, `limit` must be set explicitly, HS wont update all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.
- **return_original** (*bool*) – if set to `True`, method will return a list of original values in affected rows. Otherwise - number of affected rows (this is default behaviour).

Return type int or list

decr(*db, table, operation, fields, values, step=['1'], index_name=None, limit=0, offset=0, return_original=False*)

Decrements row(s) that meet conditions defined by `operation`, `fields` `values` in a given `table`.

Parameters

- **db** (*string*) – database name
- **table** (*string*) – table name
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **fields** (*list*) – list of table's fields to use, ordered by inclusion into the index.
- **values** (*list*) – values to compare to, ordered the same way as items in `fields`.
- **step** (*list*) – list of decrement steps, ordered the same way as items in `fields`.

- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.
- **limit** (*integer*) – optional limit of rows. Default is one row. In case multiple rows are expected to be updated, `limit` must be set explicitly, HS wont update all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.
- **return_original** (*bool*) – if set to `True`, method will return a list of original values in affected rows. Otherwise - number of affected rows (this is default behaviour).

Return type int or list

delete(*db*, *table*, *operation*, *fields*, *values*, *index_name=None*, *limit=0*, *offset=0*, *return_original=False*)

Delete row(s) that meet conditions defined by `operation`, `fields` `values` in a given `table`.

Parameters

- **db** (*string*) – database name
- **table** (*string*) – table name
- **operation** (*string*) – logical comparison operation to use over `columns`. Currently allowed operations are defined in `FIND_OPERATIONS`. Only one operation is allowed per call.
- **fields** (*list*) – list of table's fields to use, ordered by inclusion into the index.
- **values** (*list*) – values to compare to, ordered the same way as items in `fields`.
- **index_name** (*string or None*) – name of the index to open, default is `PRIMARY`.
- **limit** (*integer*) – optional limit of rows. Default is one row. In case multiple rows are expected to be deleted, `limit` must be set explicitly, HS wont delete all found rows by default.
- **offset** (*integer*) – optional offset of rows to search for.
- **return_original** (*bool*) – if set to `True`, method will return a list of original values in affected rows. Otherwise - number of affected rows (this is default behaviour).

Return type int or list

get(*db*, *table*, *fields*, *value*)

A wrapper over *find()* that gets a single row with a single field look up.

Returns a list of pairs. First item in pair is field name, second is its value.

If multiple result rows, different comparison operation or composite indexes are needed please use *find()* instead.

Parameters

- **db** (*string*) – database name.
- **table** (*string*) – table name.
- **fields** (*list*) – list of table's fields to get, ordered by inclusion into the index. First item must always be the look up field.
- **value** (*string*) – a look up value.

Return type list of tuples

purge()
Purges all read and write connections. All requests after that operation will open new connections, index caches will be cleaned too.

exceptions

Exceptions used with HandlerSocket client.

exception pyhs.exceptions.**ConnectionError**
Raised on socket connection problems.

exception pyhs.exceptions.**OperationalError**
Raised on client operation errors.

exception pyhs.exceptions.**RecoverableConnectionError**
Raised on socket connection errors that can be attempted to recover instantly.

Python Module Index

p

Index