PyStratum MySQL MariaDB Documentation

P.R. Water

Jul 08, 2022

Contents:

A stored procedure and function loader and wrapper generator for MySQL and MariaDB Python.

CHAPTER 1

Licence

This project is licensed under the terms of the MIT-licentie.

CHAPTER 2

API

2.1 pystratum_mysql package

2.1.1 Subpackages

pystratum_mysql.backend package

Submodules

pystratum_mysql.backend.MySqlBackend module

class pystratum_mysql.backend.MySqlBackend.**MySqlBackend**
Bases: pystratum_backend.Backend.Backend

PyStratum Backend for MySQL & MariaDB.

create_constant_worker(*config: configparser.ConfigParser*, *io: pystratum_backend.StratumStyle.StratumStyle*) → Optional[pystratum_backend.ConstantWorker.ConstantWorker]
Creates the object that does the actual execution of the constant command for the backend.

Parameters

- **config** (*ConfigParser*) – The settings from the PyStratum configuration file.
- **io** (*StratumStyle*) – The output object.

Return type ConstantWorker|None

create_routine_loader_worker(*config: configparser.ConfigParser*, *io: pystratum_backend.StratumStyle.StratumStyle*) → Optional[pystratum_backend.RoutineLoaderWorker.RoutineLoaderWorker]
Creates the object that does the actual execution of the routine loader command for the backend.

Parameters

- **config** (*ConfigParser*) – The settings from the PyStratum configuration file.
- **io** (*StratumStyle*) – The output object.

Return type RoutineLoaderWorker|None

create_routine_wrapper_generator_worker(*config: configparser.ConfigParser*, *io: pystratum_backend.StratumStyle.StratumStyle*) → Optional[pystratum_backend.RoutineWrapperGeneratorWorker.Rout

Creates the object that does the actual execution of the routine wrapper generator command for the backend.

Parameters

- **config** (*ConfigParser*) – The settings from the PyStratum configuration file.
- **io** (*StratumStyle*) – The output object.

Return type RoutineWrapperGeneratorWorker|None

pystratum_mysql.backend.MySqlConstantWorker module

class pystratum_mysql.backend.MySqlConstantWorker.**MySqlConstantWorker**(*io: pystratum_backend.StratumStyle.Str*, *config: configparser.ConfigParser*)

Bases: *pystratum_mysql.backend.MySqlWorker.MySqlWorker*, pystratum_common.backend.CommonConstantWorker.CommonConstantWorker

Class for creating constants based on column widths, and auto increment columns and labels for MySQL databases.

static derive_field_length(*column: Dict[str, Any]*) → Optional[int]

Returns the width of a field based on column.

Parameters **column** (*dict*) – The column of which the field is based.

Return type int|None

pystratum_mysql.backend.MySqlRoutineLoaderWorker module

class pystratum_mysql.backend.MySqlRoutineLoaderWorker.**MySqlRoutineLoaderWorker**(*io: pystratum_backend.St*, *config: configparser.ConfigPa*)

Bases: *pystratum_mysql.backend.MySqlWorker.MySqlWorker*, pystratum_common.backend.CommonRoutineLoaderWorker.CommonRoutineLoaderWorker

Class for loading stored routines into a MySQL instance from (pseudo) SQL files.

MAX_LENGTH_BINARY = 255
Maximum length of a varbinary.

MAX_LENGTH_CHAR = 255
Maximum length of a varchar.

MAX_LENGTH_VARBINARY = 4096
Maximum length of a varbinary.

MAX_LENGTH_VARCHAR = 4096
Maximum length of a varchar.

create_routine_loader_helper(*routine_name: str, pystratum_old_metadata: Optional[Dict[KT, VT]], rdbms_old_metadata: Optional[Dict[KT, VT]]*) → pystratum_mysql.helper.MySqlRoutineLoaderHelper.MySqlRoutineLoaderHelper
Creates a Routine Loader Helper object.

Parameters

- **routine_name** (*str*) – The name of the routine.
- **pystratum_old_metadata** (*dict*) – The old metadata of the stored routine from PyStratum.
- **rdbms_old_metadata** (*dict*) – The old metadata of the stored routine from MySQL.

Return type *MySqlRoutineLoaderHelper*

pystratum_mysql.backend.MySqlRoutineWrapperGeneratorWorker module

class pystratum_mysql.backend.MySqlRoutineWrapperGeneratorWorker.**MySqlRoutineWrapperGenerat**

Bases: *pystratum_mysql.backend.MySqlWorker.MySqlWorker*, pystratum_common.backend.CommonRoutineWrapperGeneratorWorker.CommonRoutineWrapperGeneratorWorker

Class for generating a class with wrapper methods for calling stored routines in a MySQL database.

pystratum_mysql.backend.MySqlWorker module

class pystratum_mysql.backend.MySqlWorker.**MySqlWorker**(*io: pystratum_backend.StratumStyle.StratumStyle, config: configparser.ConfigParser*)

Bases: object

connect() → None
Connects to the database.

disconnect() → None
Disconnects from the database.

Module contents

pystratum_mysql.helper package

Submodules

pystratum_mysql.helper.MySqlDataTypeHelper module

class pystratum_mysql.helper.MySqlDataTypeHelper.**MySqlDataTypeHelper**
Bases: pystratum_common.helper.DataTypeHelper.DataTypeHelper

Utility class for deriving information based on a MySQL data type.

column_type_to_python_type(*data_type_info: Dict[str, Any]*) → str
Returns the corresponding Python data type of a MySQL data type.

Parameters **data_type_info** (*dict*) – The MySQL data type metadata.

Return type str

column_type_to_python_type_hint(*data_type_info: Dict[str, Any]*) → str
Returns the corresponding Python data type hinting of a MySQL data type.

Parameters **data_type_info** (*dict*) – The MySQL data type metadata.

Return type str

pystratum_mysql.helper.MySqlRoutineLoaderHelper module

class pystratum_mysql.helper.MySqlRoutineLoaderHelper.**MySqlRoutineLoaderHelper**(*io: pystratum_backend.StratumStyle*, *dl: pystratum_mysql.MySqlDataLayer*, *routine_filename: str*, *routine_file_encoding: str*, *pystratum_old_metadata: Optional[Dict[KT, VT]]*, *replace_pairs: Dict[str, Any]*, *rdbms_old_metadata: Optional[Dict[KT, VT]]*, *sql_mode: str*, *character_set: str*, *collate: str*)

Bases: pystratum_common.helper.RoutineLoaderHelper.RoutineLoaderHelper

Class for loading a single stored routine into a MySQL instance from a (pseudo) SQL file.

Module contents

pystratum_mysql.wrapper package

Submodules

pystratum_mysql.wrapper.MySqlBulkWrapper module

class pystratum_mysql.wrapper.MySqlBulkWrapper.**MySqlBulkWrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.BulkWrapper.BulkWrapper

Wrapper method generator for stored procedures with large result sets.

pystratum_mysql.wrapper.MySqlFunctionsWrapper module

class pystratum_mysql.wrapper.MySqlFunctionsWrapper.**MySqlFunctionsWrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.FunctionsWrapper.FunctionsWrapper

Wrapper method generator for stored functions.

pystratum_mysql.wrapper.MySqlLogWrapper module

class pystratum_mysql.wrapper.MySqlLogWrapper.**MySqlLogWrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.LogWrapper.LogWrapper

Wrapper method generator for stored procedures with designation type log.

pystratum_mysql.wrapper.MySqlMultiWrapper module

class pystratum_mysql.wrapper.MySqlMultiWrapper.**MySqlMultiWrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.MultiWrapper.MultiWrapper

Wrapper method generator for stored procedures with designation type multi.

pystratum_mysql.wrapper.MySqlNoneWrapper module

class pystratum_mysql.wrapper.MySqlNoneWrapper.**MySqlNoneWrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.

wrapper.NoneWrapper.NoneWrapper

Wrapper method generator for stored procedures without any result set.

pystratum_mysql.wrapper.MySqlRow0Wrapper module

class pystratum_mysql.wrapper.MySqlRow0Wrapper.**MySqlRow0Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.Row0Wrapper.Row0Wrapper

Wrapper method generator for stored procedures that are selecting 0 or 1 row.

pystratum_mysql.wrapper.MySqlRow1Wrapper module

class pystratum_mysql.wrapper.MySqlRow1Wrapper.**MySqlRow1Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.Row1Wrapper.Row1Wrapper

Wrapper method generator for stored procedures that are selecting 1 row.

pystratum_mysql.wrapper.MySqlRowsWithIndexWrapper module

class pystratum_mysql.wrapper.MySqlRowsWithIndexWrapper.**MySqlRowsWithIndexWrapper**(*routine: Dict[str, Any], lob_as_strin bool*)

Bases: pystratum_common.wrapper.RowsWithIndexWrapper.RowsWithIndexWrapper, *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*

Wrapper method generator for stored procedures whose result set must be returned using tree structure using a combination of non-unique columns.

pystratum_mysql.wrapper.MySqlRowsWithKeyWrapper module

class pystratum_mysql.wrapper.MySqlRowsWithKeyWrapper.**MySqlRowsWithKeyWrapper**(*routine: Dict[str, Any], lob_as_string_flag bool*)

Bases: pystratum_common.wrapper.RowsWithKeyWrapper.RowsWithKeyWrapper, *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*

Wrapper method generator for stored procedures whose result set must be returned using tree structure using a combination of unique columns.

pystratum_mysql.wrapper.MySqlRowsWrapper module

class pystratum_mysql.wrapper.MySqlRowsWrapper.**MySqlRowsWrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.RowsWrapper.RowsWrapper

Wrapper method generator for stored procedures that are selecting 0, 1, or more rows.

pystratum_mysql.wrapper.MySqlSingleton0Wrapper module

class pystratum_mysql.wrapper.MySqlSingleton0Wrapper.**MySqlSingleton0Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.Singleton0Wrapper.Singleton0Wrapper

Wrapper method generator for stored procedures that are selecting 0 or 1 row with one column only.

pystratum_mysql.wrapper.MySqlSingleton1Wrapper module

class pystratum_mysql.wrapper.MySqlSingleton1Wrapper.**MySqlSingleton1Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: pystratum_common.wrapper.Singleton1Wrapper.Singleton1Wrapper, *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*

Wrapper method generator for stored procedures that are selecting 1 row with one column only.

pystratum_mysql.wrapper.MySqlTableWrapper module

class pystratum_mysql.wrapper.MySqlTableWrapper.**MySqlTableWrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper*, pystratum_common.wrapper.TableWrapper.TableWrapper

Wrapper method generator for printing the result set of stored procedures in a table format.

pystratum_mysql.wrapper.MySqlWrapper module

class pystratum_mysql.wrapper.MySqlWrapper.**MySqlWrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: pystratum_common.wrapper.Wrapper.Wrapper, abc.ABC

Parent class for wrapper method generators for stored procedures and functions.

is_lob_parameter(*parameters: List[Dict[str, Any]]*) → bool
Returns True of one of the parameters is a BLOB or CLOB. Otherwise, returns False.

Parameters **parameters** – The parameters of a stored routine.

Return type bool:

Module contents

pystratum_mysql.wrapper.**create_routine_wrapper**(*routine: Dict[str, Any]*, *lob_as_string_flag: bool*) → pystratum_mysql.wrapper.MySqlWrapper.MySqlWrapper
A factory for creating the appropriate object for generating a wrapper method for a stored routine.

Parameters

- **routine** (*dict[str,str]*) – The metadata of the sored routine.
- **lob_as_string_flag** (*bool*) – If True BLOBs and CLOBs must be treated as strings.

Return type *MySqlWrapper*

2.1.2 Submodules

2.1.3 pystratum_mysql.MySqlConnector module

class pystratum_mysql.MySqlConnector.**MySqlConnector**
Bases: object

Interface for classes for connecting to a MySql instances.

connect() → mysql.connector.connection.MySQLConnection
Connects to the MySql instance.

disconnect() → None
Disconnects from the MySql instance.

is_alive() → bool
Returns whether Python is (still) connected to a MySQL or MariaDB instance.

Return type bool

2.1.4 pystratum_mysql.MySqlDataLayer module

class pystratum_mysql.MySqlDataLayer.**MySqlDataLayer**(*connector: pystratum_mysql.MySqlConnector.MySqlConnector*)
Bases: object

Class for connecting to a MySQL instance and executing SQL statements. Also, a parent class for classes with static wrapper methods for executing stored procedures and functions.

commit() → None
Commits the current transaction. See https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-commit.html

connect() → None
Connects to a MySQL instance. See https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html for a complete overview of all possible keys in config.

connect_if_not_alive() → None
Connects or reconnects to the MySQL or MariaDB instance when Python is not (longer) connected to a MySQL or MariaDB instance. See https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html for a complete overview of all possible keys in config.

disconnect() → None
Disconnects from the MySQL instance. See https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-disconnect.html.

execute_multi(*sql: str*) → None
Executes a multi query that does not select any rows.

Parameters **sql** (*str*) – The SQL statements.

execute_none(*sql: str*, **params*) → int
Executes a query that does not select any rows. Returns the number of affected rows.

Parameters

- **sql** (*str*) – The SQL statement.
- **params** (*iterable*) – The values for the statement.

Return type int

execute_rows(*sql: str*, **params*) → List[Dict[str, Any]]
Executes a query that selects 0 or more rows. Returns the selected rows (an empty list if no rows are selected).

Parameters

- **sql** (*str*) – The SQL statement.
- **params** (*iterable*) – The arguments for the statement.

Return type list[dict[str,*]]

execute_singleton1(*sql: str*, **params*) → Any
Executes SQL statement that selects 1 row with 1 column. Returns the value of the selected column.

Parameters

- **sql** (*str*) – The SQL calling the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type *:

execute_sp_bulk(*bulk_handler: pystratum_middle.BulkHandler.BulkHandler*, *sql: str*, **params*) → int
Executes a stored routine with designation type "bulk". Returns the number of rows processed.

Parameters

- **bulk_handler** (*BulkHandler*) – The bulk handler for processing the selected rows.
- **sql** (*str*) – The SQL statement for calling the stored routine.
- **params** (*iterable*) – The arguments for calling the stored routine.

Return type int

execute_sp_log(*sql: str*, **params*) → int
Executes a stored routine with designation type "log". Returns the number of log messages.

Parameters

- **sql** (*str*) – The SQL statement for calling the stored routine.

- **params** (*iterable*) – The arguments for calling the stored routine.

Return type int

execute_sp_multi (*sql: str*, **params*) → List[List[Dict[str, Any]]]
Executes a stored routine with designation type "multi". Returns a list of the result sets.

Parameters

- **sql** (*str*) – The SQL statement for calling the stored routine.
- **params** (*iterable*) – The arguments for calling the stored routine.

Return type list[list[dict[str,*]]]

execute_sp_none (*sql: str*, **params*) → int
Executes a stored routine that does not select any rows. Returns the number of affected rows.

Parameters

- **sql** (*str*) – The SQL calling the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type int

execute_sp_row0 (*sql: str*, **params*) → Optional[Dict[str, Any]]
Executes a stored procedure that selects 0 or 1 row. Returns the selected row or None.

Parameters

- **sql** (*str*) – The SQL call the the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type Noneldict[str,*]

execute_sp_row1 (*sql: str*, **params*) → Dict[str, Any]
Executes a stored procedure that selects 1 row. Returns the selected row.

Parameters

- **sql** (*str*) – The SQL calling the the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type dict[str,*]

execute_sp_rows (*sql: str*, **params*) → List[Dict[str, Any]]
Executes a stored procedure that selects 0 or more rows. Returns the selected rows (an empty list if no rows are selected).

Parameters

- **sql** (*str*) – The SQL statement.
- **params** (*iterable*) – The arguments for the statement.

Return type list[dict[str,*]]

execute_sp_singleton0 (*sql: str*, **params*) → Any
Executes a stored procedure that selects 0 or 1 row with 1 column. Returns the value of selected column or None.

Parameters

- **sql** (*str*) – The SQL calling the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type

-

execute_sp_singleton1(*sql: str*, **params*) → Any
Executes a stored routine with designation type "table", i.e a stored routine that is expected to select 1 row with 1 column.

Parameters

- **sql** (*str*) – The SQL calling the the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type

- The value of the selected column.

is_alive() → bool
Returns whether Python is (still) connected to a MySQL or MariaDB instance.

Return type bool

last_sql() → str
Returns the last execute SQL statement.

line_buffered = None
If True log messages from stored procedures with designation type 'log' are line buffered (Note: In python sys.stdout is buffered by default).

rollback() → None
Rolls back the current transaction. See https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-rollback.html

start_transaction(*consistent_snapshot: bool = False*, *isolation_level: str = 'READ-COMMITTED'*, *readonly: Optional[bool] = None*) → None
Starts a transaction. See https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-start-transaction.html

Parameters

- **consistent_snapshot** (*bool*) –
- **isolation_level** (*str*) –
- **readonly** (*bool*) –

2.1.5 pystratum_mysql.MySqlDefaultConnector module

class pystratum_mysql.MySqlDefaultConnector.**MySqlDefaultConnector**(*params: Dict[str, Union[str, int]]*)

Bases: *pystratum_mysql.MySqlConnector.MySqlConnector*

Connects to a MySQL instance using username and password.

connect() → mysql.connector.connection.MySQLConnection
Connects to the MySQL instance.

disconnect() → None
Disconnects from the MySQL instance.

is_alive() → bool
Returns whether Python is (still) connected to a MySQL or MariaDB instance.

Return type bool

2.1.6 pystratum_mysql.MySqlMetadataDataLayer module

class pystratum_mysql.MySqlMetadataDataLayer.**MySqlMetadataDataLayer**(*io: pystratum_backend.StratumStyle.StratumStyle*, *connector: pystratum_mysql.MySqlConnector.MySqlConnector*)

Bases: pystratum_common.MetadataDataLayer.MetadataDataLayer

Data layer for retrieving metadata and loading stored routines.

call_stored_routine(*routine_name: str*) → int
Class a stored procedure without arguments.

Parameters **routine_name** (*str*) – The name of the procedure.

Return type int

check_table_exists(*table_name: str*) → int
Checks if a table exists in the current schema.

Parameters **table_name** (*str*) – The name of the table.

Return type int

connect() → None
Connects to a MySQL instance.

describe_table(*table_name: str*) → List[Dict[str, Any]]
Describes a table.

Parameters **table_name** (*str*) – The name of the table.

Return type list[dict[str,*]]

disconnect() → None
Disconnects from the MySQL instance.

drop_stored_routine(*routine_type: str*, *routine_name: str*) → None
Drops a stored routine if it exists.

Parameters

- **routine_type** (*str*) – The type of the routine (i.e. PROCEDURE or FUNCTION).
- **routine_name** (*str*) – The name of the routine.

drop_temporary_table(*table_name: str*) → None
Drops a temporary table.

Parameters **table_name** (*str*) – The name of the table.

execute_none(*query: str*) → int
Executes a query that does not select any rows.

Parameters **query** (*str*) – The query.

Return type int

execute_rows (*query: str*) → List[Dict[str, Any]]
Executes a query that selects 0 or more rows. Returns the selected rows (an empty list if no rows are selected).

Parameters **query** (*str*) – The query.

Return type list[dict[str,*]]

execute_singleton1 (*query: str*) → Any
Executes SQL statement that selects 1 row with 1 column. Returns the value of the selected column.

Parameters **query** (*str*) – The query.

Return type

-

get_all_table_columns () → List[Dict[str, Union[str, int, None]]]
Selects metadata of all columns of all tables.

Return type list[dict[str,*]]

get_correct_sql_mode (*sql_mode: str*) → str
Selects the SQL mode in the order as preferred by MySQL.

Parameters **sql_mode** (*str*) – The SQL mode.

Return type str

get_label_tables (*regex: str*) → List[Dict[str, Any]]
Selects metadata of tables with a label column.

Parameters **regex** (*str*) – The regular expression for columns which we want to use.

Return type list[dict[str,*]]

get_labels_from_table (*table_name: str*, *id_column_name: str*, *label_column_name: str*) → List[Dict[str, Any]]
Selects all labels from a table with labels.

Parameters

- **table_name** (*str*) – The name of the table.
- **id_column_name** (*str*) – The name of the auto increment column.
- **label_column_name** (*str*) – The name of the column with labels.

Return type list[dict[str,*]]

get_routine_parameters (*routine_name: str*) → List[Dict[str, Any]]
Selects metadata of the parameters of a stored routine.

Parameters **routine_name** (*str*) – The name of the routine.

Return type list[dict[str,*]]

get_routines () → List[Dict[str, Any]]
Selects metadata of all routines in the current schema.

Return type list[dict[str,*]]

last_sql () → str
The last executed SQL statement.

Return type str

set_character_set(*character_set: str*, *collate: str*) → None
Sets the default character set and collate.

Parameters

- **character_set** (*str*) – The name of the character set.
- **collate** (*str*) – The name of the collate

set_sql_mode(*sql_mode: str*) → None
Sets the SQL mode.

Parameters **sql_mode** (*str*) – The SQL mode.

2.1.7 Module contents

Python Module Index

p

Index

G

I

L

M

P

R

S