PyStratum PostgreSQL Documentation

P.R. Water

Oct 26, 2020

Contents:

A stored procedure and function loader and wrapper generator for PostgreSQL Python.

CHAPTER 1

Licence

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

CHAPTER 2

API

2.1 pystratum_pgsql package

2.1.1 Subpackages

pystratum_pgsql.backend package

Submodules

pystratum_pgsql.backend.PgSqlBackend module

class pystratum_pgsql.backend.PgSqlBackend.**PgSqlBackend**
Bases: pystratum_backend.Backend.Backend

Semi interface for PyStratum's backends.

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_pgsql.backend.PgSqlConstantWorker module

class pystratum_pgsql.backend.PgSqlConstantWorker.**PgSqlConstantWorker**(*io: pystratum_backend.StratumStyle.Str* *config: configparser.ConfigParser*)

Bases: *pystratum_pgsql.backend.PgSqlWorker.PgSqlWorker*, pystratum_common.backend.CommonConstantWorker.CommonConstantWorker

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

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

Returns the width of a field based on the data type of column.

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

Return type int|None

pystratum_pgsql.backend.PgSqlRoutineLoaderWorker module

class pystratum_pgsql.backend.PgSqlRoutineLoaderWorker.**PgSqlRoutineLoaderWorker**(*io: pystratum_backend.St* *config: configparser.ConfigPa*

Bases: *pystratum_pgsql.backend.PgSqlWorker.PgSqlWorker*, pystratum_common.backend.CommonRoutineLoaderWorker.CommonRoutineLoaderWorker

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

create_routine_loader_helper(*routine_name: str, pystratum_old_metadata: Optional[Dict[KT, VT]], rdbms_old_metadata: Optional[Dict[KT, VT]]*) → pystratum_pgsql.helper.PgSqlRoutineLoaderHelper.PgSqlRoutineLoaderHelper

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 PostgreSQL.

Return type *PgSqlRoutineLoaderHelper*

pystratum_pgsql.backend.PgSqlRoutineWrapperGeneratorWorker module

class pystratum_pgsql.backend.PgSqlRoutineWrapperGeneratorWorker.**PgSqlRoutineWrapperGenerat**

Bases: *pystratum_pgsql.backend.PgSqlWorker.PgSqlWorker*, pystratum_common.backend.CommonRoutineWrapperGeneratorWorker.CommonRoutineWrapperGeneratorWorker

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

pystratum_pgsql.backend.PgSqlWorker module

class pystratum_pgsql.backend.PgSqlWorker.**PgSqlWorker**(*io: pystratum_backend.StratumStyle.StratumStyle, config: configparser.ConfigParser*)

Bases: object

Class for connecting to PostgreSQL instances and reading PostgreSQL specific connection parameters from configuration files.

connect() → None
Connects to the PostgreSQL instance.

disconnect() → None
Disconnects from the PostgreSQL instance.

Module contents

pystratum_pgsql.helper package

Submodules

pystratum_pgsql.helper.PgSqlDataTypeHelper module

class pystratum_pgsql.helper.PgSqlDataTypeHelper.**PgSqlDataTypeHelper**
Bases: pystratum_common.helper.DataTypeHelper.DataTypeHelper

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

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

Parameters **data_type_info** (*dict*) – The PostgreSQL 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 PostgreSQL data type.

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

Return type str

pystratum_pgsql.helper.PgSqlRoutineLoaderHelper module

class pystratum_pgsql.helper.PgSqlRoutineLoaderHelper.**PgSqlRoutineLoaderHelper**(*io: py-s-tra-tum_backend.Str dl: py-s-tra-tum_pgsql.PgSql rou-tine_filename: str, rou-tine_file_encodin str, py-s-tra-tum_old_metada Dict[KT, VT], re-place_pairs: Dict[str, str], rdbms_old_meta Dict[KT, VT]*)

Bases: pystratum_common.helper.RoutineLoaderHelper.RoutineLoaderHelper

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

get_bulk_insert_table_columns_info() → None

Gets the column names and column types of the current table for bulk insert.

Module contents

pystratum_pgsql.wrapper package

Submodules

pystratum_pgsql.wrapper.PgSqlFunctionsWrapper module

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

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.FunctionsWrapper.FunctionsWrapper

Wrapper method generator for stored functions.

pystratum_pgsql.wrapper.PgSqlLogWrapper module

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

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.LogWrapper.LogWrapper

Wrapper method generator for stored procedures with designation type log.

pystratum_pgsql.wrapper.PgSqlNoneWrapper module

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

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.NoneWrapper.NoneWrapper

Wrapper method generator for stored procedures without any result set.

pystratum_pgsql.wrapper.PgSqlRow0Wrapper module

class pystratum_pgsql.wrapper.PgSqlRow0Wrapper.**PgSqlRow0Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.Row0Wrapper.Row0Wrapper

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

pystratum_pgsql.wrapper.PgSqlRow1Wrapper module

class pystratum_pgsql.wrapper.PgSqlRow1Wrapper.**PgSqlRow1Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.Row1Wrapper.Row1Wrapper

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

pystratum_pgsql.wrapper.PgSqlRowsWithIndexWrapper module

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

Bases: pystratum_common.wrapper.RowsWithIndexWrapper.RowsWithIndexWrapper, *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*

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

pystratum_pgsql.wrapper.PgSqlRowsWithKeyWrapper module

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

Bases: pystratum_common.wrapper.RowsWithKeyWrapper.RowsWithKeyWrapper, *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*

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

pystratum_pgsql.wrapper.PgSqlRowsWrapper module

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

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.RowsWrapper.RowsWrapper

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

pystratum_pgsql.wrapper.PgSqlSingleton0Wrapper module

class pystratum_pgsql.wrapper.PgSqlSingleton0Wrapper.**PgSqlSingleton0Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.Singleton0Wrapper.Singleton0Wrapper

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

pystratum_pgsql.wrapper.PgSqlSingleton1Wrapper module

class pystratum_pgsql.wrapper.PgSqlSingleton1Wrapper.**PgSqlSingleton1Wrapper**(*routine: Dict[str, Any], lob_as_string_flag: bool*)

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.Singleton1Wrapper.Singleton1Wrapper

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

pystratum_pgsql.wrapper.PgSqlTableWrapper module

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

Bases: *pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper*, pystratum_common.wrapper.TableWrapper.TableWrapper

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

pystratum_pgsql.wrapper.PgSqlWrapper module

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

Bases: pystratum_common.wrapper.Wrapper.Wrapper

Parent class for wrapper method generators for stored functions.

is_lob_parameter(*parameters*)

Returns True if one of the parameters is a BLOB or CLOB. Otherwise, returns False.

Parameters **parameters** (*list[dict[str,str]]*) – The parameters of a stored routine.

Return type bool

Module contents

PyStratum

pystratum_pgsql.wrapper.**create_routine_wrapper**(*routine: Dict[KT, VT], lob_as_string_flag: bool*) → pystratum_pgsql.wrapper.PgSqlWrapper.PgSqlWrapper

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 *PgSqlWrapper*

2.1.2 Submodules

2.1.3 pystratum_pgsql.PgSqlConnector module

class pystratum_pgsql.PgSqlConnector.**PgSqlConnector**
Bases: object

Interface for classes for connecting to a PostgreSQL instances.

connect() → Any
Connects to a PostgreSQL instance.

Return type psycopg2.extensions.connection

disconnect() → None
Disconnects from a PostgreSQL instance.

2.1.4 pystratum_pgsql.PgSqlDataLayer module

class pystratum_pgsql.PgSqlDataLayer.**PgSqlDataLayer**(*connector: pystratum_pgsql.PgSqlConnector.PgSqlConnector*)
Bases: object

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

commit()
Commits the current transaction.

connect() → None
Connects to a PostgreSQL instance.

copy_expert(*sql: str*, *file: object*, *size: int = 8192*) → int
Submit a user-composed COPY statement. Returns the number of rows copied.

Parameters

- **sql** (*str*) – The COPY statement to execute.
- **file** (*T*) – A file-like object to read or write (according to sql).
- **size** (*int*) – Size of the read buffer to be used in COPY FROM.

Return type int

copy_from(*file: object*, *table: str*, *sep: str = '\t'*, *null: str = '\\N'*, *size: int = 8192*, *columns: Union[None, Iterable[T_co]] = None*) → int
Read data from the file-like object file appending them to the table named table. Returns the number of rows copied.

Parameters

- **file** (*T*) – File-like object to read data from. It must have both read() and readline() methods.
- **table** (*str*) – Name of the table to copy data into.
- **sep** (*str*) – Columns separator expected in the file. Defaults to a tab.
- **null** (*str*) – Textual representation of NULL in the file. The default is the two characters string N.
- **size** (*int*) – Size of the buffer used to read from the file.

- **columns** (*iterable*) – Iterable with name of the columns to import. The length and types should match the content of the file to read. If not specified, it is assumed that the entire table matches the file structure.

Return type int

copy_to(*file: object*, *table: str*, *sep: str = '\t'*, *null: str = '\\N'*, *columns: Union[None, Iterable[T_co]] = None*) → int

Write the content of the table named table to the file-like object file. Returns the number of rows copied.

Parameters

- **file** (*T*) – File-like object to write data into. It must have a write() method.
- **table** (*str*) – Name of the table to copy data from.
- **sep** (*str*) – Columns separator expected in the file. Defaults to a tab.
- **null** (*str*) – Textual representation of NULL in the file. The default is the two characters string N.
- **columns** (*iterable*) – Iterable with name of the columns to export. If not specified, export all the columns.

Return type int

disconnect() → None

Disconnects from the PostgreSQL instance.

execute_none(*sql: str*, **params*) → int

Executes a query that does not select any rows.

Parameters

- **sql** (*string*) – The SQL statement.
- **params** (*tuple*) – The values for the statement.

Return type int

execute_rows(*sql: str*, **params*) → List[T]

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]

execute_singleton1(*sql: str*, **params*) → object

Executes query that selects 1 row with 1 column. Returns the value of the selected column.

Parameters

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

Return type object

execute_sp_log(*sql: str*, **params*) → int

Executes a stored procedure with log statements. Returns the number of lines in the log.

Parameters

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

Return type int

execute_sp_none(*sql: str*, **params*) → None
Executes a stored procedure that does not select any rows.

Unfortunately, it is not possible to retrieve the number of affected rows of the SQL statement in the stored function as with execute_none (cursor.rowcount is always 1 using cursor.execute and cursor.callproc).

Parameters

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

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

Parameters

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

Return type None|dict[str,object]

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

Parameters

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

Return type dict[str,object]

execute_sp_rows(*sql: str*, **params*) → List[T]
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 call the the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type list[dict[str,object]]

execute_sp_singleton0(*sql: str*, **params*) → object
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 call the the stored procedure.
- **params** (*iterable*) – The arguments for the stored procedure.

Return type object

execute_sp_singleton1(*sql: str*, **params*) → object
Executes a stored procedure that selects 1 row with 1 column. Returns the value of the selected column.

Parameters

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

Return type object

execute_sp_table(*sql: str*, **params*) → int
Executes a stored routine with designation type "table". Returns the number of rows.

Parameters

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

Return type int

line_buffered = True
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.

sp_log_fetch = 'stratum_log_fetch'
The name of the stored routine that must be run after a store routine with designation type "log".

sp_log_init = 'stratum_log_init'
The name of the stored routine that must be run before a store routine with designation type "log".

start_transaction(*isolation_level: str = 'READ-COMMITTED'*, *readonly: bool = False*) → None
Starts a transaction.

Parameters

- **isolation_level** (*str*) – The isolation level.
- **readonly** (*bool*) –

2.1.5 pystratum_pgsql.PgSqlDefaultConnector module

class pystratum_pgsql.PgSqlDefaultConnector.**PgSqlDefaultConnector**(*params: Dict[str, Union[str, int]]*)

Bases: *pystratum_pgsql.PgSqlConnector.PgSqlConnector*

Connects to a PostgreSQL instance using user name and password.

connect() → Any
Connects to a PostgreSQL instance.

Return type psycopg2.extensions.connection

disconnect() → None
Disconnects from a PostgreSQL instance.

2.1.6 pystratum_pgsql.PgSqlMetadataDataLayer module

class pystratum_pgsql.PgSqlMetadataDataLayer.**PgSqlMetadataDataLayer**(*io: pystratum_backend.StratumStyle.StratumStyle*, *connector: pystratum_pgsql.PgSqlConnector.PgSqlConnector*)

Bases: pystratum_common.MetadataDataLayer.MetadataDataLayer

Data layer for retrieving metadata and loading stored routines.

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

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

Return type int|None

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

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

Return type int|None

commit() → None
Commits the current transaction.

connect() → None
Connects to a PostgreSQL instance.

describe_table(*table_name: str*) → List[T]
Describes a table.

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

Return type list[dict[str,Object]]

disconnect() → None
Disconnects from the PostgreSQL instance.

drop_stored_routine(*routine_type: str*, *routine_name: str*, *routine_args: 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.
- **routine_args** (*str*) – The routine arguments types as comma separated list.

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[T]
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,Object]]

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

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

Return type Object

get_all_table_columns () → List[T]
Selects metadata of all columns of all tables.

Return type list[dict[str,Object]]

get_label_tables (*regex: str*) → List[T]
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,Object]]

get_labels_from_table (*table_name: str*, *id_column_name: str*, *label_column_name: str*) → List[T]
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,Object]]

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

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

Return type list[dict[str,Object]]

get_routines () → List[T]
Selects metadata of all routines in the current schema.

Return type list[dict[str,Object]]

rollback () → None
Rollbacks the current transaction.

2.1.7 Module contents

Python Module Index

p

Index

C

D

E

G

I

L

P

R

S