Sqlite-Integrated Documentation

Contents

Module `sqlite_integrated`

Functions

Function `dict_to_sql`

```
def dict_to_sql(
    data: dict
) -> str
```

Converts a dict into sql key value pairs. Ex: “key1 = value1, key2 = value2…”

Function `raw_table_to_table`

```
def raw_table_to_table(
    raw_table: list,
    fields: list,
    table_name: str,
    id_field
)
```

Convert a raw table (list of tuples) to a table (table of dictionaries)

```
Parameters:
    raw_entry: A tuple with the data for the entry. Ex: ´(2, "Tom", "Builder", 33)´
    fields: A list of column names for the data. Ex: ´["id", "FirstName", "LastName", "Age"]
    table_name: The name of the table (in the database) that the data belongs to. Ex: "peopl
    id_field: The name of the column which stores the id. Ex: "id". This can be set to ´None
```

Function `string_to_list`

```
def string_to_list(
    string: str
) -> list
```

Tankes a string with comma seperated values, returns a list of the values. (spaces are ignored)

Function `value_to_sql_value`

```
def value_to_sql_value(
    value
) -> str
```

Converts python values to sql values. Basically just puts quotes around strings and not ints or floats. Also converts None to null

Classes

Class `Database`

```
class Database(
    path: str,
    new=False,
    default_id_field='id',
    silent=False
)
```

Main database class for manipulating sqlite3 databases

Constructor for Database

```
Parameters:
    path:               Path to the database file

Optional
    new:                A new blank database will be created where the ´self.path´ is pointi
    default_id_field:   The default name for the id field in tables
    silent:             Disables all feedback in the form of prints
```

Instance variables

Variable `conn` The sqlite3 connection.

Variable `cursor` The sqlite3 cursor. Use ´cursor.execute(cmd)´ to execute raw sql

Variable `default_id_field` The default name for the id_field in returned DatabaseEntry.

Variable `path` Path to the database file.

Variable `silent` Disables all feedback in the form of prints.

Methods

Method `INSERT_INTO`

```
def INSERT_INTO(
    self,
    table_name
)
```

Start sql INSERT INTO query from the database. Returns a Query to build from.

```
Parameters:
    table_name: Name of the table.
```

Method `SELECT`

```
def SELECT(
    self,
    pattern='*'
)
```

Start sql SELECT query from the database. Returns a Query to build from.

```
Optional:
    pattern: Either a python list or sql list of table names.
```

Method `UPDATE`

```
def UPDATE(
    self,
    table_name
)
```

Start sql UPDATE query from the database. Returns a Query to build from.

```
Parameters:
    table_name: Name of the table.
```

Method `add_table_entry`

```
def add_table_entry(
    self,
    entry: sqlite_integrated.DatabaseEntry,
    fill_null=False,
    silent=False
)
```

Add an entry to the database. The entry must have values for all fields in the table. You can pass ´fill_null=True´ to fill remaining fields with None/null. Use ´silent=True´ to suppress warnings and messages.

```
Parameters:
    entry: The entry. The entry must NOT have an id_field (it has to be ´None´: ´entry.id_fi

Optional:
    fill_null: Fill in unpopulated fields with null values.
    silent: If True: disables prints.
```

Method `close`

```
def close(
    self
)
```

saves and closes the database. If you want to explicitly close without saving use: ´self.conn.close()´

Method `fill_null`

```
def fill_null(
    self,
    entry: sqlite_integrated.DatabaseEntry
)
```

Fills out any unpopulated fields in a DatabaseEntry (fields that exist in the database but not in the entry).

```
Parameters:
    entry: The DatabaseEntry.
```

Method `get_entry_by_id`

```
def get_entry_by_id(
    self,
    table,
    ID,
    id_field=None
)
```

Get table entry by id.

```
Parameters:
    table: Name of the table.
    ID: The entry id.

Optional:
    id_field: The field that holds the id value. Will use default if not set.
```

Method `get_table`

```
def get_table(
    self,
    name: str,
    id_field='',
    get_only=None
) -> list
```

Returns all entries in a table as python dictionaries. This function loops over all entries in the table, so it is not the best in big databases.

```
Parameters:
    name: Name of the table.

Optional:
    id_field: The id_field of the table. Will be set to the database default if not set.
    get_only: Can be set to a list of column/field names, to only retrieve those columns/fie
```

Method `get_table_columns`

```
def get_table_columns(
    self,
    name: str
)
```

Returns the column names for a given table

```
Parameters:
    name: Name of the table.
```

Method `get_table_info`

```
def get_table_info(
    self,
    name: str
)
```

Returns sql information about a table (runs ´PRAGMA TABLE_INFO(name)´).

```
Parameters:
    name: Name of the table.
```

Method `get_table_names`

```
def get_table_names(
    self
) -> list
```

Returns the names of all tables in the database.

Method `get_table_raw`

```
def get_table_raw(
    self,
    name: str,
    get_only=None
) -> list
```

Returns all entries in a table as a list of tuples

```
Parameters:
    name: Name of the table.

Optional:
    get_only: Can be set to a list of column/field names, to only retrieve those columns/fie
```

Method `is_table`

```
def is_table(
    self,
    table_name: str
) -> bool
```

Check if database has a table with a certain name.

```
Parameters:
    table_name: Name to check.
```

Method `overview`

```
def overview(
    self
)
```

Prints an overview of all the tables in the database with their fields.

Method `save`

```
def save(
    self
)
```

Writes any changes to the database file

Method `table_overview`

```
def table_overview(
    self,
    name: str,
    max_len: int = 40,
    get_only=None
)
```

Prints a pretty table (with a name).

```
Parameters:
    name: Name of the table.

Optional:
    max_len: The max number of rows shown.
    get_only: If given a list of column/field names: only shows those
```

Method `update_entry`

```
def update_entry(
    self,
    entry: dict,
    table=None,
    id_field: str = None,
    part=False,
    fill_null=False,
    silent=False
)
```

Update entry in database with a DatabaseEntry, or with a dictionary + the name of the table you want to update.

```
Parameters:
    entry: DatabaseEntry or dictionary, if dictionary you also need to provide table and id_

Optional:
    table: The table name.
    id_field: The field that holds the entry id.
    part: If True: Only updates the provided fields.
    fill_null: Fill in unpopulated fields with null values.
    silent: If True: disables prints.
```

Class `DatabaseEntry`

```
class DatabaseEntry(
    entry_dict: dict,
    table: str,
    id_field
)
```

A python dictionary that keeps track of the table where it came from, and the name and value of its id field. This class is not supposed to be created manually

" Constructs the entry by saving the table and id_field as attributes. The ´entry_dict´ is used to populate this object with data.

```
Parameters:
    id_field: The column name for the entry's id
    table:      The name of the table the entry is a part of
    entry_dict: A dictionary containing all the information. This information can be accesed
```

Ancestors (in MRO)

- builtins.dict

Static methods

`Method from_raw_entry`

```
def from_raw_entry(
    raw_entry: tuple,
    table_fields: list,
    table_name: str,
    id_field
)
```

Alternative constructor for converting a raw entry to a DatabaseEntry.

```
Parameters:
    raw_entry: A tuple with the data for the entry. Ex: ´(2, "Tom", "Builder", 33)´
    table_fields: A list of column names for the data. Ex: ´["id", "FirstName", "LastName",
    table_name: The name of the table (in the database) that the data belongs to. Ex: "peopl
    id_field: The name of the column which stores the id. Ex: "id". This can be set to ´None
```

Class `DatabaseException`

```
class DatabaseException(
    *args,
    **kwargs
)
```

Raised when the database fails to execute command

Ancestors (in MRO)

- builtins.Exception
- builtins.BaseException

Class `Query`

```
class Query(
    db=None
)
```

A class for writing sql queries. Queries can be run on the attached database or
a seperate one with the ´run´ method

Initialize query

```
Optional:
    db: The attached Database. This is the default database to run queries on.
```

Instance variables

Variable `fields` The selected fields

Variable `history` The history of commandmethods run on this object

Variable `sql` The current raw sql command

Variable `table` The table the sql query is interacting with

Methods

Method `FROM`

```
def FROM(
    self,
    table_name
)
```

Sql FROM statement. Has to be preceded by a SELECT statement.

```
Parameters:
    table_name: Name of the table you are selecting from.
```

Method `INSERT_INTO`

```
def INSERT_INTO(
    self,
    table_name
)
```

Sql INSERT INTO statement.

```
Parameters:
    table_name: Name of the table you want to insert into.
```

Method `LIKE`

```
def LIKE(
    self,
    pattern
)
```

Sql WHERE statement. Has to be preceded by a WHERE statement.

```
Parameters:
    pattern: A typical sql LIKE pattern with % and _.
```

Method `SELECT`

```
def SELECT(
    self,
    selection='*'
)
```

Sql SELECT statement.

```
Optional:
    selection: Either a python list or sql list of table names.
```

Method `SET`

```
def SET(
    self,
    data: dict
)
```

Sql SET statement. Must be preceded by an UPDATE statement.

```
Parameters:
    data: A dictionaty with key and value pairs.
```

Method `UPDATE`

```
def UPDATE(
    self,
    table_name: str
)
```

Sql UPDATE statement.

```
Parameters:
    table_name: Name of the table you are updating.
```

Method `VALUES`

```
def VALUES(
    self,
    data: dict
)
```

Sql VALUES statement. Must be preceded by INSERT_INTO statement.

```
Parameters:
    data: Dictionary with key value pairs.
```

Method `WHERE`

```
def WHERE(
    self,
    col_name: str,
    value=''
)
```

Sql WHERE statement.

```
Parameters:
    col_name: The name of the column. You can also just pass it a statement like: ´"id" = 4

Optional:
    value: The value of the column.
```

Method `run`

```
def run(
    self,
    db=None,
    raw=False
)
```

Execute the query in the attached database or in a seperate one. Returns the results in a table (list of DatabaseEntry) or ´None´ if no results.

```
Optional:
    db: The database to execute to query on.
    raw: If True: returns the raw table (list of tuples) instead of the normal table.
```

Method `valid_prefixes`

```
def valid_prefixes(
    self,
    prefixes: list
)
```

Check if a statement is valid given its prefix

Class `QueryException`

```
class QueryException(
    *args,
    **kwargs
)
```

Raised when trying to create an invalid or unsupperted query

Ancestors (in MRO)

- builtins.Exception
- builtins.BaseException

Generated by *pdoc* 0.10.0 (https://pdoc3.github.io).