python-dbschema Documentation

Release 0.1

Andi Albrecht

Aug 21, 2017

Contents

1 Installation **3**

2 Usage **5**

3 License **7**

4 Resources **9**

5 Contents **11**
- 5.1 Tutorial 11
- 5.2 API 12
- 5.3 Development 15

6 Indices and tables **17**

Python Module Index **19**

`dbschema` is a Python module to inspect database schema definitions.

CHAPTER 1

Installation

The module can be installed from the Python Package Index for example by running:

```
pip install dbschema
```

The module is compatible with Python 2.7, >=3.2 and PyPy. Depending on the database system you want to access a DB-API2 (**PEP 249**) compliant driver is required.

CHAPTER 2

Usage

The following example shows the basic usage of the `dbschema` module:

```
>>> import dbschema
>>> import psycopg2  # in this example we're using a PostgreSQL database
>>> connection = psycopg2.connect(dbname='demo', user='john', password='doe')
>>> schema = dbschema.open('postgresql', connection)
>>> tables = schema.get_tables()
>>> for table in schema.get_tables():
...     print(table.name)
...
artist
track
artist_track_rel
[...]
>>>
```

Continue with the tutorial to learn more about the database object.

CHAPTER 3

License

dbschema is licensed under the BSD license.

CHAPTER 4

Resources

Documentation http://python-dbschema.readthedocs.org/en/latest/

Bug tracker https://bitbucket.org/andialbrecht/python-dbschema/issues

Source code https://bitbucket.org/andialbrecht/python-dbschema

CHAPTER 5

Contents

Tutorial

Connection to a Database

TODO

Examining the Database Schema

TODO

Modifying the Database Schema

Example:

```
>>> import dbschema
>>> op = dbschema.operation.SchemaOperation(dbschema.BACKEND_POSTGRESQL)
>>> table = dbschema.objects.Table('foo')
>>> op += table
>>> table += dbschema.objects.Column('val1', int)
>>> table += dbschema.objects.Column('val2', str, comment='val2 is a string value')
>>> table.comment = 'An example table.'
>>> print(op.as_sql())
CREATE TABLE "foo" (
  val1 integer,
  val2 text
);
COMMENT ON TABLE "foo" IS 'An example table';
COMMENT ON COLUMN "foo"."val2" IS 'val2 is a string value';
>>> conn = psycopg2.connect(dbname='demo', user='john', password='doe')
>>> op.backend.set_connection(conn)
```

```
>>> op.execute()  # perform operation on database
>>>
```

API

The main entry point for dbschema is the *open()* function:

dbschema.**open**(*backend*, *connection=None*, *log_sql=False*, ***connect_kwargs*)
Returns a Schema instance.

Parameters

- **backend** – Backend identifier, must be one of dbschema.backends.BACKEND_NAMES.
- **connection** (*DB-API2 connection or None*) – If not None, a already opened database connection.
- **log_sql** – If True SQL statements are logged (default: False).
- **connect_kwargs** – Connection kwargs if not connection is given.

It is an error if both connection and connect_kwargs are given.

Returns A Schema instance.

Raises *dbschema.exceptions.DBSchemaError* if things went wrong.

Module Constants

dbschema.**BACKEND_MYSQL = 'mysql'**
Identifier for MySQL databases.

dbschema.**BACKEND_POSTGRESQL = 'postgresql'**
Identifier for PostgreSQL databases.

dbschema.**BACKEND_SQLITE3 = 'sqlite3'**
Identifier for SQLite3 databases.

Exceptions

The following exceptions are defined in *dbschema.exceptions*:

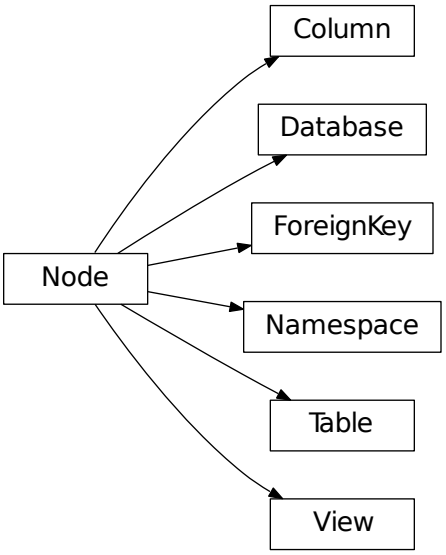
exception dbschema.exceptions.**DBSchemaError**
Base class for all exceptions in this module.

exception dbschema.exceptions.**DBSchemaOperationError**
Used when a database operation fails.

Variables **orignal_exc** – Contains the original exception raised by the underlying DB-API2 module.

Database Objects

The following graph illustrates the class inheritance in this module:



The following representations of database objects are defined in *dbschema.objects*:

class dbschema.objects.**Node**(*name*, ***kwargs*)
Base class for database objects.

name = None
The name of the object.

description = None
The description of the object.

oid = None
A unique identifier provided by the underlying backend.

add_child(*obj*)
Adds a child objects.

find(*type_cls=None*, *name=None*, *parent=None*, *recurse=True*, ***kwargs*)
Yields database objects matching search parameters.

All search parameters are optional. If not parameters are given all database objects are returned.

Parameters

- **type_cls** (Subclass of *Node*) – Find specific types.
- **name** (*str*) – The name to match.
- **parent** (Instance of *Node*) – The parent of the objects that should be yieled.
- **kwargs** – All other keyword parameters are used to compare with the attributes of the child instances.

- **recurse** – If `True` (the default) recurse into children.

Returns Iterator of *Node* instances.

find_exact(***kwargs*)
Returns exact one match or None.

For parameter reference see *find()*.

get_child_types()
Returns a set of child types for this node.

class dbschema.objects.**Database**(*name*)
Central database class.

set_connection(*connection*, ***connect_kwargs*)
Sets the connection to interact with the database.

Parameters

- **connection** – If not `None`, a already opened database connection. :type connection: DB-API2 connection or None
- **connect_kwargs** – Connection kwargs if not connection is given.

It is an error if both `connection` and `connect_kwargs` are given.

get_server_info()
Returns version information of the database.

Return type str

get_default_namespace()
Returns the default namespace or None.

Return type *Namespace* or `None`

get_tables()
Yields all tables from default namespace.

Return type Generator of *Table* instances.

get_views()
Yields all views from the default namespace.

Return type Generator of *View* instances-

set_dirty(*type_cls*, *dirty*)
Marks/unmarks a certain type as dirty.

Parameters

- **type_cls** – A *Node* subclass.
- **dirty** (*bool*) – Wether the type is dirty.

refresh_types(*type_clss*)
Instructs the backend to refresh certain types.

"type_clss" is a list of *dbschema.objects.Node* classes that should be refreshed.

class dbschema.objects.**Namespace**(*name*, ***kwargs*)
A namespace/schema in the database.

is_default_namespace()
Returns True if this namespace is a default namespace.

class dbschema.objects.**Table**(*name*, ***kwargs*)
A table in the database.

get_columns()
Yields columns of this table.

Return type Generator of *Column* instances.

get_foreign_keys()
Yields foreign key definitions.

Return type Generator of *ForeignKey* instances.

get_reverse_foreign_keys()
Yields foreign keys pointing to this table.

Return type Generaotr of *ForeignKey* instances.

class dbschema.objects.**View**(*name*, ***kwargs*)
A view in the database.

get_columns()
Yields columns of this view.

class dbschema.objects.**Column**(*name*, ***kwargs*)
A column of a view or table.

class dbschema.objects.**ForeignKey**(**args*, ***kwargs*)
A foreign key definition of a table.

Development

Testing

dbschema uses py.test for testing. Since dbschema works with various database backends a little setup is required to run the full test suite. Without any additional setup as explained below only tests for SQLite3 databases will be executed.

For testing the sqlparse Python module is required.

Copy tests/databases.conf.example to tests/database.conf and edit as needed. This configuration file needs to be adapt to the database systems you've running on your system and you need to set the parameters to access those databases accordingly. Note that the database user needs permissions to create and delete databases. Here's an example with PostgreSQL and MySQL configured, and where all created databases created during the tests are prefixed with dbschema_ to avoid conflicts.

```
[DEFAULT]
db_prefix = dbschema_

[postgresql]
user = john
password = foo

[mysql]
user = john
password = foo
host = localhost
```

If the file `tests/database.conf` doesn't exist or is empty, only the tests for SQLite databases will be run.

When ready, run **`py.test tests/`** to run the tests using the default Python interpreter.

Setting Up Databases for Testing

As noted above each DBMS you want to test requires an entry in `test/databases.conf`. This section contains some notes on how to set up various DBMS for testing on an Debian/Ubuntu system.

> **Warning:** Security
>
> The following setups assume that you run distinct database instances for testing this module. Therefore all databases users created in this examples are superusers. Please don't use a production database for testing!

MySQL

First install the MySQL server and Python module with **`sudo apt-get install mysql-server libmysqlclient-dev python-mysqldb`**. During the installation you'll be prompted for a password, note it down (or to be more secure, just remember it).

Next create a test user account. The test user needs privileges to create and drop a database. If you don't care much about security for this database installation just create a superuser with the following commands:

```
$ mysql -u root -p
mysql> CREATE USER 'test'@'localhost' IDENTIFIED BY 'secret';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'test'@'localhost';
```

Then in your `tests/databases.conf` add

```
[mysql]
user = test
password = secret
host = localhost
```

Changelog

Development Version

- Initial release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

dbschema.exceptions, 12
dbschema.objects, 13

Index

A

add_child() (dbschema.objects.Node method), 13

B

BACKEND_MYSQL (in module dbschema), 12
BACKEND_POSTGRESQL (in module dbschema), 12
BACKEND_SQLITE3 (in module dbschema), 12

C

Column (class in dbschema.objects), 15

D

Database (class in dbschema.objects), 14
dbschema.exceptions (module), 12
dbschema.objects (module), 13
DBSchemaError, 12
DBSchemaOperationError, 12
description (dbschema.objects.Node attribute), 13

F

find() (dbschema.objects.Node method), 13
find_exact() (dbschema.objects.Node method), 14
ForeignKey (class in dbschema.objects), 15

G

get_child_types() (dbschema.objects.Node method), 14
get_columns() (dbschema.objects.Table method), 15
get_columns() (dbschema.objects.View method), 15
get_default_namespace() (dbschema.objects.Database method), 14
get_foreign_keys() (dbschema.objects.Table method), 15
get_reverse_foreign_keys() (dbschema.objects.Table method), 15
get_server_info() (dbschema.objects.Database method), 14
get_tables() (dbschema.objects.Database method), 14
get_views() (dbschema.objects.Database method), 14

I

is_default_namespace() (dbschema.objects.Namespace method), 14

N

name (dbschema.objects.Node attribute), 13
Namespace (class in dbschema.objects), 14
Node (class in dbschema.objects), 13

O

oid (dbschema.objects.Node attribute), 13
open() (in module dbschema), 12

P

Python Enhancement Proposals
    PEP 249, 3

R

refresh_types() (dbschema.objects.Database method), 14

S

set_connection() (dbschema.objects.Database method), 14
set_dirty() (dbschema.objects.Database method), 14

T

Table (class in dbschema.objects), 14

V

View (class in dbschema.objects), 15