# Etcd Database Driver Documentation

***Release 1.7.0***

**Box TechOps Database Team**

**Dec 06, 2018**

# Contents

Contents:

CHAPTER 1

# Etcd Database Driver

PEP 249 compatible driver for Etcd

## 1.1 Features

- etcdb python module to query etcd as an SQL database
- Implements basic SQL: DROP/CREATE DATABASE, CREATE TABLE, SELECT, INSERT
- etcdb command line client

## 1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

# Installation

## 2.1 Stable release

To install Etcd Database Driver, run this command in your terminal:

```
$ pip install etcdb
```

This is the preferred method to install Etcd Database Driver, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for Etcd Database Driver can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/box/etcdb
```

Or download the tarball:

```
$ curl  -OL https://github.com/box/etcdb/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

# Usage

To use Etcd Database Driver in a project:

```
import etcdb
```

CHAPTER 4

# etcdb

## 4.1 etcdb package

### 4.1.1 Subpackages

#### etcdb.execute package

#### Subpackages

#### etcdb.execute.ddl package

#### Submodules

#### etcdb.execute.ddl.create module

Implement CREATE queries.

etcdb.execute.ddl.create.**create_database**(*etcd_client*, *tree*)
Create database.

**Parameters**

- **etcd_client** (*Client*) – Etcd client
- **tree** (SQLTree) – Parsing tree

etcdb.execute.ddl.create.**create_table**(*etcd_client*, *tree*, *db=None*)
Create table.

**Parameters**

- **etcd_client** (*Client*) – Etcd client
- **tree** (SQLTree) – Parsing tree

- **db** (*str*) – Database name to use if not defined in the parsing tree.

**Raises**

- ***ProgrammingError*** – If primary key is not defined, or the primary key is NULL-able.
- ***OperationalError*** – if database is not selected or table exists.

### etcdb.execute.ddl.drop module

Implement DROP queries.

etcdb.execute.ddl.drop.**drop_database**(*etcd_client*, *tree*)
Drop database.

**Parameters**

- **etcd_client** (*Client*) – Etcd client
- **tree** (SQLTree) – Parsing tree

**Raises** ***OperationalError*** – if database doesn't exist

etcdb.execute.ddl.drop.**drop_table**(*etcd_client*, *tree*, *db=None*)
Drop table.

**Parameters**

- **etcd_client** (*Client*) – Etcd client
- **tree** (SQLTree) – Parsing tree
- **db** (*str*) – Database name to use if not defined in the parsing tree.

**Raises** ***OperationalError*** – if database is not selected or if table doesn't exist.

### Module contents

Module with Data Defition Language queries.

etcdb.execute.ddl.**database_exists_or_raise**(*etcd_client*, *db*)
If database db doesn't exit raise OperationalError.

**Parameters**

- **etcd_client** (*Client*) – Etcd client.
- **db** (*str*) – Database name.

**Raises** ***OperationalError*** – if database doesn't exist

## etcdb.execute.dml package

### Submodules

### etcdb.execute.dml.delete module

Implement DELETE query.

etcdb.execute.dml.delete.**execute_delete**(*etcd_client*, *tree*, *db*)
Execute DELETE query

### etcdb.execute.dml.insert module

Implement INSERT query.

etcdb.execute.dml.insert.**get_pk_field**(*etcd_client*, *db*, *tbl*)
Get primary key column for table db.tbl.

**Parameters**

- **etcd_client** – Etcd client.
- **db** – database name.
- **tbl** – table name.

**Returns** Primary key column.

**Return type** *Column*

etcdb.execute.dml.insert.**get_table_columns**(*etcd_client*, *db*, *tbl*)
Get primary key column for table db.tbl.

**Parameters**

- **etcd_client** – Etcd client.
- **db** – database name.
- **tbl** – table name.

**Returns** Primary key column.

**Return type** *ColumnSet*

**Raises** ***ProgrammingError*** – if table or database doesn't exist

etcdb.execute.dml.insert.**insert**(*etcd_client*, *tree*, *db*)
Execute INSERT query

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client
- **tree** (SQLTree) – Parse tree
- **db** (*str*) – Current database

**Raises** ***IntegrityError*** – if duplicate primary key

### etcdb.execute.dml.select module

Implement SELECT query.

etcdb.execute.dml.select.**eval_row**(*table_columns*, *table_row*, *tree*)
Find values of a row. table_columns are fields in the table. The result columns is taken from tree.expressions.

**Parameters**

- **table_columns** (ColumnSet) – Columns in the table row.
- **table_row** (Row) – Input row.

- **tree** (SQLTree) – Parsing tree.

etcdb.execute.dml.select.**execute_select**(*etcd_client*, *tree*, *db*)
Execute SELECT query.

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client.
- **db** (*str*) – Current database.
- **tree** (SQLTree) – Parse tree.

**Returns** ResultSet instance.

**Return type** *ResultSet*

etcdb.execute.dml.select.**execute_select_no_table**(*tree*)
Execute SELECT that doesn't read from a table. SELECT VERSION() or similar.

etcdb.execute.dml.select.**execute_select_plain**(*etcd_client*, *tree*, *db*)
Execute SELECT that reads rows from table.

etcdb.execute.dml.select.**fix_tree_star**(*tree*, *etcd_client*, *db*, *tbl*)
If parsing tree contains [["*", null], null] expression it means the query was SELECT * . So, the expressions needs to be replaced with actual field names.

etcdb.execute.dml.select.**get_row_by_primary_key**(*etcd_client*, *db*, *table*, *primary_key*, ***kwargs*)
Read row from etcd by its primary key value.

**Parameters**

- **etcd_client** (*Client*) –
- **db** –
- **table** –
- **primary_key** – Primary key value.
- **kwargs** – See below.

**Returns** Row

**Return type** *Row*

**Keyword Arguments**

- **wait** (bool) - If True it will wait for a change in the key.
- **wait_index (int) - When waiting you can specify index to** wait for.

etcdb.execute.dml.select.**group_function**(*table_columns*, *table_row*, *tree*)
True if resultset should be grouped

**Returns** Grouping function or None and its position.

**Return type** tuple(*EtcdbFunction*, int)

etcdb.execute.dml.select.**group_result_set**(*func*, *result_set*, *table_row*, *tree*, *pos*)
Apply a group function to result set and return an aggregated row.

**Parameters**

- **func** (*callable*) – Aggregation function.
- **result_set** (ResultSet) – Result set to aggregate.

- **table_row** (Row) – Table row to base aggregated row on.
- **tree** (SQLTree) – Parsing tree.
- **pos** (*int*) – Aggregate function position in the resulting row.

**Returns** Result set with aggregated row.

**Return type** *ResultSet*

etcdb.execute.dml.select.**list_table**(*etcd_client*, *db*, *tbl*)
Read primary key values in table db.tbl.

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client.
- **db** – database name.
- **tbl** – table name.

**Returns** list of primary keys.

**Return type** list

etcdb.execute.dml.select.**prepare_columns**(*tree*)
Generate ColumnSet for query result. ColumnsSet doesn't include a grouping function.

**Returns** Columns of the query result.

**Return type** *ColumnSet*

### etcdb.execute.dml.show module

Implement SHOW queries.

etcdb.execute.dml.show.**desc_table**(*etcd_client*, *tree*, *db*)
Execute DESC table query#

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client
- **tree** (SQLTree) – Parse tree
- **db** (*str*) – Current database

**Returns** ResultSet instance

**Return type** *ResultSet*

etcdb.execute.dml.show.**show_databases**(*etcd_client*)
Execute SHOW [FULL] TABLES query

**Parameters** **etcd_client** (*pyetcd.client.Client*) – etcd client

**Returns** ResultSet instance

**Return type** *ResultSet*

etcdb.execute.dml.show.**show_tables**(*etcd_client*, *tree*, *db*)
Execute SHOW [FULL] TABLES query#

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client

- **db** (*str*) – Current database
- **tree** (SQLTree) – Parse tree

**Returns** ResultSet instance

**Return type** *ResultSet*

### etcdb.execute.dml.update module

Implement UPDATE query.

etcdb.execute.dml.update.**execute_update**(*etcd_client*, *tree*, *db*)
Execute UPDATE query

### etcdb.execute.dml.use module

Implement USE query.

etcdb.execute.dml.use.**use_database**(*etcd_client*, *tree*)
Return database name if it exists or raise exception.

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd client
- **tree** (SQLTree) – Parsing tree.

**Returns** Database name

**Raises** ***OperationalError*** – if database doesn't exist.

### etcdb.execute.dml.wait module

Implement WAIT query.

etcdb.execute.dml.wait.**execute_wait**(*etcd_client*, *tree*, *db*)
Execute WAIT.

**Parameters**

- **etcd_client** (*Client*) – Etcd client.
- **tree** (SQLTree) – Parsing tree.
- **db** (*str*) – Current database.

### Module contents

Data modification language routines.

etcdb.execute.dml.**get_exclusive_lock**(*etcd_client*, *tree*, *db*)
Acquire a write lock on a table. The lock may be explicitly given from a parsing tree when UPDATE or INSERT specifies it with the USE LOCK statement.

**Parameters**

- **etcd_client** (*pyetcd.client.Client*) – etcd connection

- **tree** (SQLTree) – Parsing tree
- **db** (*str*) – Database name. It doesn't necessary come from the parsing tree. That's why it has to specified.

**Returns** Write lock on a table from the parsing tree in the given database db.

**Return type** *WriteLock*

### Module contents

### etcdb.log package

### Module contents

Logging module

etcdb.log.**setup_logging**(*logger*, *logfile=None*, *debug=False*)
Configure a logger

### etcdb.sqlparser package

### Submodules

### etcdb.sqlparser.etcdb_lexer module

etcdb.sqlparser.etcdb_lexer.**t_STRING**(*t*)
[_a-zA-Z0-9]*[_a-zA-Z]+[_a-zA-Z0-9]*

etcdb.sqlparser.etcdb_lexer.**t_begin_quoted**(*t*)
‘

etcdb.sqlparser.etcdb_lexer.**t_error**(*t*)

etcdb.sqlparser.etcdb_lexer.**t_quoted_STRING_VALUE**(*t*)
[- :.a-zA-Z0-9$/+=_@]+

etcdb.sqlparser.etcdb_lexer.**t_quoted_end**(*t*)
‘

etcdb.sqlparser.etcdb_lexer.**t_quoted_error**(*t*)

### etcdb.sqlparser.parser module

**class** etcdb.sqlparser.parser.**SQLParser**
Bases: object

**parse**(**args*, ***kwargs*)

**exception** etcdb.sqlparser.parser.**SQLParserError**
Bases: exceptions.Exception

All SQL parsing errors

etcdb.sqlparser.parser.**p_AUTO_INCREMENT**(*p*)
opt_column_def_options : AUTO_INCREMENT

etcdb.sqlparser.parser.**p_DEFAULT_CLAUSE**(*p*)
opt_column_def_options : DEFAULT value

etcdb.sqlparser.parser.**p_NOT_NULL**(*p*)
opt_column_def_options : NOT NULL

etcdb.sqlparser.parser.**p_NULL**(*p*)
opt_column_def_options : NULL

etcdb.sqlparser.parser.**p_PRIMARY_KEY**(*p*)
opt_column_def_options : PRIMARY KEY

etcdb.sqlparser.parser.**p_UNIQUE**(*p*)
opt_column_def_options : UNIQUE

etcdb.sqlparser.parser.**p_bit_expr**(*p*)
bit_expr : simple_expr

etcdb.sqlparser.parser.**p_boolean_primary_comparison**(*p*)
boolean_primary : boolean_primary comparison_operator predicate

etcdb.sqlparser.parser.**p_boolean_primary_is_not_null**(*p*)
boolean_primary : boolean_primary IS NOT NULL

etcdb.sqlparser.parser.**p_boolean_primary_is_null**(*p*)
boolean_primary : boolean_primary IS NULL

etcdb.sqlparser.parser.**p_boolean_primary_predicate**(*p*)
boolean_primary : predicate

etcdb.sqlparser.parser.**p_col_expr**(*p*)
col_expr : identifier '=' expr

etcdb.sqlparser.parser.**p_col_expr_list**(*p*)
col_expr_list : col_expr_list ',' col_expr

etcdb.sqlparser.parser.**p_col_expr_list_one**(*p*)
col_expr_list : col_expr

etcdb.sqlparser.parser.**p_column_definition**(*p*)
column_definition : data_type opt_column_def_options_list

etcdb.sqlparser.parser.**p_commit_statement**(*p*)
commit_statement : COMMIT

etcdb.sqlparser.parser.**p_comparison_operator**(*p*)
comparison_operator : '=' | GREATER_OR_EQ | '>' | LESS_OR_EQ | '<' | N_EQ

etcdb.sqlparser.parser.**p_create_database_statement**(*p*)
create_database_statement : CREATE DATABASE identifier

etcdb.sqlparser.parser.**p_create_definition**(*p*)
create_definition : identifier column_definition

etcdb.sqlparser.parser.**p_create_definition_list_many**(*p*)
create_definition_list : create_definition_list ',' create_definition

etcdb.sqlparser.parser.**p_create_definition_list_one**(*p*)
create_definition_list : create_definition

etcdb.sqlparser.parser.**p_create_table_statement**(*p*)
create_table_statement : CREATE TABLE identifier '(' create_definition_list ')'

etcdb.sqlparser.parser.**p_data_type**(*p*)
data_type : INTEGER opt_UNSIGNED | VARCHAR '(' NUMBER ')' | DATETIME | DATETIME '(' NUMBER ')' | INT opt_UNSIGNED | LONGTEXT | SMALLINT opt_UNSIGNED | TINYINT | BOOL

etcdb.sqlparser.parser.**p_delete_statement**(*p*)
delete_statement : DELETE FROM identifier opt_WHERE

etcdb.sqlparser.parser.**p_desc_table_statement**(*p*)
desc_table_statement : DESC identifier

etcdb.sqlparser.parser.**p_drop_database_statement**(*p*)
drop_database_statement : DROP DATABASE identifier

etcdb.sqlparser.parser.**p_drop_table_statement**(*p*)
drop_table_statement : DROP TABLE identifier opt_IF_EXISTS

etcdb.sqlparser.parser.**p_error**(*t*)

etcdb.sqlparser.parser.**p_expr_AND**(*p*)
expr : expr AND expr

etcdb.sqlparser.parser.**p_expr_NOT**(*p*)
expr : NOT expr %prec UNOT

etcdb.sqlparser.parser.**p_expr_OR**(*p*)
expr : expr OR expr

etcdb.sqlparser.parser.**p_expr_bool_primary**(*p*)
expr : boolean_primary

etcdb.sqlparser.parser.**p_fieldlist_many**(*p*)
fieldlist : fieldlist ',' identifier

etcdb.sqlparser.parser.**p_fieldlist_one**(*p*)
fieldlist : identifier

etcdb.sqlparser.parser.**p_function_call_count_star**(*p*)
function_call : COUNT '(' '*' ')'

etcdb.sqlparser.parser.**p_function_call_version**(*p*)
function_call : VERSION '(' ')'

etcdb.sqlparser.parser.**p_identifier**(*p*)
identifier : STRING

etcdb.sqlparser.parser.**p_identifier_escaped**(*p*)
identifier : '`' STRING '`'

etcdb.sqlparser.parser.**p_insert_statement**(*p*)
insert_statement : INSERT INTO identifier opt_fieldlist VALUES '(' values_list ')' opt_USE_LOCK

etcdb.sqlparser.parser.**p_list_expr**(*p*)
list_expr : list_expr ',' expr

etcdb.sqlparser.parser.**p_list_expr_one**(*p*)
list_expr : expr

etcdb.sqlparser.parser.**p_literal**(*p*)
literal : q_STRING | NUMBER | STRING_VALUE

etcdb.sqlparser.parser.**p_opt_FULL**(*p*)
opt_FULL : FULL

etcdb.sqlparser.parser.**p_opt_FULL_empty**(*p*)
opt_FULL :

etcdb.sqlparser.parser.**p_opt_LIMIT**(*p*)
opt_LIMIT : LIMIT NUMBER

etcdb.sqlparser.parser.**p_opt_LIMIT_empty**(*p*)
opt_LIMIT :

etcdb.sqlparser.parser.**p_opt_ORDER_BY_empty**(*p*)
opt_ORDER_BY :

etcdb.sqlparser.parser.**p_opt_ORDER_BY_extended**(*p*)
opt_ORDER_BY : ORDER BY identifier '.' identifier opt_ORDER_DIRECTION

etcdb.sqlparser.parser.**p_opt_ORDER_BY_simple**(*p*)
opt_ORDER_BY : ORDER BY identifier opt_ORDER_DIRECTION

etcdb.sqlparser.parser.**p_opt_ORDER_DIRECTION**(*p*)
opt_ORDER_DIRECTION : ASC | DESC

etcdb.sqlparser.parser.**p_opt_ORDER_DIRECTION_empty**(*p*)
opt_ORDER_DIRECTION :

etcdb.sqlparser.parser.**p_opt_UNSIGNED**(*p*)
opt_UNSIGNED : | UNSIGNED

etcdb.sqlparser.parser.**p_opt_USE_LOCK**(*p*)
opt_USE_LOCK : USE LOCK STRING_VALUE

etcdb.sqlparser.parser.**p_opt_USE_LOCK_empty**(*p*)
opt_USE_LOCK :

etcdb.sqlparser.parser.**p_opt_WHERE**(*p*)
opt_WHERE : WHERE expr

etcdb.sqlparser.parser.**p_opt_WHERE_empty**(*p*)
opt_WHERE :

etcdb.sqlparser.parser.**p_opt_after**(*p*)
opt_AFTER : AFTER NUMBER

etcdb.sqlparser.parser.**p_opt_after_empty**(*p*)
opt_AFTER :

etcdb.sqlparser.parser.**p_opt_column_def_options_list**(*p*)
opt_column_def_options_list : opt_column_def_options opt_column_def_options_list

etcdb.sqlparser.parser.**p_opt_column_def_options_list_empty**(*p*)
opt_column_def_options_list :

etcdb.sqlparser.parser.**p_opt_fieldlist**(*p*)
opt_fieldlist : '(' fieldlist ')'

etcdb.sqlparser.parser.**p_opt_fieldlist_empty**(*p*)
opt_fieldlist :

etcdb.sqlparser.parser.**p_opt_from**(*p*)
opt_FROM : FROM table_reference

etcdb.sqlparser.parser.**p_opt_from_empty**(*p*)
opt_FROM :

etcdb.sqlparser.parser.**p_opt_if_exists**(*p*)
opt_IF_EXISTS : IF EXISTS

etcdb.sqlparser.parser.**p_opt_if_exists_empty**(*p*)
opt_IF_EXISTS :

etcdb.sqlparser.parser.**p_predicate**(*p*)
predicate : bit_expr

etcdb.sqlparser.parser.**p_predicate_in**(*p*)
predicate : bit_expr IN '(' list_expr ')'

etcdb.sqlparser.parser.**p_q_STRING**(*p*)
q_STRING : "" STRING ""

etcdb.sqlparser.parser.**p_q_STRING_EMPTY**(*p*)
q_STRING :

etcdb.sqlparser.parser.**p_select_alias**(*p*)
select_alias : AS identifier

etcdb.sqlparser.parser.**p_select_alias_empty**(*p*)
select_alias :

etcdb.sqlparser.parser.**p_select_item**(*p*)
select_item : select_item2 select_alias

etcdb.sqlparser.parser.**p_select_item2**(*p*)
select_item2 : table_wild | expr

etcdb.sqlparser.parser.**p_select_item_list**(*p*)
select_item_list : select_item_list ',' select_item

etcdb.sqlparser.parser.**p_select_item_list_select_item**(*p*)
select_item_list : select_item

etcdb.sqlparser.parser.**p_select_item_list_star**(*p*)
select_item_list : '*'

etcdb.sqlparser.parser.**p_select_statement**(*p*)
select_statement : SELECT select_item_list opt_FROM opt_WHERE opt_ORDER_BY opt_LIMIT

etcdb.sqlparser.parser.**p_set_names_statement**(*p*)
set_names_statement : SET NAMES STRING

etcdb.sqlparser.parser.**p_set_statement**(*p*)
set_statement : set_autocommit_statement | set_names_statement

etcdb.sqlparser.parser.**p_set_statement_autocommit**(*p*)
set_autocommit_statement : SET AUTOCOMMIT '=' NUMBER

etcdb.sqlparser.parser.**p_show_databases_statement**(*p*)
show_databases_statement : SHOW DATABASES

etcdb.sqlparser.parser.**p_show_tables_statement**(*p*)
show_tables_statement : SHOW opt_FULL TABLES

etcdb.sqlparser.parser.**p_simple_expr_function_call**(*p*)
simple_expr : function_call

etcdb.sqlparser.parser.**p_simple_expr_identifier**(*p*)
simple_expr : identifier

etcdb.sqlparser.parser.**p_simple_expr_identifier_full**(*p*)
simple_expr : identifier '.' identifier

etcdb.sqlparser.parser.**p_simple_expr_literal**(*p*)
simple_expr : literal

etcdb.sqlparser.parser.**p_simple_expr_parent**(*p*)
simple_expr : '(' expr ')'

etcdb.sqlparser.parser.**p_simple_expr_variable**(*p*)
simple_expr : variable

etcdb.sqlparser.parser.**p_statement**(*p*)
statement : select_statement | show_tables_statement | create_table_statement | create_database_statement | show_databases_statement | use_database_statement | commit_statement | set_statement | insert_statement | delete_statement | drop_database_statement | drop_table_statement | desc_table_statement | update_table_statement | wait_statement

etcdb.sqlparser.parser.**p_table_reference**(*p*)
table_reference : identifier

etcdb.sqlparser.parser.**p_table_reference_w_database**(*p*)
table_reference : identifier '.' identifier

etcdb.sqlparser.parser.**p_table_wild**(*p*)
table_wild : identifier '.' '*'

etcdb.sqlparser.parser.**p_update_table_statement**(*p*)
update_table_statement : UPDATE identifier SET col_expr_list opt_WHERE opt_USE_LOCK

etcdb.sqlparser.parser.**p_use_database_statement**(*p*)
use_database_statement : USE identifier

etcdb.sqlparser.parser.**p_value**(*p*)
value : q_STRING | NUMBER | STRING_VALUE

etcdb.sqlparser.parser.**p_values_list_many**(*p*)
values_list : values_list ',' value

etcdb.sqlparser.parser.**p_values_list_one**(*p*)
values_list : value

etcdb.sqlparser.parser.**p_variable**(*p*)
variable : '@' '@' STRING

etcdb.sqlparser.parser.**p_wait_statement**(*p*)
wait_statement : WAIT select_item_list FROM identifier opt_WHERE opt_AFTER

### etcdb.sqlparser.parsetab module

### etcdb.sqlparser.sql_tree module

**class** etcdb.sqlparser.sql_tree.**SQLTree**
Bases: object

**reset**()

#### Module contents

### 4.1.2 Submodules

### 4.1.3 etcdb.cli module

Command line functions

etcdb.cli.**get_query**(*prompt*)
Get input from a user terminated by a semicolon and return a string.

**Parameters** **prompt** (*str*) – A prompt string.

**Returns** User input without a trailing semicolon.

**Return type** str

etcdb.cli.**printf**(*fmt*, **args*)
Printf implemetnation in Python.

**Parameters**

- **fmt** (*str*) – Format string. See man 3 printf for syntax.
- **args** – Arguments to print.

### 4.1.4 etcdb.connection module

Connection class definition

etcdb.connection.**Connect**(***kwargs*)
Factory function for Connection.

**class** etcdb.connection.**Connection**(*timeout=1*, ***kwargs*)
Bases: object

Etcd connection

**static autocommit**(*autocommit*)
Set autocommit mode. Does nothing for non-transactional etcd

**client**
Return etcd client instance

**static close**()
Close the connection now (rather than whenever . __del__() is called).

**static commit**()
Commit any pending transaction to the database.

**cursor**()
Return a new Cursor Object using the connection.

**db**
Current database.

**static rollback**()
This method is optional since not all databases provide transaction support.

**timeout**
Connection timeout.

### 4.1.5 etcdb.cursor module

**class** etcdb.cursor.**ColInfo**(*name=''*, *width=None*)
Bases: object

**class** etcdb.cursor.**Cursor**(*connection*)
Bases: object

These objects represent a database cursor, which is used to manage the context of a fetch operation. Cursors created from the same connection are not isolated, i.e. , any changes done to the database by a cursor are immediately visible by the other cursors. Cursors created from different connections can or can not be isolated, depending on how the transaction support is implemented (see also the connection's .rollback () and .commit () methods).

**arraysize = 1**
This read/write attribute specifies the number of rows to fetch at a time with .fetchmany(). It defaults to 1 meaning to fetch a single row at a time.

**static close**()
Close the cursor now (rather than whenever __del__ is called).

**connection = None**
Etcd connection object

**description = None**
This read-only attribute is a sequence of 7-item sequences.

Each of these sequences contains information describing one result column:

name type_code display_size internal_size precision scale null_ok

The first two items ( name and type_code ) are mandatory, the other five are optional and are set to None if no meaningful values can be provided.

**execute**(*query*, *args=None*)
Prepare and execute a database operation (query or command).

**Parameters**

- **query** (*str*) – Query text.
- **args** (*tuple*) – Optional query arguments.

**Raises**

- ***ProgrammingError*** – if query can't be parsed.
- ***InternalError*** – If etcd is not ready to serve request

**static executemany**(*operation*, ***kwargs*)
Prepare a database operation (query or command) and then execute it against all parameter sequences or mappings found in the sequence seq_of_parameters .

**fetchall**()
Fetch all (remaining) rows of a query result, returning them as a sequence of sequences (e.g. a list of tuples). Note that the cursor's arraysize attribute can affect the performance of this operation.

**fetchmany**(*n*)
Fetch the next set of rows of a query result, returning a sequence of sequences (e.g. a list of tuples). An empty sequence is returned when no more rows are available.

**fetchone**()
Fetch the next row of a query result set, returning a single sequence, or None when no more data is available.

**static morgify**(*query*, *args*)
Prepare query string that will be sent to parser

**Parameters**

- **query** – Query text
- **args** – Tuple with query arguments

**Returns** Query text

**Return type** str

**n_cols**

**n_rows**

**result_set**

**rowcount**

**static setinputsizes**(*sizes*)
This can be used before a call to .execute*() to predefine memory areas for the operation's parameters.

**static setoutputsize**(*size*)
Set a column buffer size for fetches of large columns (e.g. LONG s, BLOB s, etc.). The column is specified as an index into the result sequence. Not specifying the column will set the default size for all large columns in the cursor.

### 4.1.6 etcdb.etcddate module

**class** etcdb.etcddate.**EtcdDate**(*year*, *month*, *day*)
Bases: object

An object holding a date value

### 4.1.7 etcdb.etcdstring module

**class** etcdb.etcdstring.**EtcdString**(*string*)
Bases: object

An object capable of holding a binary (long) string value.

### 4.1.8 etcdb.etcdtime module

**class** etcdb.etcdtime.**EtcdTime**(*hour*, *minute*, *second*)
Bases: object

An object holding a time value

### 4.1.9 etcdb.etcdtimestamp module

**class** etcdb.etcdtimestamp.**EtcdTimestamp**(*year*, *month*, *day*, *hour*, *minute*, *second*)
Bases: object

An object holding a time stamp value.

**day**

**hour**

**minute**

**month**

**second**

**year**

### 4.1.10 etcdb.eval_expr module

Module provides functions to evaluate an expression given in a WHERE statement. Grammar of expression is described on MySQL website (https://dev.mysql.com/doc/refman/5.7/en/expressions.html).

**class** etcdb.eval_expr.**EtcdbFunction**(**args*, ***kwargs*)

Bases: object

EtcdbFunction represents an SQL function.

**Parameters**

- **function_name** (*callable*) – python function that implements SQL function.
- **group** (*bool*) – True if the functions is aggregate function
- **args** – Arguments to pass to function_name
- **kwargs** – Keyword arguments

**function**

Return function name

**group**

Return whether the function is aggregate

etcdb.eval_expr.**etcdb_count**(*result_set*)

Count rows in result set

**Parameters** **result_set** (ResultSet) – ResultSet instance

**Returns** number of rows in ResultSet

**Return type** int

etcdb.eval_expr.**etcdb_version**()

Get etcdb version

etcdb.eval_expr.**eval_bit_expr**(*row*, *tree*)

Evaluate bit_expr

etcdb.eval_expr.**eval_bool_primary**(*row*, *tree*)

Evaluate bool_primary

etcdb.eval_expr.**eval_expr**(*row*, *tree*)

Evaluate expression

**Returns** Tuple with string representation and value. For example, ('id', 5).

**Return type** tuple

etcdb.eval_expr.**eval_function_call**(*row*, *tree*)

Evaluate function call :return: tuple with field name and EtcdbFunction instance

etcdb.eval_expr.**eval_identifier**(*row*, *identifier*)

Get value of identifier for a given row

**Parameters**

- **row** (*tuple(ColumnSet, Row)*) – row
- **identifier** (*str*) – Identifier

**Returns** value of identifier

etcdb.eval_expr.**eval_predicate**(*row*, *tree*)
Evaluate predicate

etcdb.eval_expr.**eval_simple_expr**(*row*, *tree*)
Evaluate simple_expr

etcdb.eval_expr.**eval_string**(*value*)
Evaluate string token

### 4.1.11 etcdb.exception module

etcdb exceptions

**exception** etcdb.exception.**DataError**
Bases: *etcdb.exception.DatabaseError*

Exception raised for errors that are due to problems with the processed data like division by zero, numeric value out of range, etc.

**exception** etcdb.exception.**DatabaseError**
Bases: *etcdb.exception.Error*

Exception raised for errors that are related to the database.

**exception** etcdb.exception.**Error**
Bases: exceptions.Exception

Exception that is the base class of all other error exceptions.

**exception** etcdb.exception.**IntegrityError**
Bases: *etcdb.exception.DatabaseError*

Exception raised when the relational integrity of the database is affected, e.g. a foreign key check fails.

**exception** etcdb.exception.**InterfaceError**
Bases: *etcdb.exception.Error*

Exception raised for errors that are related to the database interface rather than the database itself.

**exception** etcdb.exception.**InternalError**
Bases: *etcdb.exception.DatabaseError*

Exception raised when the database encounters an internal error, e.g. the cursor is not valid anymore, the transaction is out of sync, etc.

**exception** etcdb.exception.**NotSupportedError**
Bases: *etcdb.exception.DatabaseError*

Exception raised in case a method or database API was used which is not supported by the database, e.g. requesting a .rollback() on a connection that does not support transaction or has transactions turned off.

**exception** etcdb.exception.**OperationalError**
Bases: *etcdb.exception.DatabaseError*

Exception raised for errors that are related to the database's operation and not necessarily under the control of the programmer, e.g. an unexpected disconnect occurs, the data source name is not found, a transaction could not be processed, a memory allocation error occurred during processing, etc.

**exception** etcdb.exception.**ProgrammingError**
Bases: *etcdb.exception.DatabaseError*

Exception raised for programming errors, e.g. table not found or already exists, syntax error in the SQL statement, wrong number of parameters specified, etc.

**exception** etcdb.exception.**Warning**
Bases: exceptions.Exception

Exception raised for important warnings like data truncations while inserting, etc.

### 4.1.12 etcdb.lock module

Locking for etcdb

etcdb implements MyISAM style locking. There is an exclusive writer lock and there are many shared reader locks. A client can acquire a writer lock if there are no readers and no writers. A client can acquire a reader lock if there are no writers.

**class** etcdb.lock.**Lock**(*etcd_client*, *db*, *tbl*, ***kwargs*)
Bases: object

Instantiate Lock instance for a table.

**Parameters**

- **etcd_client** (*Client*) – Etcd client
- **db** – Database name.
- **tbl** – Table name.

**acquire**(*timeout=50*, *ttl=50*, ***kwargs*)
Get a lock

**Parameters**

- **timeout** (*int*) – Timeout to acquire a lock.
- **ttl** (*int*) – Place a lock on this time in seconds. 0 for permanent lock.
- **kwargs** – Keyword arguments.
  - **author (str) - Who requests the lock.** By default, 'etcdb'.
  - **reason (str) - Human readable reason to get the lock.** By default, 'etcdb internal operation'.

**Raises**

- ***InternalError*** – This class shouldn't be used directly and if user doesn't set lock_prefix the method should raise exception.
- ***OperationalError*** – If lock wait timeout expires.

**author**

**Returns** String that identifies who acquired the lock.

**Return type** str

**created_at**

**Returns** When the lock was acquired in Unix timestamp.

**Return type** int

**id**
Lock identifier

**readers**()
Get list of reader locks.

**Returns** List of ReadLock() instances

**Return type** list(*ReadLock*)

**reason**

**Returns** String that explains why lock was acquired.

**Return type** str

**release**()
Release a lock

**writers**()
Get list of writer locks.

**Returns** List of WriteLock() instances

**Return type** list(*WriteLock*)

**class** etcdb.lock.**MetaLock**(*etcd_client*, *db*, *tbl*)
Bases: *etcdb.lock.Lock*

Meta lock is needed to place a read or write lock.

**class** etcdb.lock.**ReadLock**(*etcd_client*, *db*, *tbl*, *lock_id=None*)
Bases: *etcdb.lock.Lock*

Read lock.

**acquire**(*timeout=50*, *ttl=50*, ***kwargs*)
Get a read lock

**class** etcdb.lock.**WriteLock**(*etcd_client*, *db*, *tbl*, *lock_id=None*)
Bases: *etcdb.lock.Lock*

Write lock.

**acquire**(*timeout=50*, *ttl=50*, ***kwargs*)
Get a write lock

### 4.1.13 etcdb.resultset module

Classes that represent query results

**class** etcdb.resultset.**Column**(*colname*, *coltype=None*, *options=None*)
Bases: object

Instantiate a Column

**Parameters**

- **colname** (*str*) – Column name.
- **coltype** (*str*) – Column type

- **options** (ColumnOptions) – Column options

**auto_increment**
True if column is auto_incrementing.

**default**
Column default value.

**name**
Column name

**nullable**
True if column is NULL-able.

**primary**
True if column is primary key.

**print_width**
How many symbols client has to spare to print column value. A column name can be short, but its values may be longer. To align column and its values print_width is number of characters a client should allocate for the column name so it will be as lager as the largest columns length value.

**set_print_width**(*width*)
Sets print_width.

**type**
Column type e.g. INT, VARCHAR, etc.

**unique**
True if column is unique key.

**class** etcdb.resultset.**ColumnOptions**(**options*, ***kwoptions*)
Bases: object

ColumnOptions represents column options like NULL-able or not

**auto_increment = False**

**default = None**

**nullable = None**

**primary = False**

**unique = None**

**class** etcdb.resultset.**ColumnSet**(*columns=None*)
Bases: object

Instantiate a Column set

**Parameters** **columns** (*dict*) – Optional dictionary with column definitions

**add**(*column*)
Add column to ColumnSet

**Parameters** **column** (Column) – Column instance

**Returns** Updated CoulmnSet instance

**Return type** *ColumnSet*

**columns**
Returns list of Columns

**empty**
True if there are no columns in the ColumnSet

**index**(*column*)
Find position of the given column in the set.

**next**()
Return next Column

**primary**
Return primary key column

**class** etcdb.resultset.**ResultSet**(*columns*, *rows=None*)
Bases: object

Represents query result

**Parameters**

- **columns** (ColumnSet) – Column set instance.
- **rows** (*list(Row)*) – List of Rows

**add_row**(*row*)
Add row to result set

**Parameters** **row** (Row) – Row instance

**Returns** Updated result set

**Return type** *ResultSet*

**Raises** ***InternalError*** – if row is not a Row class instance.

**n_cols**
Return number of columns in the result set.

**n_rows**
Return number of rows in the result set.

**next**()
Return next row in the result set.

**rewind**()
Move internal records pointer to the beginning of the result set. After this call .fetchone() will start returning rows again.

**class** etcdb.resultset.**Row**(*row*, *etcd_index=0*, *modified_index=0*)
Bases: object

Row class

**Parameters** **row** (*tuple*) – Row values

**etcd_index**
A row in etcdb is a key. Etcd index corresponds to X-Etcd-Index in etcd response header.

**Returns** Etcd index.

**Return type** int

**modified_index**
modifiedIndex of a key in etcd

**next**()
Return next field in the row.

**row**

**Returns** Return tuple with row values..

**Return type** tuple

### 4.1.14 Module contents

PEP-249 implementation for etcd

etcdb.**Binary**(*string*)
This function constructs an object capable of holding a binary (long) string value.

etcdb.**Date**(*year*, *month*, *day*)
This function constructs an object holding a date value.

**Parameters**

- **year** – Year, e.g. 2016
- **month** – Month, e.g. 9
- **day** – Day, e.g. 21

**Returns** EtcdDate instance

etcdb.**DateFromTicks**(*ticks*)
This function constructs an object holding a time value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details).

**Parameters** **ticks** – Seconds since Epoch

**Returns** EtcdDate

etcdb.**ETCDTABLELOCK**
alias of etcdb.EtcdTableLock

etcdb.**Time**(*hour*, *minute*, *second*)
This function constructs an object holding a time value.

**Parameters**

- **hour** – Hour, e.g. 15
- **minute** – Minute, e.g. 53
- **second** – Second, e.g. 16

**Returns** EtcdTime instance

etcdb.**TimeFromTicks**(*ticks*)
This function constructs an object holding a time value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details).

**Parameters** **ticks** – Seconds since Epoch

**Returns** EtcdTime

etcdb.**Timestamp**(*year*, *month*, *day*, *hour*, *minute*, *second*)
This function constructs an object holding a time stamp value.

**Parameters**

- **year** – See Date() and Time() arguments
- **month** –

- **day** –
- **hour** –
- **minute** –
- **second** –

**Returns** EtcdTimestamp instance

etcdb.**TimestampFromTicks**(*ticks*)

This function constructs an object holding a time stamp value from the given ticks value

(number of seconds since the epoch; see the documentation of the standard Python time module for details).

**param ticks** Seconds since Epoch

**return** EtcTimestamp

etcdb.**apilevel = '1.0'**

supported DB API level.

etcdb.**paramstyle = 'qmark'**

the type of parameter marker formatting. Question mark style, e.g. ... WHERE name=?.

etcdb.**threadsafety = 3**

the level of thread safety. Threads may share the module, connections and cursors.

CHAPTER 5

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/box/etcdb/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 5.1.4 Write Documentation

Etcd Database Driver could always use more documentation, whether as part of the official Etcd Database Driver docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/box/etcdb/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *etcdb* for local development.

1. Fork the *etcdb* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/etcdb.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv etcdb
$ cd etcdb/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make test-all
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and for PyPy. Check https://travis-ci.org/box/etcdb/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ py.test tests/unit/test_parser.py
```

To run functional tests make sure vagrant VMs are started first:

```
$ cd vagrant ; vagrant up
```

Then you can run the functional tests:

```
$ py.test tests/functional
```

CHAPTER 6

# Credits

## 6.1 Development Lead

- Box TechOps Database Team <oss@box.com>

## 6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

# History

## 7.1 0.1.0 (2016-09-21)

- First release on PyPI.

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## e

# Index

## F



## G



## H



## I



## L



## M



## N



## O



## P

## R



## S



## T

## U



## W



## Y