

---

# **VersionAlchemy Documentation**

***Release 0.1.0***

**Ryan Kirkman and Akshay Nanavati**

September 01, 2016



<b>1</b>	<b>versionalchemy</b>	<b>1</b>
1.1	versionalchemy package . . . . .	1
<b>2</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>



---

## versionalchemy

---

### 1.1 versionalchemy package

`versionalchemy.init()`

**Parameters** `Session` – the session factory class

`versionalchemy.is_initialized()`

#### 1.1.1 Subpackages

**versionalchemy.api package**

**Submodules**

**versionalchemy.api.data module**

`versionalchemy.api.data.delete(va_table, session, conds)`

**Parameters**

- **va\_table** – the model class which inherits from `VAModelMixin` and specifies the model of the user table from which we are querying
- **session** – a sqlalchemy session with connections to the database
- **conds** – a list of dictionary of key value pairs where keys are columns in the table and values are values the column should take on. If specified, this query will only return rows where the columns meet all the conditions. The columns specified in this dictionary must be exactly the unique columns that versioning pivots around.

Performs a hard delete on a row, which means the row is deleted from the versionalchemy table as well as the archive table.

`versionalchemy.api.data.get(va_table, session, va_id=None, t1=None, t2=None, fields=None, conds=None, include_deleted=True, page=1, page_size=100)`

**Parameters**

- **va\_table** – the model class which inherits from `VAModelMixin` and specifies the model of the user table from which we are querying
- **session** – a sqlalchemy session with connections to the database

- **va\_id** – if specified, the value of t1 and t2 will be ignored. If specified, this will return all records after the specified va\_id.
- **t1** – lower bound time for this query; if None or unspecified, defaults to the unix epoch. If this is specified and t2 is not, this query will simply return the time slice of data at t1. This must either be a valid sql time string or a datetime.datetime object.
- **t2** – upper bound time for this query; if both t1 and t2 are none or unspecified, this will return the latest data (i.e. time slice of data now). This must either be a valid sql time string or a datetime.datetime object.
- **fields** – a list of strings which corresponds to columns in the table; If None or unspecified, returns all fields in the table.
- **conds** – a list of dictionary of key value pairs where keys are columns in the table and values are values the column should take on. If specified, this query will only return rows where the columns meet all the conditions. The columns specified in this dictionary must be exactly the unique columns that versioning pivots around.
- **include\_deleted** – if True, the response will include deleted changes. Else it will only include changes where deleted = 0 i.e. the data was in the user table.
- **page** – the offset of the result set (1-indexed); i.e. if page\_size is 100 and page is 2, the result set will contain results 100 - 199
- **page\_size** – upper bound on number of results to display. Note the actual returned result set may be smaller than this due to the roll up.

## versionalchemy.models package

**class** versionalchemy.models.VALogMixin

Bases: `object`

A mixin providing the schema for the log table, an append only table which saves old versions of rows. An inheriting model must specify the following columns:

- **user\_id** - a column corresponding to the user that made the specified change
- 1 or more columns which are a subset of columns in the user table. These columns

must have a unique constraint on the user table and also be named the same in both tables

**classmethod** `build_row_dict` (*ut\_row, session, deleted=False, user\_id=None, use\_dirty=True*)

### Parameters

- **ut\_row** – the row from the user table
- **deleted** – whether or not the row is deleted
- **user\_id** – the user that is performing the update on this row
- **use\_dirty** – whether to use the dirty fields from ut\_row or not

**Returns** a dictionary of key value pairs to be inserted into the archive table

**Return type** `dict`

**va\_data** = `Column(None, JSONEncodedDict(), table=None, nullable=False)`

**va\_deleted** = `Column(None, Boolean(), table=None, nullable=False)`

**va\_id** = `Column(None, Integer(), table=None, primary_key=True, nullable=False)`

**va\_updated\_at** = `Column(None, DateTime(), table=None, nullable=False)`

```
va_version = Column(None, Integer(), table=None, nullable=False)
```

```
class versionalchemy.models.VAModelMixin
```

```
Bases: object
```

```
classmethod register (ArchiveTable, engine)
```

#### Parameters

- **ArchiveTable** – the model for the users archive table
- **engine** – the database engine
- **version\_col\_names** – strings which correspond to columns that versioning will pivot around. These columns must have a unique constraint set on them.

```
updated_by (user)
```

```
va_id = Column(None, Integer(), table=None, nullable=False, default=ColumnDefault(0))
```

```
va_ignore_columns = None
```

```
va_version_columns = None
```

```
version (session)
```

Returns the rows current version. This can only be called after a row has been inserted into the table and the session has been flushed. Otherwise this method has undefined behavior.

## 1.1.2 Submodules

### 1.1.3 versionalchemy.exceptions module

```
exception versionalchemy.exceptions.LogTableCreationError
```

```
Bases: exceptions.Exception
```

Thrown if an invariant is violated when registering a table for versioning with versionalchemy.

### 1.1.4 versionalchemy.utils module

```
class versionalchemy.utils.JSONEncodedDict (*args, **kwargs)
```

```
Bases: versionalchemy.utils._JSONEncoded
```

```
json_type
```

alias of *dict*

```
class versionalchemy.utils.JSONEncodedList (*args, **kwargs)
```

```
Bases: versionalchemy.utils._JSONEncoded
```

```
json_type
```

alias of *list*

```
versionalchemy.utils.generate_and_clause (cls, row, cols, use_dirty=True)
```

#### Parameters

- **cls** – the sqlalchemy ORM model
- **row** – a sqlalchemy ORM model object (must be an instance of *cls*)
- **cols** – an iterable of strings corresponding to column names
- **use\_dirty** – if *True* (default) will return the dirty value of the column

**Returns** a `sqlalchemy.and_()` clause which checks for equality of all columns in `cols` to the value they contain in `row`.

For example,

```
generate_and_clause(cls, ['foo', 'bar'], cols) =
```

would return

```
sqlalchemy.and_(cls.foo == row.foo, cls.bar == row.bar)
```

`versionalchemy.utils.generate_where_clause(cls, row, col, use_dirty=True)`

#### Parameters

- **cls** – the sqlalchemy ORM model
- **row** – a sqlalchemy ORM model object (must be an instance of `cls`)
- **col** – the column name
- **use\_dirty** – if `True` (default) will return the dirty value of the column

**Returns** a `sqlalchemy ==` clause

`versionalchemy.utils.get_bind_processor(row, col_name, dialect)`

Returns a `bind_processor` for the given column in the row based on the dialect. If dialect is `None` or there is no `bind_processor`, returns the identity function. The return value of this can be applied to `getattr(row, col_name)` and will return the sql type of that value.

`versionalchemy.utils.get_column_attribute(row, col_name, use_dirty=True, dialect=None)`

#### Parameters

- **row** – the row object
- **col\_name** – the column name
- **use\_dirty** – whether to return the dirty value of the column
- **dialect** – if not `None`, should be a `Dialect`. If specified, this function will process the column attribute into the dialect type before returning it; useful if one is using user defined column types in their mappers.

**Returns** if `use_dirty`, this will return the value of `col_name` on the row before it was changed; else this will return `getattr(row, col_name)`

`versionalchemy.utils.get_column_keys(table)`

Return a generator of names of the python attribute for the table columns.

`versionalchemy.utils.get_column_keys_and_names(table)`

Return a generator of tuples `k, c` such that `k` is the name of the python attribute for the column and `c` is the name of the column in the sql table.

`versionalchemy.utils.get_column_names(table)`

Return a generator of names of the name of the column in the sql table.

`versionalchemy.utils.get_dialect(session)`

`versionalchemy.utils.has_constraint(tbl_name, engine, *col_names)`

#### Parameters

- **tbl\_name** – a string with the name of the table to check
- **engine** – an instance of `sa.engine.Engine` from which to execute the query



- **col\_names** – the name of columns which the unique constraint should contain

**Return type** `bool`

**Returns** True if the given columns are part of a unique constraint on `tbl_name`

`versionalchemy.utils.is_modified(row, ignore=None)`

`versionalchemy.utils.result_to_dict(res)`

**Parameters** `res` – `sqlalchemy.engine.ResultProxy`

**Returns** a list of dicts where each dict represents a row in the query where the key is the column name and the value is the value of that column.



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## V

`versionalchemistry`, [1](#)  
`versionalchemistry.api`, [1](#)  
`versionalchemistry.api.data`, [1](#)  
`versionalchemistry.exceptions`, [3](#)  
`versionalchemistry.models`, [2](#)  
`versionalchemistry.utils`, [3](#)



**B**

build\_row\_dict() (versionalchemy.models.VALogMixin class method), 2

**D**

delete() (in module versionalchemy.api.data), 1

**G**

generate\_and\_clause() (in module versionalchemy.utils), 3

generate\_where\_clause() (in module versionalchemy.utils), 4

get() (in module versionalchemy.api.data), 1

get\_bind\_processor() (in module versionalchemy.utils), 4

get\_column\_attribute() (in module versionalchemy.utils), 4

get\_column\_keys() (in module versionalchemy.utils), 4

get\_column\_keys\_and\_names() (in module versionalchemy.utils), 4

get\_column\_names() (in module versionalchemy.utils), 4

get\_dialect() (in module versionalchemy.utils), 4

**H**

has\_constraint() (in module versionalchemy.utils), 4

**I**

init() (in module versionalchemy), 1

is\_initialized() (in module versionalchemy), 1

is\_modified() (in module versionalchemy.utils), 5

**J**

json\_type (versionalchemy.utils.JSONEncodedDict attribute), 3

json\_type (versionalchemy.utils.JSONEncodedList attribute), 3

JSONEncodedDict (class in versionalchemy.utils), 3

JSONEncodedList (class in versionalchemy.utils), 3

**L**

LogTableCreationError, 3

**R**

register() (versionalchemy.models.VAModelMixin class method), 3

result\_to\_dict() (in module versionalchemy.utils), 5

**U**

updated\_by() (versionalchemy.models.VAModelMixin method), 3

**V**

va\_data (versionalchemy.models.VALogMixin attribute), 2

va\_deleted (versionalchemy.models.VALogMixin attribute), 2

va\_id (versionalchemy.models.VALogMixin attribute), 2

va\_id (versionalchemy.models.VAModelMixin attribute), 3

va\_ignore\_columns (versionalchemy.models.VAModelMixin attribute), 3

va\_updated\_at (versionalchemy.models.VALogMixin attribute), 2

va\_version (versionalchemy.models.VALogMixin attribute), 2

va\_version\_columns (versionalchemy.models.VAModelMixin attribute), 3

VALogMixin (class in versionalchemy.models), 2

VAModelMixin (class in versionalchemy.models), 3

version() (versionalchemy.models.VAModelMixin method), 3

versionalchemy (module), 1

versionalchemy.api (module), 1

versionalchemy.api.data (module), 1

versionalchemy.exceptions (module), 3

versionalchemy.models (module), 2

versionalchemy.utils (module), 3