VersionAlchemy Documentation

Release 0.1.0

Ryan Kirkman and Akshay Nanavati

August 26, 2016

Contents

CHAPTER 1

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.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

V

B

D

G

H

I

J

L

R

U

V