SQLAlchemy Postgresql Audit

Release 0.3.0

Jun 24, 2019

Contents

CHAPTER 1

Description

Enables table change tracking support for tables defined by SQLAlchemy models.

Additionally, provides a flexible mechanism for enriching table change data with additional metadata (such as a request UUID or a username or ID).

After registering the relevant SQLAlchemy event listeners, whenever a table is attached to a metadata object, it's info will be checked for specific keys indicating that the table should be audited. If so, a table similar to the source table will be created, with the same name and types (but no constraints or nullability requirements). Additionally, an operation indicator (I, U, D for insert, update, delete), DB timestamp, and *current_user* will be included as columns.

A function and trigger definition are then also defined to insert a row into the audit table whenever a row is inserted, updated, or deleted. For inserts and updates, the row in the audit table is the NEW row representation. For deletes, the row in the audit table is the OLD row.

While any typical create_all/drop_all command will create/drop the relevant tables, Audit Tables info dictionary also contains the DDL necessary to create and drop the function and trigger, and any migration mechanism in usage would need to take advantage of this DDL how it sees fit.

In order to enrich the change data with relevant metadata (such as an application user id or a webrequest UUID, etc), the procedure can be configured (via the table info) to reference any number of session local variables. These variables will be written in the *audit.** namespace. Helper functions are provided for setting these these session variables, and it is recommended that you integrate these deeply in your sessionmaking logic.

This library has experimental alembic integration that installs the relevant triggers and functions. Downgrade support is limited at this time.

CHAPTER 2

Installation

Install from pypi using pip.

```
$ pip install sqlalchemy-postgresql-audit
```

Additionally, ensure that you have a postgres DBAPI library installed, such as *psycopg2*.

CHAPTER 3

Getting Started

1. Install the package (see *Installation*)
2. Enable the SQLAlchemy and/or Alembic event listeners and components. It is critical that this happens *prior* to adding any tables you want to audit to the metadata object.

```
from sqlalchemy import MetaData
import sqlalchemy_postgresql_audit

sqlalchemy_postgresql_audit.enable()

metadata = MetaData()
```

3. Add an *audit.options* key to the table *info* attribute.

ORM Style

```
class Base:

    @declared_attr
    def __table_args__(self):
        return {
            'info': {
                'audit.options': {
                    'enabled': True,
                },
            }
        }
```

Core Style

```
table = Table(
    "bar",
    metadata,
    Column("foo", String),
    info={
```

(continues on next page)

(continued from previous page)

```
        "audit.options": {
            "enabled": True
        }
    },
)
```

4. Install the triggers and functions from the metadata

```
from sqlalchemy_postgresql_audit install_audit_triggers

install_audit_triggers(metadata, engine)
```

CHAPTER 4

Example

This example represents a full workable implementation.

First, we set up naming conventions and enable the audit features

```
from sqlalchemy import Column, Integer, MetaData, String, Table, create_engine
from sqlalchemy.dialects.postgresql import UUID
from sqlalchemy.engine.url import URL
from sqlalchemy.util import immutabledict
import sqlalchemy_postgresql_audit


NAMING_CONVENTIONS = immutabledict(
    {
        "ix": "ix_%(column_0_label)s",
        "uq": "uq_%(table_name)s_%(column_0_name)s",
        "ck": "ck_%(table_name)s_%(constraint_name)s",
        "fk": "fk_%(table_name)s_%(column_0_name)s_%(referred_table_name)s",
        "pk": "pk_%(table_name)s",
        "audit.table": "%(table_name)s_audr",
    }
)

# Event listeners must be enabled before tables are added to the Metadata Object
sqlalchemy_postgresql_audit.enable()

meta = MetaData(naming_convention=NAMING_CONVENTIONS)
```

Next we add a simple table to the metadata, and enable auditing on this table by setting *enabled* to *True* in the *audit.options* dictionary of the *info* attribute.

```
bar = Table(
    "bar",
    meta,
    Column("foo", String),
```

(continues on next page)

(continued from previous page)

```
    info={
        "audit.options": {
            "enabled": True
        }
    },
)
```

We add another table, and this one we want to also capture some data we will set in the local session settings.

Notice that *username* is not nullable, so that *must* be set in the session or any changes to this table will fail.

```
t = Table(
    "foo",
    meta,
    Column("bar", String),
    Column("baz", String),
    info={
        "audit.options": {
            "enabled": True,
            "session_settings": [
                Column("username", String, nullable=False),
                Column("app_uuid", UUID),
            ],
        }
    },
)
```

With all of that set up, we can now create a connection to the database, create the tables, and install the audit triggers!

```
url = URL(
    drivername="postgresql+psycopg2",
    host="localhost",
    port=5432,
    password="postgres",
    username="postgres",
)

engine = create_engine(url)
engine.echo = True
meta.bind = engine

meta.create_all()
sqlalchemy_postgresql_audit.install_audit_triggers(meta)
```

CHAPTER 5

Alembic Integration

Alembic is supported through one of two mechanisms:

1. Create your engine with the *audit* plugin enabled

   'plugins' kwarg

   ```
   from sqlalchemy import create_engine

   engine = create_engine(
       "postgresql+psycopg2://postgres:postgres@localhost:5432/postgres",
       plugins=['audit'],
   )
   ```

   Via the connection String

   ```
   from sqlalchemy import create_engine

   engine = create_engine(
       "postgresql+psycopg2://postgres:postgres@localhost:5432/postgres?
   ↪plugin=audit"
   )
   ```

2. Calling `sqlalchemy_postgresql_audit.enable()`

CHAPTER 6

Naming Conventions

This library ships with the following default naming conventions. These can be overriden in the typical SQLAlchemy manner.

audit.table

`dialect.DEFAULT_AUDIT_TABLE_NAMING_CONVENTION = '%(table_name)s_audit'`

audit.function

`dialect.DEFAULT_AUDIT_TABLE_FUNCTION_NAMING_CONVENTION = '%(schema)s_%(table_name)s_audit'`

audit.trigger

`dialect.DEFAULT_AUDIT_TABLE_TRIGGER_CONVENTION = '%(schema)s_%(table_name)s_audit'`

The *table_name* and the *schema* are the only data elements passed in to the format string.

Additionally, if the *schema* is *None*, then *"public"* is passed in.

CHAPTER 7

Contributing

1. Submit an issue with a description of the feature or bug. Feel free to suggest your solution.
2. Open a pull request with the proposed changes.
3. Be sure to update tests or documentation.
4. Feel free to add your name (in alphabetical order) to *docs/contributing.rst*
5. ???
6. Profit

CHAPTER 8

Roadmap

- **1.0**
  - Improved alembic integration * Reflected function and trigger definition diffs * Proper downgrade support
  - Alternative column-wise audit table format that is long and narrow
- **Future**
  - Other dialects (with sponsering implementers)

CHAPTER 9

Contributors

- Hunter Senft-Grupp

CHAPTER 10

FAQ

Q: Is this library production ready?

A: While this is being used in a number of small projects, it is not yet battle tested software. Use at your own risk.

Q: Should I trust these audit tables for my bank/financial database?

A: Probably not? The author does not have particular experience with the permission and change management requirements of such systems. This library should be great for low to medium throughput in non-adversarial scenarios, where a "best effort" changelog is what we are after, not a fraud-proof audit trail.

CHAPTER 11

API

`sqlalchemy_postgresql_audit.`**`enable`**`()`

Enable the advanced inspector and enables sqlalchemy and alembic event listeners.

`sqlalchemy_postgresql_audit.`**`set_session_vars`**`(`*connectable*, ***kwargs*`)`

Wrapper to set session settings.

This must be set *in* a transaction in order for these settings to be present.

Typical use cases would be a username coming from a web request, or a request UUID or a script name.

Parameters

- **`connectable`** – A connectable that we can execute on.
- **`kwargs`** – key/value pairs of values to set.

Returns None

`sqlalchemy_postgresql_audit.`**`install_audit_triggers`**`(`*metadata*, *engine=None*`)`

Installs all audit triggers.

This can be used after calling *metadata.create_all()* to create all the procedures and triggers.

Parameters

- **`metadata`** – A `sqlalchemy.sql.schema.MetaData`
- **`engine`** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to install all audit triggers.

`sqlalchemy_postgresql_audit.`**`uninstall_audit_triggers`**`(`*metadata*, *engine=None*`)`

Uninstalls all audit triggers.

This can be used to remove all audit triggers.

Parameters

- **`metadata`** – A `sqlalchemy.sql.schema.MetaData`
- **`engine`** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to uninstall all audit triggers.

CHAPTER 12

Modules

12.1 sqlalchemy_postgresql_audit

`sqlalchemy_postgresql_audit.`**`set_session_vars`**`(`*connectable*, ***kwargs*`)`

Wrapper to set session settings.

This must be set *in* a transaction in order for these settings to be present.

Typical use cases would be a username coming from a web request, or a request UUID or a script name.

Parameters

- **`connectable`** – A connectable that we can execute on.
- **`kwargs`** – key/value pairs of values to set.

Returns None

`sqlalchemy_postgresql_audit.`**`enable`**`()`

Enable the advanced inspector and enables sqlalchemy and alembic event listeners.

`sqlalchemy_postgresql_audit.`**`install_audit_triggers`**`(`*metadata*, *engine=None*`)`

Installs all audit triggers.

This can be used after calling *metadata.create_all()* to create all the procedures and triggers.

Parameters

- **`metadata`** – A `sqlalchemy.sql.schema.MetaData`
- **`engine`** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to install all audit triggers.

`sqlalchemy_postgresql_audit.`**`uninstall_audit_triggers`**`(`*metadata*, *engine=None*`)`

Uninstalls all audit triggers.

This can be used to remove all audit triggers.

Parameters

- **metadata** – A `sqlalchemy.sql.schema.MetaData`
- **engine** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to uninstall all audit triggers.

12.2 sqlalchemy_postgresql_audit.ddl

`sqlalchemy_postgresql_audit.ddl.`**`get_audit_spec`**(*table*)

`sqlalchemy_postgresql_audit.ddl.`**`get_create_trigger_ddl`**(*target_columns*, *audit_columns*, *function_name*, *trigger_name*, *table_full_name*, *audit_table_full_name*, *session_settings=None*)

`sqlalchemy_postgresql_audit.ddl.`**`get_drop_trigger_ddl`**(*function_name*, *trigger_name*, *table_full_name*)

`sqlalchemy_postgresql_audit.ddl.`**`install_audit_triggers`**(*metadata*, *engine=None*)
Installs all audit triggers.

This can be used after calling *metadata.create_all()* to create all the procedures and triggers.

Parameters

- **metadata** – A `sqlalchemy.sql.schema.MetaData`
- **engine** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to install all audit triggers.

`sqlalchemy_postgresql_audit.ddl.`**`uninstall_audit_triggers`**(*metadata*, *engine=None*)
Uninstalls all audit triggers.

This can be used to remove all audit triggers.

Parameters

- **metadata** – A `sqlalchemy.sql.schema.MetaData`
- **engine** – A `sqlalchemy.engine.Engine` or None

Returns None or a `str` for the DDL needed to uninstall all audit triggers.

12.3 sqlalchemy_postgresql_audit.dialect

`sqlalchemy_postgresql_audit.dialect.`**`DEFAULT_AUDIT_TABLE_FUNCTION_NAMING_CONVENTION = '%(sc`**
The audit table naming convetion. Change this at naming_conventions *audit.table* key.

`sqlalchemy_postgresql_audit.dialect.`**`DEFAULT_AUDIT_TABLE_NAMING_CONVENTION = '%(table_name)s`**
The audit table naming convetion. Change this at naming_conventions *audit.table* key.

`sqlalchemy_postgresql_audit.dialect.`**`DEFAULT_AUDIT_TABLE_TRIGGER_CONVENTION = '%(schema)s_%`**
The audit table naming convetion. Change this at naming_conventions *audit.table* key.

`class` `sqlalchemy_postgresql_audit.dialect.`**`PGAdvancedInspector`**(*conn*)
A subclass of `sqlalchemy.dialects.postgresql.base.PGInspector`.

Enables advanced database reflection.

__module__ = 'sqlalchemy_postgresql_audit.dialect'

reflecttable (*table*, *include_columns*, **args*, ***kwargs*)
Given a Table object, load its internal constructs based on introspection.

This is the underlying method used by most dialects to produce table reflection. Direct usage is like:

```
from sqlalchemy import create_engine, MetaData, Table
from sqlalchemy.engine.reflection import Inspector

engine = create_engine('...')
meta = MetaData()
user_table = Table('user', meta)
insp = Inspector.from_engine(engine)
insp.reflecttable(user_table, None)
```

Parameters

- **table** – a `Table` instance.
- **include_columns** – a list of string column names to include in the reflection process. If `None`, all columns are reflected.

12.4 sqlalchemy_postgresql_audit.install

12.5 sqlalchemy_postgresql_audit.plugin

class sqlalchemy_postgresql_audit.plugin.**AuditPlugin** (*url*, *kwargs*)

__init__ (*url*, *kwargs*)
Construct a new `CreateEnginePlugin`.

The plugin object is instantiated individually for each call to `create_engine()`. A single `Engine` will be passed to the `CreateEnginePlugin.engine_created()` method corresponding to this URL.

Parameters

- **url** – the `URL` object. The plugin should inspect what it needs here as well as remove its custom arguments from the `URL.query` collection. The URL can be modified in-place in any other way as well.
- **kwargs** – The keyword arguments passed to :func‘.create_engine‘. The plugin can read and modify this dictionary in-place, to affect the ultimate arguments used to create the engine. It should remove its custom arguments from the dictionary as well.

__module__ = 'sqlalchemy_postgresql_audit.plugin'

sqlalchemy_postgresql_audit.plugin.**enable** ()
Enable the advanced inspector and enables sqlalchemy and alembic event listeners.

12.6 sqlalchemy_postgresql_audit.session

sqlalchemy_postgresql_audit.session.**set_session_var_stmt** (***kwargs*)
Returns proper sql statements for setting session settings.

Namespaces all settings under *audit.** namespace.

e.g.

```
set_session_var_stmt(foo='bar', baz='foobaz')
# set local "audit.foo" = 'bar'; set local "audit.baz" = 'foobaz';
```

Parameters **kwargs** – key/value pairs of values to set.

Returns a `str`, valid to set the relevant settings.

`sqlalchemy_postgresql_audit.session.`**set_session_vars**(*connectable*, ***kwargs*)

Wrapper to set session settings.

This must be set *in* a transaction in order for these settings to be present.

Typical use cases would be a username coming from a web request, or a request UUID or a script name.

Parameters

- **connectable** – A connectable that we can execute on.
- **kwargs** – key/value pairs of values to set.

Returns None

12.7 sqlalchemy_postgresql_audit.templates

`sqlalchemy_postgresql_audit.templates.`**make_audit_procedure**(*procedure_name*, *trigger_name*, *check_settings*, *audit_table_full_name*, *table_full_name*, *audit_columns*, *deletion_elements*, *updation_elements*, *insertion_elements*)

Return the string

Parameters

- **procedure_name** – The name for the procedure
- **trigger_name** – The name for the trigger
- **check_settings** – A list of settings checks. Used to validate that the settings are non-empty.
- **audit_table_full_name** – The full name (including schema) of the audit table
- **table_full_name** – The full name (including schema) of the audited table
- **audit_columns** – A list of columns to be audited
- **deletion_elements** – A list of column expressions suitable to use in the VALUES block of an insert statement.
- **updation_elements** – A list of column expressions suitable to use in the VALUES block of an insert statement.

- **insertion_elements** – A list of column expressions suitable to use in the VALUES block of an insert statement.

Returns A `str` of the full DDL needed to be executed to create the procedure and trigger.

`sqlalchemy_postgresql_audit.templates.`**make_drop_audit_procedure**(*function_name*, *trigger_name*, *table_full_name*)

12.8 sqlalchemy_postgresql_audit.event_listeners

`sqlalchemy_postgresql_audit.event_listeners.`**_enable_alembic_event_listeners**()

`sqlalchemy_postgresql_audit.event_listeners.`**_enable_sqlalchemy_event_listeners**()

`sqlalchemy_postgresql_audit.event_listeners.`**enable_event_listeners**()

12.9 sqlalchemy_postgresql_audit.event_listeners.sqlalchemy

Defines event listeners for:

- creating table objects
- generating trigger/procedure DDL for audit tables.

`sqlalchemy_postgresql_audit.event_listeners.sqlalchemy.`**create_audit_table**(*target*, *parent*)

Create an audit table and generate procedure/trigger DDL.

Naming conventions can be defined for a few of the named elements:

- audit.table: Controls the name of the table
- audit.function: Controls the name of the function
- audit.trigger: Controls the name of the trigger on the table

This function creates a new companion table to store row versions.

Any `sqlalchemy.sql.schema.Column`s specified in `table.info['session_settings']` will be copied and included in the audit table.

This function will leave a key in the audited table:

```
table.info['audit.is_audited']
```

And a key in the audit table:

```
table.info['audit.is_audit_table']
```

Additionally you can find the relevant create/drop ddl at the followng keys:

```
table.info['audit.create_ddl']
table.info['audit.drop_ddl']
```

Parameters

- **target** – The `sqlalchemy.sql.schema.Table` to make an audit table for
- **parent** – The `sqlalchemy.sql.schema.MetaData` to associate the audit table with.

Returns None

12.10 sqlalchemy_postgresql_audit.event_listeners.alembic

class `sqlalchemy_postgresql_audit.event_listeners.alembic.`**`ReversableExecute`**(*sqltext*, *reverse_ddl*, *execution_options=None*)

`__init__`(*sqltext*, *reverse_ddl*, *execution_options=None*)
Initialize self. See help(type(self)) for accurate signature.

`__module__ = 'sqlalchemy_postgresql_audit.event_listeners.alembic'`

`reverse`()

`sqlalchemy_postgresql_audit.event_listeners.alembic.`**`compare_for_table`**(*autogen_context*, *modify_table_ops*, *schema*, *tname*, *conn_table*, *metadata_table*)

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

e

i

p

s

t

Index

Symbols

A

C

D

E

G

I

M

P

R