Chrononaut Documentation

Release 0.2.3

Reference Genomics, Inc.

Apr 05, 2019

Contents

Chrononaut is a simple package to provide versioning, change tracking, and record locking for applications using Flask-SQLAlchemy. It currently supports Postgres as a database backend.

CHAPTER 1

Getting started

Getting started with Chrononaut is a simple two step process. First, replace your `FlaskSQLAlchemy` database object with a Chrononaut *`VersionedSQLAlchemy`* database connection:

```
from flask_sqlalchemy import SQLAlchemy
from chrononaut import VersionedSQLAlchemy

# A standard, FlaskSQLAlchemy database connection without support
# for automatic version tracking
db = SQLAlchemy(app)

# A Chrononaut database connection with automated versioning
# for any models with a `Versioned` mixin
db = VersionedSQLAlchemy(app)
```

After that, simply add the *`Versioned`* mixin object to your standard Flask-SQLAlchemy models:

```
# A simple User model with versioning to support tracking of, e.g.,
# email and name changes.
class User(db.Model, Versioned):
    __tablename__ = 'appuser'
    __chrononaut_untracked__ = ['login_count']
    __chrononaut_hidden__ = ['password']

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), unique=False)
    email = db.Column(db.String(255), unique=True)
    password = db.Column(db.Text())
    ...
    login_count = db.Column(db.Integer())
```

This creates an `appuser_history` table that provides prior record values, along with JSON `change_info` and a `changed` microsecond-level timestamp.

CHAPTER 2

Using model history

Chrononaut automatically generates a history table for each model into which you mixin *Versioned*. This history table facilitates:

```
# See if the user has changed their email
# since they first signed up
user = User.query.first()
original_user_info = user.versions()[0]
if user.email == original_user_info.email:
    print('User email matches!')
else:
    print('The user has updated their email!')
```

Trying to access fields that are untracked or hidden raises an exception:

```
print(original_user_info.password)      # Raises a HiddenAttributeError
print(original_user_info.login_count)   # Raises an UntrackedAttributeError
```

For more information on fetching specific version records see *Versioned.versions()*.

CHAPTER 3

Fine-grained versioning

By default, Chrononaut will automatically version every column in a model.

In the above example, we do not want to retain past user passwords in our history table, so we add `password` to the model's `__chrononaut_hidden__` property. Changes to a user's password will now result in a new model version and creation of a history record, but the automatically generated `appuser_history` table will not have a `password` field and will only note that a hidden column was changed in its `change_info` JSON column.

Similarly, Chrononaut's `__chrononaut_untracked__` property allows us to specify that we do not want to track a field at all. This is useful for changes that are regularly incremented, toggled, or otherwise changed but do not need to be tracked. A good example would be a `starred` property on an object or other UI state that might be persisted to the database between application sessions.

CHAPTER 4

Migrations

Chrononaut automatically generates a SQLAlchemy model (and corresponding table) for each *Versioned* mixin. By default, this table is named `tablename_history` where `tablename` is the name of the table for the model. A custom table name may be specified by using the `__chrononaut_tablename__` property in the model.

In order to use Chrononaut, it's important to keep your `*_history` tables in sync with your main tables. We recommend using Alembic for migrations which should automatically generate the `*_history` tables when you first add the *Versioned* mixins and subsequent updates to your models.

CHAPTER 5

More details

More in-depth information on Chrononaut's API is available below:

5.1 Chrononaut's API

5.1.1 Core library classes

class chrononaut.**Versioned**

A mixin for use with Flask-SQLAlchemy declarative models. To get started, simply add the *Versioned* mixin to one of your models:

```
class User(db.Model, Versioned):
    __tablename__ = 'appuser'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(255))
    ...
```

The above will then automatically track updates to the User model and create an appuser_history table for tracking prior versions of each record. By default, *all* columns are tracked. By default, change information includes a user_id and remote_addr, which are set to automatically populate from Flask-Login's current_user in the _capture_change_info() method. Subclass *Versioned* and override a combination of _capture_change_info(), _fetch_current_user_id(), and _get_custom_change_info(). This change_info is stored in a JSON column in your application's database and has the following rough layout:

```
{
    "user_id": "A unique user ID (string) or None",
    "remote_addr": "The user IP (string) or None",
    "extra": {
        ...  # Optional extra fields
    },
    "hidden_cols_changed": [
```

```
        ...  # A list of any hidden fields changed in the version
    ]
}
```

Note that the latter two keys will not exist if they would otherwise be empty. You may provide a list of column names that you do not want to track using the optional `__chrononaut_untracked__` field or you may provide a list of columns you'd like to "hide" (i.e., track updates to the columns but not their values) using the `__chrononaut_hidden__` field. This can be useful for sensitive values, e.g., passwords, which you do not want to retain indefinitely.

diff(*from_model*, *to=None*, *include_hidden=False*)
Enumerate the changes from a prior history model to a later history model or the current model's state (if `to` is `None`).

Parameters

- **from_model** – A history model to diff from.
- **to** – A history model or `None`.

Returns A dict of column names and `(from, to)` value tuples

has_changed_since(*since*)
Check if there are any changes since a given time.

Parameters **since** – The DateTime from which to find any history records

Returns `True` if there have been any changes. `False` if not.

previous_version()
Fetch the previous version of this model (or None)

Returns A history model, or `None` if no history exists

version_at(*at*)
Fetch the history model at a specific time (or None)

Parameters **at** – The DateTime at which to find the history record.

Returns A history model at the given point in time or the model itself if that is current.

versions(*before=None*, *after=None*, *return_query=False*)
Fetch the history of the given object from its history table.

Parameters

- **before** – Return changes only _before_ the provided `DateTime`.
- **before** – Return changes only _after_ the provided `DateTime`.
- **return_query** – Return a SQLAlchemy query instead of a list of models.

Returns List of history models for the given object (or a query object).

class chrononaut.**VersionedSQLAlchemy**(*app=None*, *use_native_unicode=True*, *session_options=None*, *metadata=None*, *query_class=<class 'flask_sqlalchemy.BaseQuery'>*, *model_class=<class 'flask_sqlalchemy.model.Model'>*)
A subclass of the `SQLAlchemy` used to control a SQLAlchemy integration to a Flask application.

Two usage modes are supported (as in Flask-SQLAlchemy). One is directly binding to a Flask application:

```
app = Flask(__name__)
db = VersionedSQLAlchemy(app)
```

The other is by creating the `db` object and then later initializing it for the application:

```
db = VersionedSQLAlchemy()

# Later/elsewhere
def configure_app():
    app = Flask(__name__)
    db.init_app(app)
    return app
```

At its core, the *VersionedSQLAlchemy* class simply ensures that database `session` objects properly listen to events and create version records for models with the *Versioned* mixin.

class chrononaut.**RecordChanges**
A mixin that records change information in a `change_info` JSON column and a `changed` timezone-aware datetime column. Creates change records in the same format as the *Versioned* mixin, but stores them directly on the model vs. in a separate history table.

5.1.2 Helper functions

chrononaut.**extra_change_info**(**args*, ***kwds*)
A context manager for appending extra `change_info` into Chrononaut history records for *Versioned* models. Supports appending changes to multiple individual objects of the same or varied classes.

Usage:

```
with extra_change_info(change_rationale='User request'):
    user.email = 'new-email@example.com'
    letter.subject = 'Welcome New User!'
    db.session.commit()
```

Note that the `db.session.commit()` change needs to occur within the context manager block for additional fields to get injected into the history table `change_info` JSON within an `extra` info field. Any number of keyword arguments with string values are supported.

The above example yields a `change_info` like the following:

```
{
    "user_id": "admin@example.com",
    "remote_addr": "127.0.0.1",
    "extra": {
        "change_rationale": "User request"
    }
}
```

chrononaut.**append_change_info**(**args*, ***kwds*)
A context manager for appending extra `change` info directly onto a single model instance. Use *extra_change_info()* for tracking multiple objects of the same or different classes.

Usage:

```
with append_change_info(user, change_rationale='User request'):
    user.email = 'new-email@example.com'
    db.session.commit()
```

Note that `db.session.commit()` does *not* need to occur within the context manager block for additional fields to be appended. Changes take the same form as with *extra_change_info()*.

chrononaut.**rationale**(**args*, ***kwds*)

A simplified version of the *extra_change_info()* context manager that accepts only a rationale string and stores it in the extra change info.

Usage:

```
with rationale('Updating per user request, see GH #1732'):
    user.email = 'updated@example.com'
    db.session.commit()
```

This would yield a change_info like the following:

```
{
    "user_id": "admin@example.com",
    "remote_addr": "127.0.0.1",
    "extra": {
        "rationale": "Updating per user request, see GH #1732"
    }
}
```

Index