Temporal Data Structures with SQLAlchemy and Postgres

Joey Leingang
Engineering Manager at Clover Health
https://technology.cloverhealth.com

“People assume that time is a strict progression of cause to effect, but actually from a non-linear, non-subjective viewpoint, it’s more like a big ball of wibbly-wobbly, timey-wimey stuff.”

—The Doctor

Agenda

1. What does temporal mean?
2. Some things we tried
3. Postgres: ranges
4. Postgres: exclusion constraints
5. SQLAlchemy: modeling data
6. SQLAlchemy: initializing temporality
7. SQLAlchemy: recording history
8. Take Aways
9. QA

The Dream

```
new_friend = Friend("Rachel", location="New York")
session.add(new_friend)
session.commit()

# some times passes

friend = session.query(Friend).filter_by(name="Rachel").one()
with friend.clock_tick():
  friend.location = "San Francisco"

session.commit()
friend.location_history[0] == "New York"
friend.location_history[1] == friend.location == "San Francisco"
```

The Log File

14:38:04.142597	opendirectoryd	Client: <private>, UID: 0, EUID: 0, GID: 0, EGID: 0
14:38:04.142693	opendirectoryd	queuing request to connection - '<private>'
14:38:04.163463	opendirectoryd	Client: <private>, UID: 0, EUID: 0, GID: 0, EGID: 0
14:38:04.166501	mds	directQueryOpenReply 29078 1795162144 0
14:38:04.166567	mds	QueryOpen succeeded (0) for store <private>
14:38:04.167284	mds	directQueryFetchResultsReply 29078 1795162144 0
14:38:04.445107	opendirectoryd	queuing request to connection - '<private>'
14:38:04.445285	opendirectoryd	queuing request to connection - '<private>'
14:38:04.447881	opendirectoryd	initiating reconnect of module connection
14:38:04.448095	opendirectoryd	queuing request to connection - '<private>'
14:38:04.448389	opendirectoryd	queuing request to connection - '<private>'
14:38:04.452904	opendirectoryd	cache miss with id type: 1
14:38:04.466975	opendirectoryd	queuing request to connection - '<private>'
14:38:04.467196	opendirectoryd	queuing request to connection - '<private>'
14:38:05.019922	kernel	SmartBattery: finished polling type 4
14:38:05.021698	powerd	Battery time remaining posted with value 0x20000000048017b
14:38:05.037565	coreduetd	CDDBatteryMonitor: received batterycallback, currentPercentage:93.804581!
14:38:07.284621	com.apple.WebKit.WebContent	Current memory footprint: 473 MB
14:38:10.310142	AppleIDAuthAgent	SERVER Doing account check for "mu...gl@??.???". (scheduledAccountCheckDispatcher()/Apple…
14:38:10.312557	AppleIDAuthAgent	Scheduled account check '<private>'
14:38:10.312632	AppleIDAuthAgent	Next time for '<private>': Sun Apr 16 14:38:10 2017 in -0.000002 seconds
14:38:10.313136	AppleIDAuthAgent	AppleIDCheckAccount: '<private>'
14:38:10.313224	AppleIDAuthAgent	checkAccountAndDetermineNextAction: '<private>'

Versioned Tables

Friends Table

id	name	location
1	Dave	Berkeley
2	Rachel	San Francisco
3	Will	Oakland

Friends History Table

id	name	location	timestamp
1	Dave	Tucson	20100913
1	Dave	San Francisco	20130502
2	Rachel	New York	20141008
3	Will	San Francisco	20150922
2	Rachel	San Francisco	20160108
1	Dave	Berkeley	20160819
3	Will	Oakland	20160929

Per Property Tracking

Friends Table

id	name	location	vclock
1	Dave	Berkeley	3
2	Rachel	San Francisco	2
3	Will	Oakland	2

Clock Table

friend	tick	timestamp
1	1	20100913
1	2	20130502
2	1	20141008
3	1	20150922
2	2	20160108
1	3	20160819
3	3	20160929

Location History

friend	location	vclock
1	Tucson	1, 2
1	San Francisco	2, 3
2	New York	1, 2
3	San Francisco	1, 2
2	San Francisco	2, ∞
1	Berkeley	3, ∞
3	Oakland	3, ∞

The Basics

```
CREATE TABLE friend (
    id serial PRIMARY KEY,
    name text,
    location text,
    vclock int
);

CREATE TABLE friend_clock (
    friend_id integer REFERENCES friend (id),
    tick int,
    timestamp timestamp WITH time zone,
    PRIMARY KEY (friend_id, tick)
);
```

Ranges

```
-- Containment
SELECT int4range(10, 20) @> 3;

-- Overlaps
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);

-- Extract the upper bound
SELECT upper(int8range(15, 25));

-- Compute the intersection
SELECT int4range(10, 20) * int4range(15, 25);

-- Is the range empty?
SELECT isempty(numrange(1, 5));
```

Exclusion (the only good kind)

```
CREATE EXTENSION btree_gist;

CREATE TABLE friend_location_history (
    id serial PRIMARY KEY,
    friend_id integer REFERENCES friend (id),
    vclock int4range,
    location text,
    EXCLUDE USING gist (friend_id WITH =, vclock WITH &&)
);
```

Exclusion in Action

```
-- start recording history for Dave
INSERT INTO friend_location_history (friend_id, location, vclock)
VALUES
  (1, 'Tucson', int4range(1, null));
INSERT 0 1

-- Can't live in two places at the same time!
INSERT INTO friend_location_history (friend_id, location, vclock)
VALUES
  (1, 'San Francisco', int4range(2, null));
ERROR:  conflicting key value violates exclusion constraint
"friend_location_history_friend_id_vclock_excl"
DETAIL:  Key (friend_id, vclock)=(1, [2,)) conflicts with existing key (friend_id,
vclock)=(1, [1,)).
```

Skip the SQL, Go Straight to Alchemy

```
import sqlalchemy as sa
import sqlalchemy.ext.declarative as sa_decl
import temporal_sqlalchemy as temporal

Base = sa_decl.declarative_base()

class Friend(Base, temporal_sqlalchemy.TemporalModel):
    __tablename__ = 'friend'

    id = sa.Column(sa.Integer, primary_key=True)

    name = sa.Column(sa.Text)
    location = sa.Column(sa.Text)

    class Temporal:
        track = ('name', 'location', )
        schema = 'history_schema'
```

Under the Hood: TemporalModel

```
class TemporalModel:
    vclock = sa.Column(sa.Integer, default=1)

    @contextlib.contextmanager
    def clock_tick(self, activity: TemporalActivityMixin = None):
        ...

    @staticmethod
    def temporal_map(mapper: orm.Mapper, cls):
        ...

    @staticmethod
    def init_clock(clocked: 'TemporalModel', args, kwargs):
        ...

    @declarative.declared_attr
    def __mapper_cls__(cls):
        ...
```

Declared Attributes

```
@declarative.declared_attr
def __mapper_cls__(cls):
    assert hasattr(cls, 'Temporal')

    def mapper(cls, *args, **kwargs):
        mp = orm.mapper(cls, *args, **kwargs)
        cls.temporal_map(mp, cls)
        return mp

    return mapper
```

Temporal Map

```
@staticmethod
def temporal_map(mapper: orm.Mapper, cls):

  # 1. get things defined on Temporal
  # 2. make sure all temporal properties have active_history (always loaded)
  # 3. used to construct a new clock model for this entity
  # 4. construct history models for each property

  event.listen(cls, 'init', TemporalModel.init_clock)
```

Starting History

```
@staticmethod
def init_clock(clocked, args, kwargs):
   kwargs.setdefault('vclock', 1)
   initial_tick = clocked.temporal_options.clock_model(
      tick=kwargs['vclock'],
      entity=clocked,
   )

   if 'activity' in kwargs:
      initial_tick.activity = kwargs.pop('activity')
```

Updating History

```
@contextlib.contextmanager
def clock_tick(self, activity: TemporalActivityMixin = None):
    """Increments vclock by 1 with changes scoped to the session"""
    if self.temporal_options.activity_cls is not None and activity is None:
        raise ValueError("activity is missing on edit") from None

    session = orm.object_session(self)
    with session.no_autoflush:
        yield self

    if session.is_modified(self):
        self.vclock += 1

    new_clock_tick = self.temporal_options.clock_model(entity=self, tick=self.vclock)
    if activity is not None:
        new_clock_tick.activity = activity

    session.add(new_clock_tick)
```

Decorator Too!

```
def add_clock(*props: typing.Iterable[str],
              activity_cls: nine.Type[TemporalActivityMixin] = None,
              temporal_schema: typing.Optional[str] = None):
    # all of the same stuff as before
```

The Dream

```
new_friend = Friend("Rachel", location="New York")
session.add(new_friend)
session.commit()

# some times passes

friend = session.query(Friend).filter_by(name="Rachel").one()
with friend.clock_tick():
  friend.location = "San Francisco"

session.commit()
friend.location_history[0] == "New York"
friend.location_history[1] == friend.location == "San Francisco"
```

Take Aways

If you need to keep history, trade offs apply.

For a lot of models, and a lot of properties, you get A LOT of tables.

Doing things in bulk is very challenging.

QA

Temporal Sqlalchemy is open source:
https://github.com/CloverHealth/temporal-sqlalchemy
(not published to pypi yet!)

Requires:
Python 3.3+, Sqlalchemy 1.0.15+, Postgres 9.3+,
psycopg2 2.6.2+