Wicked Production SQLAlchemy

Lying

Caching


There is only one thing
I hate more than lying: Skim Milk.
Which is water that's lying about being milk.
HANNIBAL
TONIGHT 10/9c

```
SELECT  titel, 2011-Jahr AS alt, 'Jahre alt' AS Text
FROM    buch
WHERE   jahr > 1997
ORDER BY alt DESC, titel

SELECT  B.buchid, B.titel, V.name, V.ort, B.jahr
FROM    buch B NATURAL JOIN verlag V
WHERE   V.name='Springer' AND B.jahr>=1990
ORDER   BY V.ort

SELECT  DISTINCT A.nachname, A.vornamen, A.autorid
FROM    autor A NATURAL JOIN buch_aut BA
        NATURAL JOIN buch_sw BS NATURAL JOIN schlagwort SW
WHERE   SW.schlagwort = 'Datenbank'
ORDER BY A.nachname

SELECT buchid, titel, jahr
FROM buch
WHERE jahr=(SELECT MIN(jahr) FROM buch)
```


SENORGIF.COM

“Parallel” Queries...

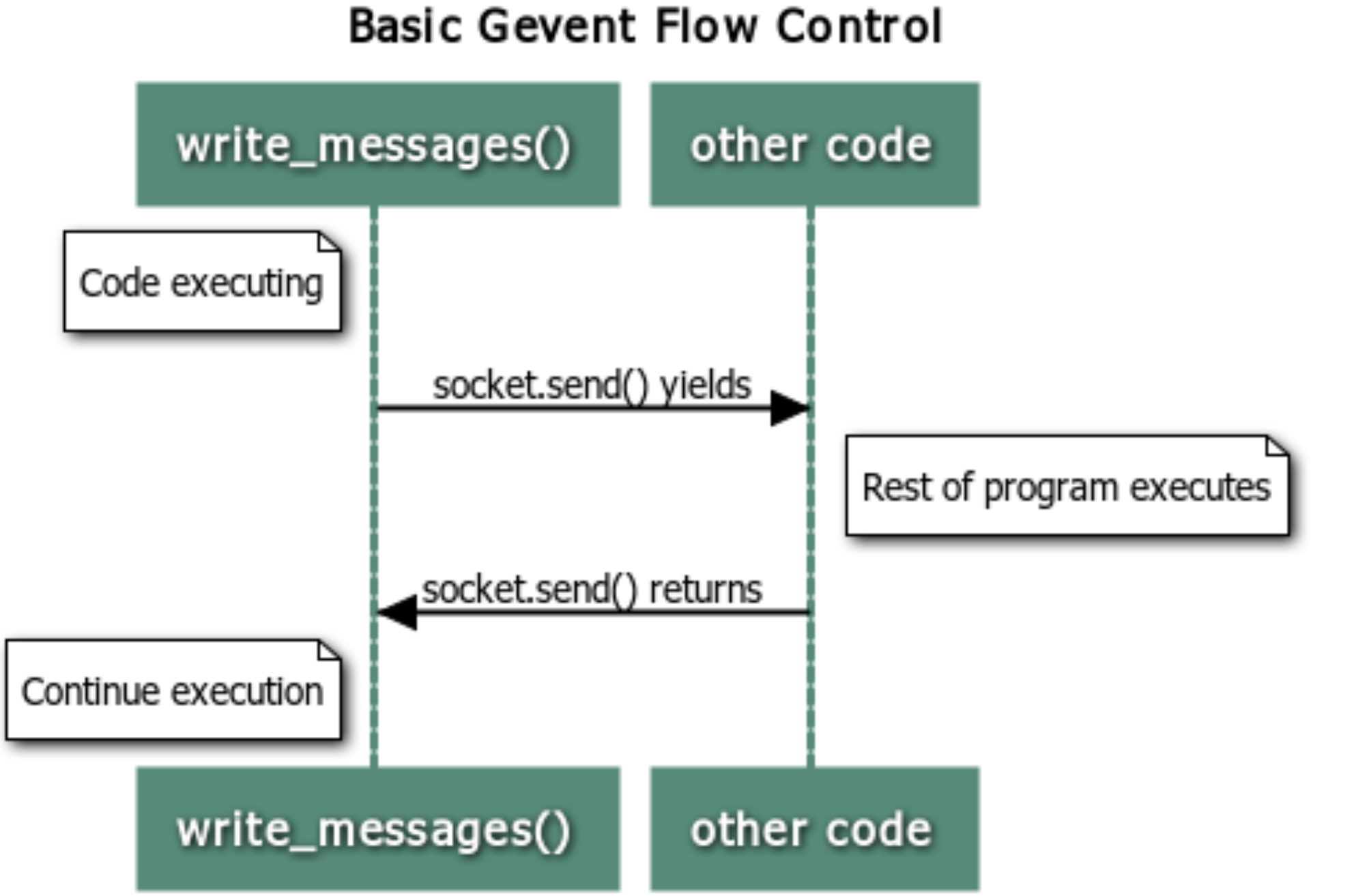
sort of...

not really...

it's lying

gevent

- Fast event loop
- Lightweight execution units
- Monkey patching utility


Basic Gevent Flow Control
write_messages()
other code
Code executing
socket.send() yields
Rest of program executes
socket.send() returns
Continue execution
write_messages()
other code
www.websequencediagrams.com

Coloring PostgreSQL Green

```
def gevent_wait_callback(conn, timeout=None):
    """A wait callback useful to allow gevent to work with Psycopg."""
    while True:
        state = conn.poll()
        if state == extensions.POLL_OK:
            break
        elif state == extensions.POLL_READ:
            wait_read(conn.fileno(), timeout=timeout)
        elif state == extensions.POLL_WRITE:
            wait_write(conn.fileno(), timeout=timeout)
        else:
            raise psycopg2.OperationalError(
                "Bad result from poll: %r" % state)
```

Coloring PostgreSQL Green

```
def make_psycopg_green():
    """Configure Psycopg to be used with gevent in non-blocking way."""
    if not hasattr(extensions, 'set_wait_callback'):
        raise ImportError(
            "support for coroutines not available in this Psycopg version (%s)"
            % psycopg2.__version__)

    extensions.set_wait_callback(gevent_wait_callback)
```

Building a Query Pool (**init**)

```
import gevent
from gevent.queue import JoinableQueue, Queue

class QueryPool(object):
    def __init__(self, queries, pool_size=5):
        self.queries = queries
        self.POOL_MAX = pool_size
        self.tasks = JoinableQueue()
        self.output_queue = Queue()
```

Building a Query Pool (work)

```
def __query(self, query):
    conn = engine.connect()
    results = conn.execute(query).fetchall()
    return results
```

Building a Query Pool (Executor)

```
def executor(self, number):
    while not self.tasks.empty():
        query = self.tasks.get()
        try:
            results = self.__query(query)
            self.output_queue.put(results)
        except Exception as exc_info:
            print exc_info
            print 'Query failed :('
        self.tasks.task_done()
```

Building a Query Pool (Overseer)

```
def overseer(self):
    for query in self.queries:
        self.tasks.put(query)
```

Building a Query Pool (runner)

```
def run(self):
    self.running = []
    gevent.spawn(self.overseer).join()
    for i in range(self.POOL_MAX):
        runner = gevent.spawn(self.executor, i)
        runner.start()
        self.running.append(runner)

    self.tasks.join()
    for runner in self.running:
        runner.kill()
    return [x for x in self.output_queue]
```

Queries

```
query1 = select([pgbench_tellers])
query2 = select([pgbench_accounts.c.bid, func.count(1)]).group_by(pgbench_accounts.c.bid)
query3 = select([pgbench_branches])
query4 = select([pgbench_accounts.c.bid]).distinct()
query5 = select([pgbench_accounts]).limit(1000)
query6 = select([pgbench_accounts.c.bid, func.count(1)]
                ).group_by(pgbench_accounts.c.bid).limit(5000)

queries = [query1, query2, query3, query4, query5, query6]
```

Putting it all together

```
make_psycopg_green()
results = QueryPool(queries).run()
```

Well... is it worth it...

Executing the 6 queries

- 57s: Serially
- 49.7s: Query pool 2 workers
- 31.4s: Query pool 3 workers
- 30s: Query pool 5 workers
- 27.5s: Query pool with 6 workers


amc

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.008	0.174	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.720..11.310 rows=1 width=4955) (actual time=0.160..0.174 rows=1 loops=1) Join Filter: (victor_seven0kilo_oscar.three = charlie_lima8kilo_oscar.quebec) Rows Removed by Join Filter: 7
2.	0.003	0.158	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.720..10.150 rows=1 width=4758) (actual time=0.144..0.158 rows=1 loops=1) Join Filter: (six_seven6kilo_oscar.six_kilo = foxtrot_mike7kilo_oscar.quebec)
3.	0.004	0.153	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.720..9.130 rows=1 width=3717) (actual time=0.139..0.153 rows=1 loops=1)
4.	0.005	0.136	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.590..8.220 rows=1 width=3185) (actual time=0.124..0.136 rows=1 loops=1) Join Filter: (victor_seven0kilo_oscar.four = charlie_lima4kilo_oscar.quebec) Rows Removed by Join Filter: 6
5.	0.004	0.122	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.590..7.060 rows=1 width=2988) (actual time=0.113..0.122 rows=1 loops=1) Join Filter: (charlie_lima1kilo_oscar.oscar_juliet = romeo3kilo_oscar.quebec) Rows Removed by Join Filter: 4
6.	0.004	0.115	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.590..6.000 rows=1 width=1431) (actual time=0.106..0.115 rows=1 loops=1) Join Filter: (charlie_lima1kilo_oscar.six_kilo = foxtrot_mike2kilo_oscar.quebec) Rows Removed by Join Filter: 1
7.	0.007	0.108	↑ 1.0	1	1	→ Nested Loop Left Join (cost=1.590..4.970 rows=1 width=390) (actual time=0.099..0.108 rows=1 loops=1) Join Filter: (victor_seven0kilo_oscar.papa = charlie_lima1kilo_oscar.quebec) Rows Removed by Join Filter: 7
8.	0.026	0.088	↑ 1.0	1	1	→ Hash Join (cost=1.590..3.820 rows=1 width=193) (actual time=0.079..0.088 rows=1 loops=1) Hash Cond: (whiskey5kilo_oscar.quebec = victor_seven0kilo_oscar.alpha)
9.	0.029	0.029	↑ 1.0	16	1	→ Seq Scan on whiskey kilo_two (cost=0.000..2.160 rows=16 width=119) (actual time=0.009..0.029 rows=16 loops=1)
10.	0.005	0.033	↑ 1.0	1	1	→ Hash (cost=1.580..1.580 rows=1 width=74) (actual time=0.033..0.033 rows=1 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 1kB
11.	0.028	0.028	↑ 1.0	1	1	→ Seq Scan on victor_seven echo (cost=0.000..1.580 rows=1 width=74) (actual time=0.028..0.028 rows=1 loops=1) Filter: (quebec = 705) Rows Removed by Filter: 45
12.	0.013	0.013	↑ 1.0	7	1	→ Seq Scan on seven_romeo uniform (cost=0.000..1.070 rows=7 width=197) (actual time=0.007..0.013 rows=7 loops=1)
13.	0.003	0.003	↑ 1.0	1	1	→ Seq Scan on foxtrot_mike foxtrot_two (cost=0.000..1.010 rows=1 width=1041) (actual time=0.003..0.003 rows=1 loops=1)
14.	0.003	0.003	↓ 1.3	4	1	→ Seq Scan on juliet seven_golf (cost=0.000..1.030 rows=3 width=1557) (actual time=0.002..0.003 rows=4 loops=1)
15.	0.009	0.009	↑ 1.0	7	1	→ Seq Scan on seven_romeo charlie_bravo (cost=0.000..1.070 rows=7 width=197) (actual time=0.003..0.009 rows=7 loops=1)
16.	0.013	0.013	↑ 1.0	1	1	→ Index Scan using charlie_echo on six_seven victor_three (cost=0.130..0.900 rows=1 width=532) (actual time=0.012..0.013 rows=1 loops=1) Index Cond: (whiskey5kilo_oscar.six_india = quebec)

Dogpile (regions)

```
regions = {}

regions['default'] = make_region(async_creation_runner=async_creation_runner,
                                 key_mangler=mangle_key).configure(
    'dogpile.cache.redis',
    arguments={
        'host': redis_host,
        'port': redis_port,
        'db': 0,
        'redis_expiration_time': 60*60*2,   # 2 hours
        'distributed_lock': True,
        'lock_timeout': 120,
        'lock_sleep': 5
    }
)
```

Dogpile (Creating cache objects)

```
def async_creation_runner(cache, somekey, creator, mutex):
    def runner():
        try:
            value = creator()
            cache.set(somekey, value)
        finally:
            mutex.release()

    thread = threading.Thread(target=runner)
    thread.start()
```

Dogpile (Building Cache Keys)

```
def unicode_sha1_mangle_key(key):
    return sha1_mangle_key(key.encode('ascii', 'ignore'))


def mangle_key(key):
    prefix, key = key.split(':', 1)
    base = 'cookie:cache:'
    if prefix:
        base += '{}'.format(prefix)
    else:
        raise ValueError(key)
    return '{}:{}'.format(base, unicode_sha1_mangle_key(key))
```


Shut up. I'm rich.

CachingQuery (__init__, __iter__)

```
class CachingQuery(Query):

    def __init__(self, regions, *args, **kw):
        self.cache_regions = regions
        self.saved_to_cache = False
        Query.__init__(self, *args, **kw)

    def __iter__(self):
        if hasattr(self, '_cache_region'):
            return self.get_value(
                createfunc=lambda: list(Query.__iter__(self)))
        else:
            return Query.__iter__(self)
```

CachingQuery (regions)

```
def _get_cache_plus_key(self):
    dogpile_region = self.cache_regions[self._cache_region.region]
    if self._cache_region.cache_key:
        key = self._cache_region.cache_key
    else:
        key = _key_from_query(self)
    return dogpile_region, key
```

CachingQuery (Getter)

```
def get_value(self, merge=True, createfunc=None,
              expiration_time=None, ignore_expiration=False):
    dogpile_region, cache_key = self._get_cache_plus_key()

    assert not ignore_expiration or not createfunc, \
        "Can't ignore expiration and also provide createfunc"

    if ignore_expiration or not createfunc:
        cached_value = dogpile_region.get(
            cache_key,
            expiration_time=expiration_time,
            ignore_expiration=ignore_expiration
        )
```

CachingQuery (Getter - cont)

```
else:
    try:
        cached_value = dogpile_region.get_or_create(
            cache_key,
            createfunc,
            expiration_time=expiration_time
        )
    except ConnectionError:
        logger.error('Cannot connect to query caching backend!')
        cached_value = createfunc()
if cached_value is NO_VALUE:
    raise KeyError(cache_key)
if merge:
    cached_value = self.merge_result(cached_value, load=False)
return cached_value
```

CachingQuery (Setter)

```
def set_value(self, value):
    dogpile_region, cache_key = self._get_cache_plus_key()
    try:
        dogpile_region.set(cache_key, value)
        self.saved_to_cache = True
    except ConnectionError:
        logger.error('Cannot connect to query caching backend!')
```

CachingQuery (Key Generator)

```
def _key_from_query(query, qualifier=None):
    stmt = query.with_labels().statement
    compiled = stmt.compile()
    params = compiled.params

    return " ".join([str(compiled)] +
                    [str(params[k]) for k in sorted(params)])
```

SQLAlchemy Options (FromQuery)

```
class FromCache(MapperOption):
    """Specifies that a Query should load results from a cache."""

    propagate_to_loaders = False

    def __init__(self, region="default", cache_key=None, cache_prefix=None):
        self.region = region
        self.cache_key = cache_key
        self.cache_prefix = cache_prefix

    def process_query(self, query):
        query._cache_region = self
```

Callable

```
def query_callable(regions, query_cls=CachingQuery):
    def query(*arg, **kw):
        return query_cls(regions, *arg, **kw)
    return query
```

Putting it together (Session)

```
def init_caching_session(engine=None):
    if not engine:
        return

    return sessionmaker(
        bind=engine, autoflush=False, autocommit=False,
        query_cls=query_callable(regions)
    )

CachingSession = init_caching_session(engine)
caching_session=CachingSession()
```

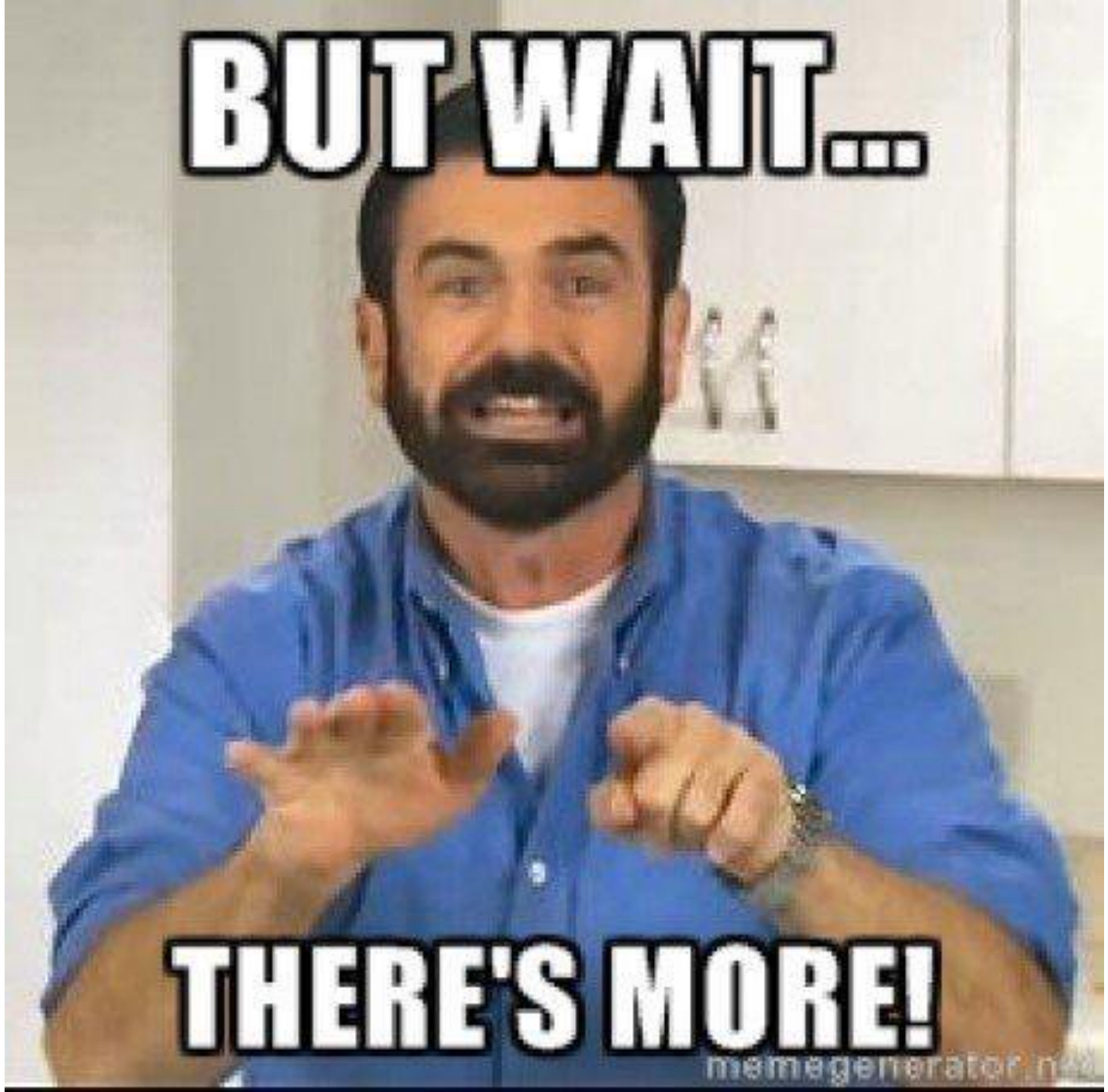
Putting it together (Query)

```
query = caching_session.query(Accounts.bid, func.count(1)
    ).group_by(Accounts.bid).limit(5000).options(
        FromCache('default'))
```

Well... is it worth it...

Executing the query

- 24.9s: Uncached
- 24.9s: Initial run of caching_query
- 4.32 ms: Second run of caching_query


BUT WAIT...
THERE'S MORE!
memegenerator.net

Baked queries

Bulk Operations

Thank You

http://github.com/jasonamyers/wicked-sqlalchemy