Python and

Relational / Non-relational Databases

Andrew Godwin

Introduction

Python for 5 years

Django core developer

Data modelling / visualisation

"Andrew speaks English like a machine gun speaks bullets."

Reinout van Rees

If I speak too fast -
tell me!

What is a relational database?

A relational database is a “collection of relations”

It’s what a lot of people are used to.

Relational Databases

PostgreSQL
MySQL
SQLite

Let’s pick PostgreSQL (it’s a good choice)

Usage

```
conn = psycopg2.connect(
    host="localhost",
    user="postgres"
)
cursor = conn.cursor()
cursor.execute('SELECT * FROM users WHERE
username = "andrew";')
for row in cursor.fetchall():
    print row
```

You've probably seen all that before.

Now, to introduce some non-relational databases

Document Databases

MongoDB

CouchDB

Key-Value Stores

Cassandra

Redis

Message Queues

AMQP

Celery

Various Others

Graph databases
Filesystems
VCSs

Redis and MongoDB are
two good examples here

Redis: Key-value store with strings, lists, sets, channels and atomic operations.

Redis Example

```
conn = redis.Redis(host="localhost")
print conn.get("top_value")
conn.set("last_user", "andrew")
conn.inc("num_runs")
conn.sadd("users", "andrew")
conn.sadd("users", "martin")
for item in conn.smembers("users"):
    print item
```

MongoDB: Document store with indexing and a wide range of query filters.

MongoDB Example

```
conn = pymongo.Connection("localhost")
db = conn['mongo_example']
coll = db['users']
coll.insert({
    "username": "andrew",
    "uid": 1000,
})
for entry in coll.find({"username":
"andrew"}):
    print entry
```

These all solve different problems - you can't easily replace one with the other.

"When all you have is a hammer, everything looks like a nail"

Abraham Manslow (paraphrased)

JOIN - your best friend,
and your worst enemy.

Denormalising your data speeds up reads, and slows down writes.

Schemaless != Denormalised

Atomic operations are nice.

conn.incrby('num_users', 2)

But SQL can do some of them.

```
UPDATE foo SET bar = bar + 1 WHERE baz;
```

Redis, the datastructures server.

SETNX, GETSET, EXPIRES and friends

Locks / Semaphores

```
conn.setnx('lock:foo', time.time() + 3600)
val = conn.decr('sem:foo')
if val >= 0: ... else: conn.incr('sem:foo')
```

Queues

conn.lpush('myqueue', 'workitem')

todo = conn.lpop('myqueue')

(or publish/subscribe)

Priority Queues

```
conn.zadd('myqueue', 'handle-meltdown', 1)
conn.zadd('myqueue', 'feed-cats', 5)
todo = conn.zrange('myqueue', 0, 1)
conn.zrem(todo)
```

Lock-free linked lists!

```
new_id = 'bgrdsd'
old_end = conn.getset(':end', new_id)
conn.set('%s:next' % old_end, new_id)
```

Performance-wise, the less checks/integrity the faster it goes.

Maturity can sometimes be an issue, but new features can appear rapidly.

You can also use databases for the wrong thing - it often only matters "at scale"

But how does this all relate to Python?

Most databases - even new ones - have good Python bindings

Postgres: PsycoPG2
Redis: redis-py
MongoDB: pymongo

(and more - neo4j, VCSen, relational, etc.)

Some databases have
Python available inside
(Postgres has it as an option)

Document databases map really well to Python dicts

You may find non-relational databases a nicer way to store state - for any app

Remember, you might still need transactions/reliability.

(Business logic is probably better off on mature systems for now)

Overall? Just keep all the options in mind.

Don’t get caught by trends, and don’t abuse your relational store

Thanks.

Andrew Godwin
@andrewgodwin
http://aeracode.org