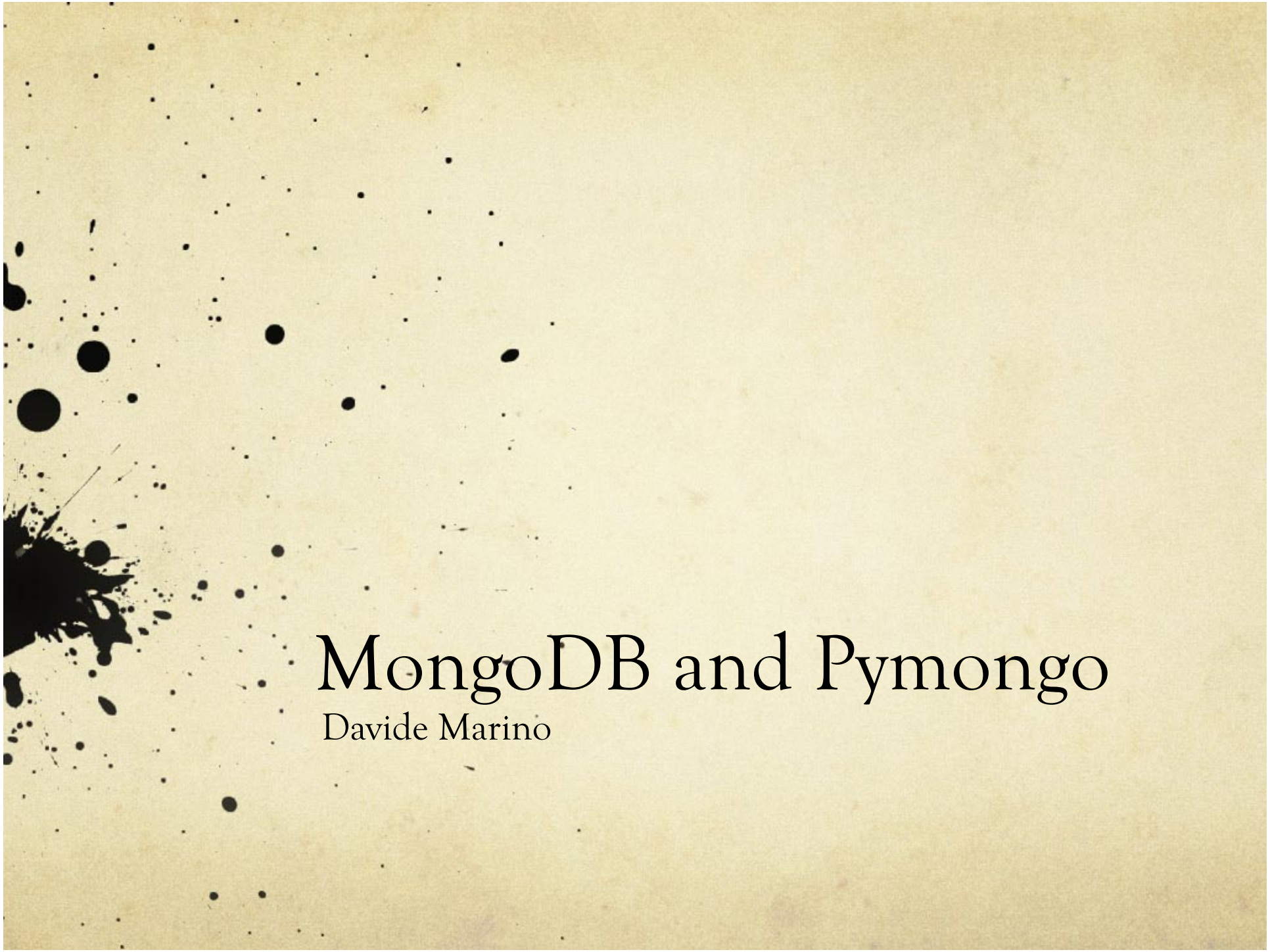
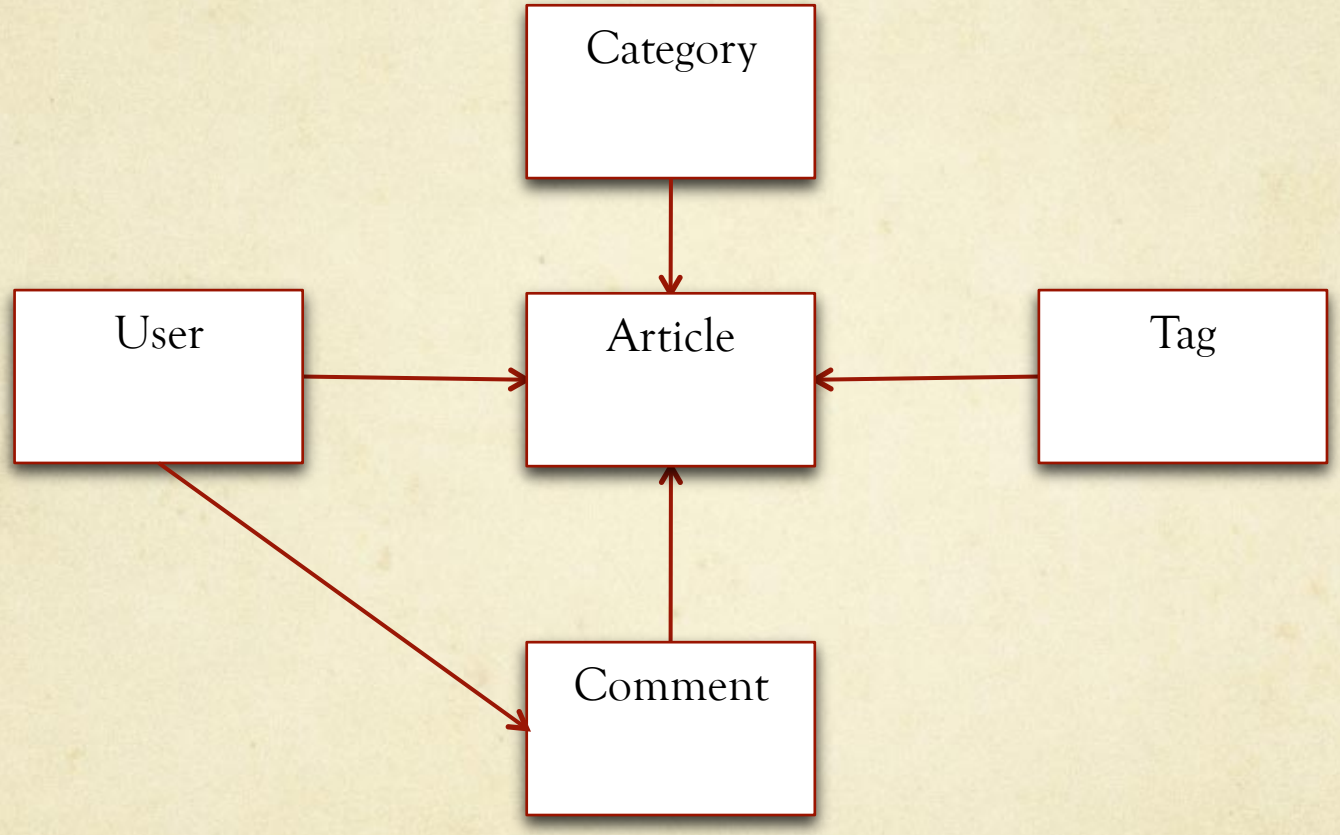


MongoDB and Pymongo

Davide Marino

Relational DB



Relational DB advantages

- Data stored compact
- Rigid schema
- Data optimized

Mapping big data

- Relational DB:
  - Difficult to store
  - Scaling via big iron
  - Schema must be known
- Goal
  - Scale horizontally
  - Incorporate what works
  - Delete what doesn't

No sql and CAP theoreme

- by Brewer basically says that a distributed systems can only have two of the following three properties:
  - Consistency i.e. each node has the same data
  - Availability i.e. a node will always answer queries if possible
  - Partition tolerance i.e. work despite a network failure so nodes cannot communicate with one another
- MongoDB comes in the “AP” category

Mongo DB

- MongoDB (from "humongous") is an open-source document database, and the leading NoSQL database. Written in C++, MongoDB features:
  - Document-Oriented Storage »
    - JSON-style documents with dynamic schemas offer simplicity and power.
  - Full Index Support »
    - Index on any attribute, just like you're used to.
  - Auto-Sharding »
    - Scale horizontally without compromising functionality.
  - Querying »
    - Rich, document-based queries.
- MongoDB library for python pymongo

Adding data

```
import pymongo
from datetime import datetime

conn = pymongo.MongoClient()
db = conn.test
# Clear the test database
conn.drop_database("test")
post = {"author": "Ross",
        "text": "My first blog post!",
        "tags": ["mongodb", "python", "pymongo"],
        "date": datetime.utcnow()}
db.posts.insert(post)

list(db.posts.find())
```

Query 1

```
import pymongo
db = pymongo.MongoClient().training
#where score = 90
db.scores.find({"score": 90})
# limit to 10 results
db.scores.find().limit(10)
#count results
db.scores.find({"score": 90}).count()
# where score > 90
db.scores.find({"score": {"$gt": 90}}).count()
#where score < 90
db.scores.find_one({"score": {"$lte": 60}})
# in and not in examples
db.scores.find({"name": {"$in": ["quiz", "exam"]}})
db.scores.find({"name": {"$nin": ["quiz", "exam"]}})
```

Query 2

```
#regular expressions
import re
rgx = re.compile("^qu")
db.scores.find_one({"name": rgx})
#descengin order
db.scores.find().sort([("score", -1)])
#ascending order
db.scores.find().sort([("score", -1)])
#use distinct() to find all distinct values of a field
db.scores.distinct("name")
```

Update and Remove

```
#Update
db.things.update({"_id": 123}, {"hello": "world"})
# update using  $set
db.things.update({"_id": 123}, {"$set": {"hello": "PyCon Ireland"}})
#retrieve the document, modify it client side and save it
#back again using the save() method.
doc = db.things.find_one({"_id": 123})
doc['ts'] = datetime.datetime.now()
db.things.save(doc)
db.things.find_one({"_id": 123})
#Remove
db.things.remove({"name": "Rozza"})
```

Indexes

```
import pymongo
conn = pymongo.MongoClient()
db = conn.training
db.scores.drop_indexes()
# add index on score
db.scores.ensure_index([("score",
pymongo.ASCENDING )])
#or mulitindex
db.scores.ensure_index([("score", pymongo.ASCENDING),
("name", pymongo.DESCENDING)])
```

Map Reduce

```
from bson.code import Code
mymap = Code("function () {"
             "  this.tags.forEach(function(z) {"
             "    emit(z, 1);"
             "  });"
             "}")
myreduce = Code("function (key, values) {"
                "  var total = 0;"
                "  for (var i = 0; i < values.length; i++) {"
                "    total += values[i];"
                "  }"
                "  return total;"
                "}")
coll = db.things.map_reduce(mymap, myreduce, "myresults")
```

Example 1

```
import datetime
import mongoengine as db

conn = db.connect('tumblelog')
conn.drop_database('tumblelog')


class Post(db.Document):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=True)
    title = db.StringField(max_length=255, required=True)
    slug = db.StringField(max_length=255, required=True)
    body = db.StringField(required=True)
    author = db.ReferenceField('User', required=True)
    comments = db.ListField(db.EmbeddedDocumentField('Comment'))

    def __unicode__(self):
        return unicode(self.title) or u"New Post"


class Comment(db.EmbeddedDocument):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=True)
    body = db.StringField(verbose_name="Comment", required=True)
    author = db.StringField(verbose_name="Name", max_length=255, required=True)

    def __unicode__(self):
        return (u"comment by %s" % self.author) if self.author else "New Comment"
```

Example 2

```
class User(db.Document):
    email = db.StringField(required=True)
    first_name = db.StringField(max_length=50)
    last_name = db.StringField(max_length=50)

ross = User(email="ross@10gen.com",
            first_name="Ross",
            last_name="Lawley").save()

Post(title="mongoengine post",
     slug="mongoengine-post",
     body="Welcome to Europython 2012!",
     author=ross).save()

comment_1 = Comment(author="Bob",
                    body="Nice post thanks")
comment_2 = Comment(author="Ross",
                    body="Florence rocks!")

Post.objects.update(push_all__comments=[comment_1, comment_2])
```

Thank you

Any questions?