GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 4: NOSQL & PYTHON

I. INTRO TO PYTHON
II. NOSQL DATABASES
III. CASSANDRA
IV. REDIS
V. MONGODB

II. NO-SQL DATABASES

NO-SQL databases are a new trend in databases

*The title **NOSQL** refers to the lack of a relational structure between stored objects*

NO-SQL databases are a new trend in databases

*The title **NOSQL** refers to the lack of a relational structure between stored objects*

*Most importantly, they often attempt to minimize the need for **JOIN** operations*

NO-SQL databases are a new trend in databases

http://db-engines.com/en/ranking_trend

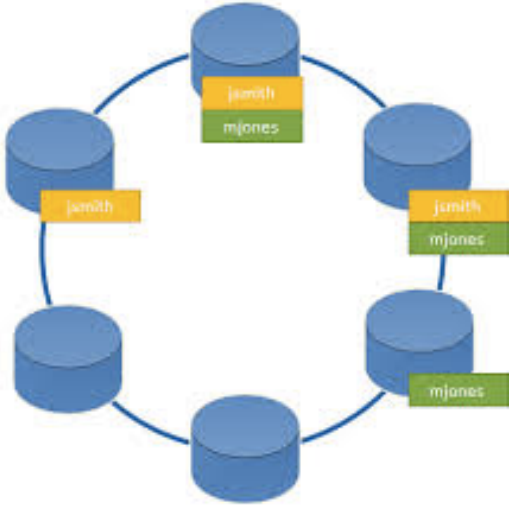
No-SQL DB Types (most popular):

- **Document Stores - *MongoDB, CouchDB, etc.***
- **Wide Column Stores - *Cassandra, BigTable, HBase, etc.***
- **Search engines - Solr, ElasticSearch, etc.**
- **Key-value Stores - Redis, Riak, DynamoDB, etc.**
- **Graph DBMS - Neo4J, OrientDB, etc.**

III. CASSANDRA

Apache Cassandra

- Hybrid between a key-value store and a column-oriented database
- Horizontally scalable
- Values both performance and high availability, replication is configurable at the table level.



Apache Cassandra

- Queried using CQL (Cassandra Query Language)
  - Very similar syntax to SQL, but different performance characteristics
  - No JOIN, complex WHERE clauses, GROUP BY, etc

```
SELECT text FROM posts WHERE
  id = 'ff19bcaa-36e5-4bcc-a3cd-6298b114d38e';
```

Must uniquely identify row with primary key

Unique keys achieved with UUIDs (no autoincrement!)

IV. REDIS

THE BIG HASH IN THE SKY

- Redis
- Memcached
- Riak
- Amazon DynamoDB

Keys	Values
“page:index.html”	“<html><head>…”
“user:123:session_key”	“xDrSdEwd4dS…”
“login_count”	“45435”

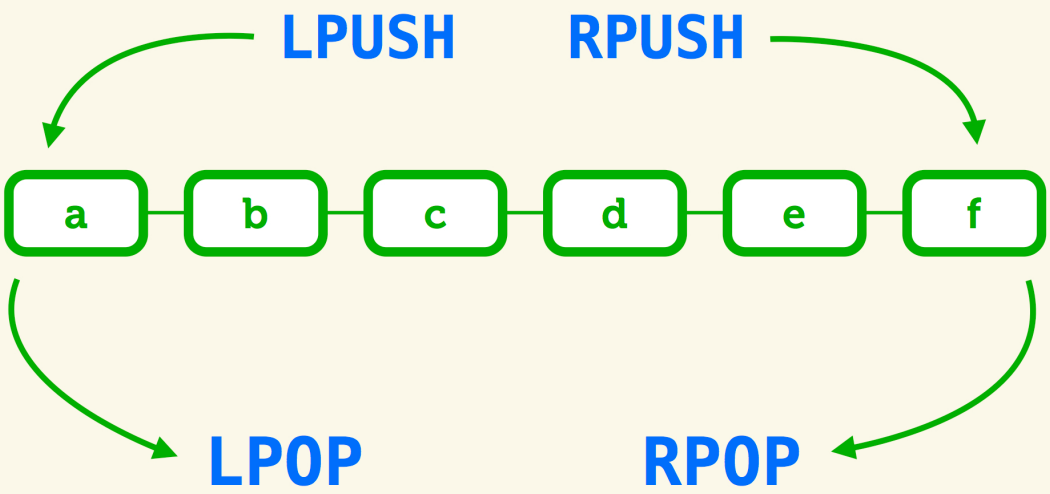
Memcached was:

- *developed by LiveJournal*
- *distributed key-value store (HashMap or Python Dict)*
- *Support two **very fast** operations:*
  - ***get** and **set***

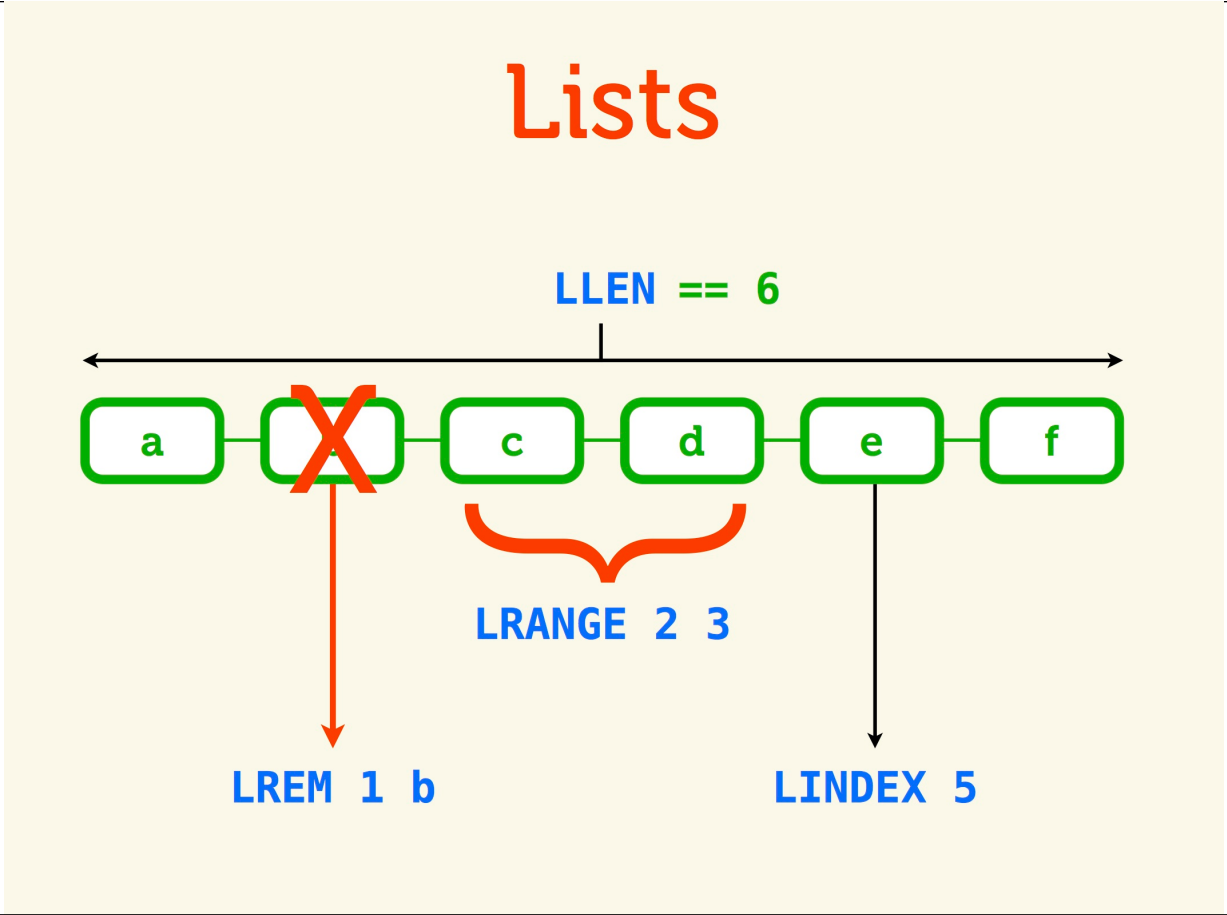
Redis is like Memcache, but add more data types, persistence, scalability, functions, etc.

Keys	Values	Type
“page:index.html”	“<html><head>…”	string
“login_count”	“45435”	string
users_logged_in_today	{“bob”, “sue”, “jane”, …}	set
latest_post_ids	[201, 204, 209, …]	list
user:123:session_info	{name: “makena”, “date”: “2014-03-11”, … }	hash (or dict)
users_and_scores	(“jane” ~ 1.232, “bob” ~ 2.342, “lucy” ~ 34.534	scored set

Lists

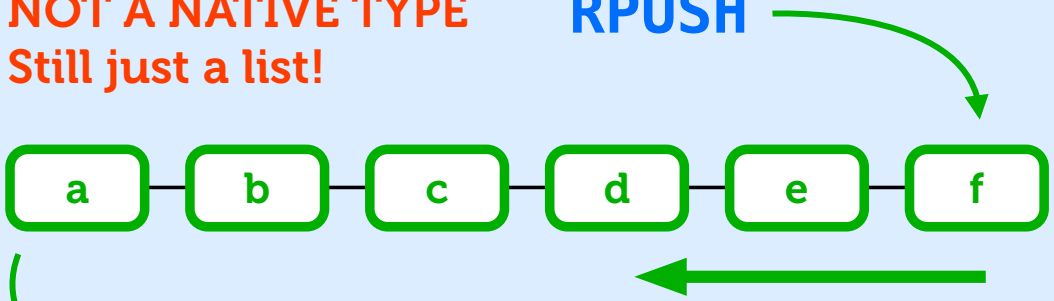


e.g. RPUSH my_q f


Lists
LLEN == 6
a
b
c
d
e
f
LRANGE 2 3
LREM 1 b
LINDEX 5

Queues

NOT A NATIVE TYPE
Still just a list!

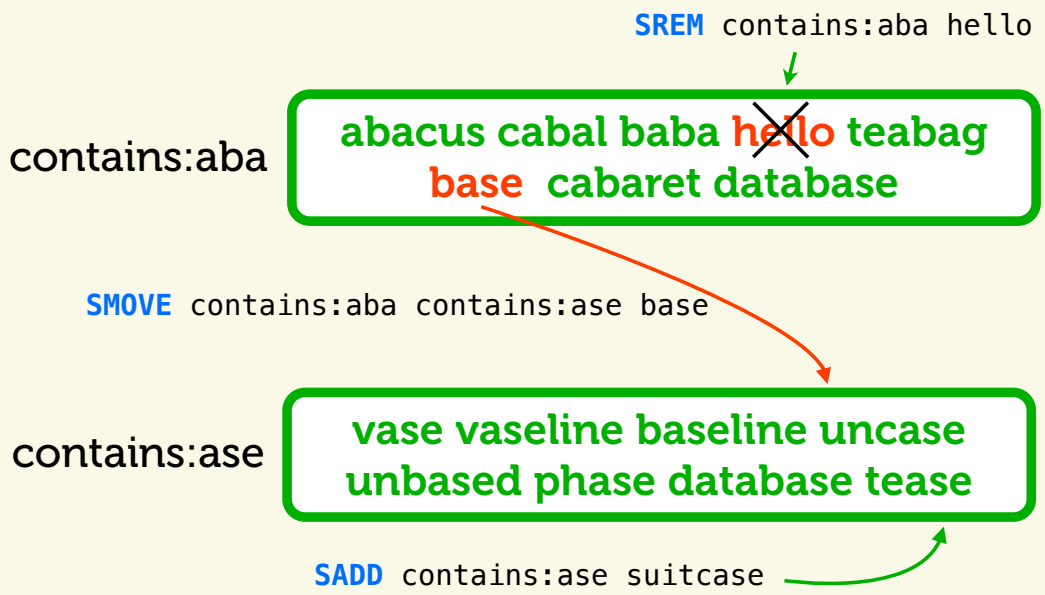


LPOP

Or BLPOP to block (wait) until something can be popped

```
RPUSH my_q abc
RPUSH my_q def
LPOP my_q      == “abc”
LPOP my_q      == “def”
LPOP my_q      == (nil)
```

Sets


SREM contains:aba hello
contains:aba
abacus cabal baba hello teabag
base cabaret database
SMOVE contains:aba contains:ase base
contains:ase
vase vaseline baseline uncase
unbased phase database tease
SADD contains:ase suitcase

Sets

contains:aba

abacus cabal baba teabag
cabaret database

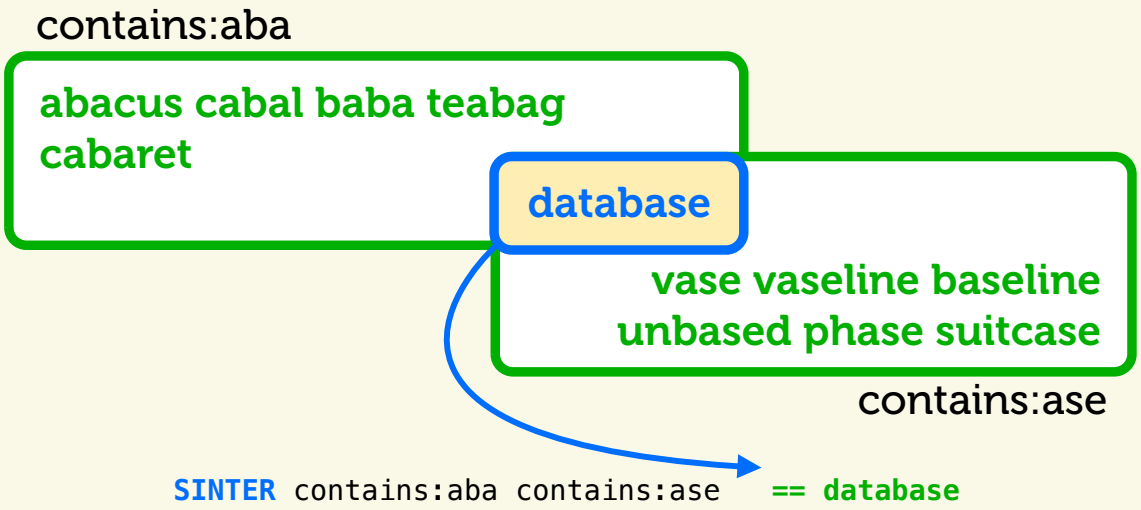
```
SCARD contains:aba            == 6
SISMEMBER contains:aba chips  == 0 (meaning false)
SRANDMEMBER contains:aba      == “teabag”
```

contains:ase

vase vaseline baseline unbased
phase database suitcase

```
SMEMBERS contains:ase         == vase, vaseline,
                                 baseline, unbased,
                                 phase, database,
                                 suitcase
```

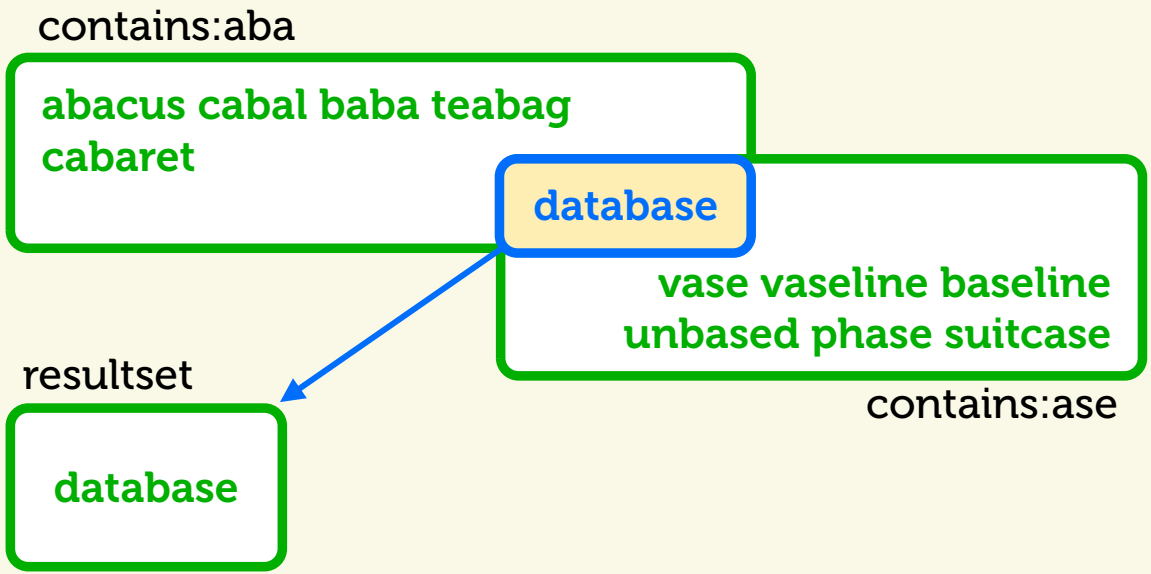
Sets




```
SINTER contains:aba contains:ase    == database
```

This is only a simple example. **SINTER** can take any number of arguments!
SUNION is another command that will join sets together.

Sets




```
SINTERSTORE resultset contains:aba contains:ase
```

SUNIONSTORE does the same for set unions.

Hashes

product:1

created_at	102374657
product_id	1
name	Twinkies
available	10

```
HSET product:1 created_at 102374657
HSET product:1 product_id 1
HSET product:1 name “Twinkies”
HSET product:1 available 10

HGET product:1 name        == Twinkies
HLEN product:1             == 4
HKEYS product:1            == created_at, product_id,
                              name, available
HGETALL product:1          == created_at => 102374657
                              product_id => 1
                              [.. etc ..]
```

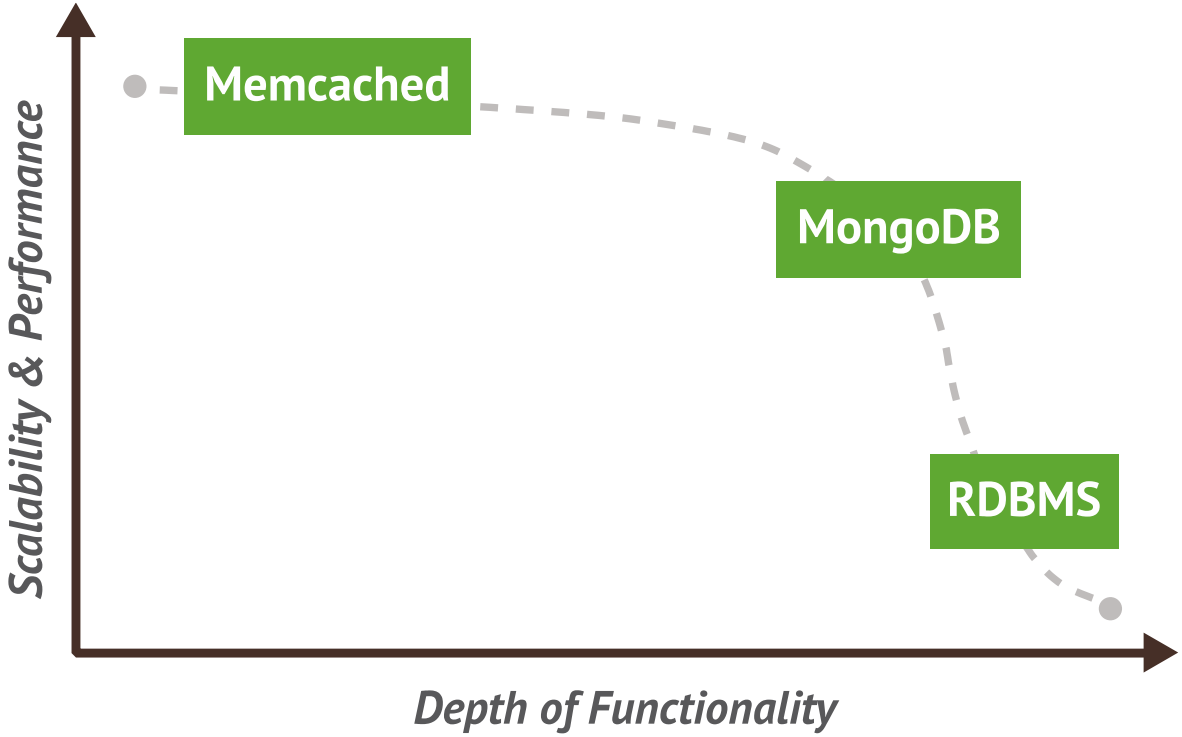
Also...

HVALS HEXISTS HINCRBY HMGET HMSET

http://try.redis.io

V. MONGODB

Document Stores



- MongoDB is the most popular NoSQL system
- Used by
  - Craigslist
  - MTV Networks
  - Intuit
  - foursquare ...
- CouchDB and Couchbase are also popular

Document Stores

- Not for .PDF & .DOC files
- A document is essentially an associative array
- Document = JSON object
- Document = PHP Array
- Document = Python Dict
- Document = Ruby Hash

MongoDB

Queries	• Find Paul's cars • Find everybody in London with a car built between 1970 and 1980
Geospatial	• Find all of the car owners within 5km of Trafalgar Sq.
Text Search	• Find all the cars described as having leather seats
Aggregation	• Calculate the average value of Paul's car collection
Map Reduce	• What is the ownership pattern of colors by geography over time? (is purple trending up in China?)

Instead of "rows", you have a document like this.

```
{ first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: {
     type: "Point",
     coordinates: [-0.128, 51.507] },
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  }
}
```

Rich Queries

- Find Paul's cars
- Find everybody in London with a car built between 1970 and 1980

```
db.cars.find({
    first_name: 'Paul'
})

db.cars.find({
    city: 'London',
    "cars.year" : {
        $gte : 1970,
        $lte : 1980
    }
})
```

```
{ first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: {
     type: "Point",
coordinates :
          [-0.128, 51.507]
     },
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  }
}
```

Geospatial

- **Find all of the car owners within 5km of Trafalgar Sq.**

```
db.cars.find( {
      location:
      { $near :
         { $geometry :
             { type: 'Point' ,
               coordinates :
                     [-0.128, 51.507]
             }
         },
         $maxDistance :5000
         }
} )
```

```
{ first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: {
     type: "Point",
     coordinates :
          [-0.128, 51.507]
     },
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  }
}
```

Aggregation

- **Calculate the average value of Paul's car collection**

```
db.cars.aggregate( [
      {$match : {"first_name" : "Paul"}},
      {$project : {"first_name":1,"cars":1}},
      {$unwind : "$cars"},
      { $group : {_id:"$first_name",
                  average : {
                        $avg : "$cars.value"}}}
])

# Result…
{ "_id" : "Paul", "average" : 215000 }
```

```
{ first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: {
     type: "Point",
coordinates :
          [-0.128, 51.507]
     },
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  }
}
```

_id

- _id is the primary key in MongoDB
- Automatically indexed
- Automatically created as an ObjectId if not provided
- Any unique immutable value could be used

ObjectId

- ObjectId is a special 12 byte value
- Guaranteed to be unique across your cluster
- ObjectId("50804d0bd94ccab2da652599")

```
|----ts-----||---mac---||-pid-||----inc-----|
     4          3        2          3
```

http://try.mongodb.org/