



NoSQL: MongoDB and Redis

School of Information Studies
Syracuse University

Introduction to MongoDB

MongoDB



Document Database

- Database -> Collections -> Documents
- Documents are JSON Schema

Single Master Architecture

- Scales well horizontally, single point of failure
- Consistent reads
- Supports sharding with replication

Data Model

- Document data model—schema-less
- Every document stored has a key
- Uses JavaScript as a query language and JSON as a data format
- There are only integrity constraints on the key; it must be unique per collection

Developer Friendly

Store what you need

The way you need to store it


Easy-to-read JSON format

Without the complexities of:

- Relational design
- Data normalization
- Foreign key constraints



RELATIONAL VS. DOCUMENT



Relational vs. Document

Relational

Designed for use by database administrators

Complex data model, normalized

Flexible, performant query operations

Related information stored in spread tables

Data redundancy is exceptional

Document

Designed for use by programmers/developers

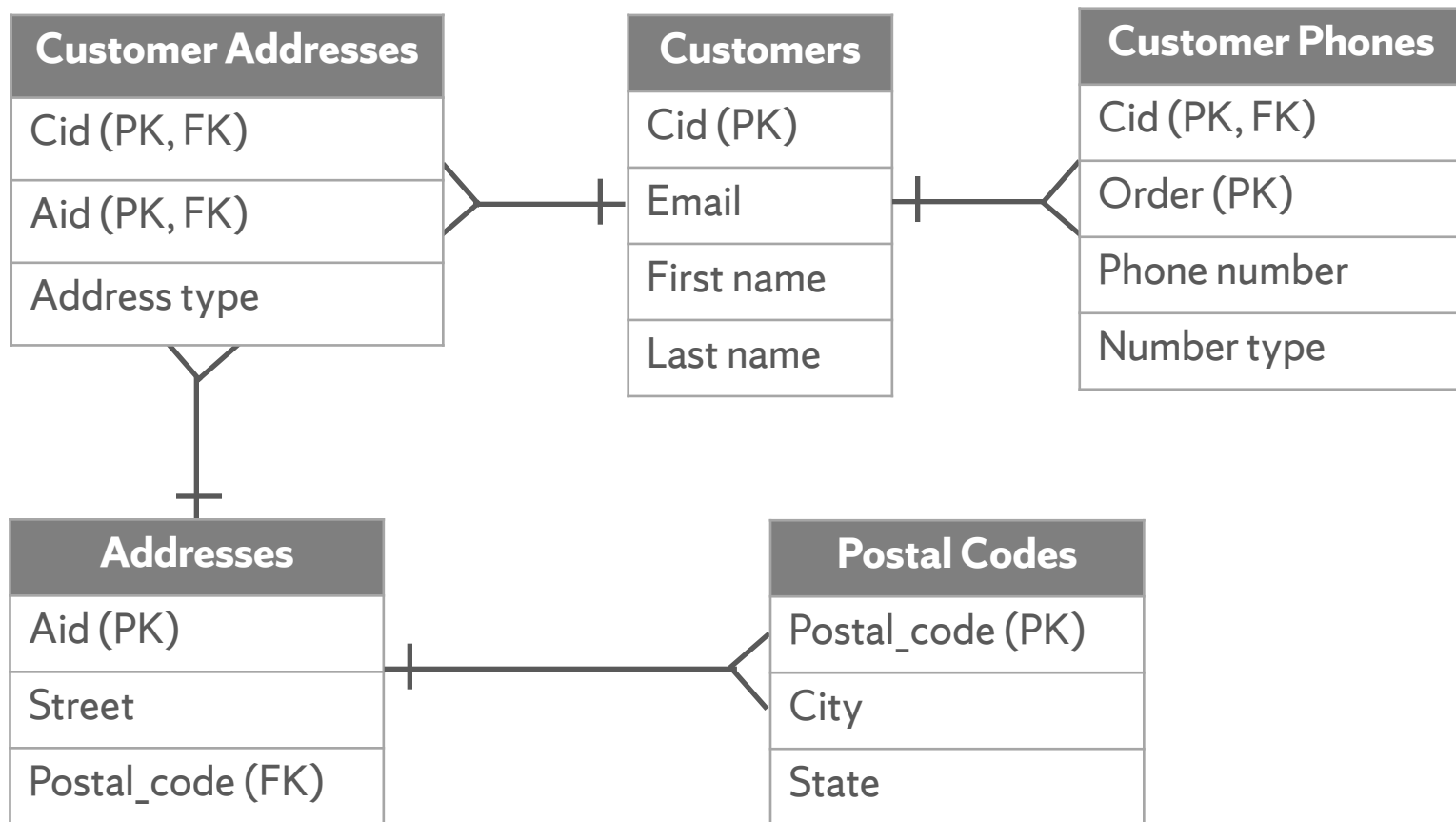
Simplified data model, denormalized

Query not as performant

Related information stored with item

Data redundancy is expected

Customer: The Relational Way



Customer: The Document Way

```
{  cid : 1,
  email: mafudge@syr.edu,
  first name: Michael,
  last name: Fudge,
  phones: [
    { type : home, number: 555-1234 },
    { type : cell, number: 555-9283 }
  ],
  addresses: [ { type: billing,
    street: 1313 Mockinbird Ln,
    city: Syracuse,
    state: NY,
    zip: 13244 }
  ]
}
```





MONGO DB CLUSTERING

Sharding and Replication

Sharding

Splits the database into partitions so that each node hosts part of it

Distributes the I/O over several hosts

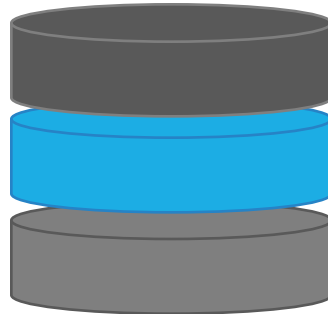
Replica

Synced copies of nodes

Each replica contains an exact mirror of its primary node

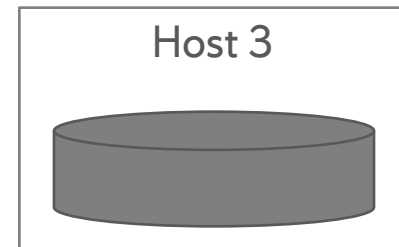
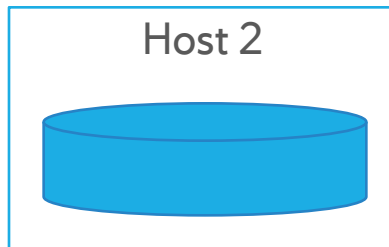
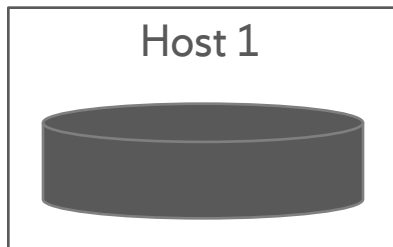
MongoDB Cluster Visualized

Consider a mongo database...



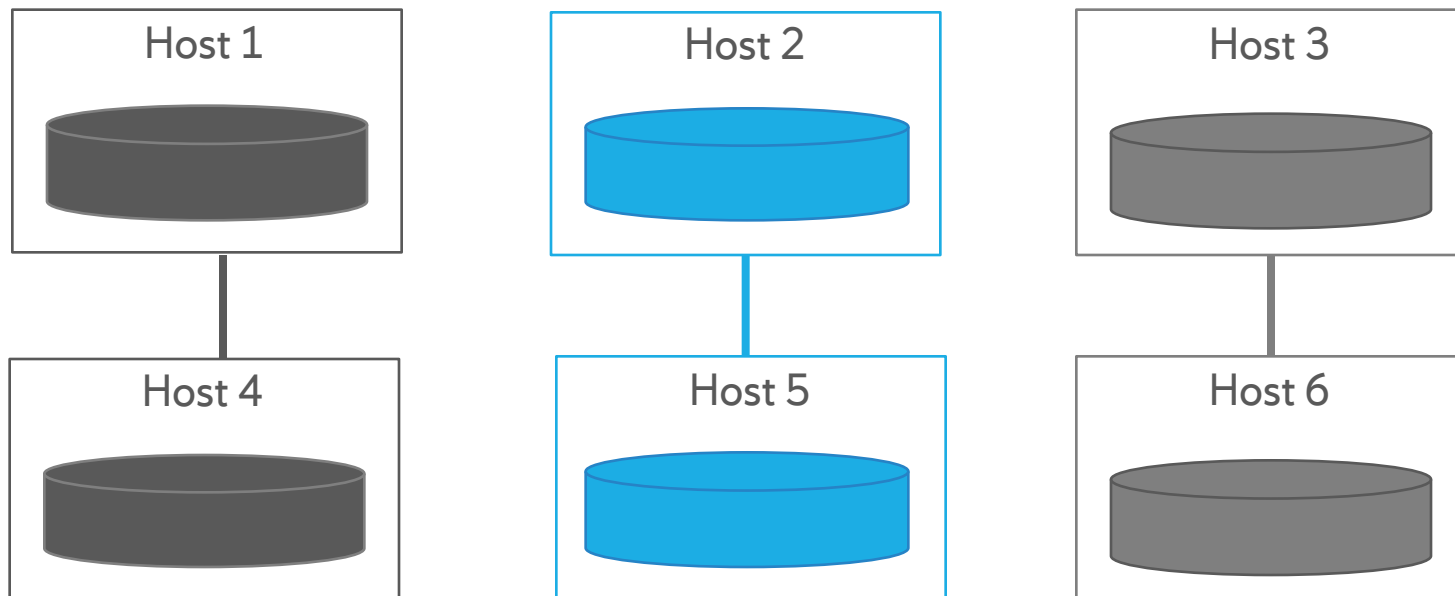
MongoDB Cluster Visualized

It is sharded into three nodes. Each node (hosts 1-3) has different data—no single node has the same data.



MongoDB Cluster Visualized

Each of the three nodes has one replica. Hosts 4-6 are a mirror copy of hosts 1-3, respectively.





MongoDB Data Model And Basic Commands

MongoDB Data Model



Database—boundary for one or more collections



Collection—a subject area for documents to be stored



Document—an individual subject in a collection

Basic MongoDB Commands

`use database`

`show collections`

`db.collection.insert(jsonData)`

`db.collection.find()`

Demo: MongoDB Basics

Current database

See all databases

Use a database—does not need to exist!

See collections in a DB

Insert data

See data



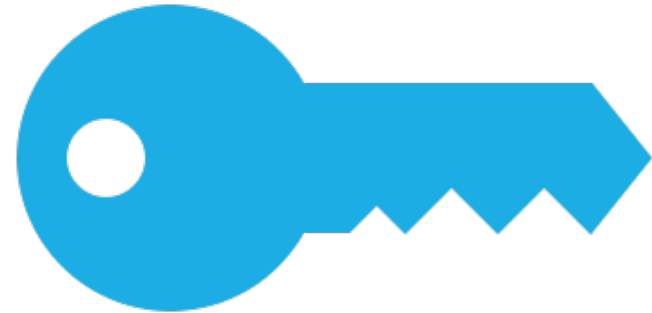


DEMO: _ID

| `_id`

Every document in a MongoDB collection must have a unique ID.

This can be specified when you add the data. If you do not, one will be auto-generated.



Demo: _id

Insert same data—just makes another ID

Use explicit ID

Cannot insert data again



DEMO: UPDATES AND DELETES

Updates and Deletes

You find a document and replace it with **update**.
Partial updates are not allowed.

```
db.collection.update(query, object)
```

You find documents and remove them with **remove**.

```
db.collection.remove(query)
```

Demo: Updating and Deleting

Delete a group on documents

Update the CRV and make the vehicle type SUV

Partial updates don't work!

The document gets replaced

JavaScript for partial updates



Demo: Find

Find Queries

The find method is used to query documents in collections.

The first argument is the column and values to match.

The second argument, which is optional, specifies the columns to return.

```
db.collection.find(query, [columns])
```

Demo: Find Queries

Column projections

Dot notation ex. products.department

Items that exist or do not exist

Demo: Find Methods

Methods can be attached to the results of the query

Count—still counts documents

Limiting output

Sorting output

Really Advanced Queries

Need to be written in JavaScript

Or at least require an
understanding of JavaScript

This is beyond the scope of
the course





DEMO: INDEXING

Simple Indexing

MongoDB indexes work similarly to RDBMS indexes.

They improve query performance at the expense of creating an index table of data.

You are limited to 64 indexes per collection.

To create an index over a column in a collection:

```
db.collection.createIndex(columnName:1)
```

Or over several columns:

```
db.collection.createIndex(colA:1, colB:2)
```


Demo: Indexing

Let's use **explain("executionStats")** to demonstrate the need to index.

Add an index over the query column.

Now the query only accesses the documents in the filter!

Introduction To Redis

Redis



Persistent Data Structure Store

- All data are stored under a key, but values can be of complex types
- Operations are in memory then persisted to disk asynchronously for high performance

Dual Architecture—Your Choice

- Scales horizontally as a consistent single-master cluster
- Can also scale horizontally with high availability and eventual consistency

Data Model

- Flat, global namespace
- Data stored under keys, which can be anything
- A simple query language

Use Cases



Session store—keeping track of logged-in users and their data across a web farm



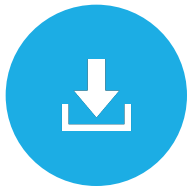
Page cache—storing rendered HTML page output on the server to save on CPU page processing



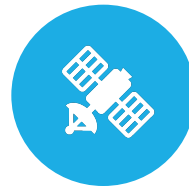
Message queues—the list feature is a very effective queue



Leaderboard—the Redis increment feature keeps the processing in the data layer



Pub/sub—Redis has publish/subscribe message features



Geolocation—Redis can “do the math” for you

Redis Persistence Architectures

RDB: Redis Database File

A snapshot of the data in memory at a point in time

Several snapshots can be taken to return the data to a previous snapshot

On restart, the snapshot is loaded to return the data to its current state

AOF: Append Only File

Similar to a commit/transaction log

Stores each change as it happens, when it happens

Grows in size over time

On restart, the commits are replayed to return the data to their current state

Redis Scalability



Sentinel

Cluster

Sentinel

High-availability solution

Single-master

Consistent reads

Data replicated to other nodes



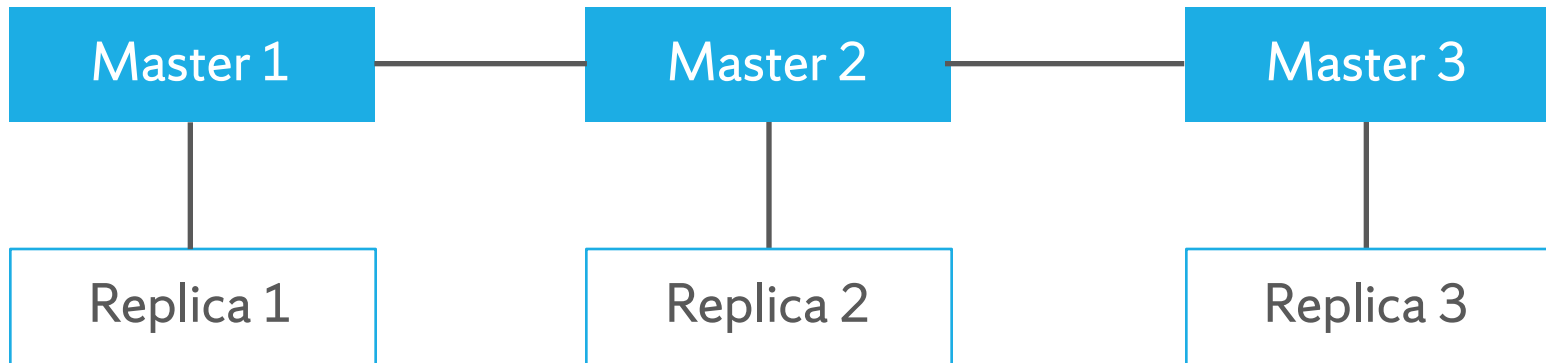
Cluster

Data is sharded/split evenly across nodes through a hashing algorithm.

Shards are replicated asynchronously.

There is no single master.

There is eventual consistency, not guarantees of consistency.







REDIS DATA MODEL AND TYPES

The Redis Data Model

All keys live in a global keyspace/database.

You can have multiple databases, but it's not cluster friendly and should be avoided.

Keys can be anything, for example:

- 100
- -----
- set
- "This is a key"

But, keys should be semantically namespaced, as a best practice:

- Instead of mafudge try, session:user:mafudge
- Instead of about, try pagecache:/home/about

Data Types

Redis is a data structure database and has unique data types, which provide for a variety of data-oriented activities:

Strings—arbitrary textual data

Lists—collections of strings that can be treated like a queue or stack

Sets—unordered collection of strings, no duplicates

Hashes—a set of fields/values under a single key





KEY / VALUE STRINGS AND DEMO

String Key/Value Commands

GET—get a value for the given key

SET—the value for a given key

MSET—set multiple keys/values

MGET—get multiple

EXISTS—check if a key exists

GETSET—get the current value, then set a new value

DEL—delete a key and its value

KEYS—find all keys matching a pattern

String Key/Value Timeouts

Set an expiration on the key in seconds

Set x test—set key x to value “test”

Expire x 5—expire key x in 5 seconds

Set x test ex 5—same thing in a single command

Demo: Strings

Demonstrate basic Redis commands

Why namespacing is important in Redis, even though it's not required!

Example of how a page cache might be implemented in Redis





HASHES AND DEMO

Hashes

Hashes allow for storage of several fields under a single key

HSET key field value

HMSET key field1 value1 field2 value2...

HGET key field1

HGETALL key

HMGET key field1 field3 field4

HEXISTS key field

Demo: Session Data

Demo of using Redis to store data about the current logged-on user





LISTS AND DEMO

Lists

For building lists, queues, and stacks:

L/RPUSH key item1 item2 item3 add to beginning or end

LRANGE 0 -1 (all items)

LRANGE 0 2 (first three items)

LPUSH key item0 (add to beginning)

LPOP key (remove from beginning)

RPOP key (remove from end)

LLEN key (length of list)

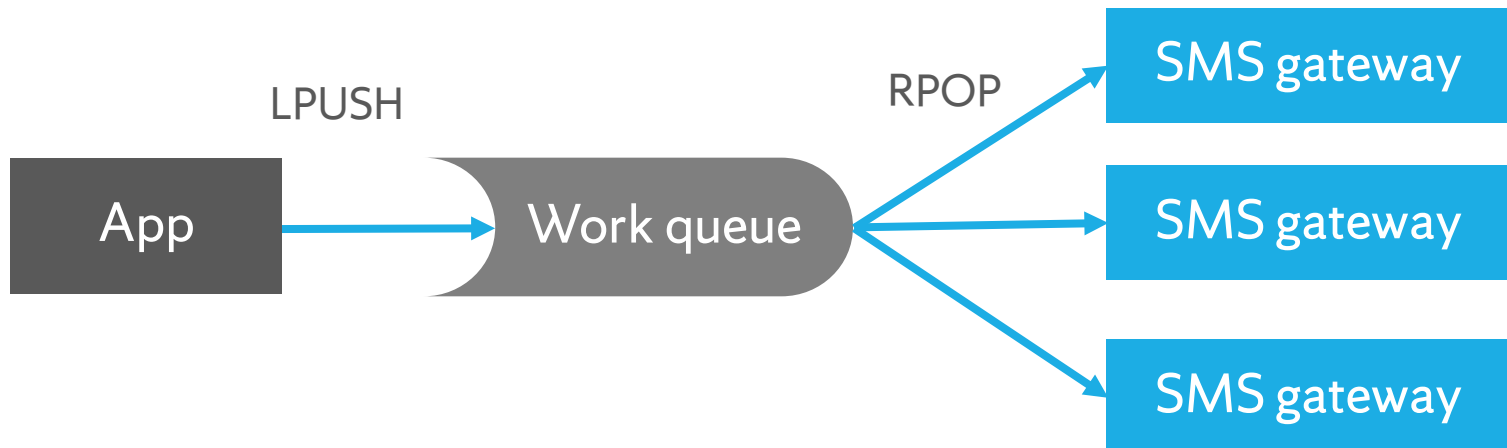
LINDEX key number (get value at index)

Demo: Lists

Simple list example

Work/command queue example: sending TXT messages, emails, and so on when events must be queued

Multiple clients can process from the work queue as ready







ORDERED SETS AND DEMO

Ordered Sets

Useful for leaderboards and dashboards

ZADD key score1 member1 score2 member2

ZRANGE key 0-1 WITHSCORES (shows members + scores)

ZADD key INCR score1 member1 (adds to member)

ZADD key score1 existing1 (overwrites)

ZRANK key member (return index in set)

ZSCORE key member (return score)

Demo: Sales Leaderboard

Demonstrate a sales leaderboard in Redis

Add scores

Show leaderboard

Increment scores

Set scores

Get vales and indexes





PUBLISH / SUBSCRIBE AND DEMO

Publish/Subscribe

Pub/sub model: subscriber sees messages as they are published

SUBSCRIBE channel

PUBLISH channel message

Demo: Pub/Sub

Quick demo of the publish/subscribe feature