NoSQL Systems

CS 5513

University of Oklahoma

Contents

NOSQL SYSTEMS: MOTIVATIONS, FEATURES AND CATEGORIES

NoSQL Motivation

- Originally motivated by Web 2.0 applications
- Goal is to scale simple OLTP (OnLine Transaction Processing)-style workloads to thousands or millions of users
- Users are doing both updates and reads

What is the Problem?

- Scaling a relational DBMS is hard
- Much more difficult to scale ***transactions***
- ***Because we need to ensure ACID properties***
  - Hard to do beyond a single machine
- Current DBMS technology does not provide adequate tools to scale-out a database from a small number of machines to a large number of machines.

NoSQL Key Feature Decisions

- Want a data management system that is
  - Elastic and highly scalable
  - Flexible (different records have different schemas)
- To achieve the above goals, willing to give up
  - Complex queries: e.g., give up on joins
  - Multi-object transactions
  - ACID guarantees: e.g., eventual consistency is OK
  - *Not all NoSQL systems give up all these properties*

“Not Only SQL” or “Not Relational”

- Six key features:
  1. Scale horizontally “simple operations”
     – key lookups, reads and writes of one record or a small number of records, simple selections
  2. Replicate/distribute data over many servers
  3. Less powerful query language
  4. Weaker concurrency model than ACID
  5. Efficient use of distributed indexes and RAM
  6. Flexible schema

Terminology

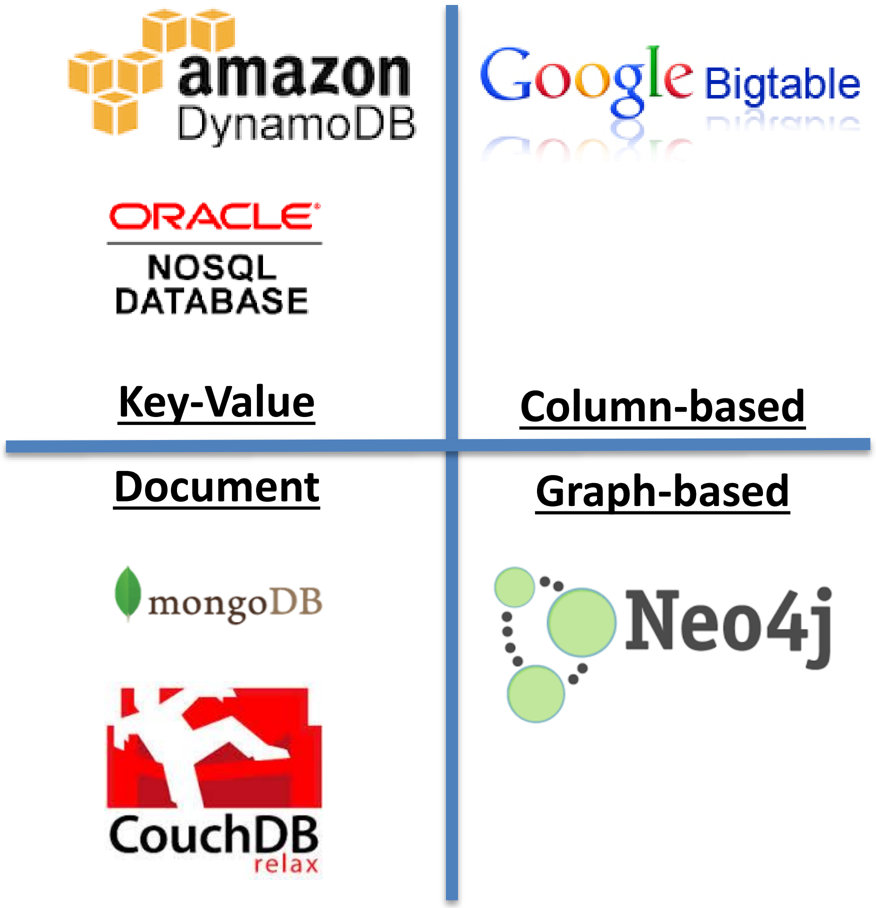
- Sharding = horizontal partitioning by some key, and storing records on different servers in order to improve performance
- Horizontal scalability = distribute both data *and* load over many servers => scale out
- Vertical scaling = when a DBMS uses multiple cores and/or CPUs => scale up

NoSQL systems vs. SQL systems

NoSQL system	SQL system
Usually do not use SQL language	Use SQL language
Does not support relational model	Supports relational model
Usually no join queries	Have join queries
Distributed systems can be eventually consistent	Strong consistency in distributed systems
ACID properties not enforced	ACID properties enforced
Easy to scale horizontally	Difficult to scale horizontally

Categories of NoSQL Systems

- The main categories of NoSQL Systems are:
  - Key-Value stores
    - E.g. DynamoDB, Voldemort, Oracle NoSQL
  - Document-stores
    - E.g. MongoDB, CouchDB
  - Column-based stores
    - E.g. BigTable, Hbase,
  - Graph-based
    - E.g. Neo4j, GraphBase



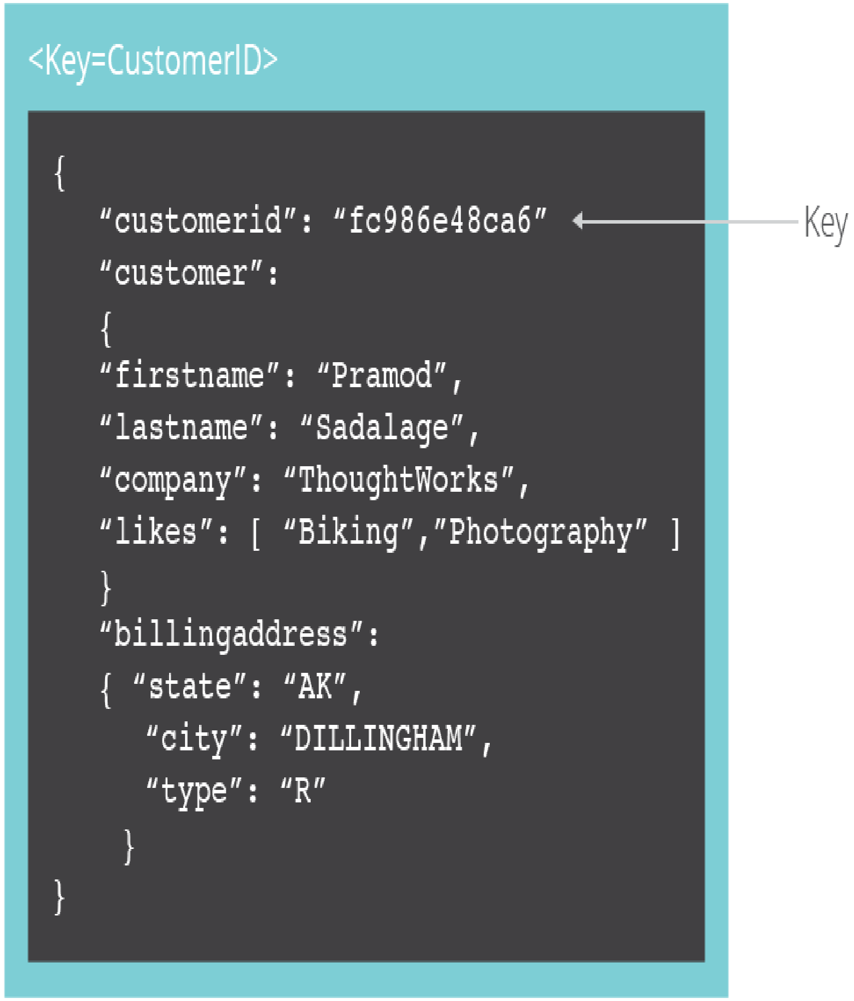
NOSQL SYSTEMS: DOCUMENT STORES

Document-Stores Features

- Example systems: MongoDB, SimpleDB, CouchDB
- Document: encapsulates and encodes data (or information) in some standard formats or encodings (XML, JSON, BSON, etc.)
- A document is addressed by its key:
  - key-document
- DBMS offers APIs/query language to retrieve document contents
- Different implementations have different ways to organize documents (e.g. collection, tag, etc.)
- Documents within a collection can have different fields (unlike records in a relational table)

Document Databases

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)



```
{
   “customerid”: “fc986e48ca6”
   “customer”:
   {
   “firstname”: “Pramod”,
   “lastname”: “Sadalage”,
   “company”: “ThoughtWorks”,
   “likes”: [ “Biking”,”Photography” ]
   }
   “billingaddress”:
   { “state”: “AK”,
     “city”: “DILLINGHAM”,
     “type”: “R”
    }
}
```

DOCUMENT STORE EXAMPLE: MONGODB

Document store example: MongoDB (http://www.mongodb.org/)

- Data Model and Schema:
  - A document is in BSON (Binary JSON) format: an object is a BSON document.
  - A MongoDB database stores its data as collections of documents, each document has one or more fields.
  - Documents in the same collection may have different fields.
  - A field can be added to one document without being added to other documents in the same collection.
  - MongoDB uses dynamic schemas
    - You can create collections without defining the structure, i.e. the fields or the types of their values, of the documents in the collection.
    - You can change the structure of documents simply by adding new fields or deleting existing ones.
    - Documents in a collection need not have an identical set of fields.

Document-Store System: MongoDB

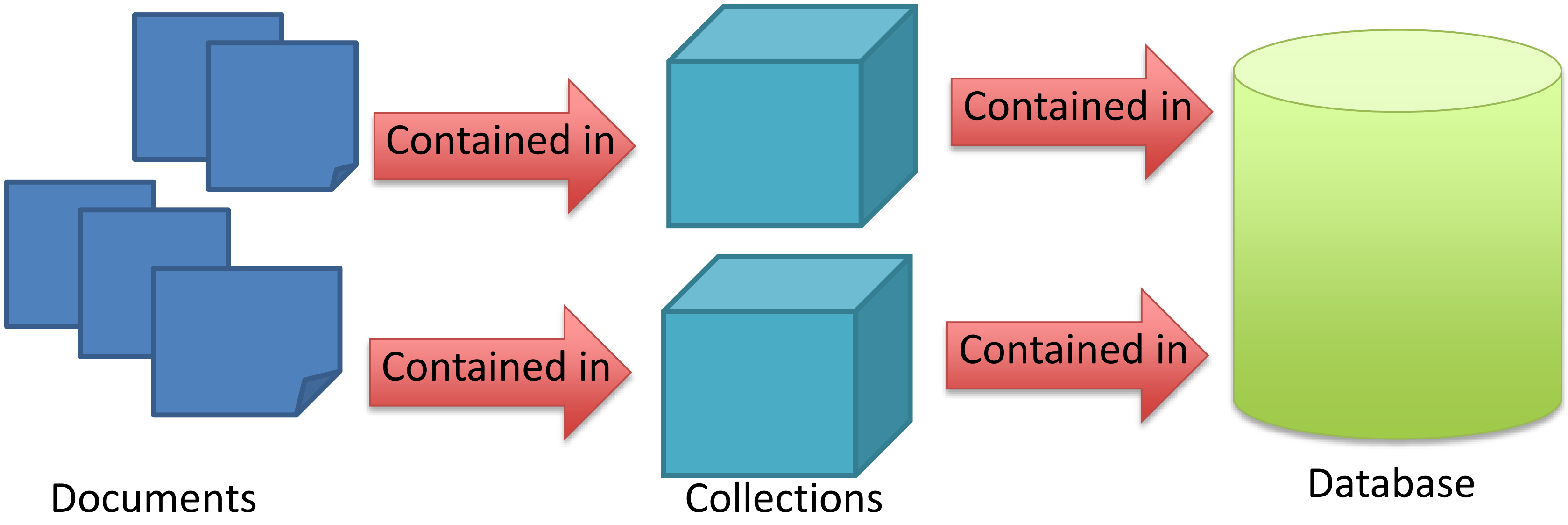
SQL Terms/Concepts	MongoDB Terms/Concepts
database	*database*
table	*collection*
row	*document* or *BSON* document
column	*field*
index	*index*
table joins	embedded documents and linking
primary key Specify any unique column or column combination as primary key.	*primary key* In MongoDB, the primary key is automatically set to the *_id* field.
aggregation (e.g. group by)	aggregation framework

Document Store System: MongoDB

- Queries
  - Search by field, range queries, regular expression searches.
  - Queries can return specific fields of documents.
  - No support for joins and ACID transactions.
  - No support for SQL; Mongo DB has its own query language.
- Indexing
  - Primary and secondary indices can be done for any field in a MongoDB document.
- Replication
  - Master-slave replication.
- Caching
  - MongoDB keeps the most recently used data in RAM
- Load balancing
  - MongoDB scales horizontally using sharding.
  - The data is split into ranges (based on the given shard key) and distributed across multiple shards.
  - A shard is a master with one or more slaves.
  - Do load balancing among nodes automatically

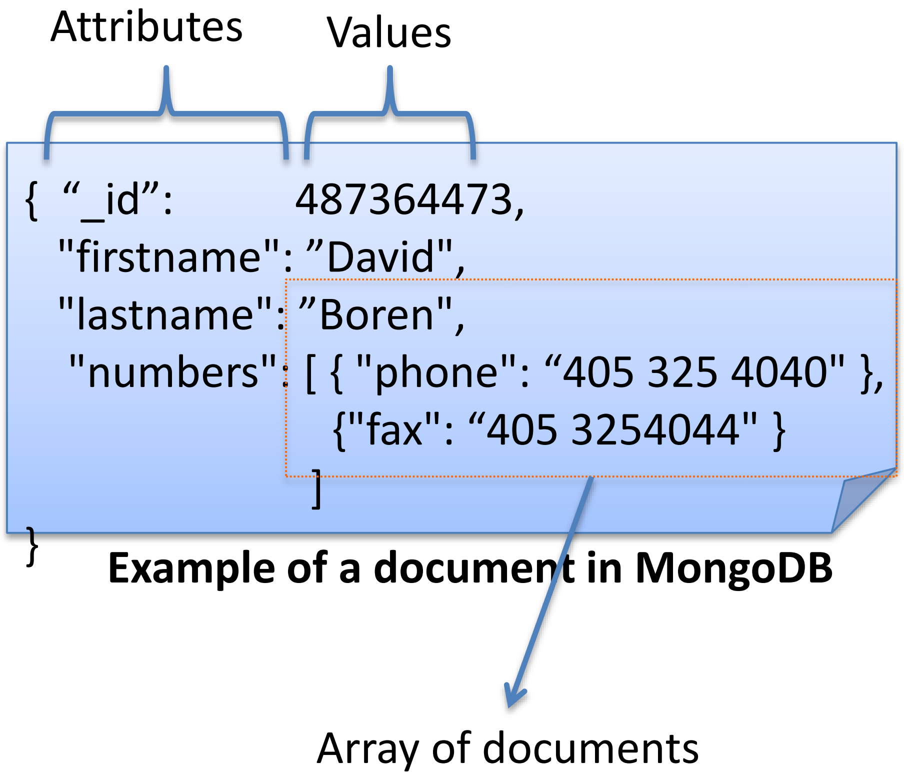
MongoDB’s Data Model

- Main elements in the data model
  - Document (similar to a row in relational model)
  - Collection (similar to a table in relational model)
  - Database



MongoDB’s Data Model

- Documents:
  - MongoDB’s unit of storage
  - Analog to a row
  - No schema enforced
  - Can have nested documents
  - The primary key is always the “_id” attribute. If “_id” is not set by the user, then MongoDB will set it (unless it’s a nested document).



```
{ “_id”:        487364473,
  "firstname": ”David",
  "lastname": ”Boren",
   "numbers": [ { "phone": “405 325 4040" },
               {"fax": “405 3254044" }
              ]
}
```

Example of a document in MongoDB

MongoDB’s Data Model

- Examples of Documents:

```
{ "firstname": ”David",
  "lastname": ”Boren",
   “age” : 75
}
```

Plain document

```
{ "Type" : "Book”,
  "Title" : “Database System Concepts, 6th Edition",
  "Publisher" : ”Oxford”,
   "Author" : [”Silberschatz, Abraham",
              “Korth, Henry F.",
               ”Sudarshan, S.”]
}
```

Document with
an attribute with
array value

MongoDB’s Data Model

- Examples of Documents (cont’d):

```
{ "Type" : "Book”,
  "Title" : “The Art of Computer Programming, 1st
vol",
  "Publisher" : ”Addison Wesley”,
   "Author" : “Knuth, Donald”
   “Chapters”: [
                  {“Num” : 1,
                   “Title” : “Basic Concepts”,
                   “Num Pages” : 231  },
                   {“Num” : 2,
                   “Title” : “Information Structures”,
                    “Num Pages” : 234  }
                  ]
}
```

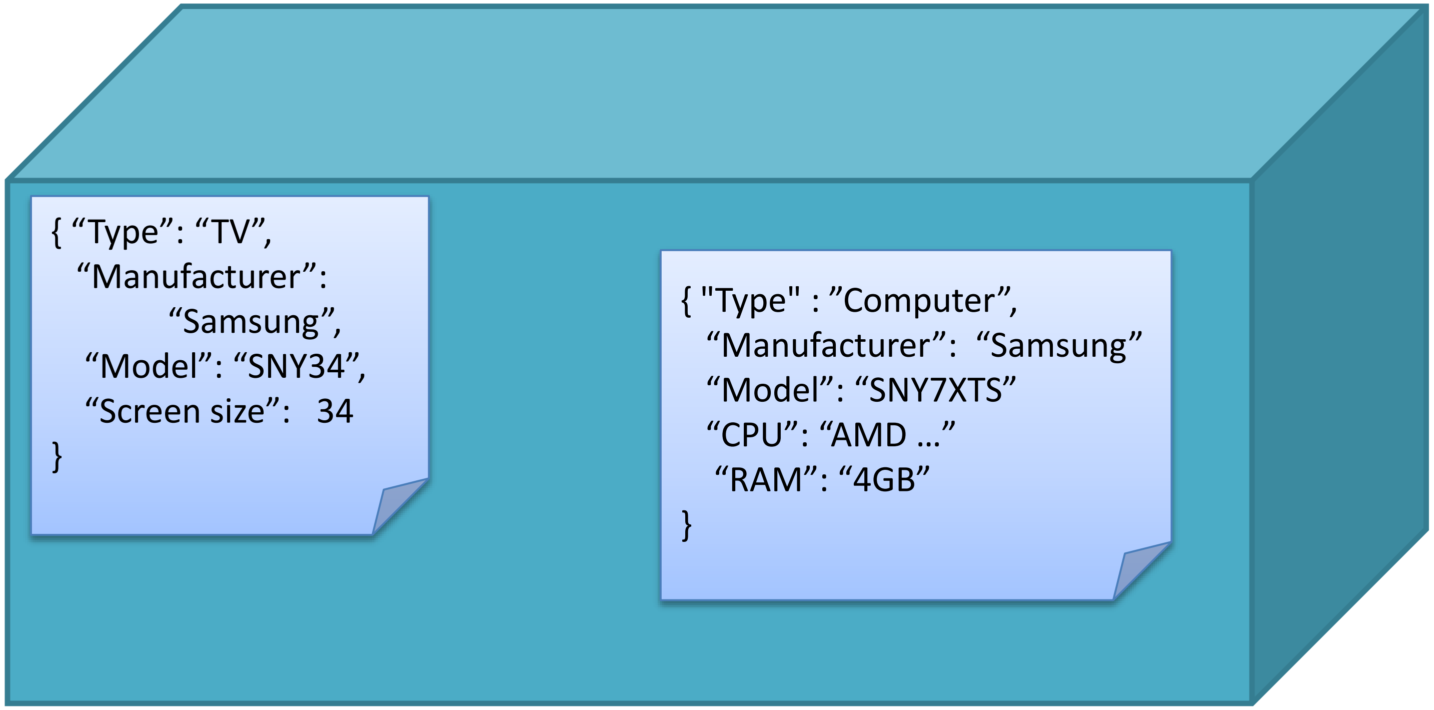
Document with an attribute which is an array of nested documents

MongoDB’s Data Model

- Collections:
  - Its elements are documents
  - Similar to a table
  - No schemas

MongoDB’s Data Model

- Examples of Collections:
  - Collections may have documents with different schemas



Electronics Collection

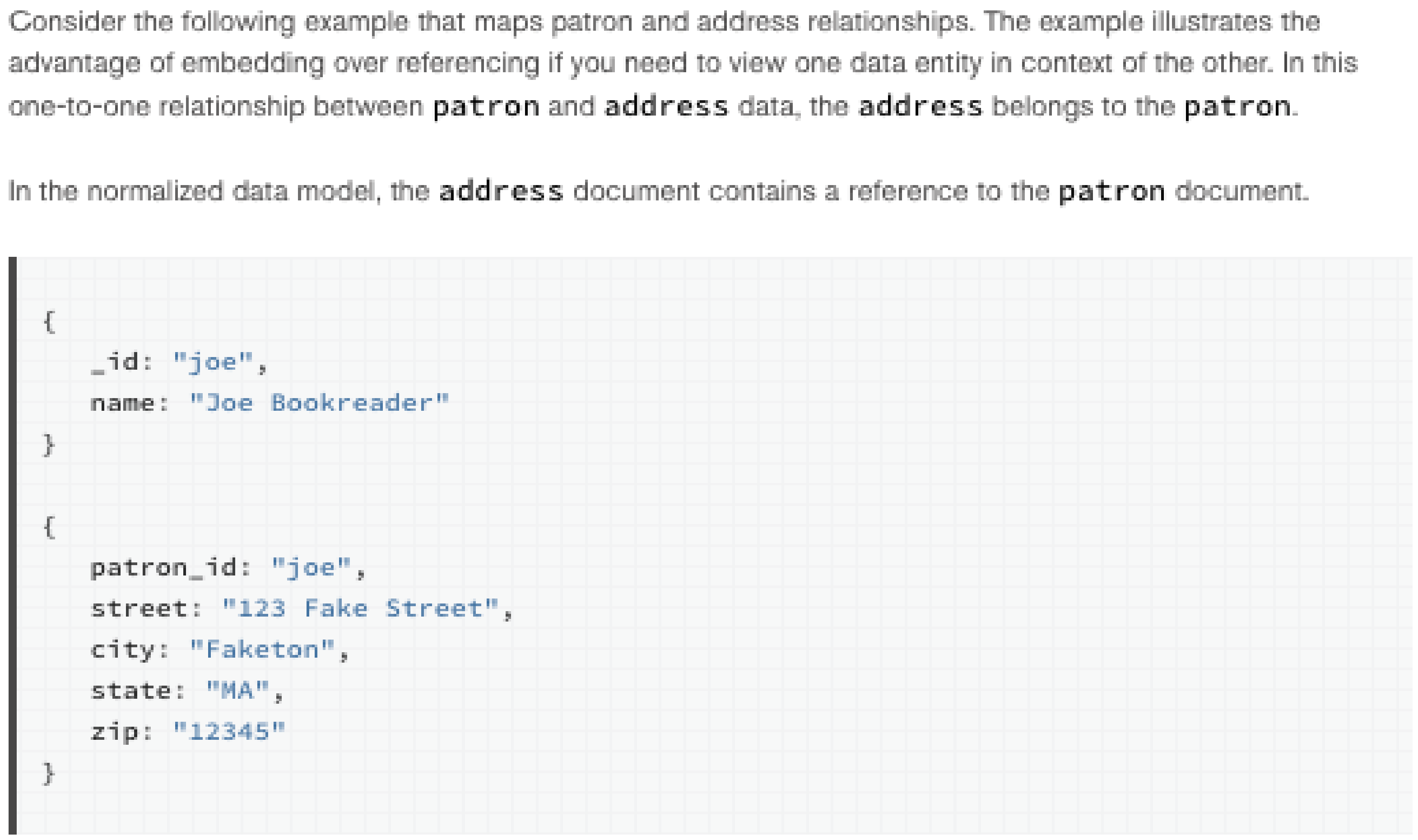
```
{ “Type”: “TV”,
  “Manufacturer”:
          “Samsung”,
   “Model”: “SNY34”,
   “Screen size”:   34
}
```

```
{ "Type" : ”Computer”,
  “Manufacturer”:  “Samsung”
  “Model”: “SNY7XTS”
  “CPU”: “AMD …”
   “RAM”: “4GB”
}
```

MODELING RELATIONSHIPS BETWEEN DOCUMENTS WITH MONGODB

https://docs.mongodb.org/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/

Modeling One-to-One Relationships in MongoDB (i)

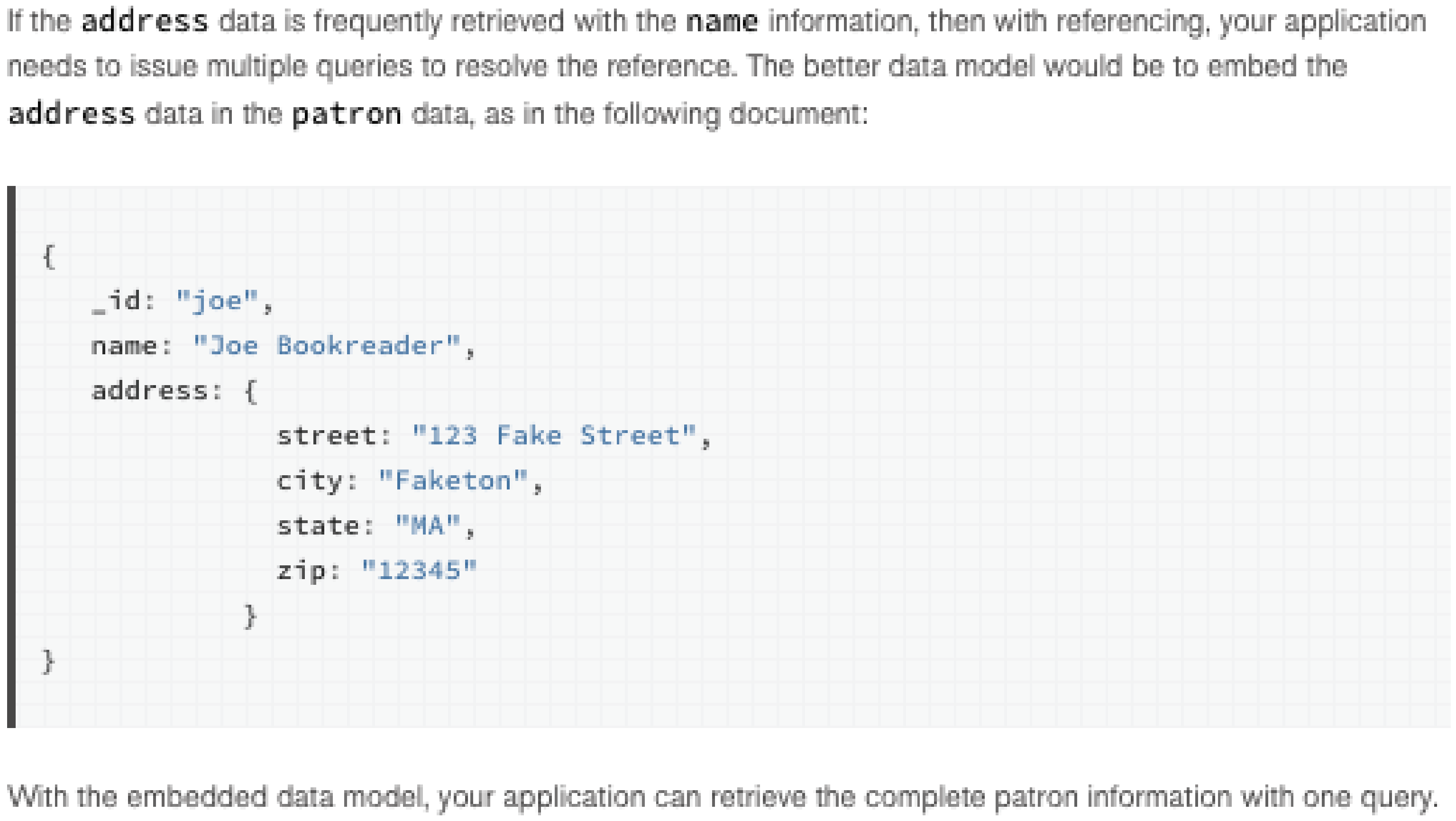


Consider the following example that maps patron and address relationships. The example illustrates the advantage of embedding over referencing if you need to view one data entity in context of the other. In this one-to-one relationship between `patron` and `address` data, the `address` belongs to the `patron`.

In the normalized data model, the `address` document contains a reference to the `patron` document.

```
{
   _id: "joe",
   name: "Joe Bookreader"
}

{
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

Modeling One-to-One Relationships in MongoDB (ii)

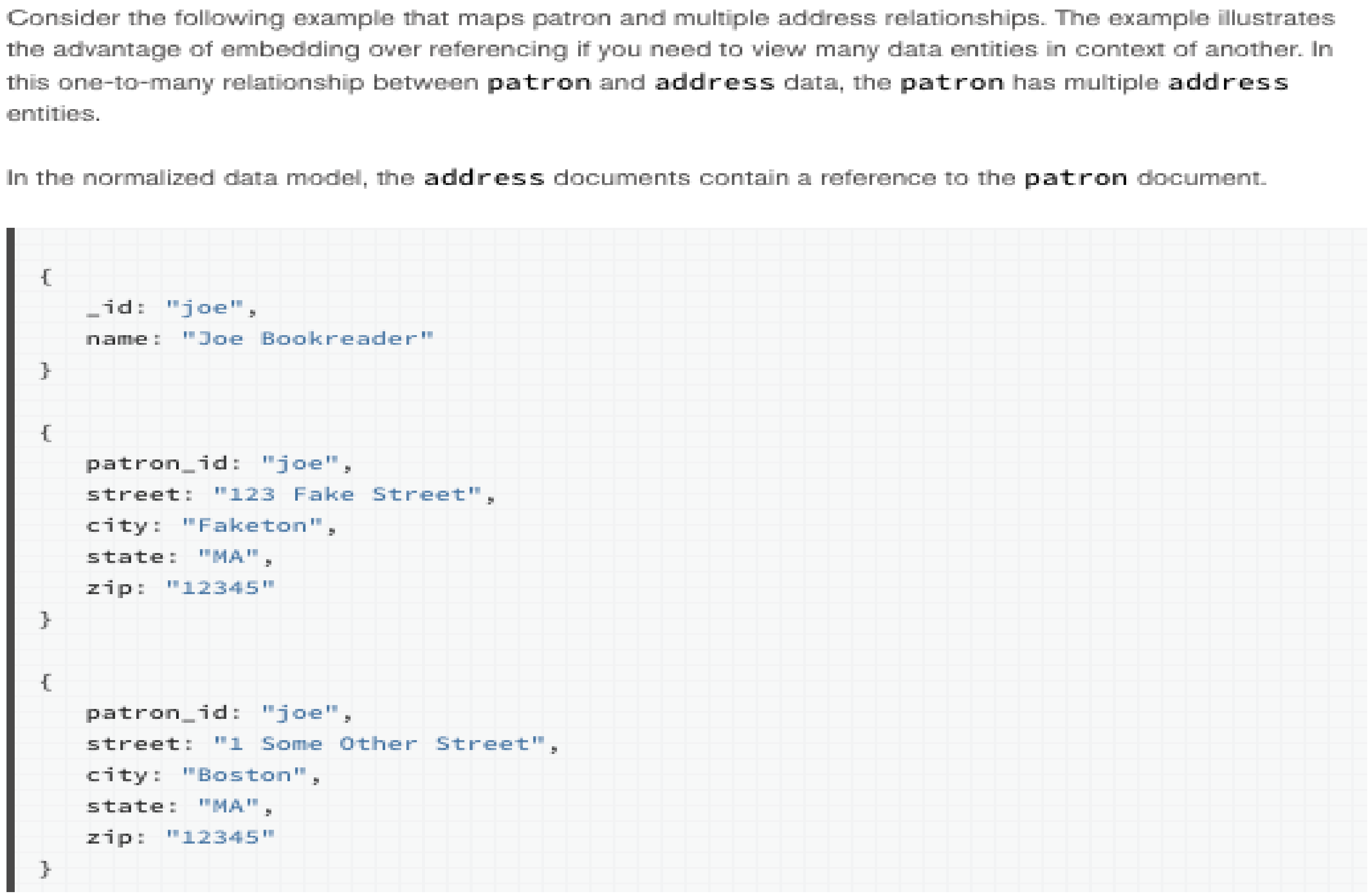


If the **address** data is frequently retrieved with the **name** information, then with referencing, your application needs to issue multiple queries to resolve the reference. The better data model would be to embed the **address** data in the **patron** data, as in the following document:

```
{
   _id: "joe",
   name: "Joe Bookreader",
   address: {
              street: "123 Fake Street",
              city: "Faketon",
              state: "MA",
              zip: "12345"
            }
}
```

With the embedded data model, your application can retrieve the complete patron information with one query.

Modeling One-to-Many Relationships in MongoDB with References (i)

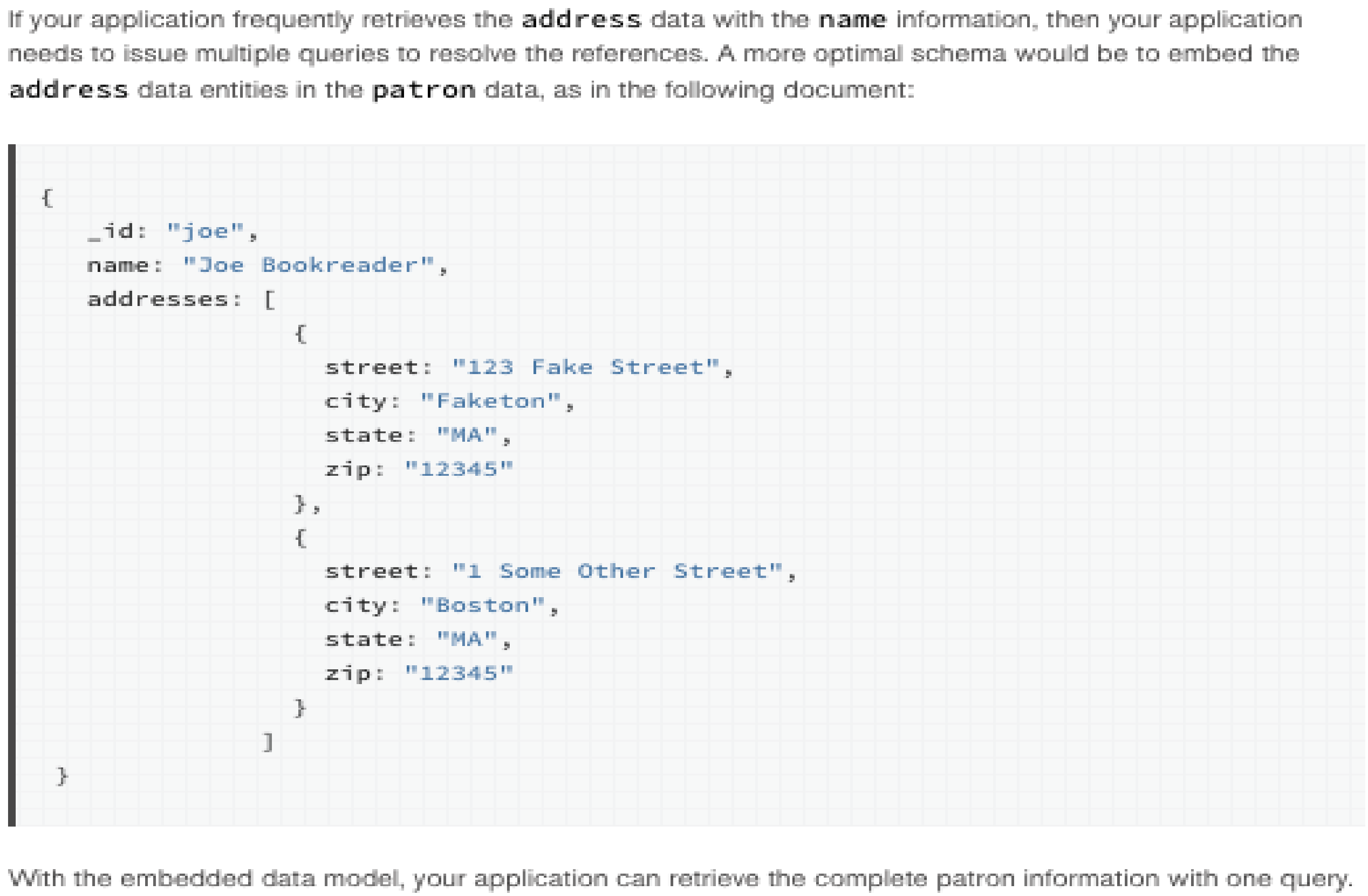


Consider the following example that maps patron and multiple address relationships. The example illustrates the advantage of embedding over referencing if you need to view many data entities in context of another. In this one-to-many relationship between `patron` and `address` data, the `patron` has multiple `address` entities.

In the normalized data model, the `address` documents contain a reference to the `patron` document.

```
{
   _id: "joe",
   name: "Joe Bookreader"
}

{
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

Modeling One-to-Many Relationships in MongoDB with Embedded Documents (ii)

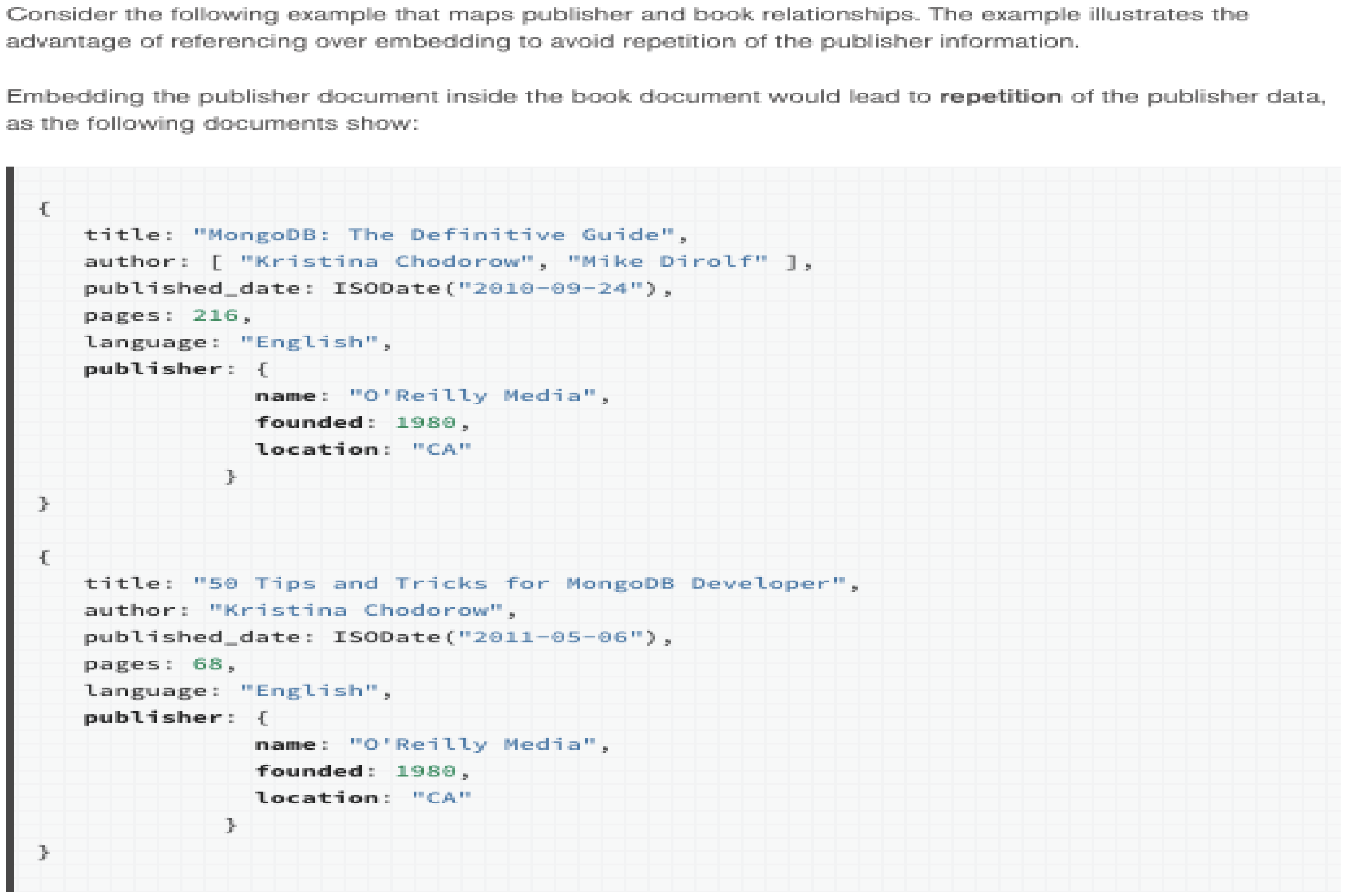


If your application frequently retrieves the `address` data with the `name` information, then your application needs to issue multiple queries to resolve the references. A more optimal schema would be to embed the `address` data entities in the `patron` data, as in the following document:

```
{
   _id: "joe",
   name: "Joe Bookreader",
   addresses: [
                {
                  street: "123 Fake Street",
                  city: "Faketon",
                  state: "MA",
                  zip: "12345"
                },
                {
                  street: "1 Some Other Street",
                  city: "Boston",
                  state: "MA",
                  zip: "12345"
                }
              ]
 }
```

With the embedded data model, your application can retrieve the complete patron information with one query.

Modeling One-to-Many Relationships in MongoDB with Document References (i)

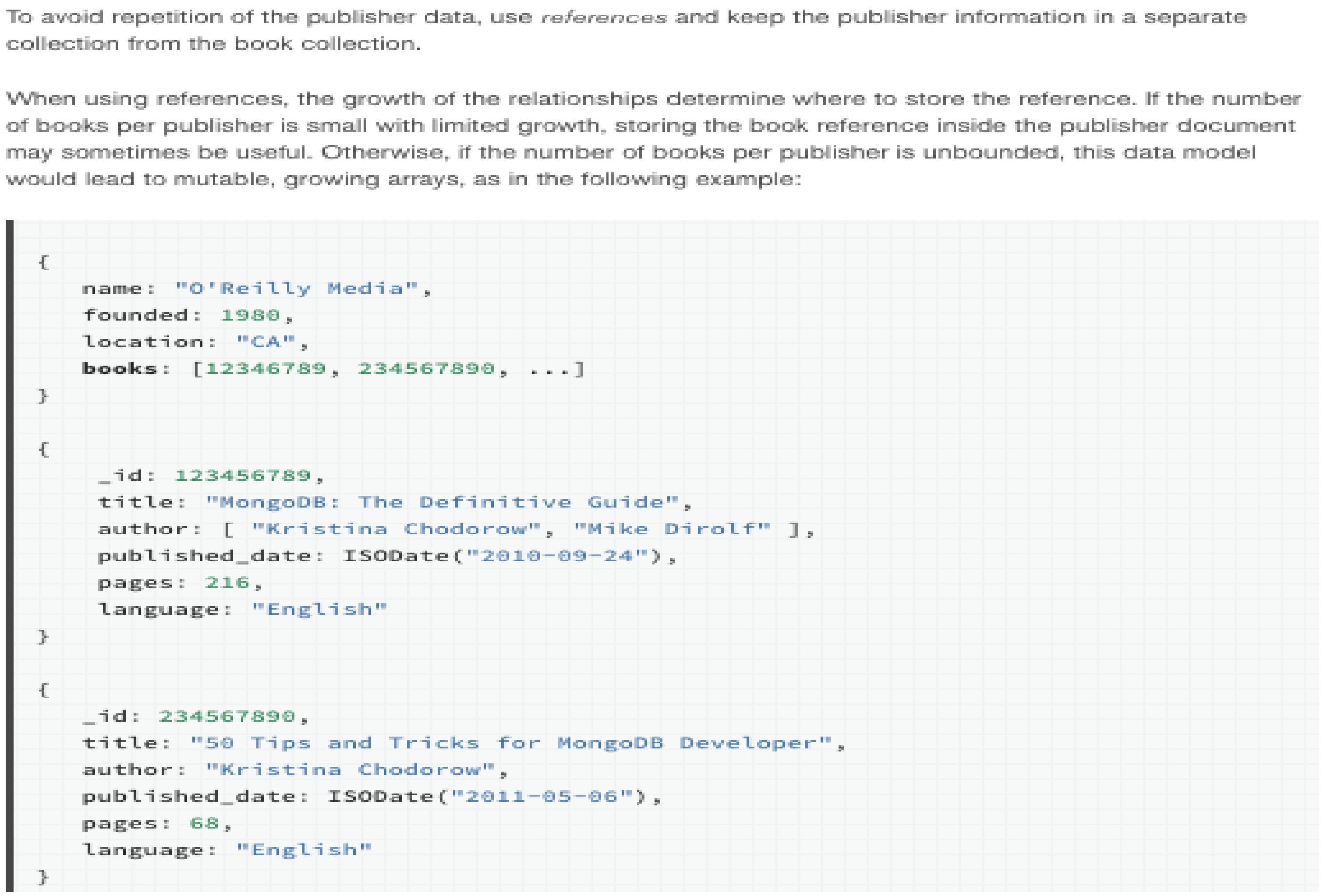


Consider the following example that maps publisher and book relationships. The example illustrates the advantage of referencing over embedding to avoid repetition of the publisher information.

Embedding the publisher document inside the book document would lead to **repetition** of the publisher data, as the following documents show:

```
{
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}

{
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}
```

Modeling One-to-Many Relationships in MongoDB with Document References (ii)



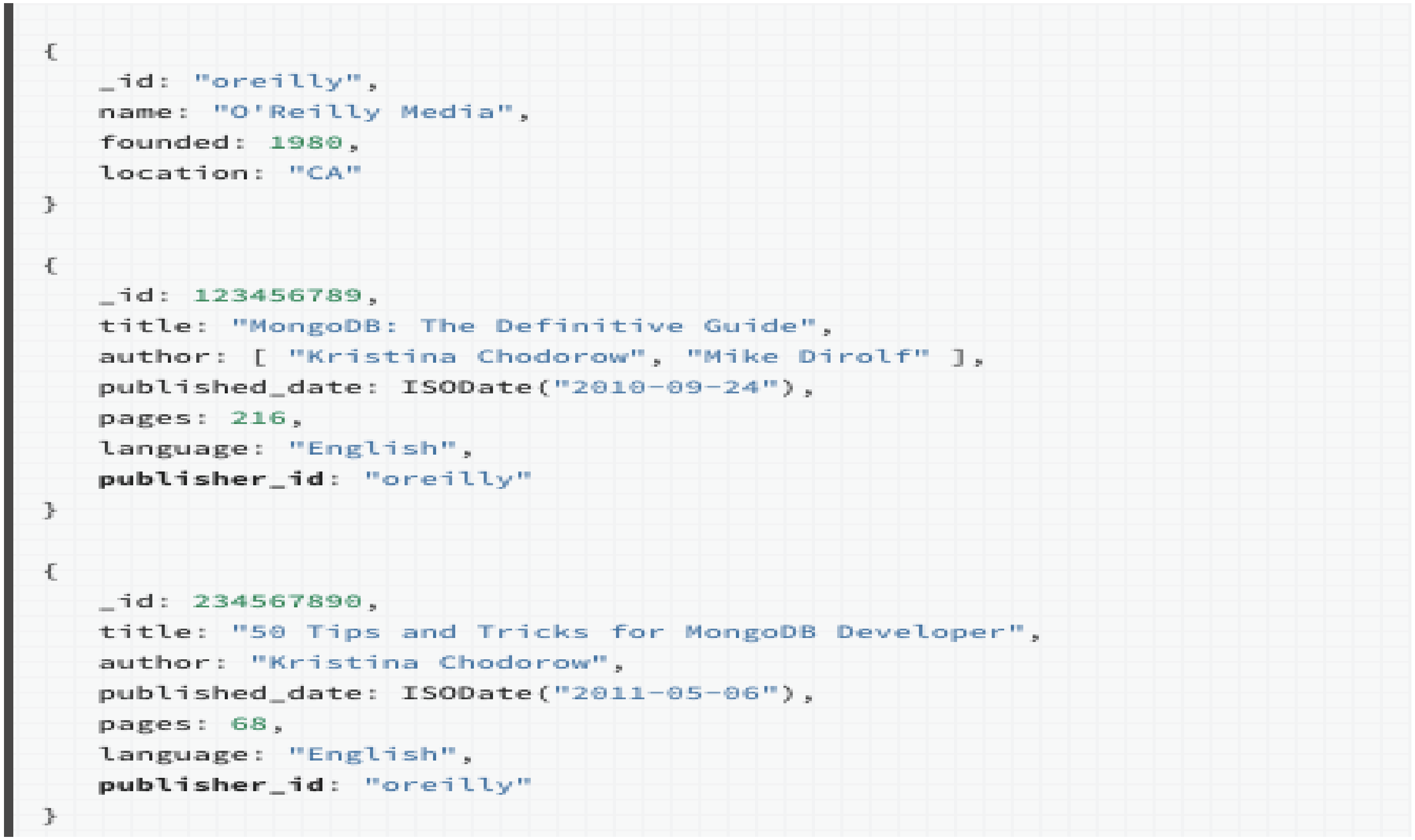
To avoid repetition of the publisher data, use *references* and keep the publisher information in a separate collection from the book collection.

When using references, the growth of the relationships determine where to store the reference. If the number of books per publisher is small with limited growth, storing the book reference inside the publisher document may sometimes be useful. Otherwise, if the number of books per publisher is unbounded, this data model would lead to mutable, growing arrays, as in the following example:

```
{
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA",
   books: [12346789, 234567890, ...]
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English"
}
```

Modeling One-to-Many Relationships in MongoDB with Document References (iii)

To avoid mutable, growing arrays, store the publisher reference inside the book document:



```
{
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher_id: "oreilly"
}
```

MONGODB CRUDE OPERATIONS (CREATE, READ, UPDATE, DELETE)

MongoDB’s Basic Commands

- To list all the databases in the system

```
> show databases
```

- To create a database named “myDB” in case it doesn’t exist, or to switch to it in case it exists so that you can execute all database commands for this database

```
> use myDB
```

- To create a collection named “people” in the database “myDB”

```
> db.createCollection(“people”)
```

MongoDB’s Basic Commands (ii)

- To show all the collections in the current database

```
> show collections
```

- To delete a collection named ‘people’ type

```
> db.people.drop()
```

- To delete a database named ‘myDB’ type

```
> use myDB
> db.dropDatabase()
```

Insertion

- To insert a new document into the collection named “people”

```
> db.people.insert({“_id”:1, “firstname”: “Don”, “lastname”: “Tillman” , “age” : 30})
```

- To create a new document and associate with a variable

```
> don = {_id: 1, “firstname”: “Don”, “lastname”: “Tillman”, “age” : 30}
```

- To insert a document associated with a variable

```
> db.people.insert(don)
```

Note: In case the collection “people” does not exist, it will be created automatically when you enter the command above

Insertion (ii)

- To insert multiple documents (bulk insertion) into the collection named ‘people’

```
> db.people.insert( [ {“_id” : 32,
                       “firstname”: “Don”,
                       “lastname”: “Tillman” ,
                       “age” : 30} ,
                     {“_id” : 5454,
                       “firstname”: “David”,
                       “lastname”: “Boren” ,
                       “age” : 74,
                       “salary”: 3000.00}
                     ]
)
```

Selection Queries

- To select all the documents contained in a collection named “people”

```
> db.people.find({})
```

The equivalent in SQL would be:

```
SELECT *
FROM people;
```

Not valid in MongoDB

Selection Queries (ii)

- To select all documents in the collection ‘people’ that have “lastname” = “Tillman”

```
> db.people.find( {“lastname” : “Tillman”} )
```

Argument is a document!

- To select all documents in the collection ‘people’ that have “lastname” = “Tillman” **and** “firstname” = “Don”

```
> db.people.find( {“lastname” : “Tillman”, “firstname” = “Don”} )
```

Selection Queries (iii)

- To select all the “people” with last name “Tillman” displaying only first names and ages

```
> db.people.find({“lastname”: “Tillman”}, {“firstname”:1, “age”:1, “_id”: 0})
```

The equivalent in SQL would be:

```
SELECT firstname, age
FROM people
WHERE lastname = “Tillman”;
```

Not valid in MongoDB

Deletion Queries

- To delete all documents in the collection ‘people’ that have “lastname” = “Tillman”

```
> db.people.remove( {“lastname” : “Tillman”} )
```
Argument is a document!

- To delete all documents in the collection ‘people’ that have “lastname” = “Tillman” **and** “firstname” = “Don”

```
> db.people.remove( {“lastname” : “Tillman”, “firstname” : “Don”} )
```

Conditional Operators

- When performing selections, we can use other conditional operators too: $lt (less than) , $gt (greater than), $lte (less than or equal), $gte greater than or equal), $ne (not equal)
- To find the documents in “people” whose “lastname” is “Tillman” and age < 30

```
> db.people.find( {“lastname” :“Tillman”, “age” : {$lt : 30 }} )
```

```
SELECT *
FROM people
WHERE lastname = “Tillman” and age < 30;
```

Not valid in MongoDB

Boolean Operators

- To select the documents in “people” whose “lastname” is “Tillman” or “Morrison”

```
> db.people.find( {$or : [{“lastname” : “Tillman”}, {“lastname”: “Morrison”}] } )
```

```
SELECT *
FROM people
WHERE lastname = “Tillman” OR lastname = “Morrison”;
```

Not valid in MongoDB

Boolean Operators (ii)

- To select the documents in “people” whose “lastname” is “Tillman” and whose is age is not < 30

```
db.people.find( {$or: [{"lastname" : "Tillman"}, {"age": {$not:{$lt: 30}}}] } )
```

```
SELECT *
FROM people
WHERE lastname = “Tillman” AND NOT (age < 30);
```

Not valid in MongoDB

Sorting

- To retrieve all documents in the collection "people" sorted in ascending order by "firstname":

```
> db.people.find().sort({"firstname" : 1})
```

```
SELECT * FROM people
ORDER BY firstname ASC;
```

Not valid in MongoDB

- To retrieve all docs in the collection "people" sorted in descending order by "firstname":

```
> db.people.find().sort({"firstname" : -1})
```

```
SELECT * FROM people
ORDER BY Publisher DESC;
```

Not valid in MongoDB

Exists, In, and Limit

- To select all those people that have an attribute “salary”

```
> db.people.find({“salary” : {$exists: true}})
```

- To select all people with “firstname” equal to “Joe” or “Katie”

```
> db.people.find({“firstname” : {$in: [“Joe”, “Katie”]}})
```

```
SELECT *
FROM people
WHERE firstname IN (“Joe”, “Katie”);
```

Not valid in MongoDB

Exists, In, and Limit (ii)

- To select 4 people with “firstname” equal to “Joe” or “Katie”

```
> db.people.find({“firstname” : {$in: [“Joe”, “Katie”]}}).limit(4)
```

```
SELECT *
FROM people
WHERE firstname IN (“Joe”, “Katie”)
LIMIT 4;
```

Not valid in MongoDB

Updates

- To update a set of documents in a collection we use the $set operator.
  - For example, for all people with last name “Tillman” set their ages to 74

```
> db.people.update( {“lastname” : “Tillman”}, {$set: {“age” : 74 }} )
```

is equivalent to the SQL update

```
UPDATE people
SET age = 74
WHERE lastname = “Tillman”;
```

Not valid in MongoDB

Indices

- To create an (ascending) index on the key “age” of the collection “people”

```
> db.people.createIndex({"age": 1})
```

Remember that in MongoDB, schemas are not enforced, so you can create an index on “age” even if some documents in that collection do not have that attribute defined!

Indices (ii)

- Suppose you have a “books” collection. You can then create indices on attributes of nested documents

```
> db.books.createIndex({“Chapters.Title”: 1})
```

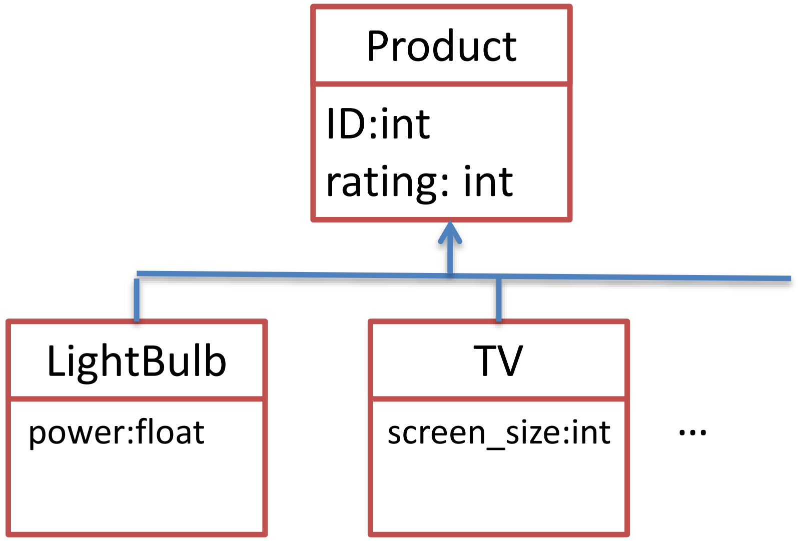
```
{ "Type" : "Book”,
  "Title" : “The Art of Computer Programming, 1st
vol",
  "Publisher" : ”Addison Wesley”,
   "Author" : “Knuth, Donald”
   “Chapters”: [
                    {“Num” : 1,
                     “Title” : “Basic Concepts”,
                     “Num Pages” : 231  },
                     {“Num” : 2,
                     “Title” : “Information Structures”,
                      “Num Pages” : 234  }
                    ]
}
```

When to use MongoDB?

- Applications that need the flexibility of not enforcing a schema
  - For example, when we need to store semi-structured data
- Applications that do not need strong consistency
  - For example, storing the tweets of users in Twitter, or posts of users in Facebook

Use Case: Keeping a Product Catalog

- Relational model Solution 1:
  - At least 1 table per type of object
  - Many table schemas
  - Need to do a join to obtain all attributes of a product

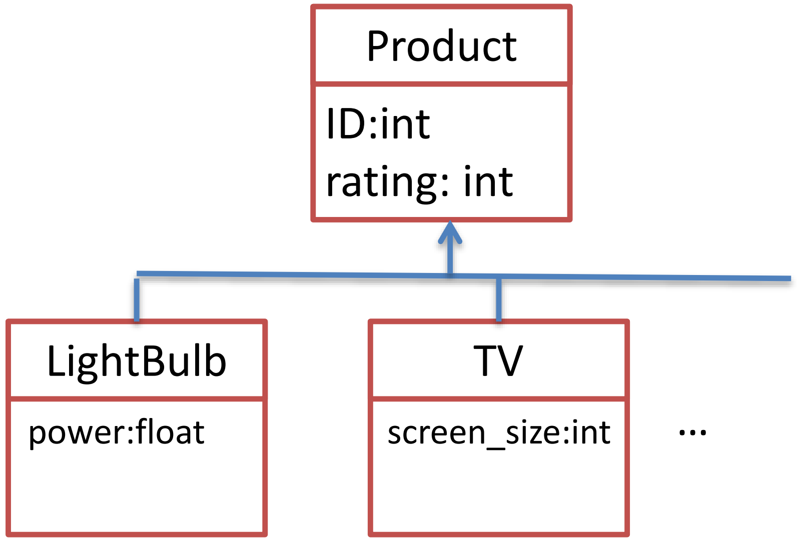


SQL Tables:
Product(ID, rating)
LightBulb(ID, power)
TV(ID, screen_size)
…

Solution 1 for keeping a product catalog in SQL

Use Case: Keeping a Product Catalog (ii)

- Relational model Solution 2:
  - Single table with all possible attributes
  - Many rows filled with nulls → wastes space

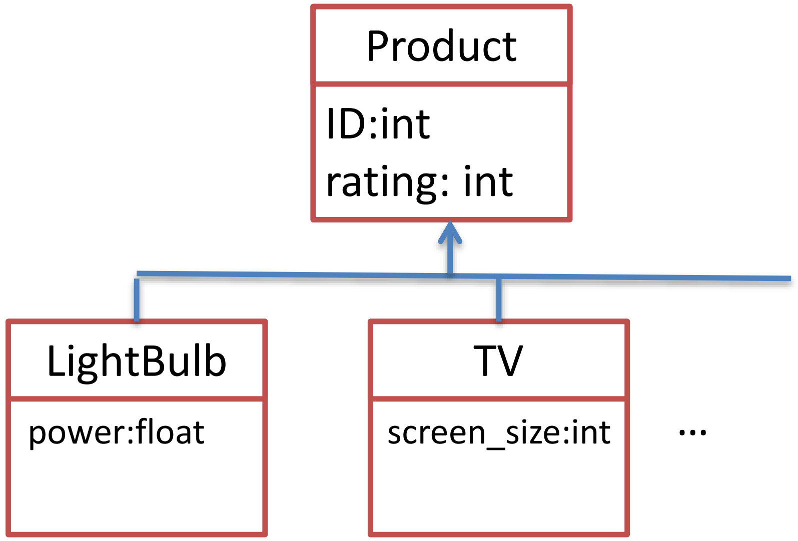


SQL Tables:
Product(ID, rating, power, screen_size,…)

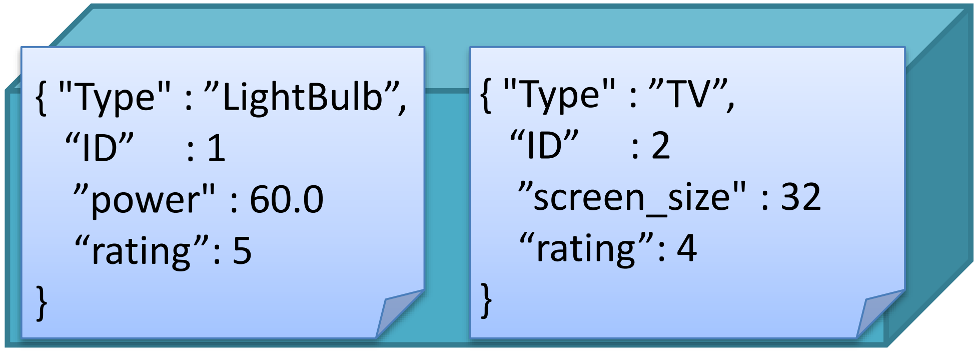
Solution 2 for keeping a product catalog in SQL

Use Case: Keeping a Product Catalog (iii)

- MongoDB Solution:
  - Keep a collection (analog of SQL table in MongoDB) “Product” and insert all document products there
  - No nulls
  - Just one collection



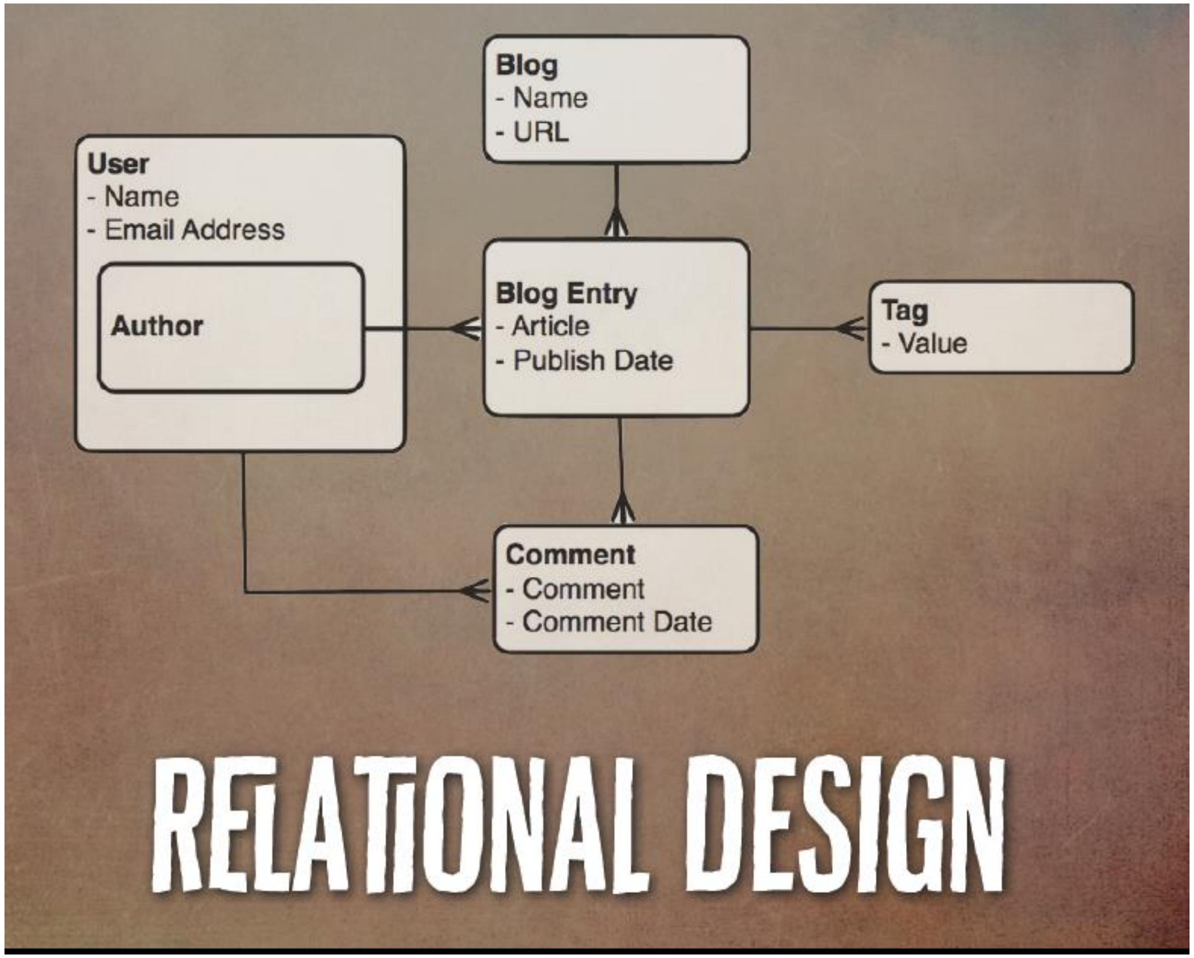
MongoDB collection:
Product



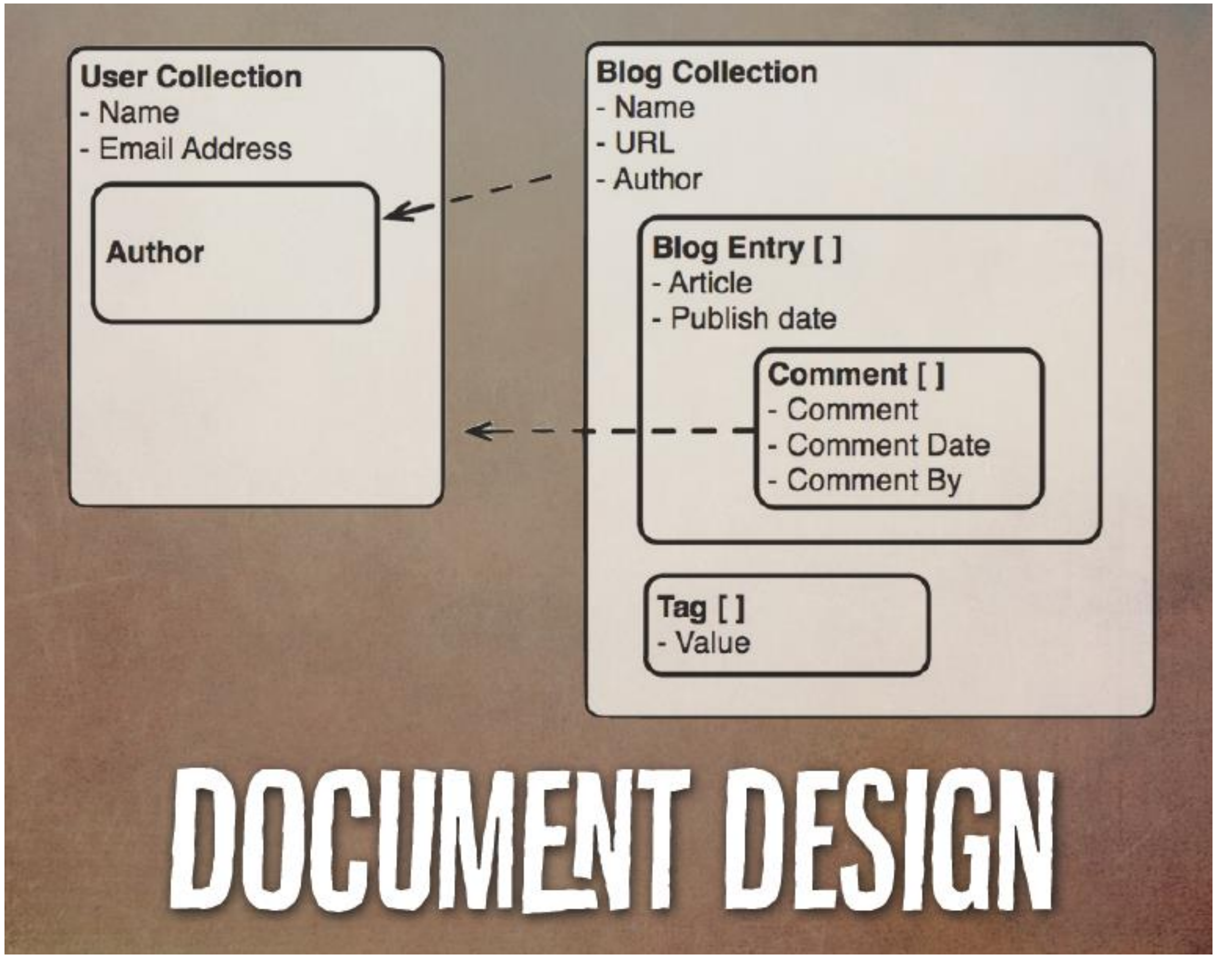
```
{ "Type" : ”LightBulb”,
  “ID”    : 1
  ”power" : 60.0
  “rating”: 5
}
```

```
{ "Type" : ”TV”,
  “ID”    : 2
  ”screen_size" : 32
  “rating”: 4
}
```

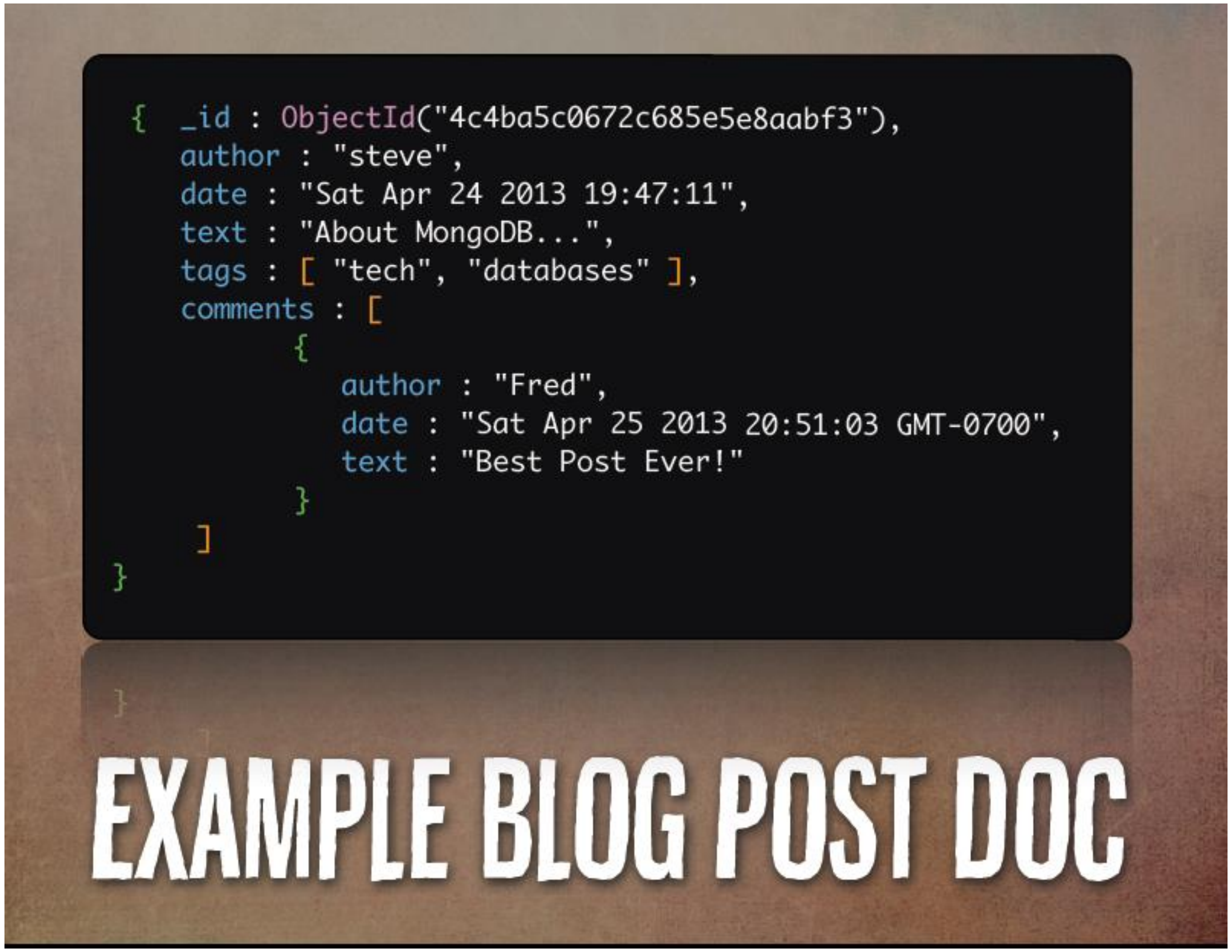
Use Case: Blog Application (i)



Use Case: Blog Application (ii)



Use Case: Blob Application (iii)



```
{  _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
   author : "steve",
   date : "Sat Apr 24 2013 19:47:11",
   text : "About MongoDB...",
   tags : [ "tech", "databases" ],
   comments : [
          {
             author : "Fred",
             date : "Sat Apr 25 2013 20:51:03 GMT-0700",
             text : "Best Post Ever!"
          }
     ]
}
```

EXAMPLE BLOG POST DOC

RUNNING MONGODB ON OU’S SYSTEM

GPEL Machines Accounts

- To use mongoDB in OU’s system, you need an account in the gpel machines in the School of Computer Science.
- Contact the CS system administrator Mr. John Mueller (jmueller@ou.edu) to obtain an account in these machines (if you haven’t done so yet).

Launching a MongoDB Server

- Assume that you want to launch a MongoDB server running at gpel9.cs.ou.edu port 27105

1. Connect to gpel9.cs.ou.edu

```
$ ssh your_username@gpel9.cs.ou.edu
```

2. From gpel9, create an empty directory for the database files

```
gpel9$ mkdir -p ~/data/db
```

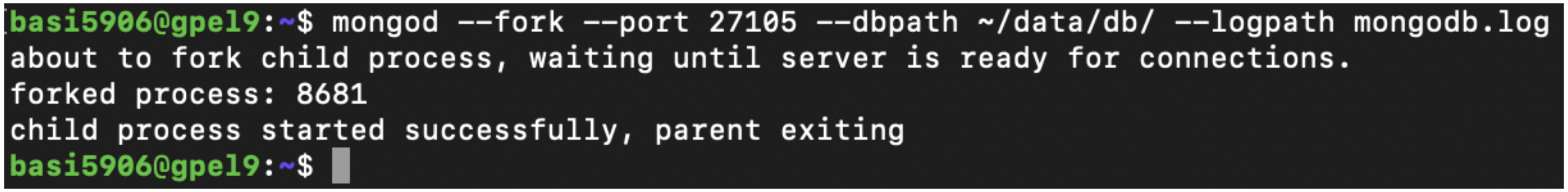
Launching a MongoDB Server (ii)

3. Change the ownership of the folders

```
gpel9$ chown your_username ~/data/db/
```

4. Launch the mongoDB server

```
gpel9$ mongod --fork --port 27105 --dbpath ~/data/db/ --logpath mongodb.log
```



```
basi5906@gpel9:~$ mongod --fork --port 27105 --dbpath ~/data/db/ --logpath mongodb.log
about to fork child process, waiting until server is ready for connections.
forked process: 8681
child process started successfully, parent exiting
basi5906@gpel9:~$
```

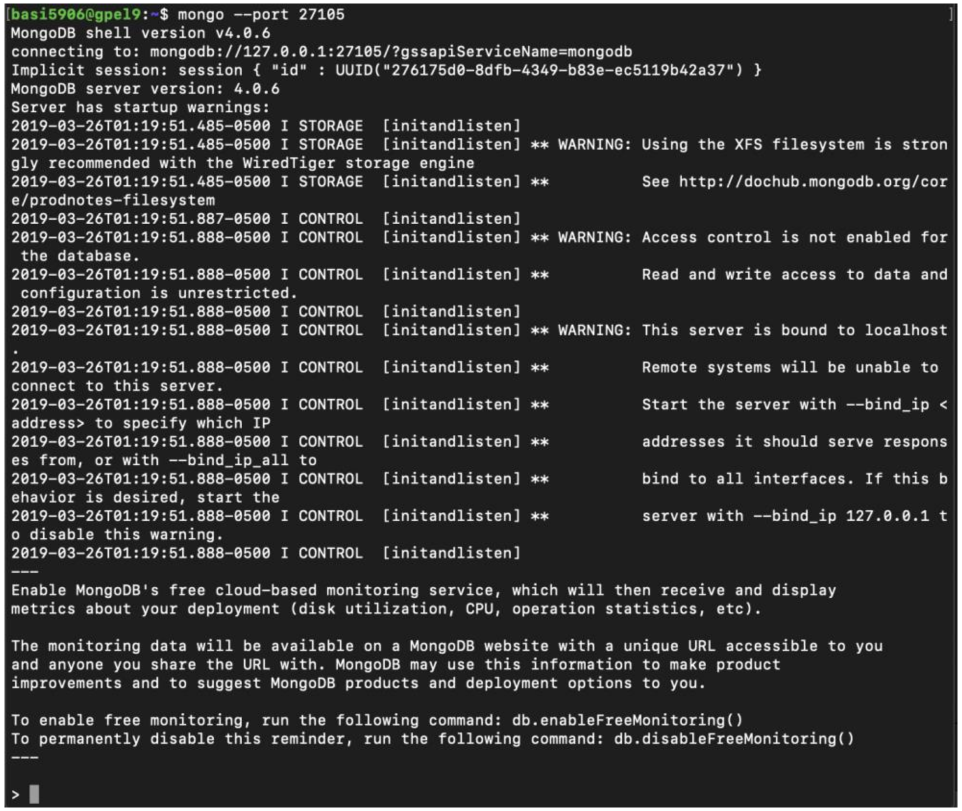
5. To stop a mongoDB server

```
gpel9$ mongod --shutdown --port 27105 --dbpath ~/data/db/
```

Connecting to a MongoDB Server

1. To connect to a server running locally type

```
$ mongo --port 27105
```

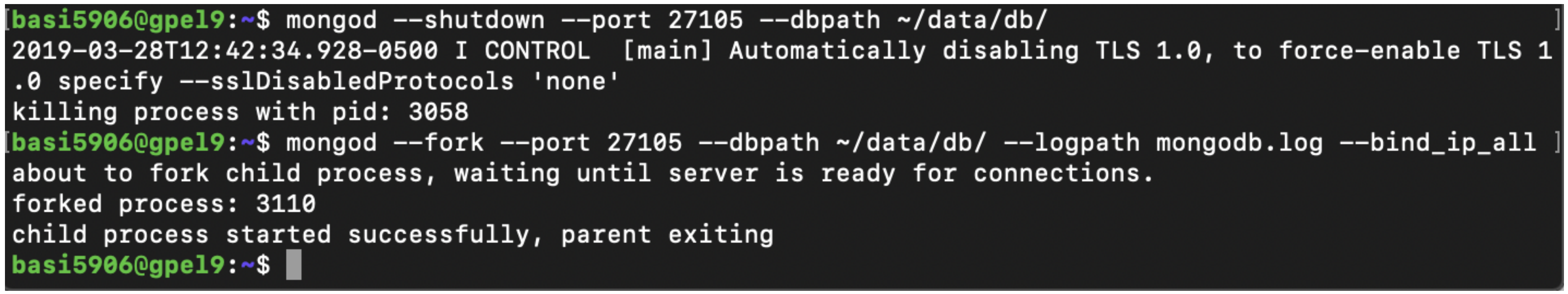


```
basi5906@gpel9:~$ mongo --port 27105
MongoDB shell version v4.0.6
connecting to: mongodb://127.0.0.1:27105/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("276175d0-8dfb-4349-b83e-ec5119b42a37") }
MongoDB server version: 4.0.6
Server has startup warnings:
2019-03-26T01:19:51.485-0500 I STORAGE  [initandlisten]
2019-03-26T01:19:51.485-0500 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is stron
gly recommended with the WiredTiger storage engine
2019-03-26T01:19:51.485-0500 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/cor
e/prodnotes-filesystem
2019-03-26T01:19:51.887-0500 I CONTROL  [initandlisten]
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for
 the database.
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          Read and write access to data and
 configuration is unrestricted.
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten]
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost
.
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          Remote systems will be unable to
connect to this server.
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          Start the server with --bind_ip <
address> to specify which IP
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          addresses it should serve respons
es from, or with --bind_ip_all to
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          bind to all interfaces. If this b
ehavior is desired, start the
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 t
o disable this warning.
2019-03-26T01:19:51.888-0500 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

Connecting to a MongoDB Server

2. To allow remote connections, re-launch the local server with --bind_ip_all options

```
$ mongod --shutdown --port 27105 --dbpath ~/data/db/
$ mongod --fork --port 27105 --dbpath ~/data/db/ --logpath
mongodb.log --bind_ip_all
```

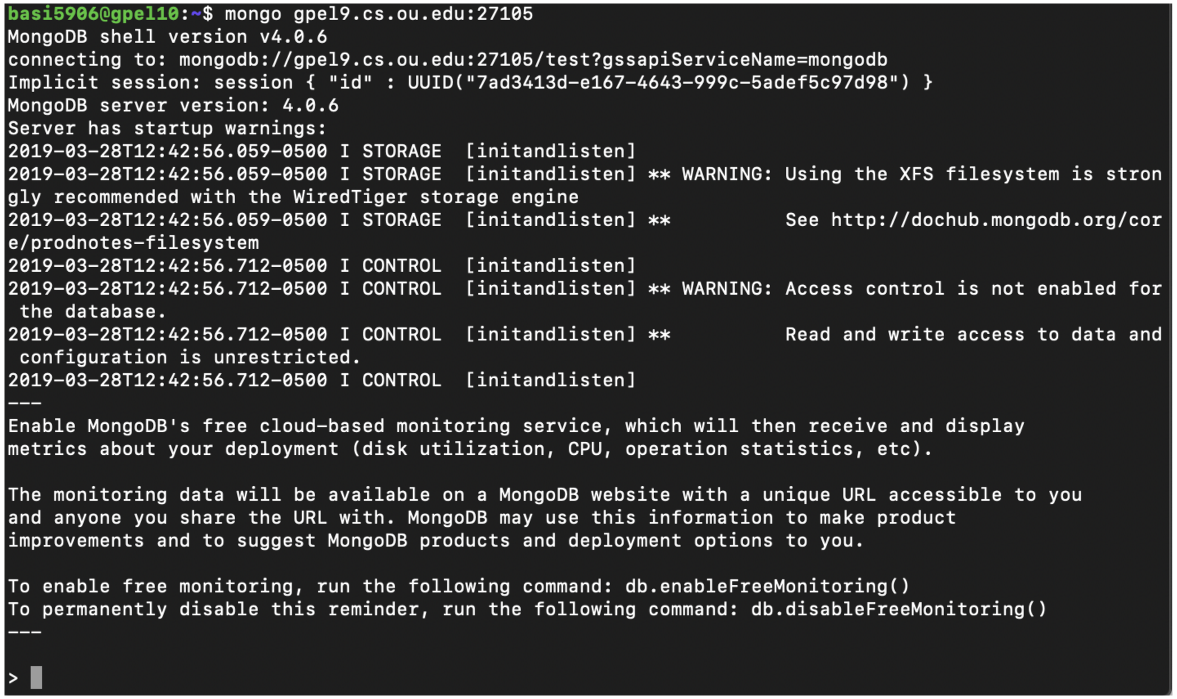


```
basi5906@gpel9:~$ mongod --shutdown --port 27105 --dbpath ~/data/db/
2019-03-28T12:42:34.928-0500 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1
.0 specify --sslDisabledProtocols 'none'
killing process with pid: 3058
basi5906@gpel9:~$ mongod --fork --port 27105 --dbpath ~/data/db/ --logpath mongodb.log --bind_ip_all
about to fork child process, waiting until server is ready for connections.
forked process: 3110
child process started successfully, parent exiting
basi5906@gpel9:~$
```

Connecting to a MongoDB Server

3. Finally, to connect to a database server remotely (from gpel10.cs.ou.edu machine in this example)

```
$ mongo gpel9.cs.ou.edu:27105
```



```
basi5906@gpel10:~$ mongo gpel9.cs.ou.edu:27105
MongoDB shell version v4.0.6
connecting to: mongodb://gpel9.cs.ou.edu:27105/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("7ad3413d-e167-4643-999c-5adef5c97d98") }
MongoDB server version: 4.0.6
Server has startup warnings:
2019-03-28T12:42:56.059-0500 I STORAGE  [initandlisten]
2019-03-28T12:42:56.059-0500 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is stron
gly recommended with the WiredTiger storage engine
2019-03-28T12:42:56.059-0500 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/cor
e/prodnotes-filesystem
2019-03-28T12:42:56.712-0500 I CONTROL  [initandlisten]
2019-03-28T12:42:56.712-0500 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for
 the database.
2019-03-28T12:42:56.712-0500 I CONTROL  [initandlisten] **          Read and write access to data and
 configuration is unrestricted.
2019-03-28T12:42:56.712-0500 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

Running a Script in MongoDB

- Suppose that we want to use a script named ‘myscript.txt’ to insert two records into the collection ‘people’ belonging to the database ‘mydb’. Also suppose the database is running in gpel9.cs.ou.edu:27105.

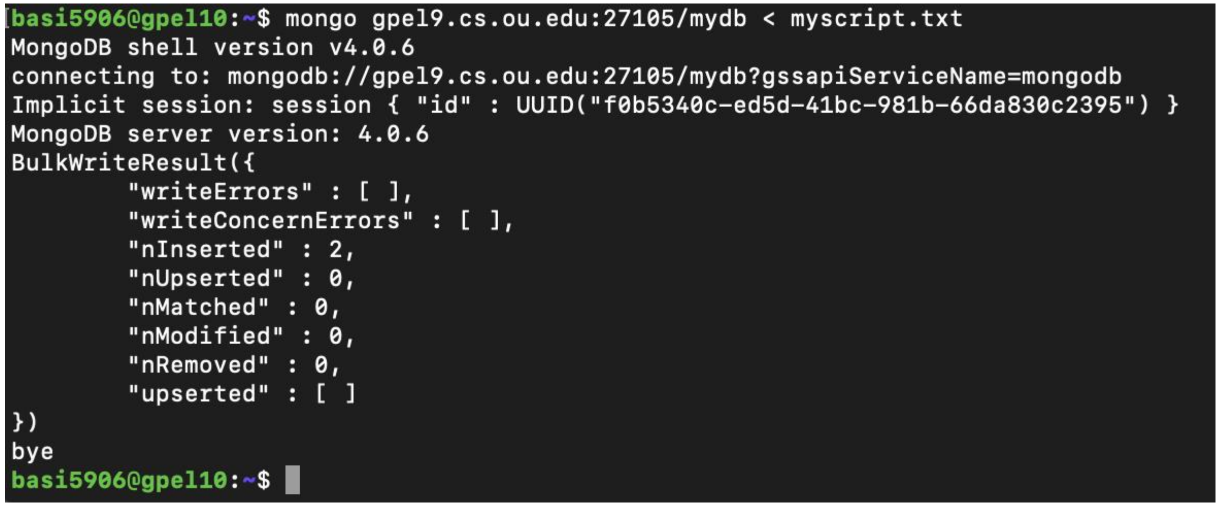
1. Type the following text in a new file named ‘myscript.txt’

```
db.people.insert([
	{"firstname": "Don", "lastname": "Tillman", "age" : 30},
	{"firstname": "David", "lastname": "Boren", "age" : 74}
]);
```

Running a Script in MongoDB (ii)

2. Then, to run the commands in ‘myscript.txt’ in mongo, type the following in your terminal (not in the mongo shell)

```
gpel10: ~ $ mongo  gpel9.cs.ou.edu:27105/mydb < myscript.txt
```

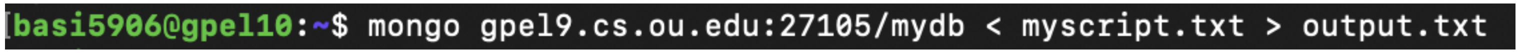


```
basi5906@gpel10:~$ mongo gpel9.cs.ou.edu:27105/mydb < myscript.txt
MongoDB shell version v4.0.6
connecting to: mongodb://gpel9.cs.ou.edu:27105/mydb?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f0b5340c-ed5d-41bc-981b-66da830c2395") }
MongoDB server version: 4.0.6
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
bye
basi5906@gpel10:~$
```

Output to File in MongoDB

- To output the complete mongodb session generated by running the script ‘myscript.txt’ to a file, type the following command in your terminal

```
gpel10: ~ $ mongo gpel9.cs.ou.edu:27105/mydb < myscript.txt > output.txt
```

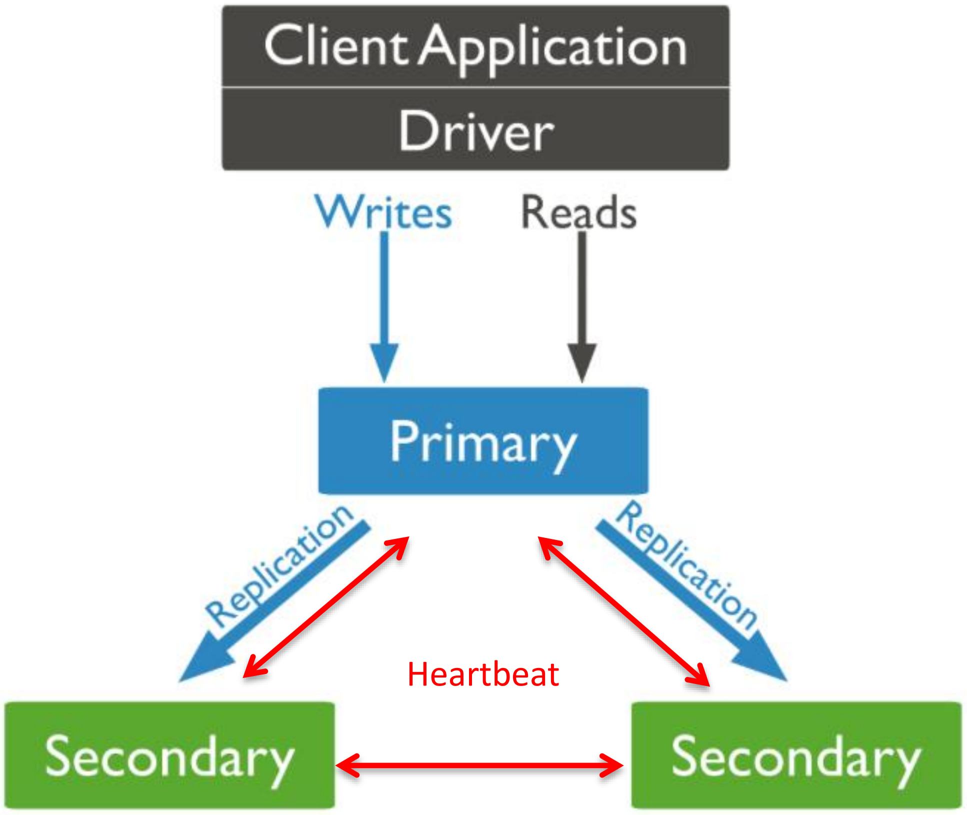


```
basi5906@gpel10:~$ mongo gpel9.cs.ou.edu:27105/mydb < myscript.txt > output.txt
```

MONGODB’S DISTRIBUTED SYSTEMS CHARACTERISTICS AND USING THEM ON OU’S GPEL MACHINES

Replication in MongoDB

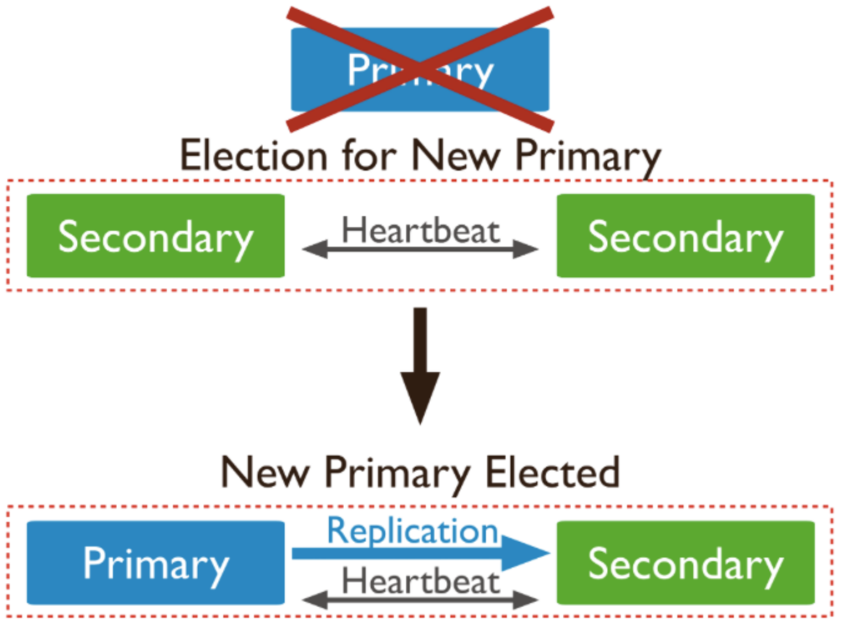
- Replica sets: Primary, Secondaries, Arbiters
- At most one primary per replica set
- **Writes → Primary** nodes
- **Reads →** Primary or **Secondary** nodes
- Replication → Asynchronous



Replica Set

Figure adapted from [Mongo]

Replication in MongoDB (ii)



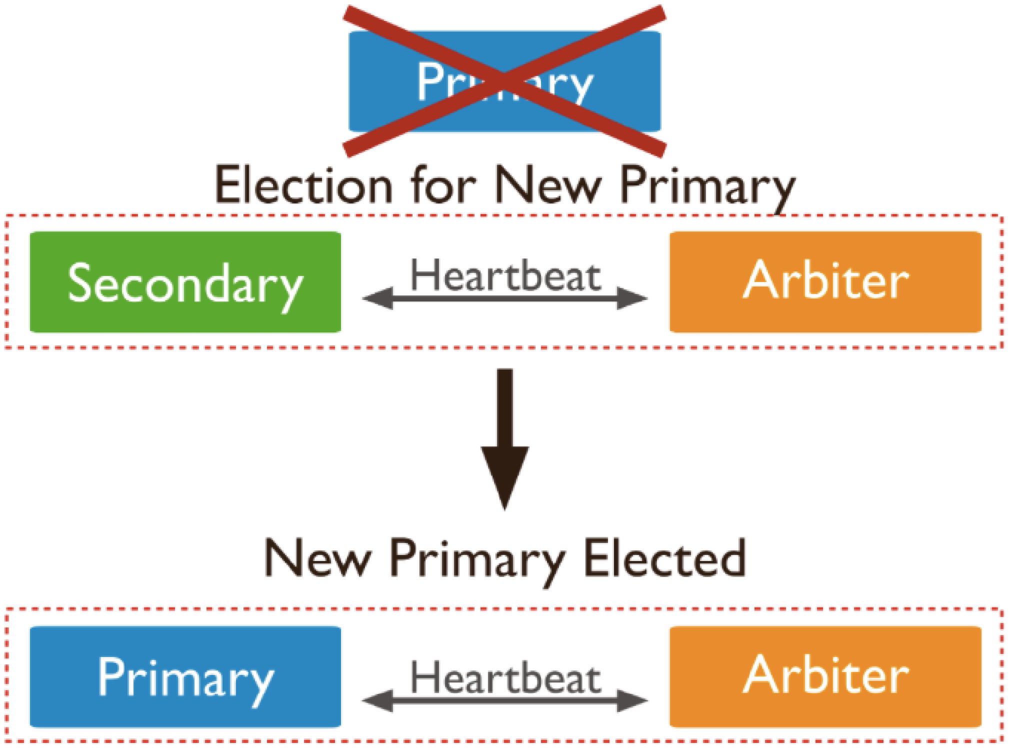
Failure of Primary Node and Elections

- In case of failure of primary → **majority voting**, where secondaries and arbiters vote
- Replica sets → odd number of nodes (add arbiter if needed)
- Arbiters → to elect a primary after network partitioning
- Arbiters do not contain data
- Secondaries can become primaries, but arbiters can only vote and can't become primaries.

Figure from [Mongo]

Replication in MongoDB (iii)

- Arbiter needed in case there is just 1 primary and 1 secondary



Failure of Primary Node and Elections with Arbiter

Implementing Replication in MongoDB in OU’s gpel Machines (i)

1. Suppose we have 3 machines gpel9.cs.ou.edu, gpel10.cs.ou.edu and gpel11.cs.ou.edu. Suppose you want to replicate the database “myDB”. In the following, replace ‘#’ with the machine’s number.
2. In each machine gpel#, create a file

```
gpel#: ~ $ mkdir -p /home/user/data/gpel#
```

3. In each machine gpel#, run the mongod server using the command to create the replica set “myDBReplicaSet” (we allow un-authenticated connections for next examples)

```
gpel#: ~ $ mongod --fork --dbpath ~/data/gpel#/ --port 2710# --logpath
mongodb.log --replSet myDBReplicaSet --bind_ip_all
```

Implementing Replication in MongoDB in OU's gpel Machines (ii)

4. Assuming gpel9 is your primary, connect to its mongo instance

```
gpel9: ~ $ mongo –host gpel9.cs.ou.edu --port 27109 myDB
```

5. In mongoDB console (at gpel9), initiate the replica set

```
> rs.initiate()
```

6. Then, add the secondaries gpel10 and gpel11

```
myDBReplicaSet:PRIMARY> rs.add("gpel10.cs.ou.edu:27110")
myDBReplicaSet:PRIMARY> rs.add("gpel11.cs.ou.edu:27111")
```

Implementing Replication in MongoDB in OU's gpel Machines (iii)

7. Check the replica set status with the command

```
>use admin
>rs.status()
```

Other commands related to replication that you can check are:

- rs.stepDown() → to take a primary out of the replica set and force the election of a new primary
- rs.addArb(host) → add an arbiter
- db.isMaster() → check if it is primary

Connecting to a Replica Set in MongoDB

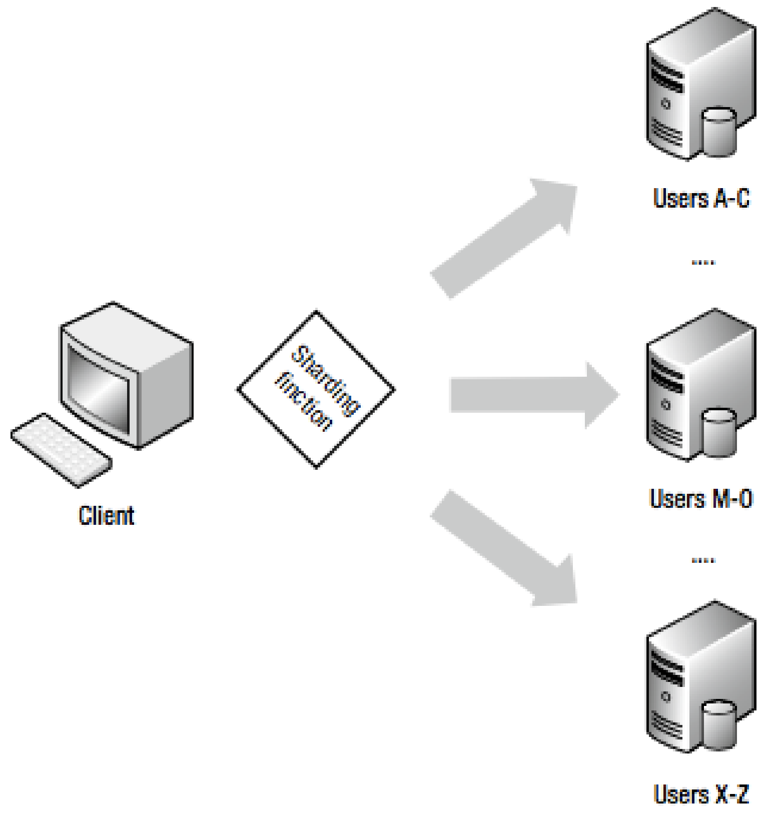
- Once configured your “myDBReplicaSet” replica set, you can connect to it by

```
$ mongo –host
‘myDBReplicaSet/gpel9.cs.ou.edu:27109,gpel10.cs.ou.edu:27110,gpel11.cs.ou.e
du:27111’
```

- By default, all queries to a replica set are sent to the primary. To change that, change the “read preferences.” For details see:

https://docs.mongodb.org/manual/core/read-preference/

Sharding in MongoDB



- Sharding→ horizontal partitioning on a special attribute called "shard key"
- The **shard key** is an attribute that all documents in the sharded collection must have, and there must be an index on this attribute
- Sharding is done at the collection level

Figure from [HPMH15]

Sharding in MongoDB (ii)

- Queries are sent to the shard controller
- The **config server** is aware of the sharding configuration
- The **shard controller** sends queries to the shard servers
- The **shard controller aggregates** query **results** from the shard servers

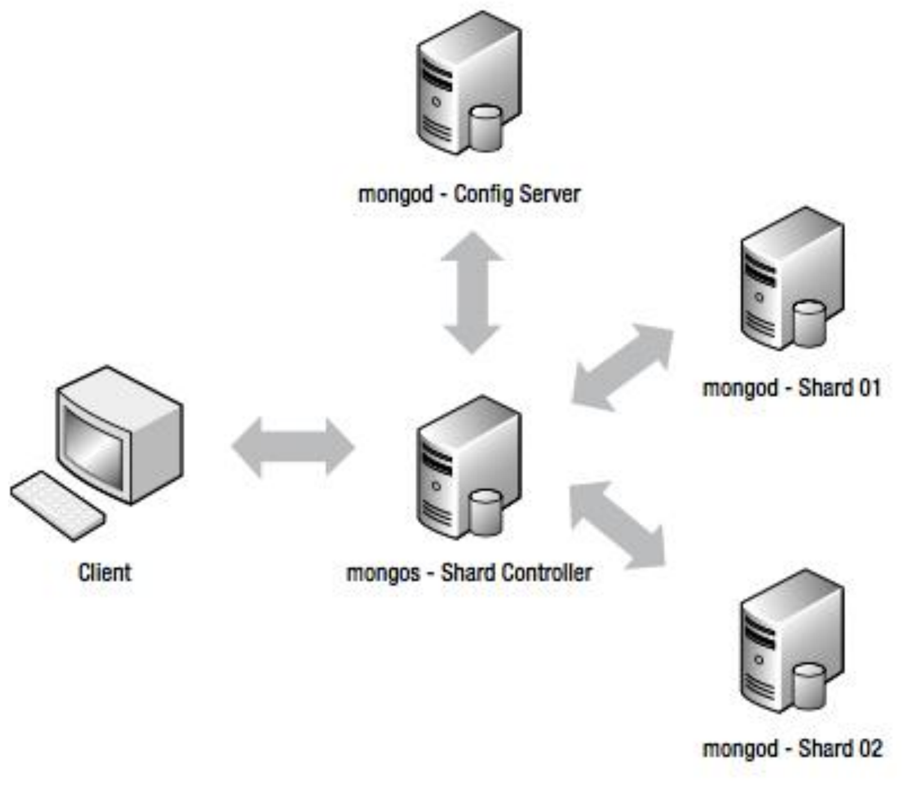


Figure from [HPMH15]

Sharding in MongoDB in OU’s gpel Machines (i)

- Assume we want to shard the collection “people” in the replicated database “myDB” using “_id” as shard key, and that we want
  - gpel9.cs.ou.edu -→ shard controller (on port 27000) and config server (on port 27009)
  - gpel10.cs.ou.edu → shard server 1 (port 27010)
  - gpel11.cs.ou.edu → shard server 2 (port 27011)

1. Bring up the gpel9 config server (we assume the replica set is already initiated)

```
gpel9: ~ $ mongod --fork --port 27009 --replSet configSrvRS --dbpath
~/data/config/ --logpath mongodb.log --configsvr --bind_ip_all
```

Sharding in MongoDB in OU's gpel Machines (ii)

2. Bring up the shard controller in gpel9 with (default) chunk size for sharding of 64MB

```
gpel9: ~ $ mongos --fork --configdb configSrvRS/gpel9.cs.ou.edu:27009 --port
27000 --logpath mongos.log
```

3. Launch mongoDB in these servers gpel10, gpel11 to prepare them to become shard servers in step 4

```
gpel10: ~ $ mongod --fork --port 27010 --dbpath ~/data/gpel10shard/ --logpath
mongodb.log --shardsvr --bind_ip_all
```

```
gpel11: ~ $ mongod --fork --port 27011 --dbpath ~/data/gpel11shard/ --logpath
mongodb.log --shardsvr --bind_ip_all
```

Sharding in MongoDB in OU’s gpel Machines (iii)

4. Then add the shard servers to the shard controller

```
gpel9: ~ $ mongo --port 27000
mongos> sh.addShard(“gpel10.cs.ou.edu:27010”)
mongos> sh.addShard(“gpel11.cs.ou.edu:27011”)
```

5. Check the sharding configuration

```
mongos> db.printShardingStatus()
```

6. Tell the shard controller about the database and the collection that you want to shard

```
gpel9: ~ $ mongo –port 27000 myDB
> sh.enableSharding(“myDB”)
> sh.shardCollection(“myDB.people”, {“_id” : 1})
```

Sharding in MongoDB in OU’s gpel Machines (iv)

7. Connect to the shard servers and periodically check how the number of documents changes over time as the sharding system rebalances the data across the shard servers

```
$ mongo gpel10.cs.ou.edu:27010
> use myDB
> db.people.count()

$ mongo gpel11.cs.ou.edu:27011
> use myDB
> db.people.count()
```

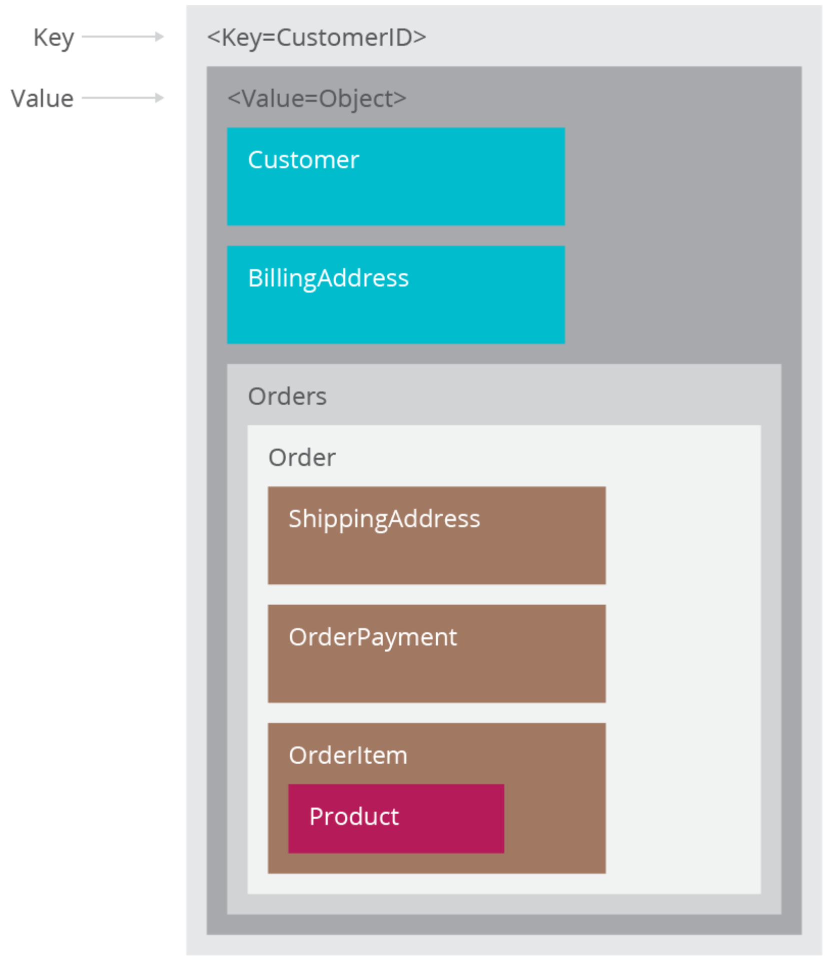
NOSQL SYSTEMS: KEY-VALUE STORES

Key-Value Stores Features

- Example systems: Project Voldemort, Memcached
- Data model: (key, value) pairs
  - A single key-value index for all the data
- Operations
  - Insert, delete, and lookup operations on keys
- Distribution / Partitioning
  - Distribute keys across different nodes
- Other features
  - Versioning
  - Sorting

Key Value Stores

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)



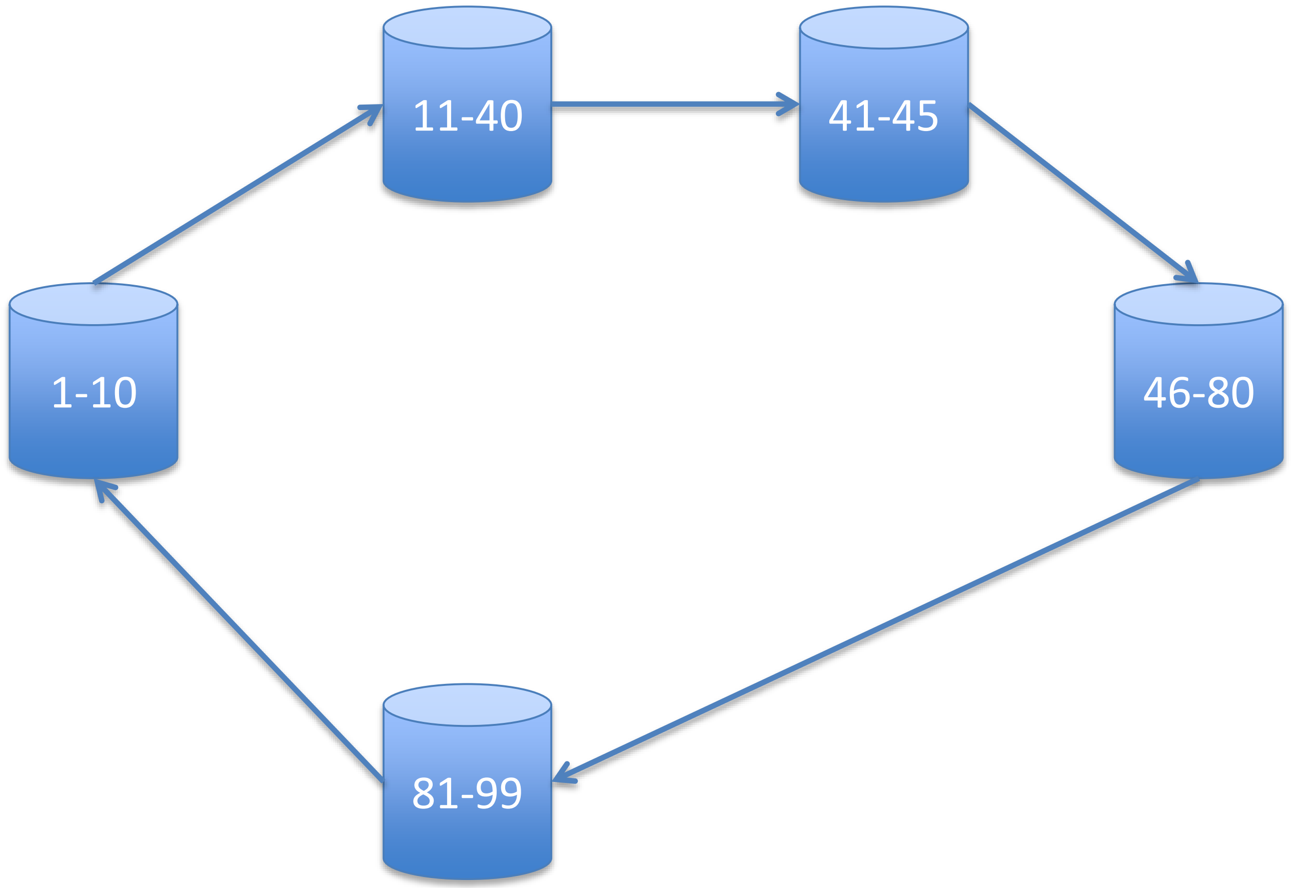
Key-Value Stores Internals

- Data remains in main memory
- One type of implementation: distributed hash table (DHT)
- Most systems also offer a persistence option
- Others use replication to provide fault-tolerance
  - Asynchronous or synchronous replication
  - Tunable consistency: read/write one replica or majority
- Some offer ACID transactions, others do not
- Multiversion concurrency control or locking

Distributed Hash Tables

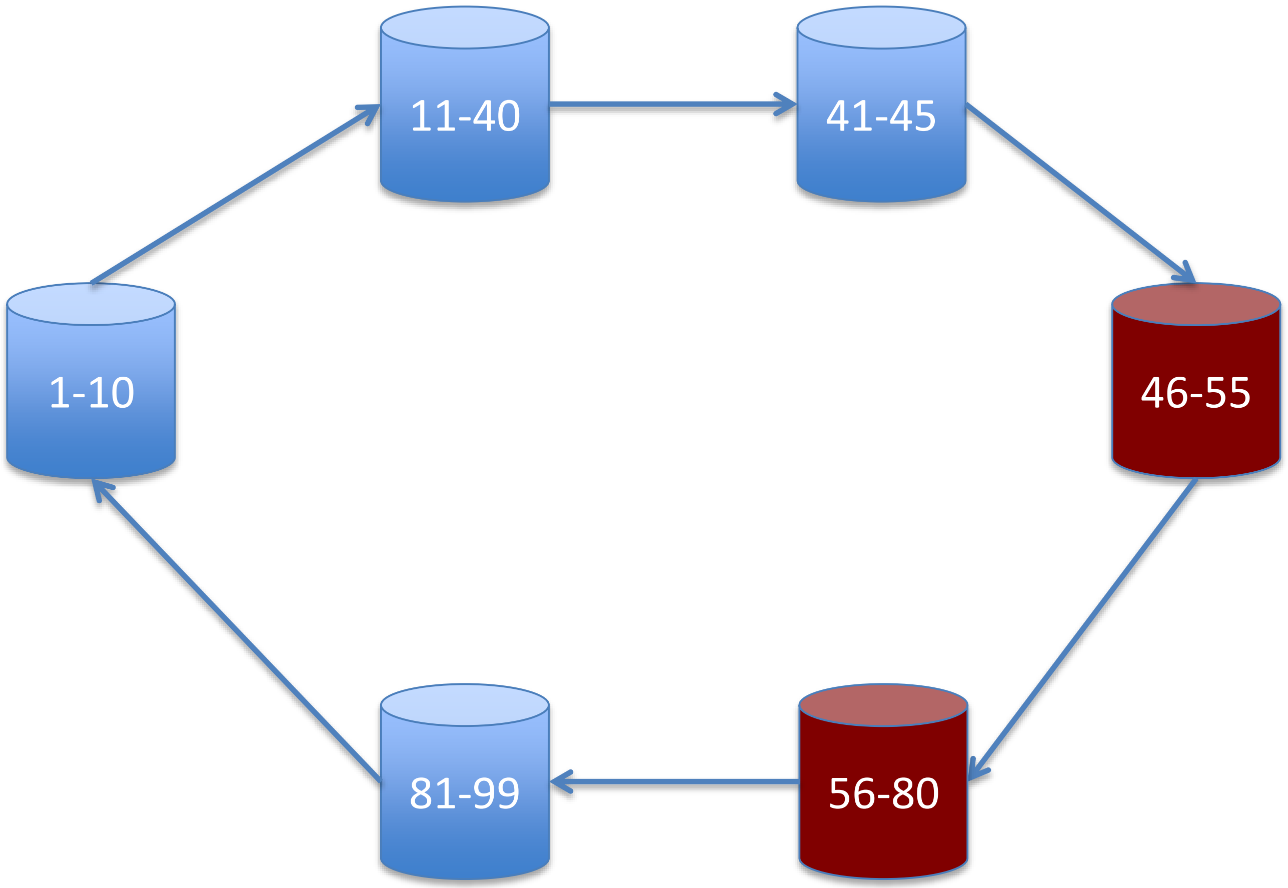
- Simple interface
  - get(key)
  - put(key, value)   // two flavors: insert and/or replace
- Principles
  - consistent hashing, allows to extend / shrink "ring"
  - "finger tables": support logarithmic access times
  - chain replication: support fault tolerance
- Examples
  - late 90s (research): Chord, Pastry, …
  - mid 2000s (applied research, product): Cassandra, Dynamo, ..
- **Virtualizes storage layer: fault tolerant, elastic, fast**
  - decouples machine from "storage service"

Consistent Hashing



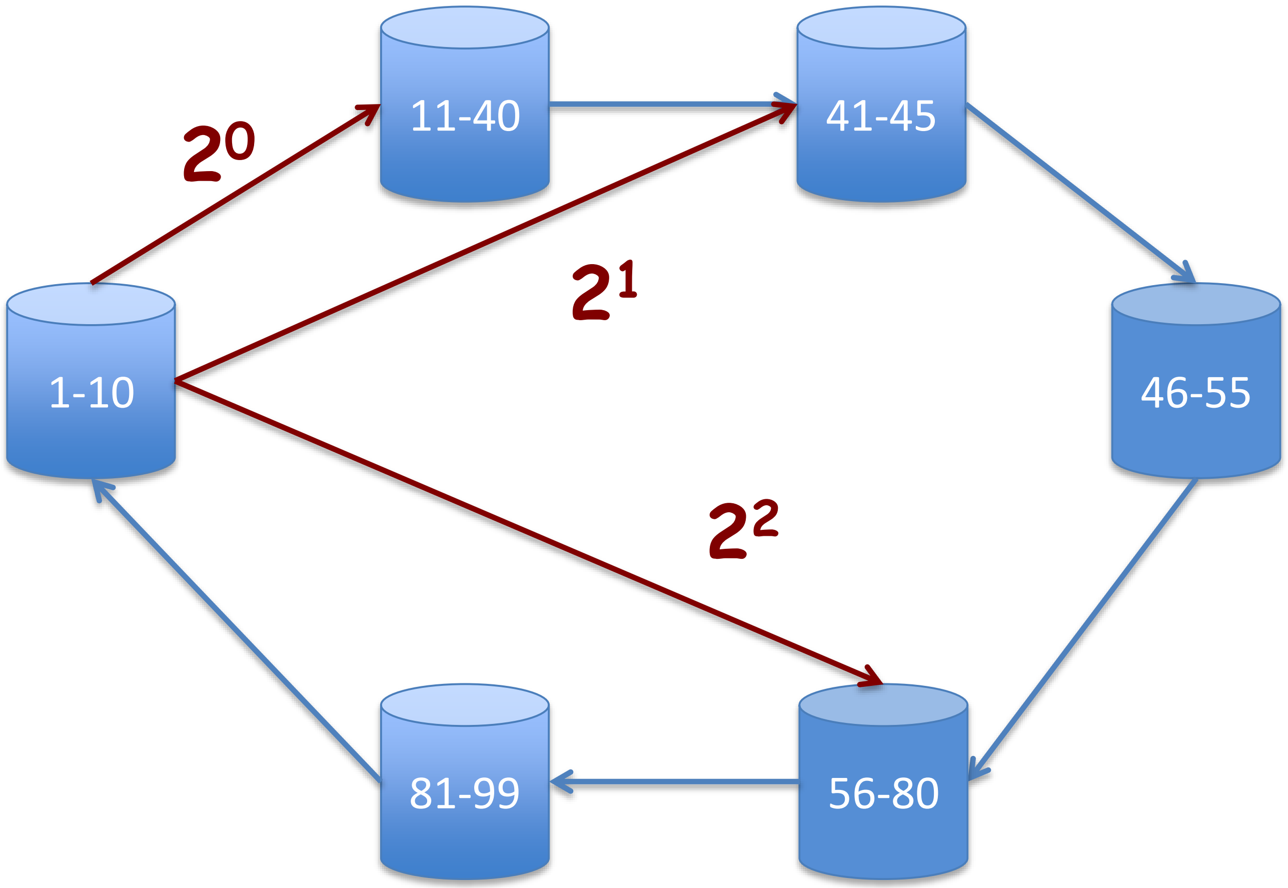
- Each machine can store several partitions
- Partitions should have even load -> machines have even load

Consistent Hashing



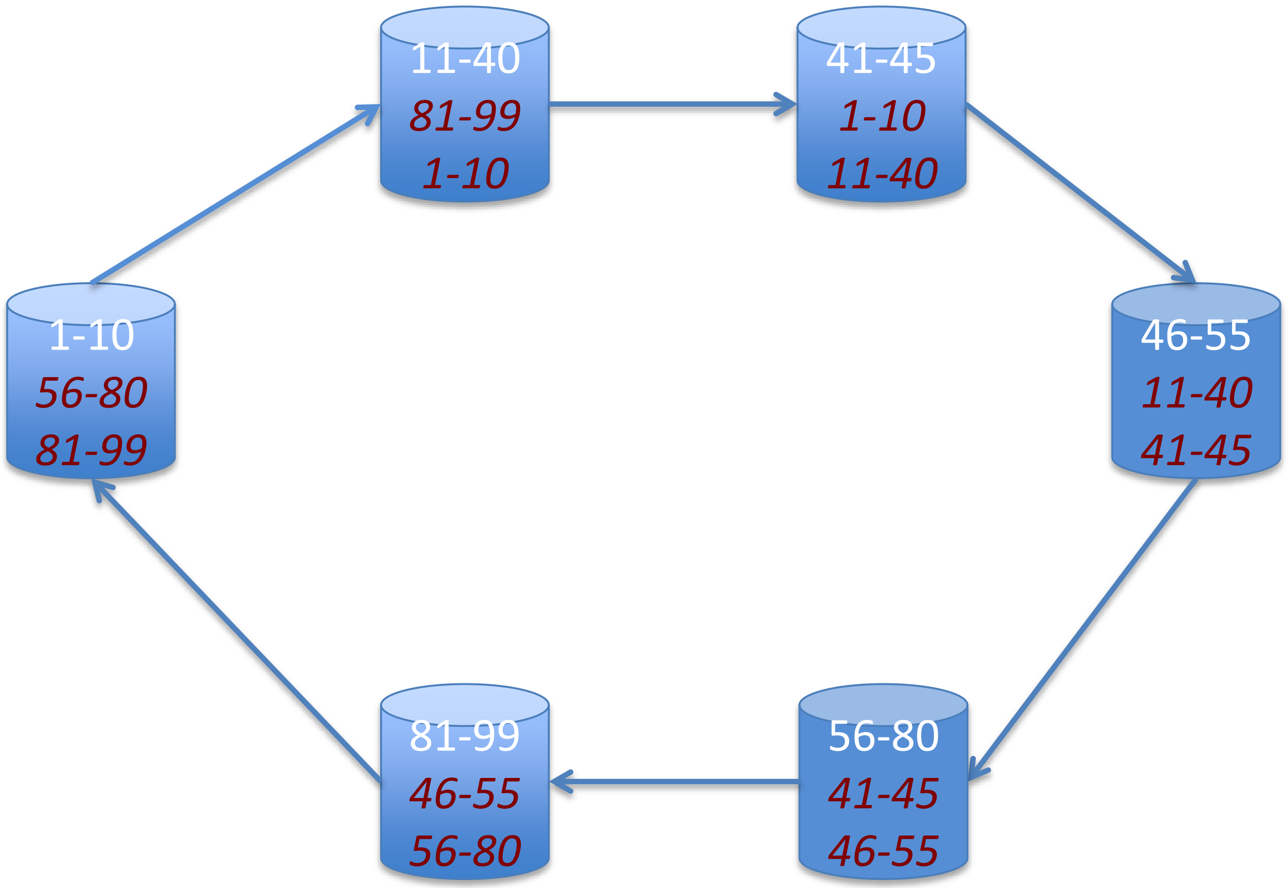
- Overload in one partition -> split that partition
- Split does not affect other partitions

Finger Tables



- Clients can send requests to any node in ring
- Nodes serve as routers within ring (logarithmic time and space)

Chain Replication



- Failure of one/two machines does not result in loss of data
- Many protocols to keep finger table, replication, etc. up to date

Why DHTs?

- They virtualize storage
  - client needs to know of only one node
- They tolerate failure in a cheap and simple way
  - e.g., no fancy technology for failure detection
- They are elastic
  - can add and remove machines at any point in time
- They are okay fast
  - logarithmic access guaranteed
  - caching at client improves situation significantly

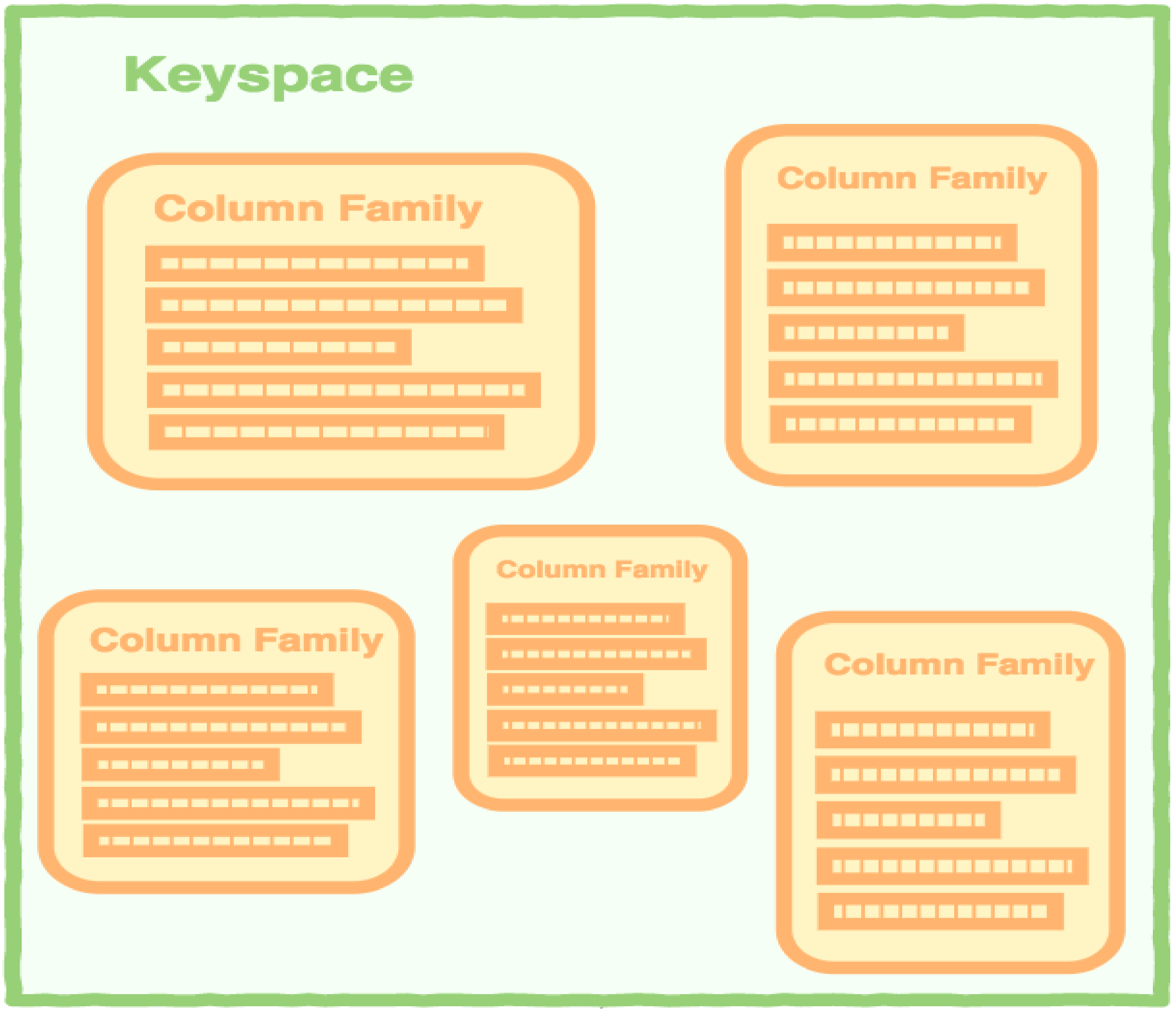
NOSQL SYSTEMS: COLUMN FAMILY STORES

Extensible Record Stores (Column Family Stores)

- Example systems: HBase, Cassandra, PNUTS
- Based on Google's BigTable
- Data model is rows and columns
- Scalability by splitting rows and columns over nodes
  – Rows partitioned through sharding on primary key
  – Columns of a table are distributed over multiple nodes by using "column groups"
- HBase is an open source implementation of BigTable

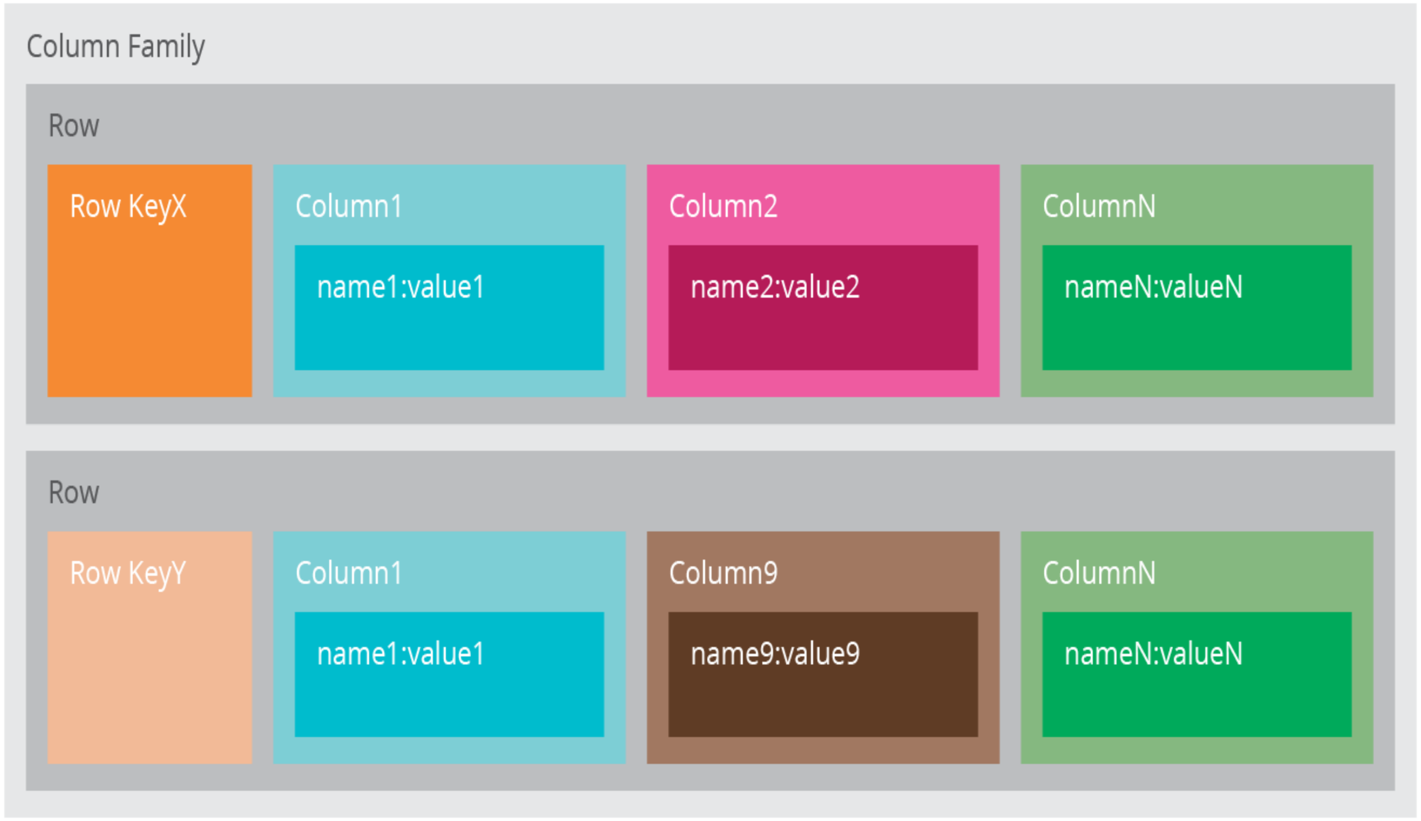
Column Family Stores

https://database.guide/what-is-a-column-store-database/



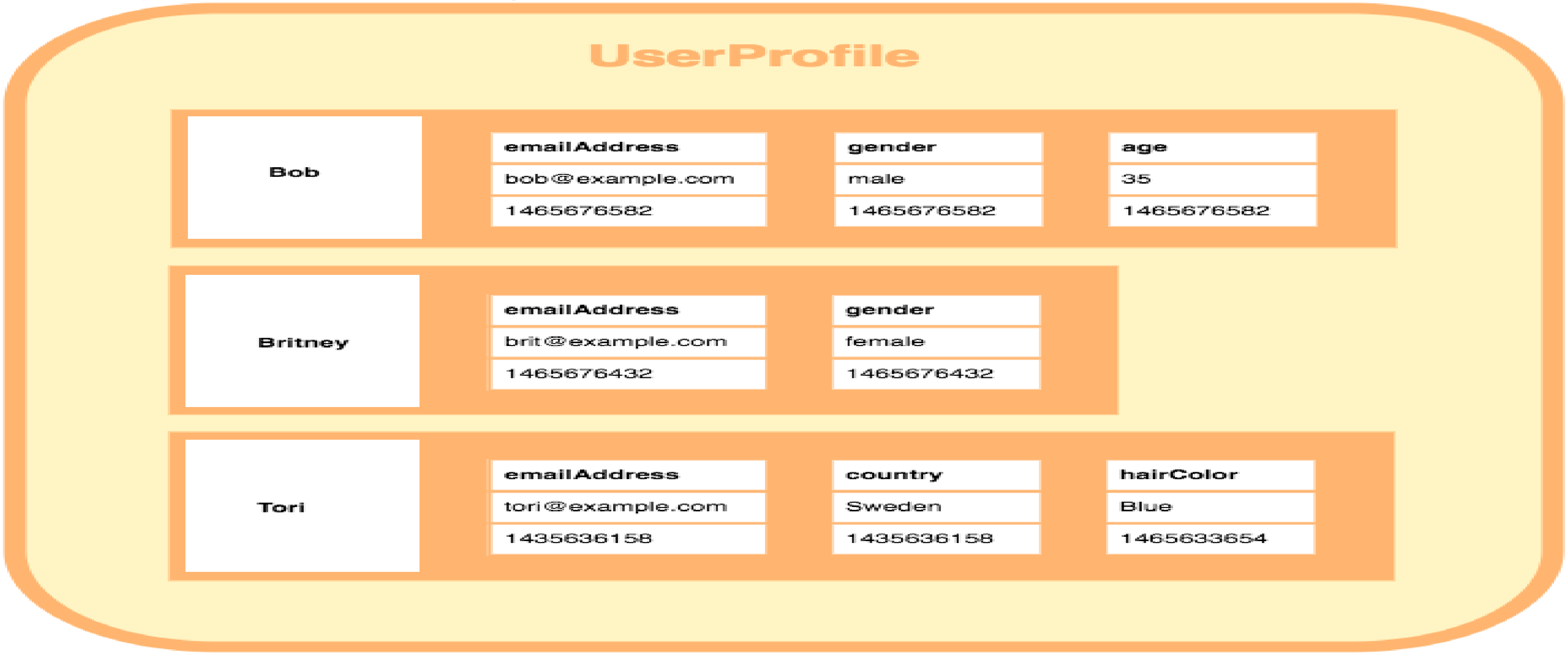
Column Family Stores

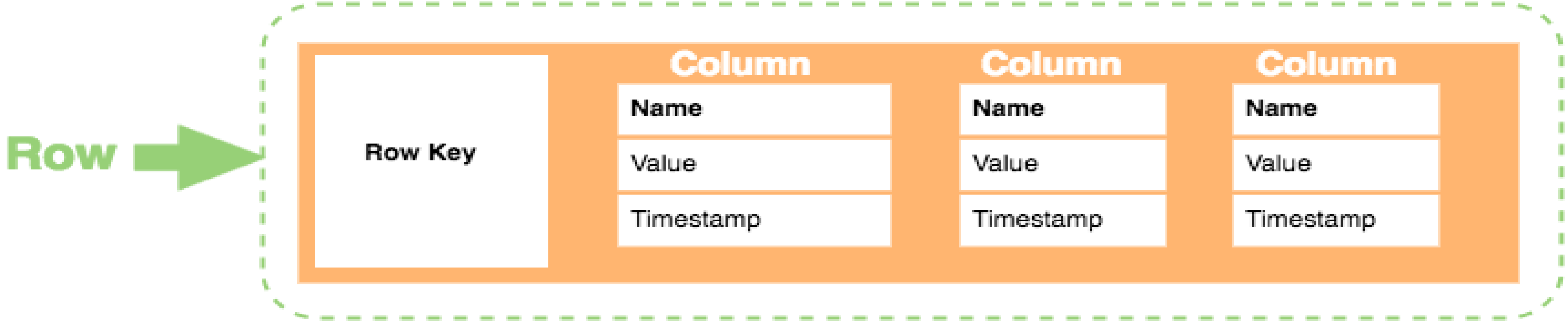
(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)



A Column Family Example

https://database.guide/what-is-a-column-store-database/





What is BigTable?

- Distributed storage system
- Designed to
  - Hold structured data
  - Scale to thousands of servers
  - Store up to several hundred TB (maybe even PB)
  - Perform backend bulk processing
  - Perform real-time data serving
- To scale, BigTable has a limited set of features

BigTable Key Features

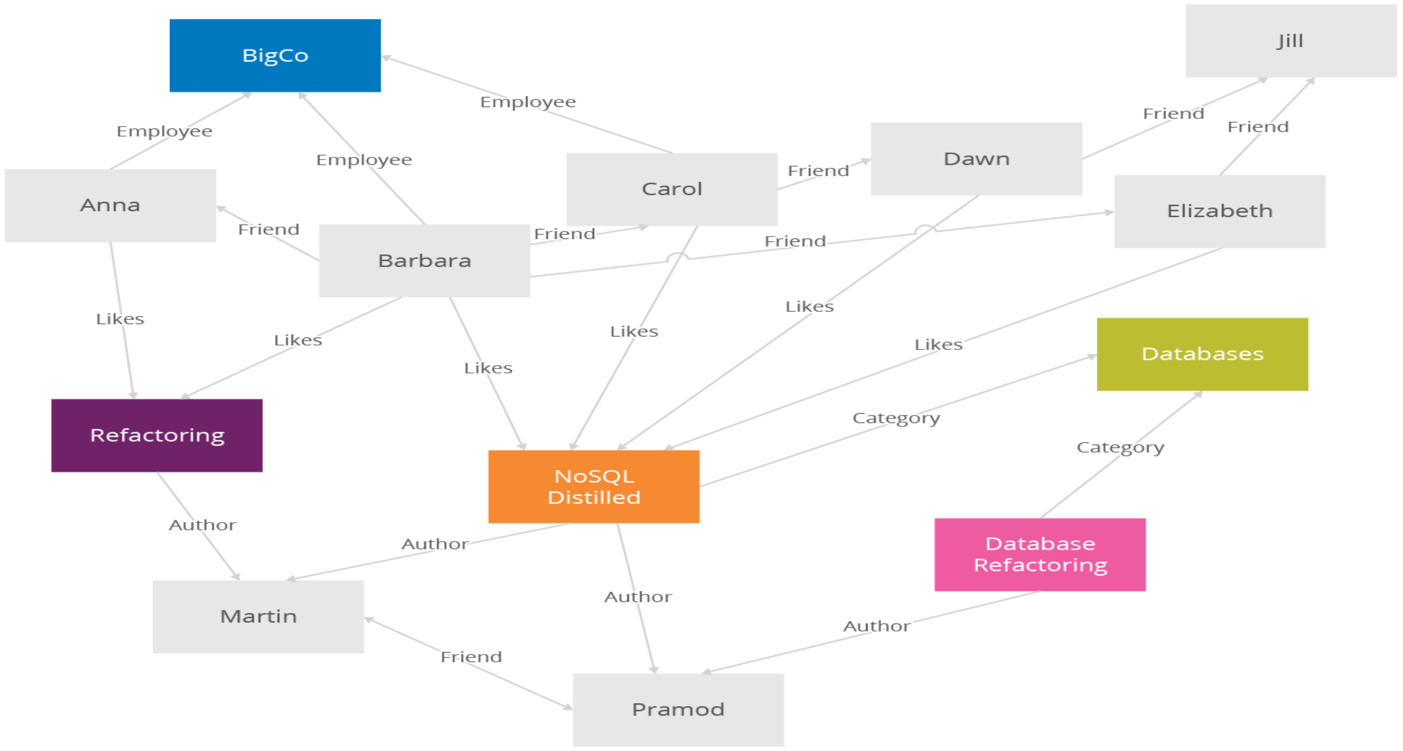
- Reads/writes of data under single row key is atomic
  - Only single-row transactions!
- Data is stored in lexicographical order
  - Improves data access locality
- Column families are unit of access control
- Data is versioned (old versions garbage collected)
  - Ex: most recent three crawls of each page, with times

NOSQL SYSTEMS: GRAPH DATABASES

Graph Databases (1/3)

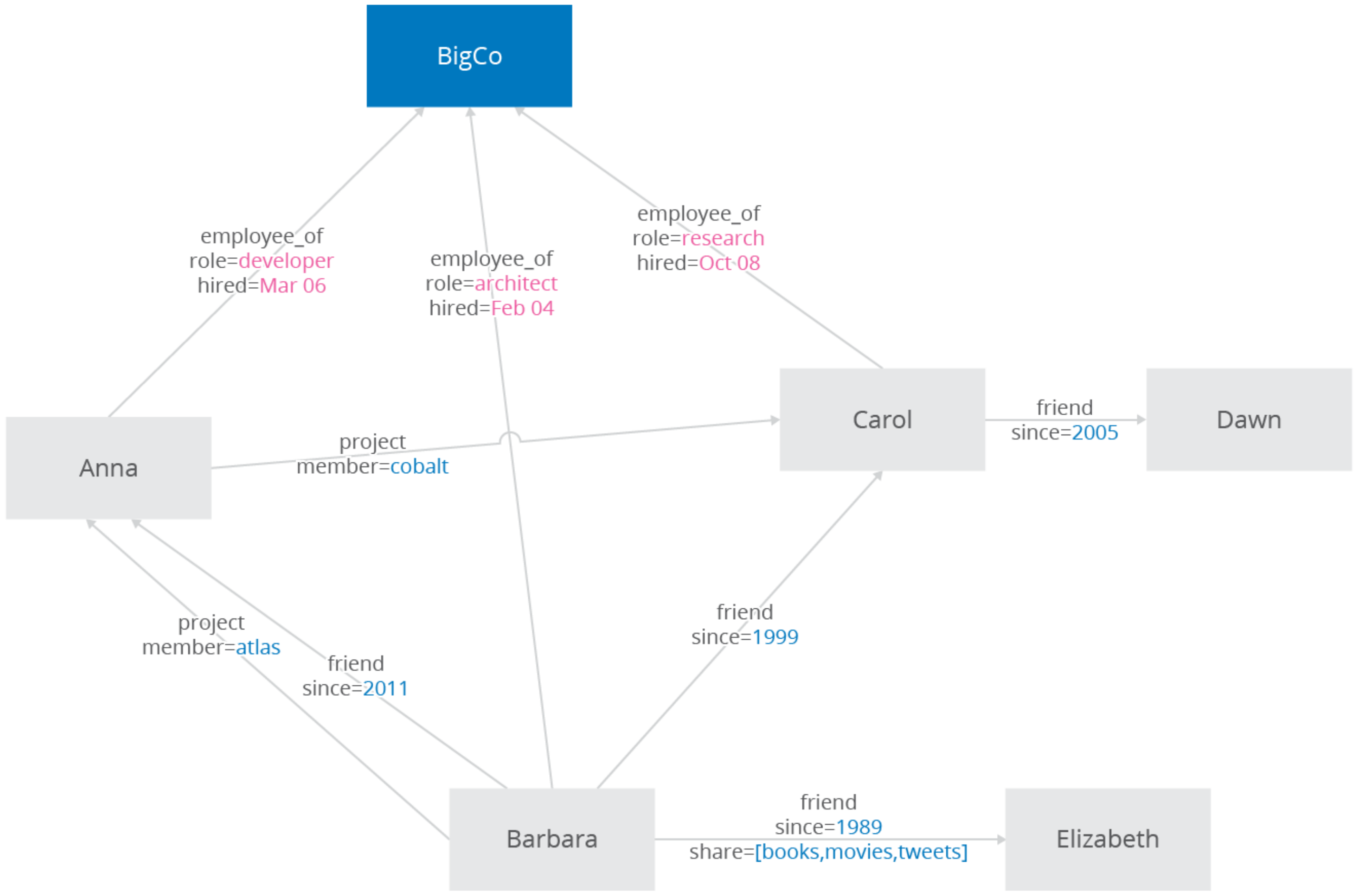
(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)

- Example systems: Neo4J, Infinite Graph, OrientDB
- Store graph-structured databases: nodes (entities) and edges (relationships) between nodes with directional significance
- Nodes have properties and edges can have properties



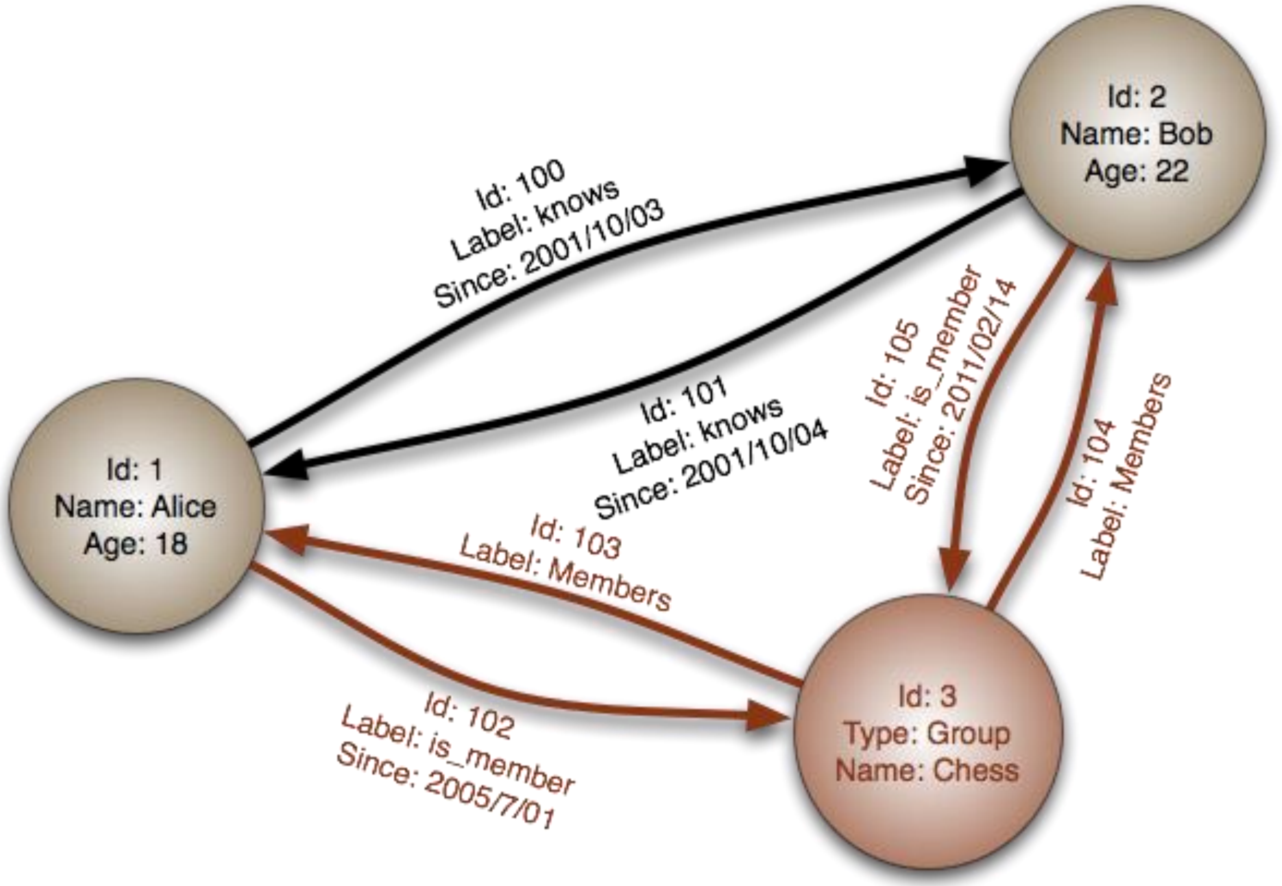
Graph Databases (2/3)

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)



Graph Databases (3/3)

(http://en.wikipedia.org/wiki/Graph_database)



CHOOSING NOSQL AND RDMBS DATABASES

Choosing NoSQL Database Guidelines (1/2)

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)

- Key-value databases:
  - Are useful for storing session info, user profiles, preferences, shopping cart data
  - Avoid using them when we need to query by data, have relationships between the data being stored, or need to operate on multiple keys at the same time
- Document databases:
  - Are useful for content management systems, blogging platforms, web analytics
  - Avoid using them for systems that need complex transactions spanning multiple operations or queries against varying aggregate structures

Choosing NoSQL Database Guidelines (2/2)

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)

- Column Family databases:
  - Are useful for data warehouses, customer relationship management (CRM) systems, maintaining counters, expiring usage
  - Avoid using them for systems that are in early development, changing query patterns
- Graph databases:
  - Are useful for problems with graph structured data, e.g. social networks, spatial data, routing info for goods and money, recommendation engines.
  - Avoid using them for problems that do not have graph structured data.

Schema-Less Ramifications

(http://www.thoughtworks.com/insights/blog/nosql-databases-overview)

- All NoSQL databases claim to be schema-less, which means there is no schema enforced by the databases themselves.
- Databases with strong schemas, such as relational databases, can be migrated by saving each schema change, plus its data migration, in a version-controlled sequence.
- Schema-less databases still need careful migration due to the implicit schema in any code that accesses the data.

Choosing RDBMS, NoSQL Database or Both?

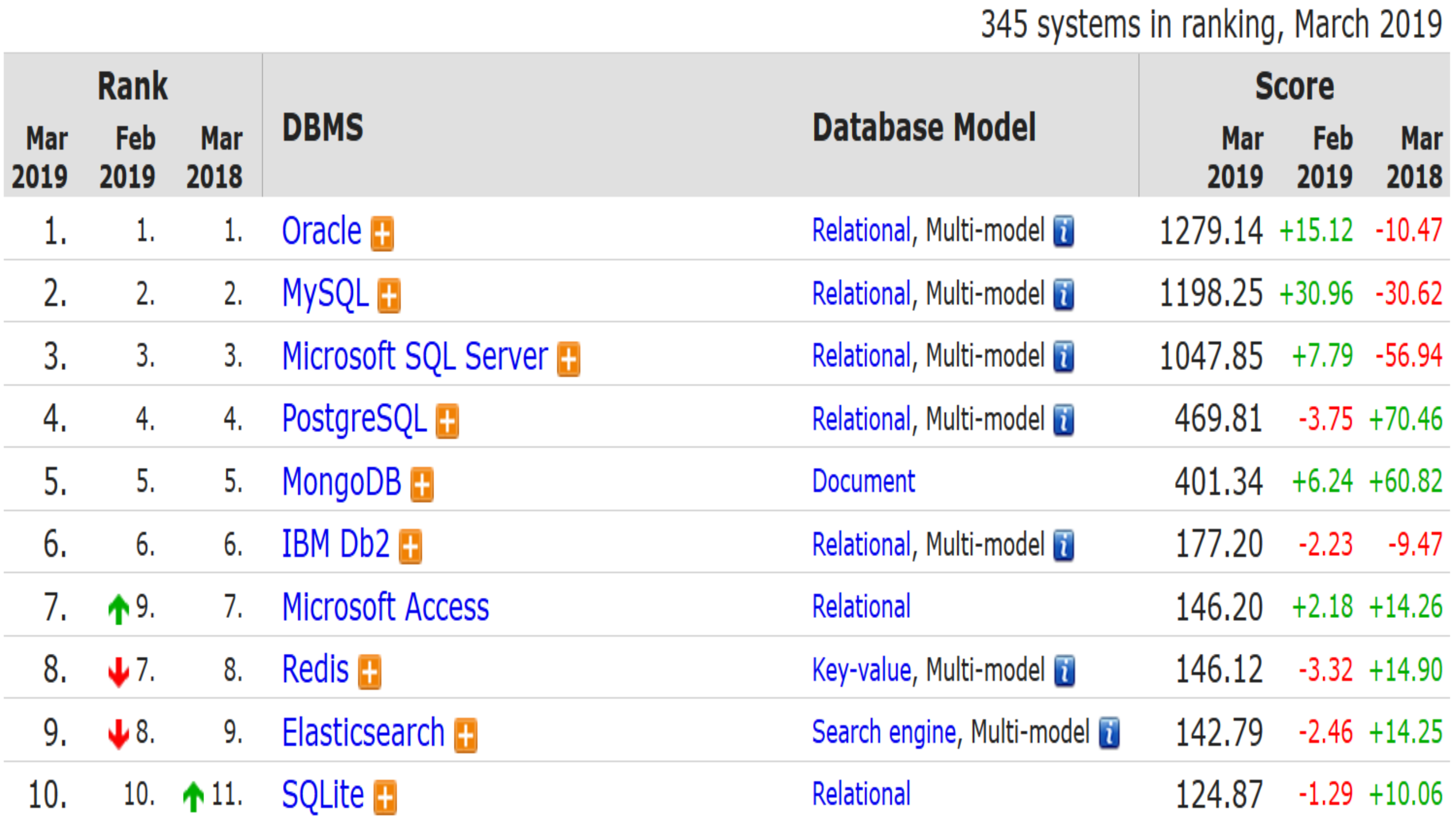
(http://www.informit.com/articles/article.aspx?p=2247310)

- Factors to consider when choosing a database for balancing speed, accuracy and reliability:
  - **What does my data look like?**
    - Does your data favor a table/row structure of a RDBMS, a document structure, a simple key-value pair structure, a column-based structure, or a graph structure?
  - **How is the current data stored?**
    - What is the cost for database migration between a RDBMS and a NoSQL database?
  - **How important is the guaranteed accuracy of database transactions?**
    - Are ACID properties important?
  - **How important is the speed of the database?**
  - **What happens when the data is not available?**
    - How critical it is for customers when data is not available?
  - **How is the database being used?**
    - Are most of database operations are Writes to store data or Reads?
  - **Should I split up the data to leverage the advantages of both RDBMS and NoSQL?**
    - After you have looked at the previous questions, you might want to consider putting some of the data, such as critical transactions, in an RDBMS while putting other data, such as blog posts, in a NoSQL database.

Ranking of Database Engines

- There are websites that rank the popularity of Database engines monthly (Relational DBMS vs. Key-Value Stores vs. Document Stores vs…)
  - Example:
    - http://db-engines.com/en/ranking

Ranking of DB Engine Popularity (http://db-engines.com/en/ranking)



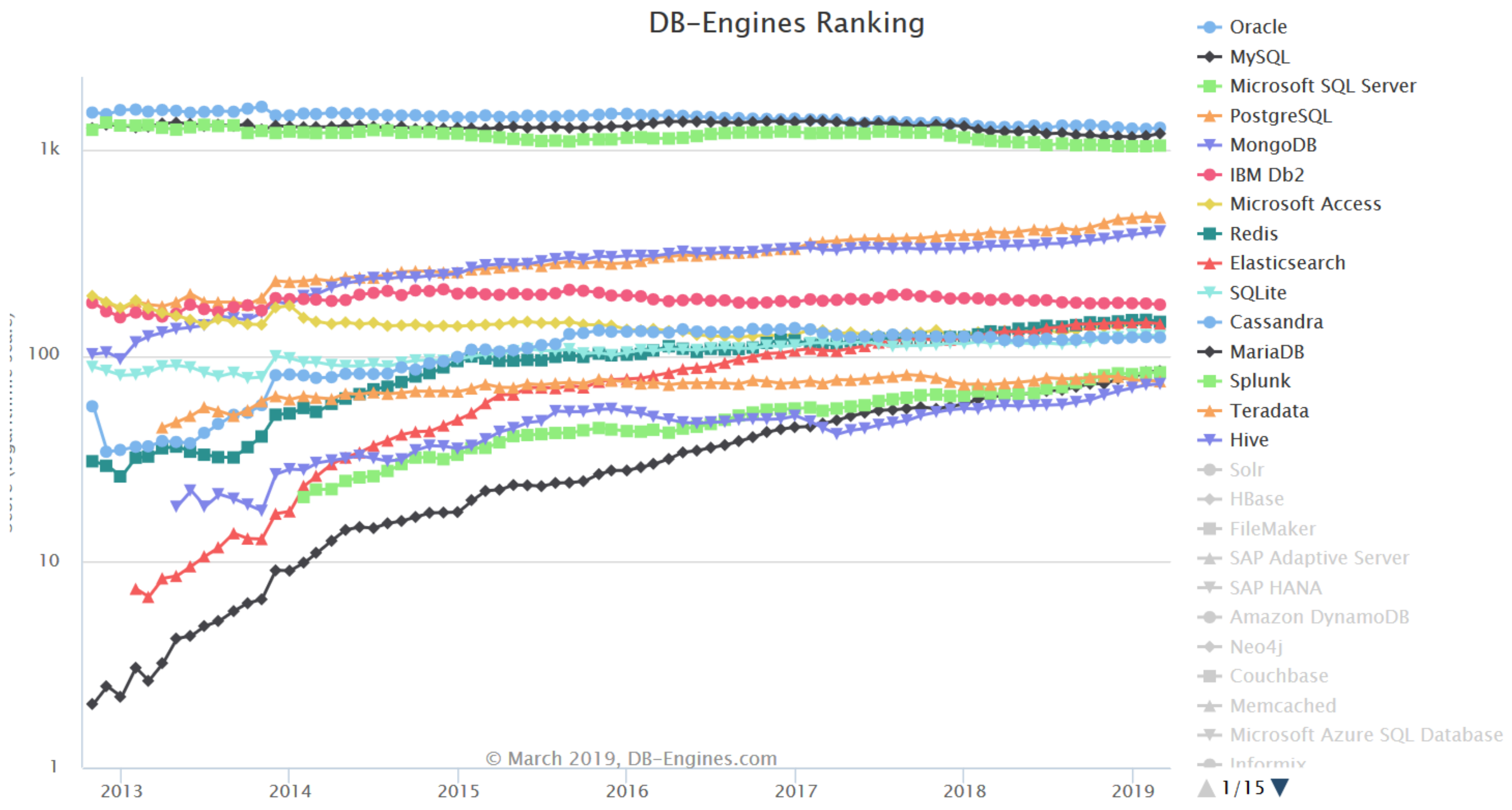
345 systems in ranking, March 2019

Rank			DBMS	Database Model	Score		
Mar 2019	Feb 2019	Mar 2018			Mar 2019	Feb 2019	Mar 2018
1.	1.	1.	Oracle	Relational, Multi-model	1279.14	+15.12	-10.47
2.	2.	2.	MySQL	Relational, Multi-model	1198.25	+30.96	-30.62
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1047.85	+7.79	-56.94
4.	4.	4.	PostgreSQL	Relational, Multi-model	469.81	-3.75	+70.46
5.	5.	5.	MongoDB	Document	401.34	+6.24	+60.82
6.	6.	6.	IBM Db2	Relational, Multi-model	177.20	-2.23	-9.47
7.	↑ 9.	7.	Microsoft Access	Relational	146.20	+2.18	+14.26
8.	↓ 7.	8.	Redis	Key-value, Multi-model	146.12	-3.32	+14.90
9.	↓ 8.	9.	Elasticsearch	Search engine, Multi-model	142.79	-2.46	+14.25
10.	10.	↑ 11.	SQLite	Relational	124.87	-1.29	+10.06

Ranking of DB Engines – Trend Popularity

(http://db-engines.com/en/ranking)

March 2019



References

The slides and materials in this lecture are from the following presentations/articles:

1. [Balazinska, 2012] Magda Balazinska, “NoSQL” , University of Washington, Fall 2012.
2. [Cattell, 2010] Rick Cattell, “Scalable SQL and NoSQL Data Stores”, SIGMOD Record, December 2010.
3. [Dayley, 2014] Brad Dayley, “Introducing NoSQL and MongoDB,” InformIT, http://www.informit.com/articles/article.aspx?p=2247310, September 2014.
4. [Francia, 2013] Steve Francia, “Building a Modern Web Application”. http://www.slideshare.net/spf13/modern-web-applications-mongo-db, 2013.
5. [Hows, 2015]]David Hows, Eelco Plugge, Peter Membrey, and Tim Hawkins, The Definitive Guide to MongoDB. A complete guide to dealing with Big Data using MongoDB, Appress, 2nd edition, 2015. *Available at the OU Digital Library*.
6. [MongoDB, 2019] MongoDB, “The MongoDB manual”, https://docs.mongodb.org/manual/ , March 2019.
7. [Sadalage, 2014] Pramod Sadalage, “NoSQL Databases – An Overview”, ThoughtWorks, http://www.thoughtworks.com/insights/blog/nosql-databases-overview, October 2014.

11. [SolidIT, 2016] Solid IT, “DB-Engines Ranking,” http://db-engines.com/en/ranking, April 2018.
12. (http://www.slideshare.net/spf13/modern-web-applications-mongo-db).

THE END