Document Stores

23D020: Big Data Management for Data Science

Barcelona School of Economics

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Knowledge objectives

1. Explain the main difference between key-value and document stores
2. Explain the main resemblances and differences between XML and JSON documents
3. Explain the design principle of documents
4. Name 3 consequences of the design principle of a document store
5. Explain the difference between relational foreign keys and document references
6. Exemplify 6 alternatives in deciding the structure of a document
7. Explain the difference between JSON and BSON
8. Name the main functional components of the MongoDB architecture
9. Explain the role of "mongos" in query processing
10. Explain what a replica set is in MongoDB
11. Name the three storage engines of MongoDB
12. Explain what shards and chunks are in MongoDB
13. Explain the two horizontal fragmentation mechanisms in MongoDB
14. Explain how the catalog works in MongoDB
15. Identify the characteristics of the replica synchronization management in MongoDB
16. Explain how primary copy failure is managed in MongoDB
17. Name the three query mechanisms of MongoDB
18. Explain the query optimization mechanism of MongoDB

Understanding objectives

1. Given two alternative structures of a document, explain the performance impact of the choice in a given setting
2. Simulate splitting and migration of chunks in MongoDB
3. Configure the number of replicas needed for confirmation on both reading and writing in a given scenario

Application objectives

1. Perform some queries on MongoDB through the shell and aggregation framework
2. Compare the access costs given different document designs
3. Compare the access costs with different indexing strategies (i.e., hash and range based)
4. Compare the access costs with different sharding distributions (i.e., balanced and unbalanced)

Semi-structured database model

XML and JSON

Semi-structured data

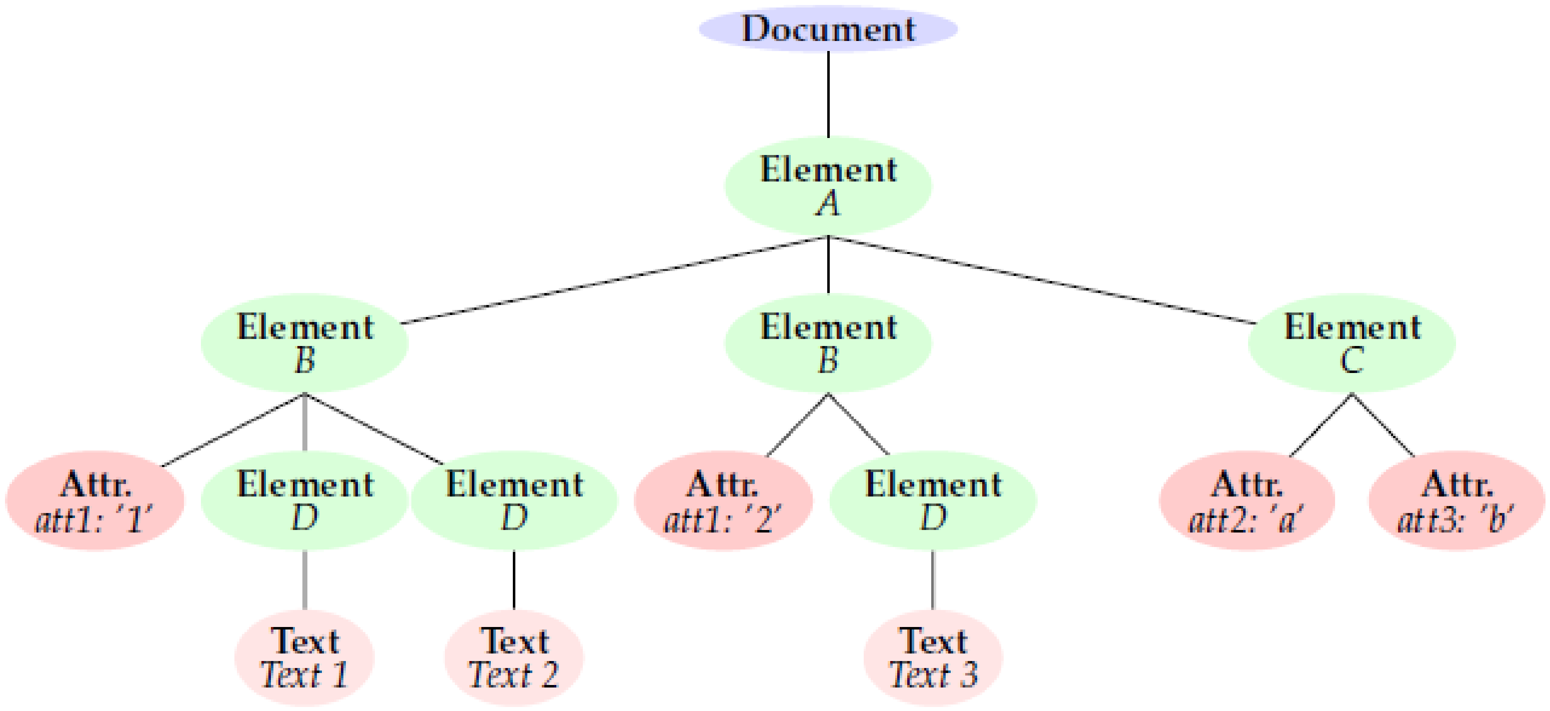
- Document stores are essentially key-value stores
  - The value is a document
    - Allow secondary indexes
- Different implementations
  - eXtensible Markup Language (XML)
  - JavaScript Object Notation (JSON)
- Tightly related to the web
  - Easily readable by humans and machines
  - Data exchange formats for REST APIs

XML Documents

- Tree data structure
  - Document: the root node of the XML document
  - Element: nodes that correspond to the tagged nodes in the document
  - Attribute: nodes attached to Element nodes
  - Text: text nodes, i.e., untagged leaves of the XML tree
- XML-oriented databases storage
  - eXist-db
  - MarkLogic
  - Relational extensions for Oracle, PostgreSQL, etc.

XML Document Example

```
<?xml version="1.0"
      encoding="utf-8"?>
<A>
  <B att1='1'>
    <D>Text 1</D>
    <D>Text 2</D>
  </B>
  <B att1='2'>
    <D>Text 3</D>
  </B>
  <C att2="a"
     att3="b"/>
</A>
```



S. Abiteboul et al.

JSON Documents

- Lightweight data interchange format
- Can contain unbounded nesting of arrays and objects
  - Brackets ([ ]) represent ordered lists
  - Curly braces ({ }) represent key-value dictionaries
    - Keys must be strings, delimited by quotes (")
    - Values can be strings, numbers, booleans, lists, or key-value dictionaries
- Natively compatible with JavaScript
  - Web browsers are natural clients
- JSON-like storage
  - MongoDB
  - CouchDB
  - Relational extensions for Oracle, PostgreSQL, etc.

JSON Example (I)


```json
{
  "title": "The Social network",
  "year": "2010",
  "genre": "drama",
  "country": "USA",
  "director": {
    "last_name": "Fincher",
    "first_name": "David",
    "birth_date": "1962"
  },
  "actors": [
    {
      "first_name": "Jesse",
      "last_name": "Eisenberg",
      "birth_date": "1983",
      "role": "Mark Zuckerberg"
    },
    {
      "first_name": "Rooney",
      "last_name": "Mara",
      "birth_date": "1985",
      "role": "Erica Albright"
    }
  ]
}
```

JSON Example (II)

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

source: MongoDB

JSON Example (III)

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
            level: 5,
            group: "dev"
          }
}
```

Embedded sub-document

Embedded sub-document

Data structure alternatives

Designing Document Stores

Do not think relational-wise

- Break 1NF to avoid joins
  - Get all data needed with one single fetch
  - Use indexes to identify finer data granularities

Consequences:

- Massive denormalization
- Independent documents
  - Avoid pointers (i.e., **we may have references but not FKs**)
- Massive rearrangement of documents on changing the application layout (e.g., queries)

Metadata representation

JSON

```
{
   _id: 123,
   A1: "x",
   …
   An: "x"
}
```

Tuple

_id	A1	…	An
123	"x"	…	"x"

Attribute optionality

J-666	J-NULL	J-Abs
`{ _id: 123, A1: 666, … An: 666 }`	`{ _id: 123, A1: null, … An: null }`	`{ _id: 123 }`

J-666

```
{
   _id: 123,
   A₁: 666,
   …
   Aₙ: 666
}
```

J-NULL

```
{
   _id: 123,
   A₁: null,
   …
   Aₙ: null
}
```

J-Abs

```
{
   _id: 123
}
```

T-666

_id	A1	…	An
123	666	…	666

T-NULL

_id	A1	…	An
123	null	…	null

Structure and Data Types

JSON Type

```
{
   _id: 123,
   A1: k,
   …
   An: k
}
```

```
{
"type": "object",
 "properties":{
  "A1": {
    "type": "number”
    },
   …
 "A1": {
    "type": "number”
    },
 required: ["A1",…, "An"]
 }
}
```

Tuple Type

_id	A_1	…	A_n
123	k	…	k

```
CREATE TABLE T (
_id INTEGER,
A1 INTEGER,
…
An INTEGER,
);
```

Integrity Constraints

JSON-IC

```
{
   _id: 123,
   A1: k,
   …
   An: k
}
```

```
{
"type": "object",
 "properties":{
  "A1": {
    "type": "number”
    "minimum": -k’
    "type": k’},
   …
 "An": {
    "type": "number”
    "minimum": -k’
    "maximum": k’}
 }
}
```

Tuple-IC

_id	A_1	…	A_n
123	k	…	k

```
ALTER TABLE T ADD CONSTRAINT
val_A1 CHECK
(A1 BETWEEN -k’ AND k’);
…

ALTER TABLE T ADD CONSTRAINT
val_An CHECK
(An BETWEEN -k’ AND k’);
```

Structure complexity

JSON-Attrib

```
{ _id: 123,
   A1: k,
   …
   An: k
}
```

JSON-Array

```
{ _id: 123,
   A: [1,…,n]
}
```

JSON-Nest

```
{ _id: 123
  L1:{
   …
   Ln:{
    An+1: k}
  }
}
```

Tuple-Attrib

_id	A_1	…	A_n
123	k	…	k

Tuple-Array

_id	A
123	[1,…,n]

MongoDB architecture

Abstraction

- **Documents**
  - Definition: JSON documents (serialized as BSON)
    - Basic atom
    - Identified by `"_id"` (user or system generated)
    - May contain
      - References (not FKs!)
      - Embedded documents
- **Collections**
  - Definition: A grouping of MongoDB documents
    - A collection exists within a single database
    - Collections do not enforce a schema
  - MongoDB Namespace: `database.collection`

JSON vs. BSON (Binary JSON)

```
{
  "id": 179,
  "name": "The Wire",
  "type": "Scripted",
  "language": "English",
  "genres": [ "Drama", "Crime", "Thriller" ],
  "status": "Ended",
  "runtime": 60,
  "premiered": "2002-06-02",
  "schedule": {
    "time": "21:00",
    "days": [
      "Sunday"
    ]
  },
  "rating": {
    "average": 9.4
  }
}
```

```
{
  "_id": ObjectId(99a88b77c66d),
  "name": "The Wire",
  "type": "Scripted",
  "language": "English",
  "genres": [ "Drama", "Crime", "Thriller" ],
  "status": "Ended",
  "runtime": 60,
  "premiered": ISODate("2002-06-02"),
  "schedule": {
    "time": "21:00",
    "days": [
      "Sunday"
    ]
  },
  "rating": {
    "average": 9.4
  }
}
```

A. Hogan

Shell commands

- `show dbs`
- `show collections`
- `show users`
- `use <database>`
- `coll = db.<collection>`
- `find([<criteria>], [<projection>])`
- `insert(<document>)`
- `update(<query>, <update>, <options [e.g., upsert]>)`
- `remove(<query>, [justOne])`
- `drop()`
- `createIndex(<keys>, <options>)`

- Notes:
  - *db* refers to the current database
  - *query* is a document (query-by-example)

MongoDB syntax

Global variable

Query-by-example (Depending on the method: document, array of documents, etc.)

```
db.[collection-name].[method]([query],[options])
```

- Collection methods: insert, update, remove, find, …

  ```
  db.restaurants.find({"name": "x"})
  ```

- Cursor methods: forEach, hasNext, count, sort, skip, size, ...

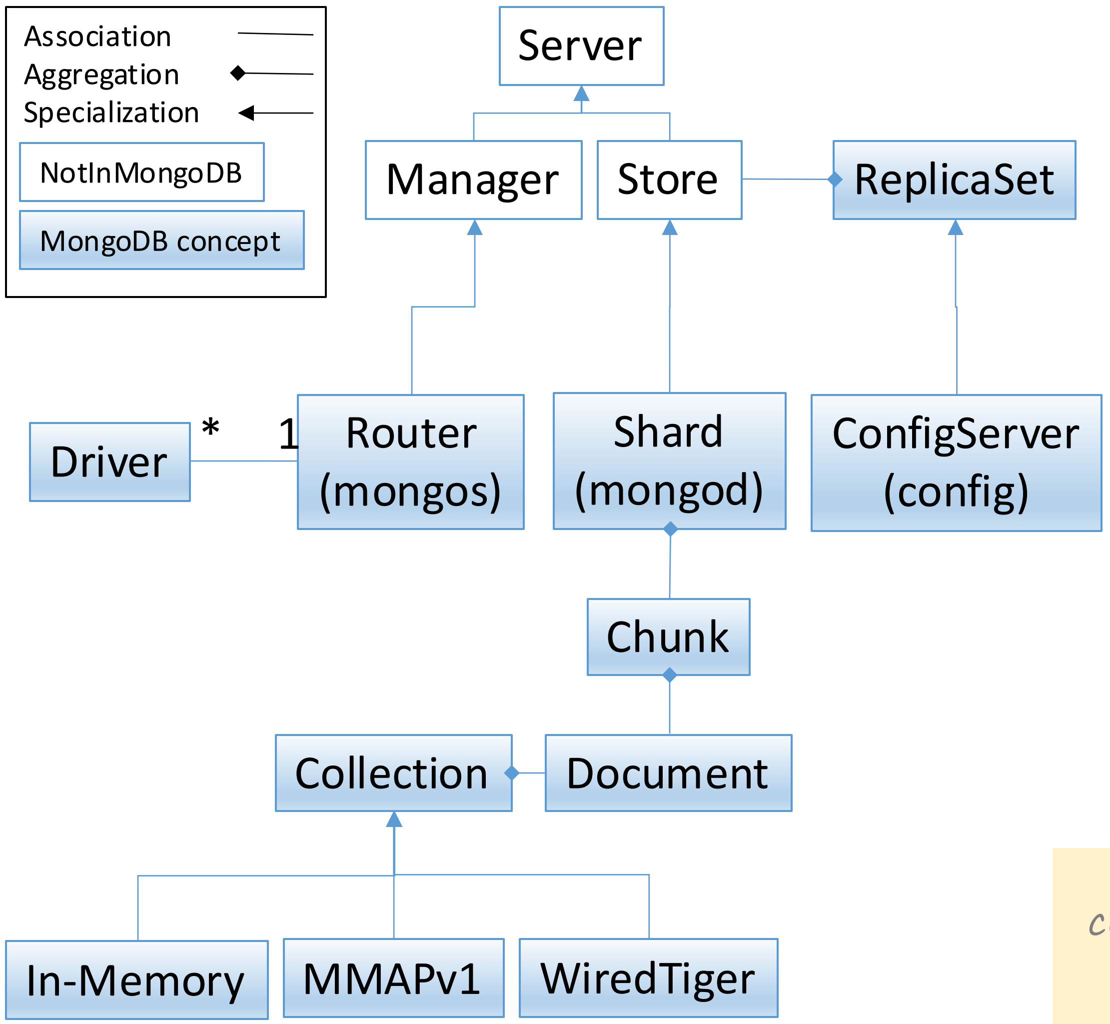
  ```
  db.restaurants.find({"name": "x"}).count()
  ```

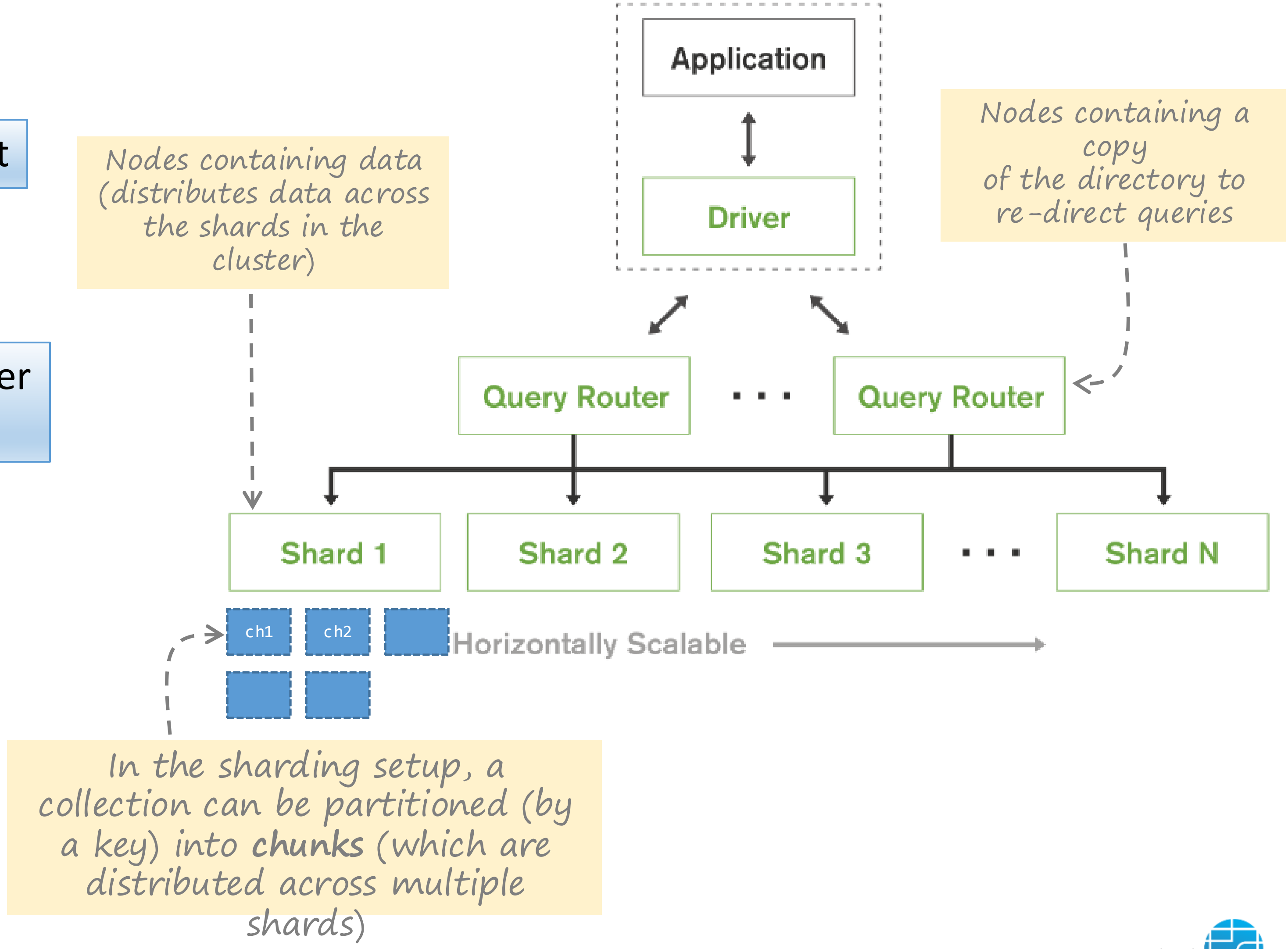
- Database methods: createCollection, copyDatabase, ...

  ```
  db.createCollection("collection-name")
  ```

- …

MongoDB functional components



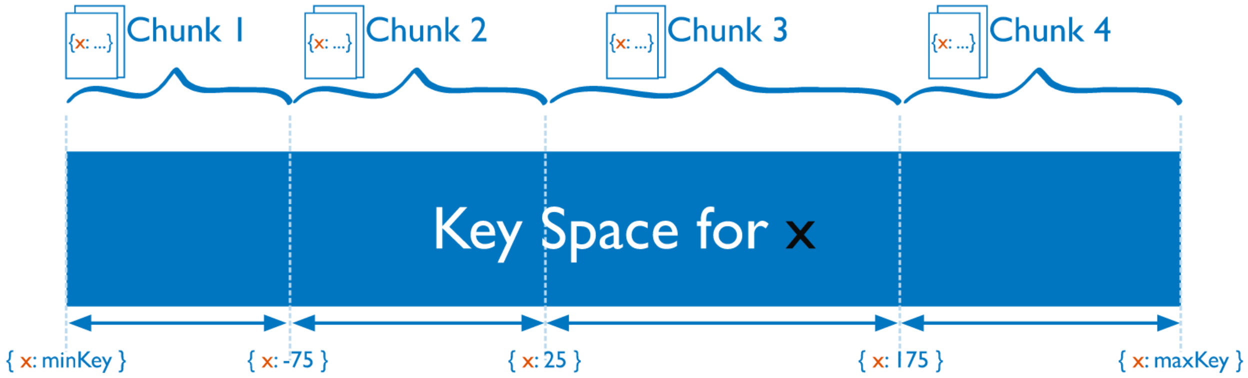


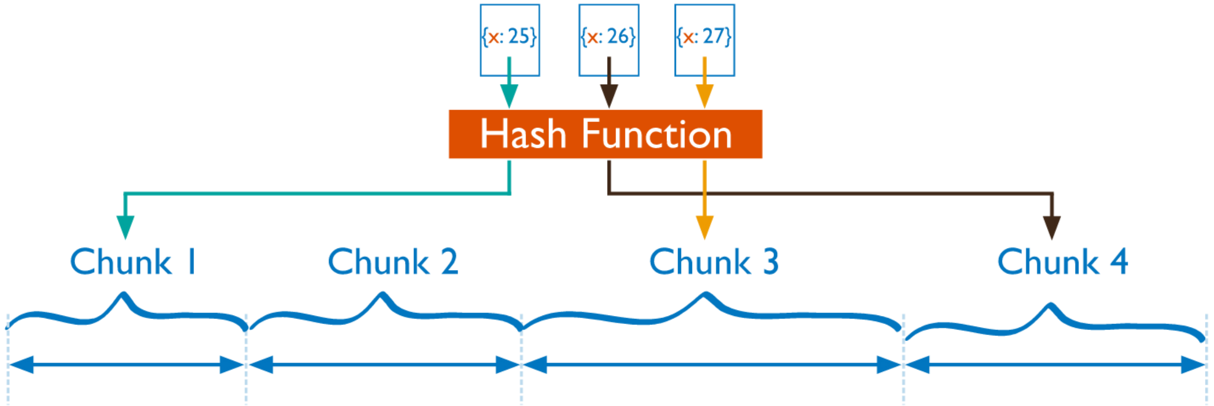
Data Design

Challenge I

Sharding (horizontal fragmentation)

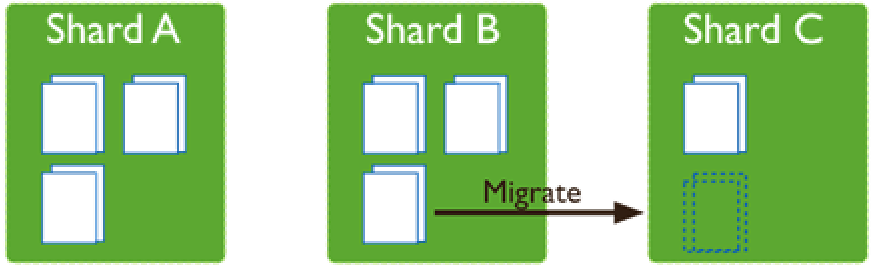
- **Shard key**
  - Must be indexed (`sh.shardCollection(namespace, key)`)
  - If not existing in a document, treated as null
- **Chunk (64MB)**
  - Horizontal fragment according to the shard key
    - Range-based: Range of values determines the chunks
      - Adequate for range queries
    - Hash-based: Hash function determines the chunks
      - Consistent hashing





Splitting and migrating chunks

- Inserts and updates above a threshold trigger splits
  - Not in single-key chunks (same value in the shard keys)
- Uneven distributions in the number of chunks per shard trigger migrations
  1. A new chunk is created in an underused shard
  2. Per document requests are sent to the origin shard
  3. Origin keeps working as usual
     - Changes made during the migration are applied *a posteriori* in the destination shard
  4. Changes are annotated in the config servers, which enables the new chunk
  5. Chunk at origin is dropped
  6. Client cache in query routers is inconsistent
     - Eventually synchronized



Catalog Management

Challenge II

Catalog structure

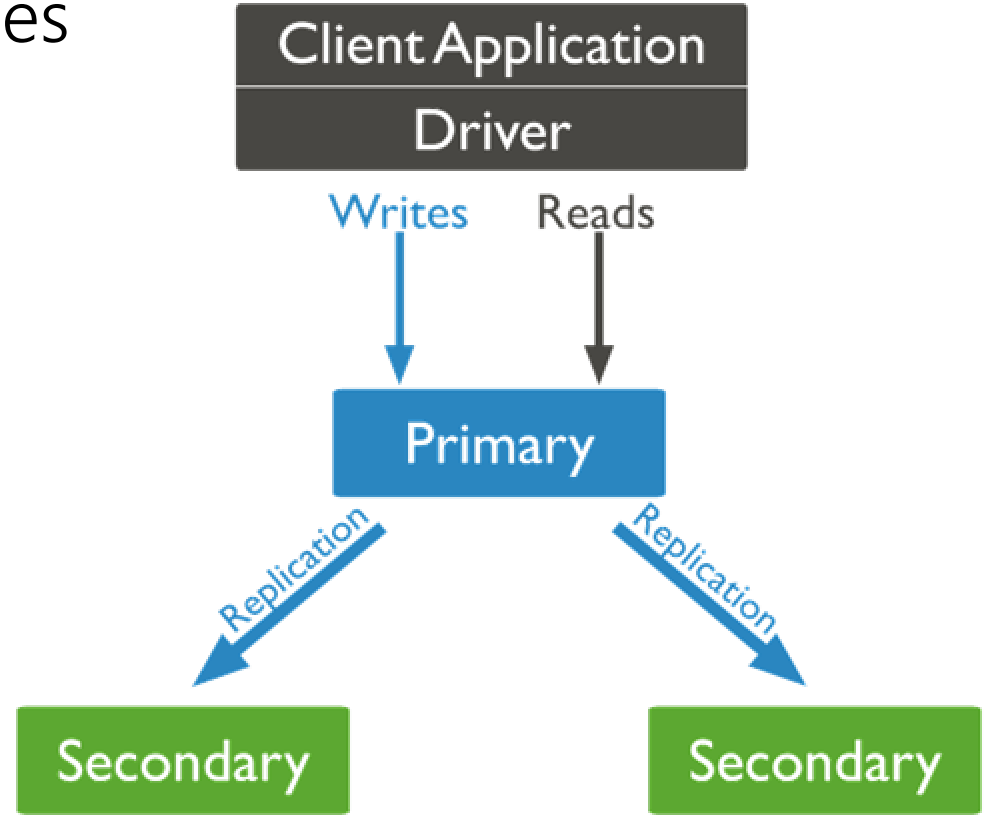
- Content
  - List of chunks in every shard
- Implemented in a replica set (as any other data)
- Client cache in the query routers
  - Lazy/Primary-copy replication maintenance

Transaction Management

Challenge III

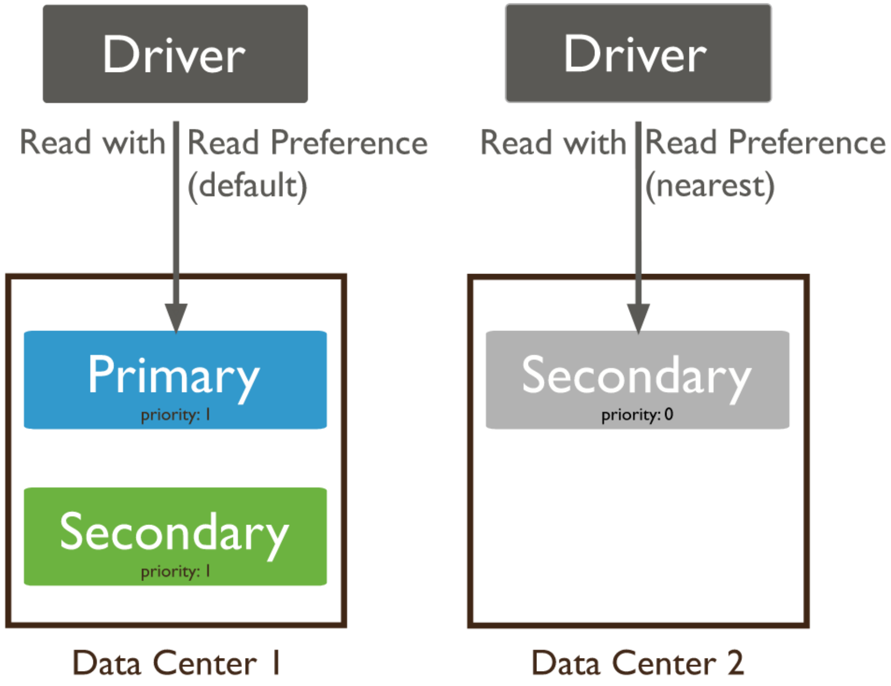
Replica sets

- A replica set is a set of 3 **`mongod`** instances
- Primary copy with lazy replication
  - One primary copy
    - Inserts, writes, updates
    - Reads
  - Secondary copies
    - Reads



Read preference

- By default, applications will try to read the primary replica
- It can also specify a read preference
  - `primary`
  - `primaryPreferred`
  - `secondary`
  - `secondaryPreferred`
  - `nearest`
    - Least network latency

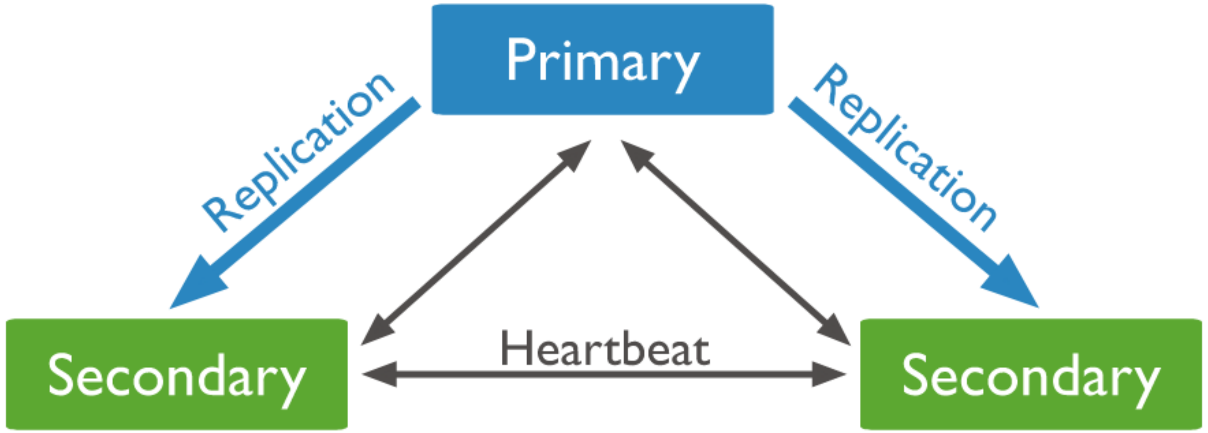


Required read and writes

- **ReadConcern**
  - Specifies how many copies need to be read before confirmation
    - They should coincide
- **WriteConcern**
  - Specifies how many copies need to be writen before confirmation
    - Might be zero

Handling failures

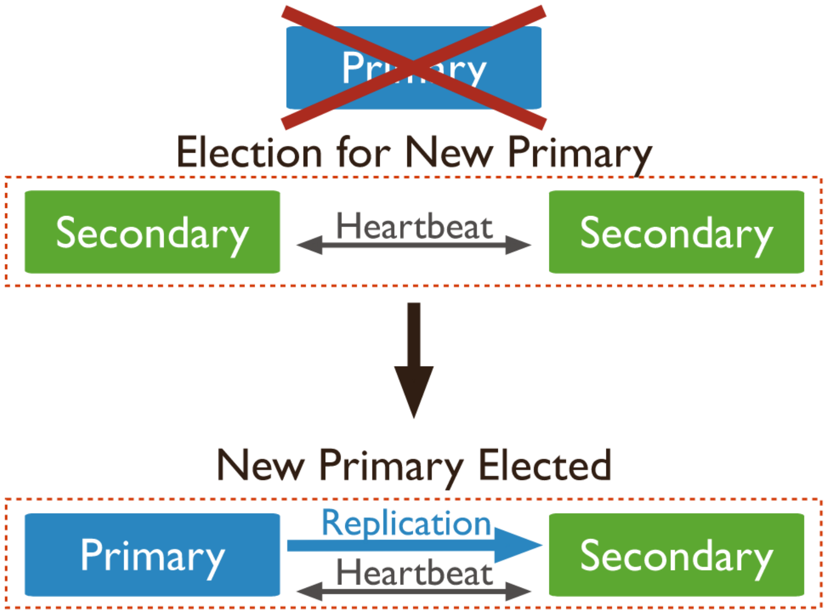
- Heartbeat system
  - Primary does not communicate with the other members for 10sec → Failure



source: MongoDB

Handling failures

- Heartbeat system
  - Primary does not communicate with the other members for 10sec → Failure
- New primary is decided based on consensus protocols
  - PAXOS



Query Processing

Challenge IV

Query mechanisms

a) JavaScript API
- `find` and `findOne` methods (Query By Example)
  - `db.collection.find()`
  - `db.collection.find( { qty: { $gt: 25 } } )`
  - `db.collection.find( { field: { $gt: value1, $lt: value2 } } )`

b) Aggregation Framework
- Documents enter a multi-stage pipeline that transforms them
  - Filters that operate like queries
  - Transformations that reshape the output document
  - Grouping
  - Sorting
  - Other stage operations

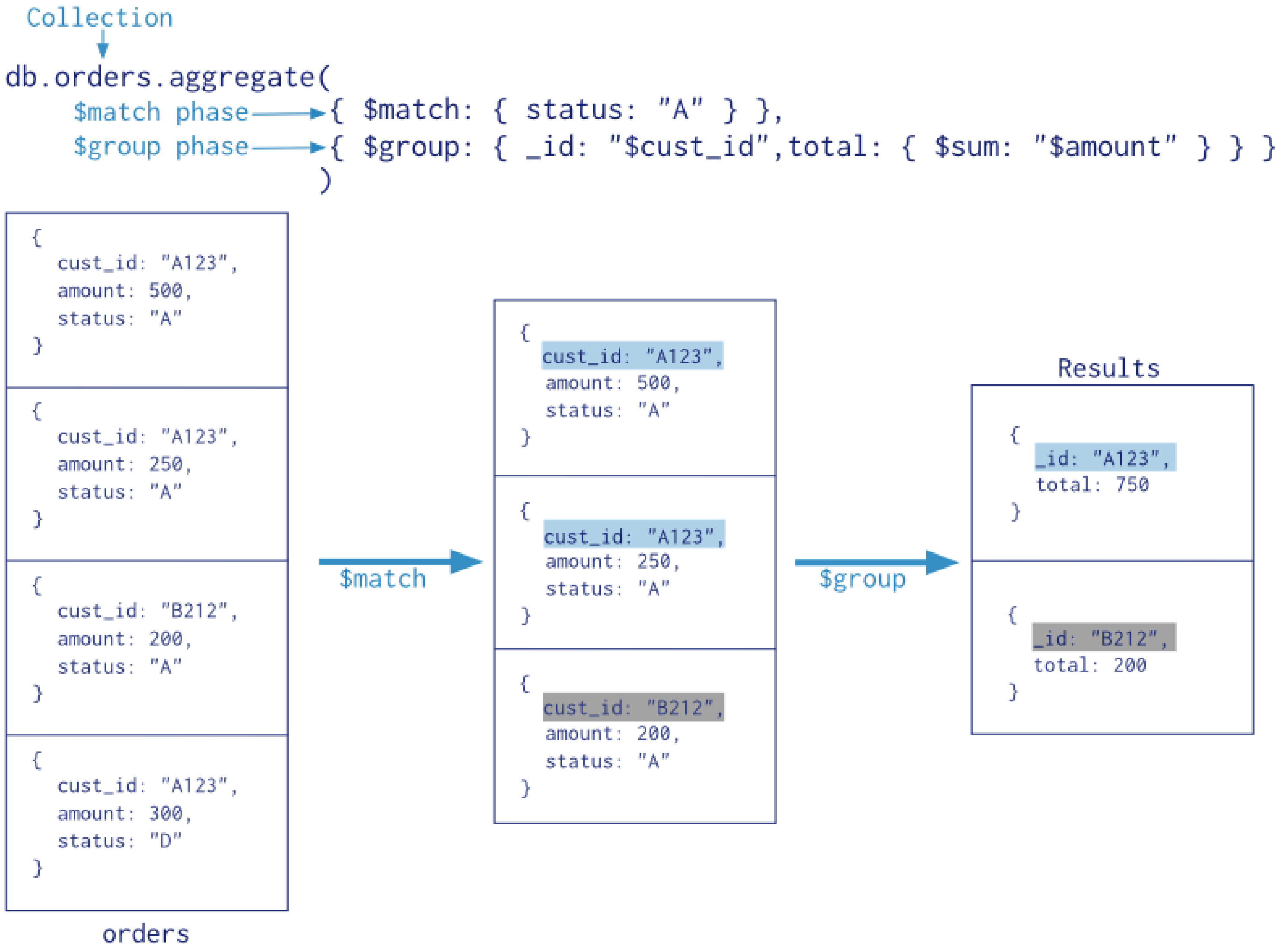
c) MapReduce

Example queries

```sql
1. SELECT * FROM users;
2. SELECT * FROM users WHERE age > 25;
3. SELECT name, age FROM users;
4. INSERT INTO users (name, age) VALUES ('Alice', 30);
5. UPDATE users SET age = 31 WHERE name = 'Alice';
```

```javascript
1. db.users.find({});
2. db.users.find({ age: { $gt: 25 } });
3. db.users.find({}, { name: 1, age: 1, _id: 0 });
4. db.users.insertOne({ name: "Alice", age: 30 });
5. db.users.updateOne({ name: "Alice" }, { $set: { age: 31 } });
```

Aggregation Framework Steps



Aggregation Framework Syntax

Pipeline stages: ($match, $group, $addfields, $sort, $unwind …)

The name of the computed field

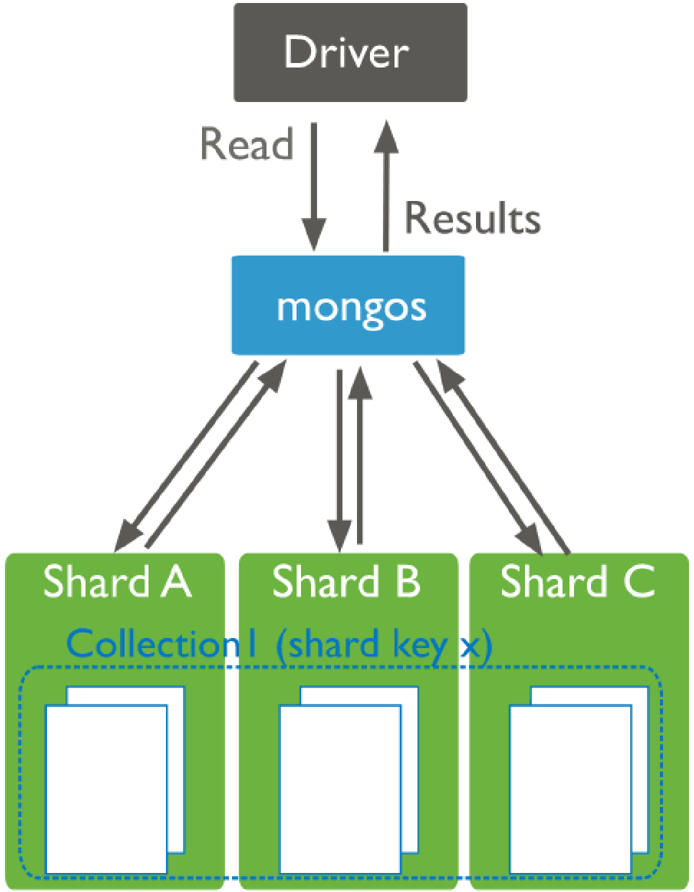
```
db.orders.aggregate(
                  {$match: {status:"A"}},
                  {$group: {_id: "$cust_id", total:{$sum: "$amount"}}}
                 )
```

Pipeline operators: $sum, $max, $min …

References the field

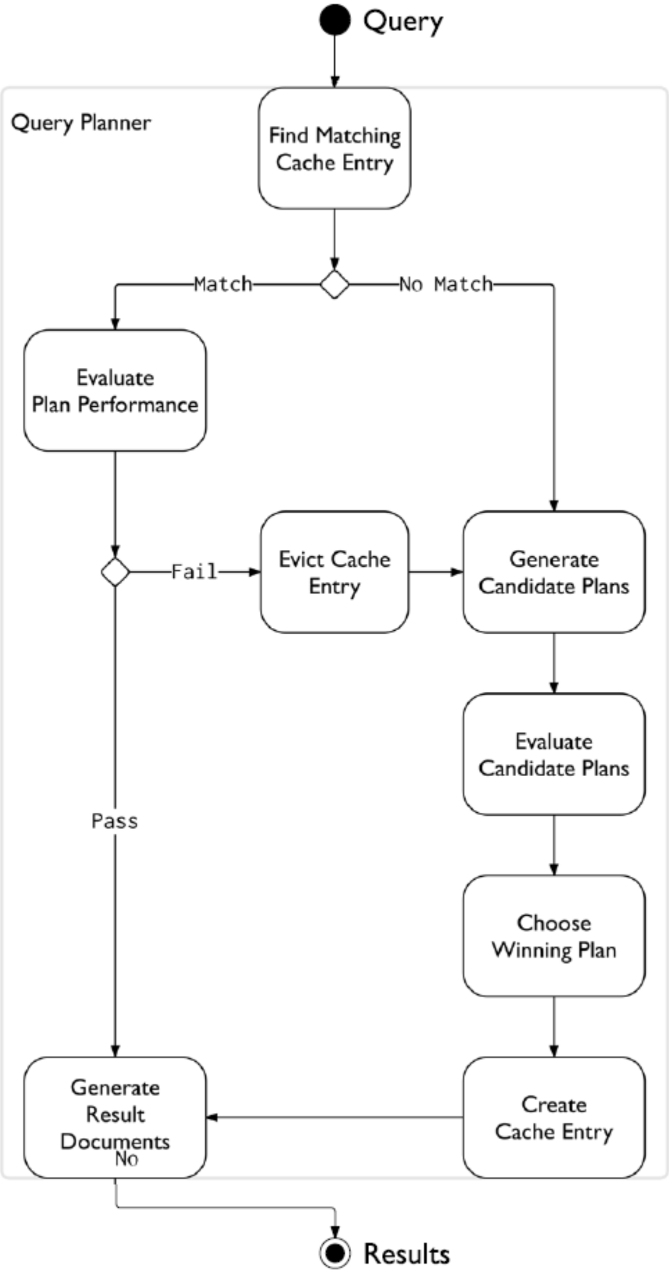
Required field: to identify the field for the group by

Query routing



Indexing

- Kinds
  - B+
  - Hash
  - Geospatial
  - Text
- Allow
  - Multi-attribute indexes
  - Multi-valued indexes
    - On arrays
  - Index-only query answering
- Usage
  - Best plan is cached
  - Performance is evaluated on execution
    - New candidate plans are evaluated for some time



Closing

Summary

- Document-stores
  - Semi-structured database model
  - Indexing
- MongoDB
  - Architecture
  - Interfaces

References

- E. Brewer. *Towards Robust Distributed Systems*. PODC'00
- L. Liu and M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- S. Abiteboul et al. *Web Data Management*. Cambridge University Press, 2012
- M. Hewasinghage et al. On the Performance Impact of Using JSON, Beyond Impedance Mismatch. ADBIS 2020
- A. Hogan: *Procesado de Datos Masivos*. U. de Chile.

http://aidanhogan.com/teaching/cc5212-1-2020

Lab 2

Document Stores

Lab 2: Document Stores - Teams

- Teams of two
  - You cannot repeat the teammate
- Assign yourself to a team, otherwise to be assigned randomly
  - https://docs.google.com/spreadsheets/d/1jEzgsNGEEHR6yeS0HsQuynAo2IkHi0731aNMF8pV6bI/edit?usp=sharing

Lab 2: Document Stores - Training

Training [not evaluated]

- Installing MongoDB
  - MongoDB Community Server: https://www.mongodb.com/try/download/community
  - MongoDB Compas (GUI): https://www.mongodb.com/try/download/compass
  - How To/FAQs: https://diligent-skirt-36b.notion.site/MongoDB-2f1db119176c4be7886edfac2062d3cc?pvs=4
- Tasks:
  - Importing data
  - Querying data
    - Inserte, Delete, Update, Select
  - Geospatial queries

Lab 2: Document Stores - Assignment

Lab Assignment

- Deadline: **Week 8 (**27/05/2025, 12:25)
- Tasks:
  - Model data in MongoDB
  - Querying data in MongoDB
  - Reporting query latencies
  - Discussion of modeling alternatives
- Deliverables
  - Python Code
  - PDF Document