



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 5.1: NoSQL Database - MongoDB

Janardhanan PS

janardhanan.ps@wilp.bits-pilani.ac.in

What is document database

- While relational databases rely on rigid structures, document databases are much more natural to work with and can be used for a variety of use cases across industries.
- Document databases give developers a better experience for building applications with **flexible data** schemas and **lightning-fast** development cycles.
- Advantages of document database:
 - ✓ An intuitive data model that is fast and easy for developers to work with
 - ✓ A flexible schema that allows for the data model to evolve as application need change
 - ✓ Supports MapReduce interface
 - ✓ The ability to horizontally scale out

Typical Document Database: MongoDB, CouchDB

What are documents ?

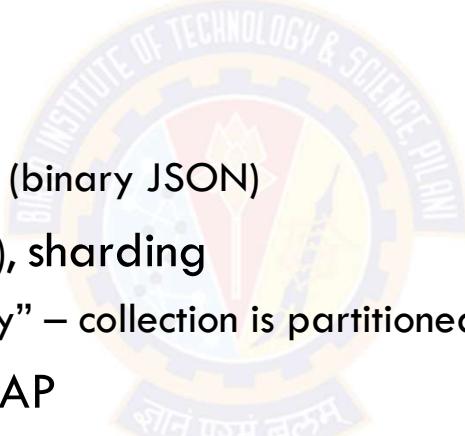
- A document is a record in a document database.
- A document typically stores information about related metadata (field) and one object (value)
- Documents store data in **field-value** pairs.
- The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects.
- Documents can be stored in formats like JSON, BSON, and XML

Sample JSON Document

```
{  
    FirstName: John,  
    LastName: Mathews,  
    ContactNo: [+123 4567 8900, +123 4444 5555]  
}
```

MongoDB (1)

- Open source (Community edition)
- Name derived from the word humongous (extremely large)
- 1st release 2009
- Document store
 - ✓ An extended format called BSON (binary JSON)
- Supports replication (master/slave), sharding
 - ✓ Developer provides the “shard key” – collection is partitioned by ranges of values of this key
- Consistency guarantees, CP of CAP
- Used by Adobe (experience tracking), Craigslist, eBay, FIFA (video game), LinkedIn, McAfee
- Provides connector to Hadoop
 - ✓ Cloudera provides the MongoDB connector in distributions



MongoDB (2)

- Database is a set of collections
- A collection is like a table in RDBMS
- A collection stores documents
 - ✓ BSON or Binary JSON with hierarchical key-value pairs
 - ✓ Documents are similar to rows in a table
 - ✓ Max 16MB documents stored in [WiredTiger](#) storage engine
- For larger than 16MB documents uses [GridFS](#)
 - ✓ Support for binary data
 - ✓ Large objects can be stored in ‘chunks’ of 255KB
 - ✓ Stores Meta-data in a separate collection
 - ✓ Does not support multi-document transactions

WiredTiger storage engine* * <https://docs.mongodb.com/manual/core/wiretiger/>

MongoDB (3)

- Data is partitioned in shards
 - ✓ For horizontal scaling
 - ✓ Reduces amount of data each shard handles as the cluster grows
 - ✓ Reduces number of operations on each shard
- Data is replicated
 - ✓ Writes to primary in oplog. “write-concern” setting used to tweak write consistency.
 - ✓ Secondaries use oplog to get local copies updated
 - ✓ Clients usually read from primary but “read-preference” setting can tweak read consistency
- Data updates happen in place and not versioned / timestamped

cloud.mongodb.com

Clusters

Find a cluster... 

SANDBOX

● Cluster0
Version 4.4.6

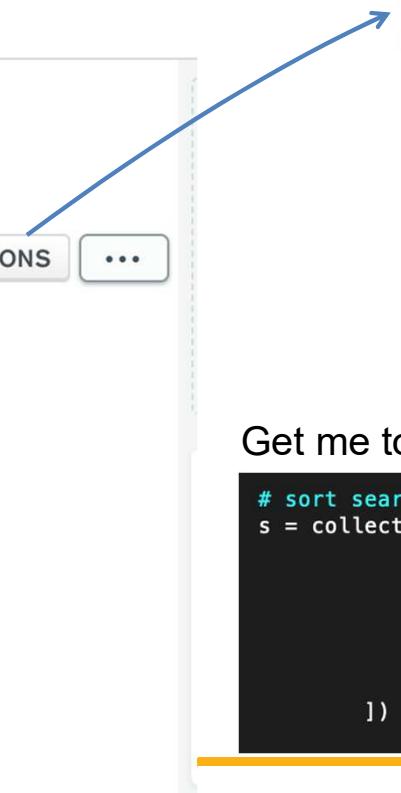
CONNECT METRICS COLLECTIONS ...

CLUSTER TIER
M0 Sandbox (General)

REGION
AWS / Mumbai (ap-south-1)

TYPE
Replica Set - 3 nodes

LINKED REALM APP
None Linked



Find Indexes Schema Anti-Patterns 0 Aggreg

FILTER {"filter": "example"}

QUERY RESULTS 1-20 OF MANY

sample_airbnb

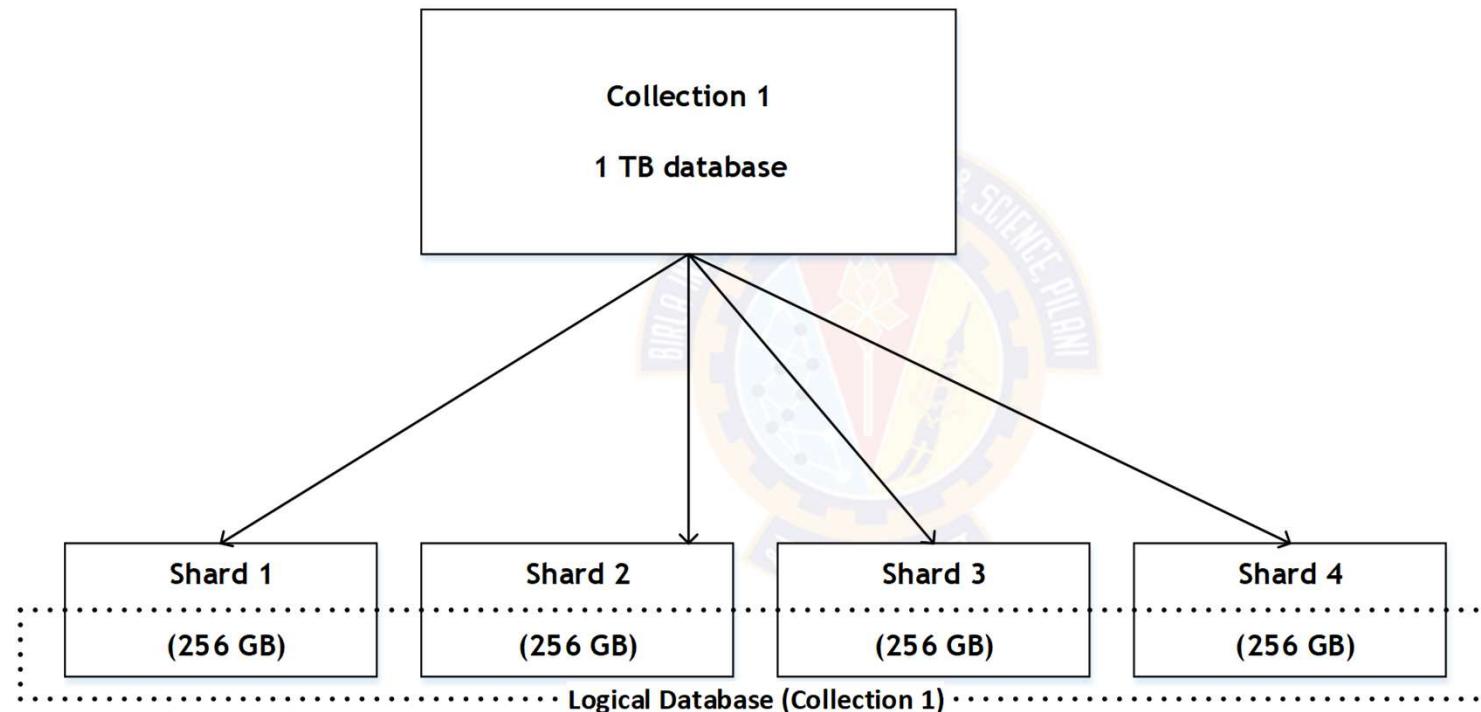
- listingsAndReviews
- sample_analytics
- sample_geospatial
- sample_mflix
- sample_restaurants
- sample_supplies
- sample_training
- sample_weatherdata

_id: "10006546"
listing_url: "https://www.airbnb.com/rooms/10006546"
name: "Ribeira Charming Duplex"
summary: "Fantastic duplex apartment with three bedrooms, lc
space: "Privileged views of the Douro River and Ribeira squa
description: "Fantastic duplex apartment with three bedrooms
neighborhood_overview: "In the neighborhood of the river, yc
notes: "Lose yourself in the narrow streets and staircases 2
transit: "Transport: • Metro station and S. Bento railway 5m
access: "We are always available to help guests. The house i
interaction: "Cot - 10 € / night Dog - € 7,5 / night"
house_rules: "Make the house your home..."
property_type: "House"
room_type: "Entire home/apt"
bed_type: "Real Bed"
minimum_nights: "2"
maximum_nights: "30"
cancellation_policy: "moderate"
last_scraped: 2019-02-16T05:00:00.000+00:00
calendar_last_scraped: 2019-02-16T05:00:00.000+00:00
first_review: 2016-01-03T05:00:00.000+00:00
last_review: 2019-01-20T05:00:00.000+00:00
accommodates: 8
bedrooms: 3
beds: 5

Get me top 10 beach front homes

```
# sort search results by score
s = collection.aggregate([
    { "$match": { "$text": { "$search": "beach front" } } },
    { "$project": { "name": 1, "_id": 0, "score": { "$meta": "textScore" } } },
    { "$match": { "score": { "$gt": 1.0 } } },
    { "$sort": { "score": -1}},
    { "$limit": 10}
])
```

Sharding in MongoDB



Terms used in RDBMS and MongoDB

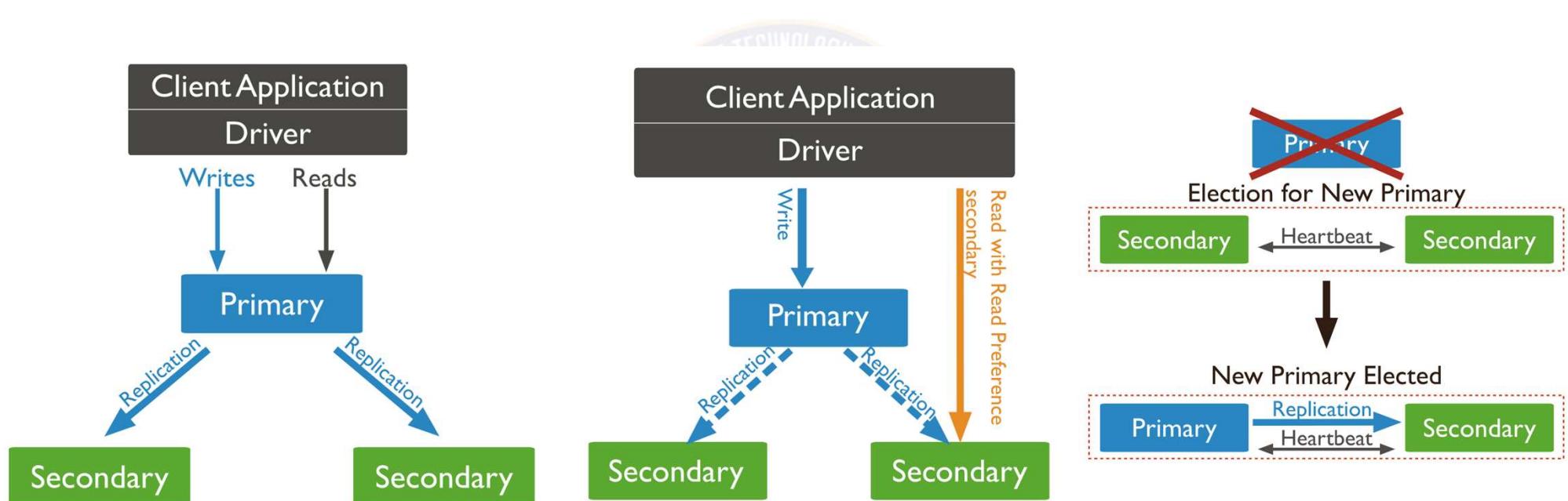
RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields / Key Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is an identifier)

MongoDB Data Model

- JavaScript Object Notation (JSON) model
- *Database* = set of named *collections*
- *Collection* = sequence of *documents*
- *Document* = {attribute₁:value₁,...,attribute_k:value_k}
- *Attribute* = string (attribute_i≠attribute_j)
- *Value* = **primitive** value (string, number, date, ...), or a **document**, or an *array*
- *Array* = [value₁,...,value_n]
- Key properties: **hierarchical** (like XML), **no schema**
 - ✓ Collection docs may have different attributes

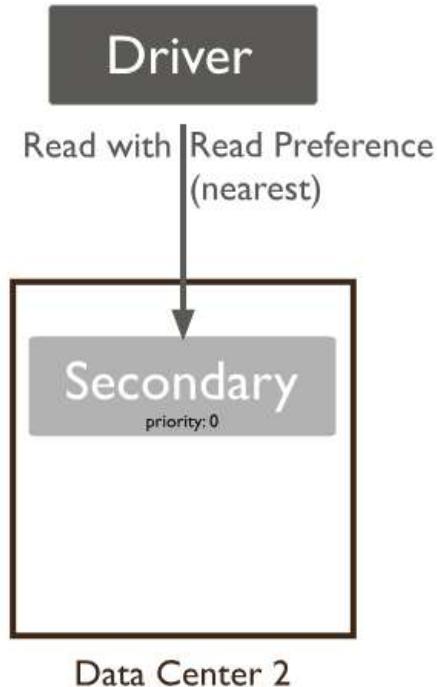
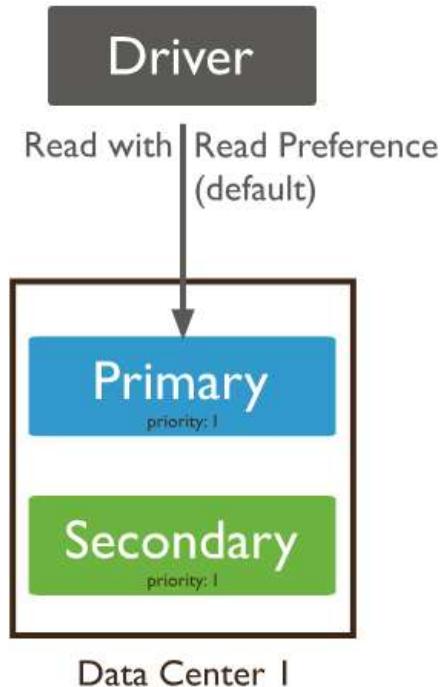
MongoDB

- Document oriented DB
- Various read and write choices for flexible consistency tradeoff with scale / performance and durability
- Automatic primary re-election on primary failure and/or network partition



MongoDB Read Preference

Read preference describes how MongoDB clients route read operations to the members of a replica set



Read Preference Modes

- Primary
- PrimaryPreferred
- secondary
- secondaryPreferred
- nearest

<https://www.mongodb.com/docs/manual/core/read-preference/>

MongoDB “read concerns”

Read Concern option allows you to control the consistency and latency properties of the data read from replica sets and sharded clusters.

- **local :**

- ✓ Client reads primary replica
 - ✓ Client reads from secondary in causally consistent sessions

- **available:**

- ✓ Read on secondary but causal consistency not required

- **majority :**

- ✓ If client wants to read what majority of nodes have. Best option for fault tolerance and durability.

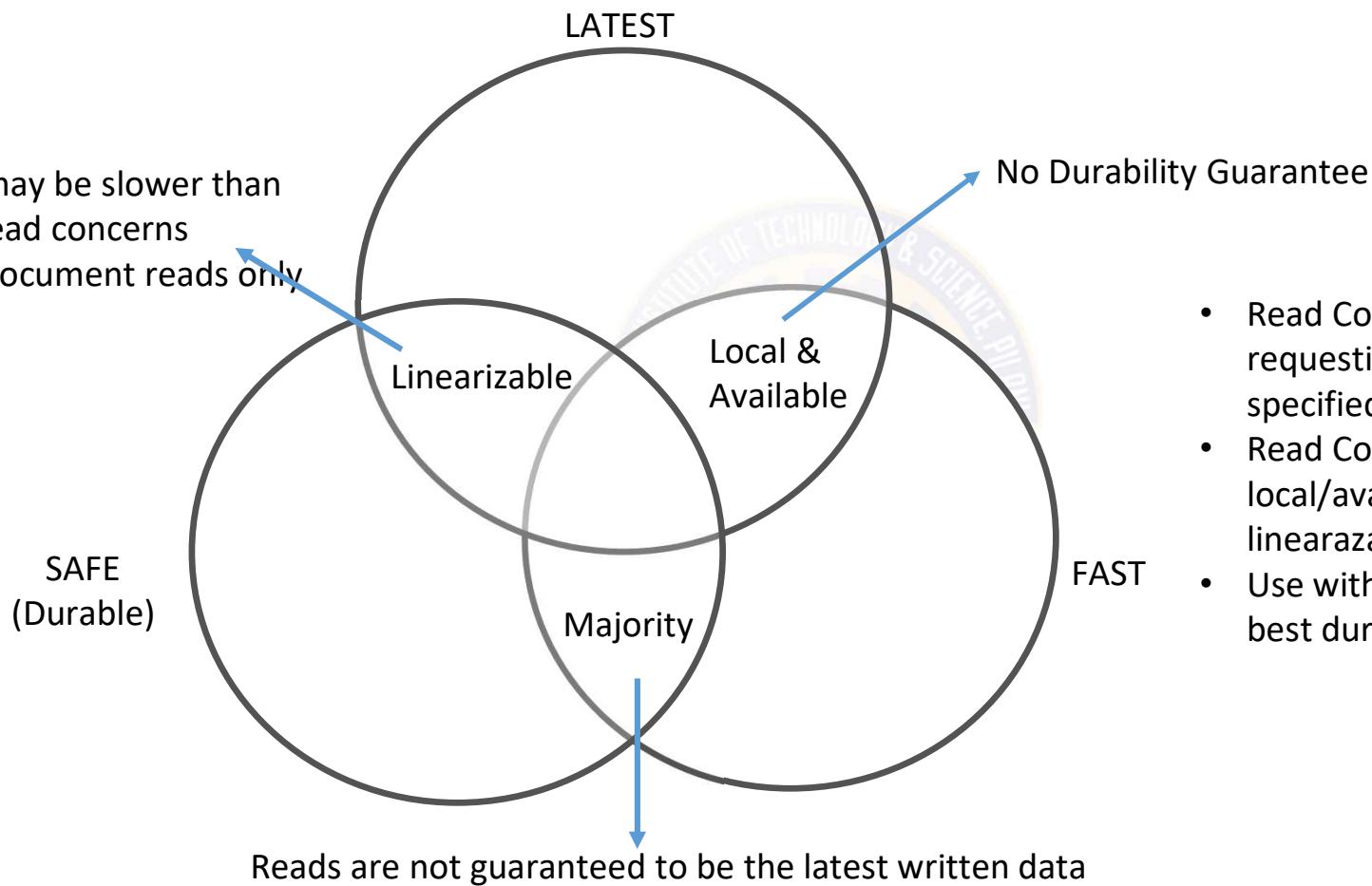
- **linearizable :**

- ✓ If client wants to read what has been written to majority of nodes before the read started.
 - ✓ Has to be read on primary
 - ✓ Only single document can be read

<https://www.mongodb.com/docs/manual/reference/read-concern/>

Latency and Read Concerns

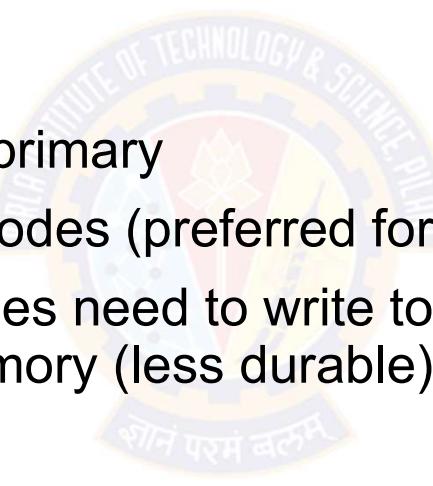
- Reads may be slower than other read concerns
- Single document reads only



- Read Concern is a way of requesting data that meets a specified level of durability
- Read Concern options: local/available, majority, and linearizable
- Use with write concern for best durability guarantees

MongoDB “write concerns”

- how many replicas should ack
 - ✓ 1 - primary only
 - ✓ 0 - none
 - ✓ n - how many including primary
 - ✓ majority - a majority of nodes (preferred for durability)
- journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)
- timeout for write operation



<https://www.mongodb.com/docs/manual/reference/write-concern/>

MongoDB Data Example

Collection inventory

```
{  
  item: "ABC2",  
  details: { model: "14Q3", manufacturer: "M1 Corporation" },  
  stock: [ { size: "M", qty: 50 } ],  
  category: "clothing"  
}  
  
{  
  item: "MNO2",  
  details: { model: "14Q3", manufacturer: "ABC Company" },  
  stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L", qty: 1 } ],  
  category: "clothing"  
}
```

```
db.inventory.insert(  
  {  
    item: "ABC1",  
    details: {model: "14Q3", manufacturer: "XYZ Compa  
    stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],  
    category: "clothing"  
  })
```

Document insertion

Example of Simple Query

Collection orders

```
{  
  _id: "a",  
  cust_id: "abc123",  
  status: "A",  
  price: 25,  
  items: [ { sku: "mmm", qty: 5, price: 3 },  
           { sku: "nnn", qty: 5, price: 2 } ]  
}  
  
{  
  _id: "b",  
  cust_id: "abc124",  
  status: "B",  
  price: 12,  
  items: [ { sku: "nnn", qty: 2, price: 2 },  
           { sku: "ppp", qty: 2, price: 4 } ]  
}
```

db.orders.find(

 { status: "A" },

 { cust_id: 1, price: 1, _id: 0 }

)

selection

projection

In SQL it would look like this:

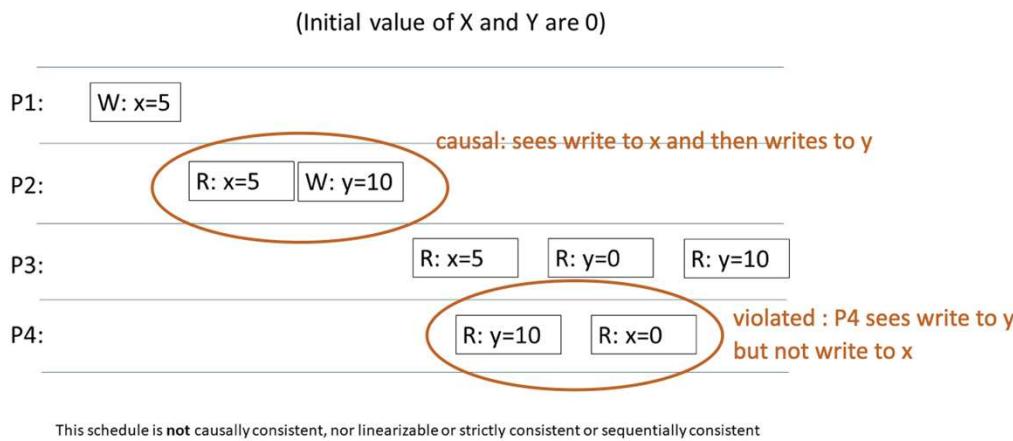
```
SELECT cust_id, price  
FROM orders  
WHERE status="A"
```

Results

```
{  
  cust_id: "abc123",  
  price: 25  
}
```

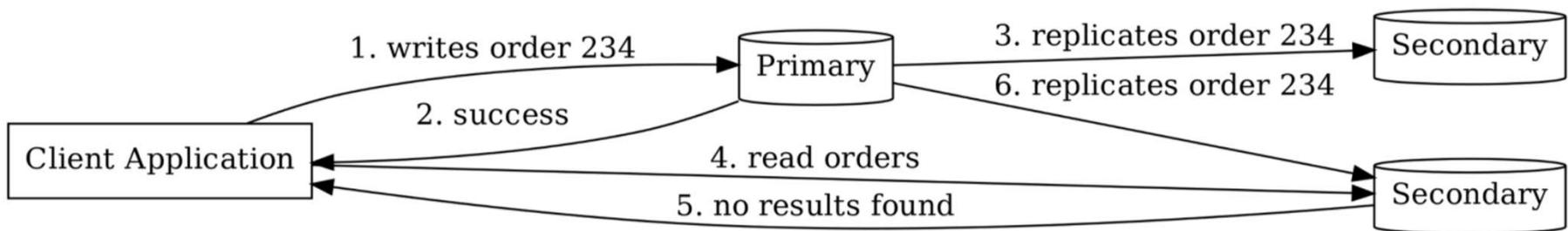
What is Causal Consistency

- | | |
|----------------------------|--|
| Read your writes | Read operations reflect the results of write operations that precede them. |
| Monotonic reads | Read operations do not return results that correspond to an earlier state of the data than a preceding read operation. |
| Monotonic writes | Write operations that must precede other writes are executed before those other writes. |
| Writes follow reads | Write operations that must occur after read operations are executed after those read operations. |

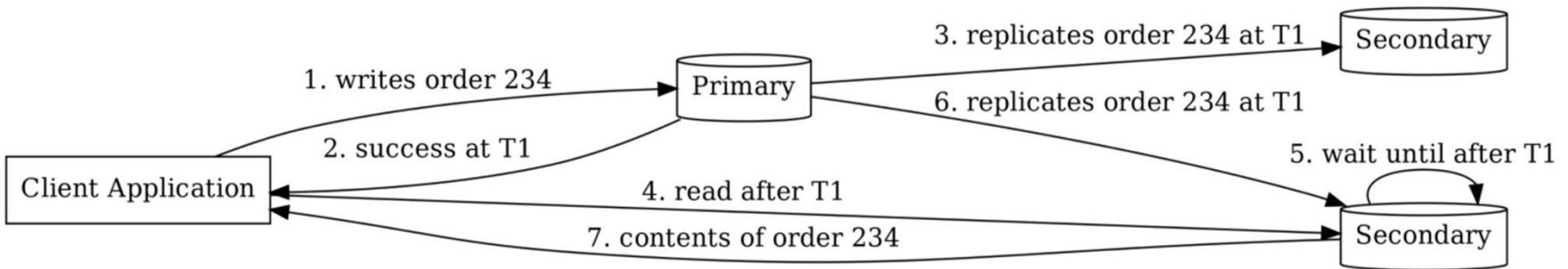


Example in MongoDB

- Case 1 : No causal consistency



- Case 2: Causal consistency by making read to secondary wait



MongoDB: Indexing

- Can create index on any field of a collection or a sub-document fields
- e.g. document in a collection

```
{  
    "address": {  
        "city": "New Delhi",  
        "state": "Delhi",  
        "pincode": "110001"  
    },  
    "tags": [  
        "football",  
        "cricket",  
        "badminton"  
    ],  
    "name": "Ravi"  
}
```

- indexing a field in ascending order and find

```
> db.users.createIndex({"tags":1})  
> db.users.find({tags:"cricket"}).pretty()
```

- indexing a sub-document field in ascending order and find

```
> db.users.createIndex({"address.city":1,"address.state":1,"address.pincode":1})  
> db.users.find({"address.city":"New Delhi"}).pretty()
```

MongoDB: Joins

Mongo 3.2+ it is possible to join data from 2 collections using aggregate

If you have two collections (users , comments) and want to pull all the comments with pid=444 along with the user info for each comments

```
{ uid:12345, pid:444, comment="blah" }  
{ uid:12345, pid:888, comment="asdf" }  
{ uid:99999, pid:444, comment="qwer" }
```

users

```
{ uid:12345, name:"john" }  
{ uid:99999, name:"mia" }
```

Join command - Join using \$lookup

```
db.users.aggregate({  
  $lookup:{  
    from:"comments",  
    localField:"uid",  
    foreignField:"uid",  
    as:"users_comments"  
  }  
})
```

MongoDB: MapReduce

orders collection

Sample document:

```
{ _id: 1, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-01"), price: 25, items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 } ], status: "A" }
```

Define the **map function** to process each input document:

```
var mapFunction1 = function() {
  emit(this.cust_id, this.price);
};
```

Define the **reduce function** with two arguments keyCustomerId and valuesPrices:

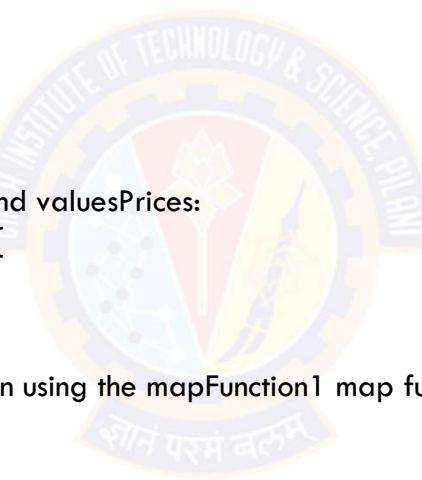
```
var reduceFunction1 = function(keyCustomerId, valuesPrices) {
  return Array.sum(valuesPrices);
};
```

Perform map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function:

```
db.orders.mapReduce(
  mapFunction1,
  reduceFunction1,
  {out: "map_reduce_example"}
)
```

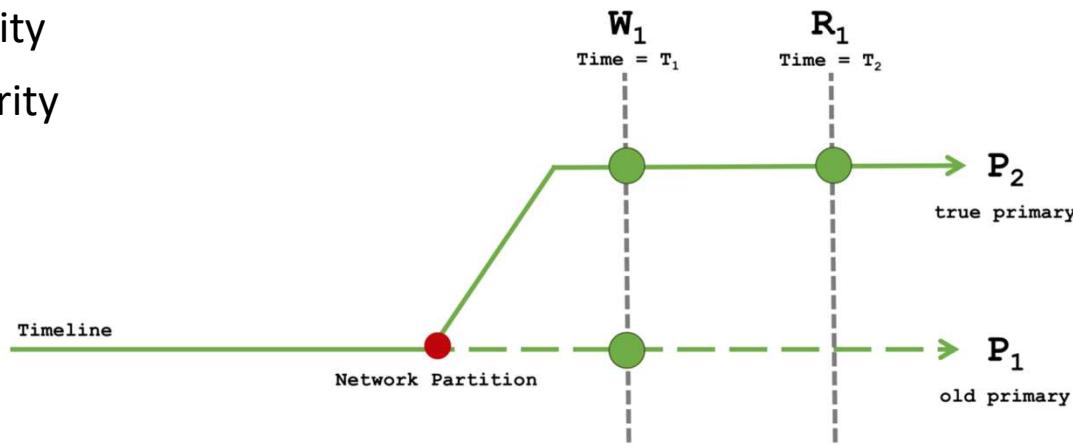
Query the map_reduce_example collection to verify the results:

```
db.map_reduce_example.find().sort({id:1});
```

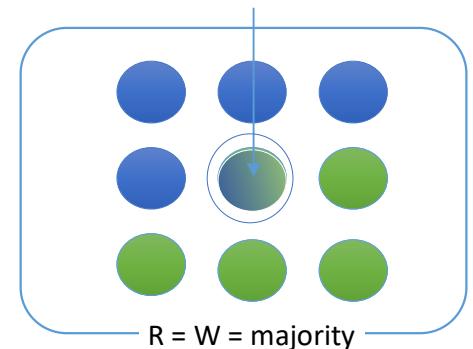


Consistency scenarios - causally consistent and durable

read=majority
write=majority



Read latest written value
from common node



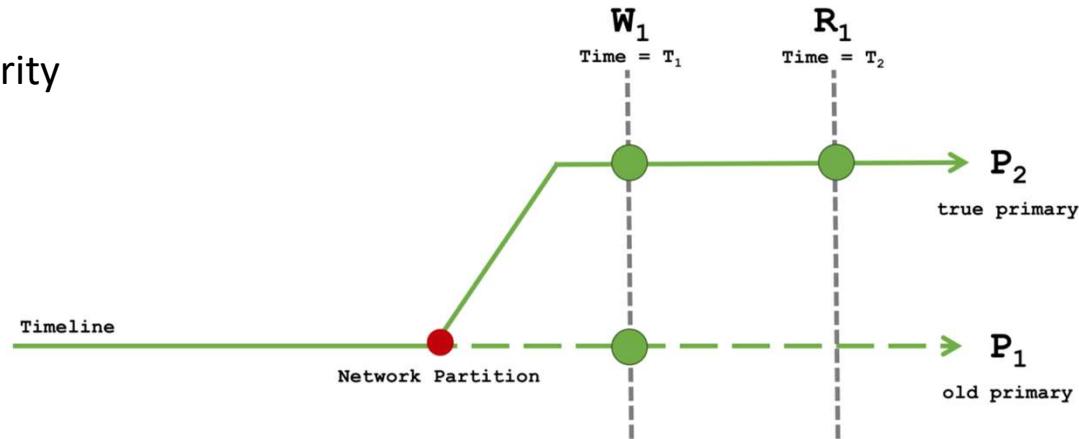
- W_1 and R_1 for P_1 will fail and will succeed in P_2
- So causally consistent, durable even with network partition sacrificing performance
- *Example:* Used in critical transaction oriented applications, e.g. stock trading

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

Consistency scenarios - causally consistent but not durable

read=majority

write=1



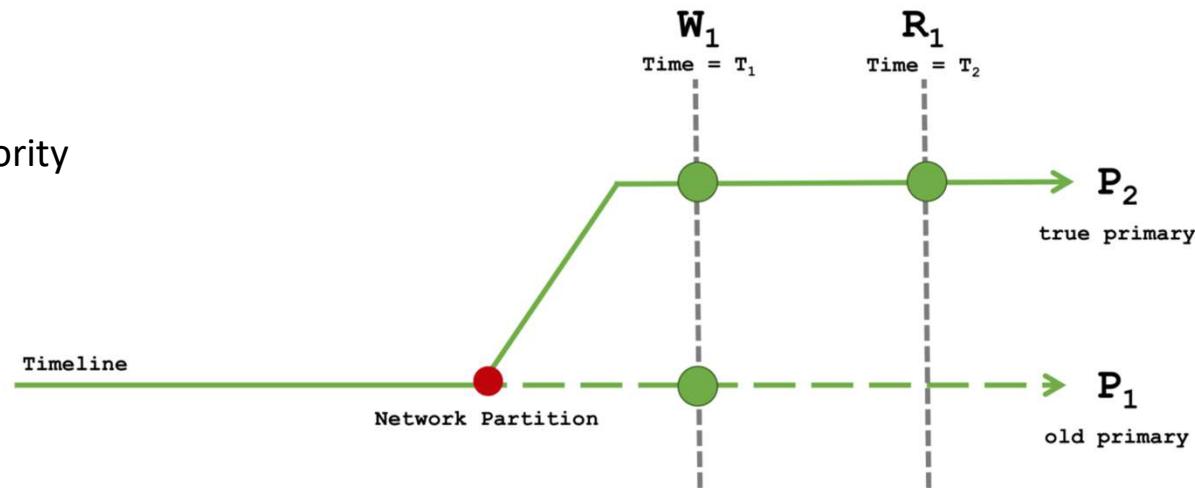
- W_1 may succeed on P_1 and P_2 . R_1 will succeed only on P_2 . W_1 on P_1 may roll back.
- So causally consistent but not durable with network partition. Fast writes, slower reads.
- *Example:* Twitter - a post may disappear but if on refresh you see it then it should be durable, else repost.

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

Consistency scenarios: eventual consistency with durable writes

read=local

write=majority



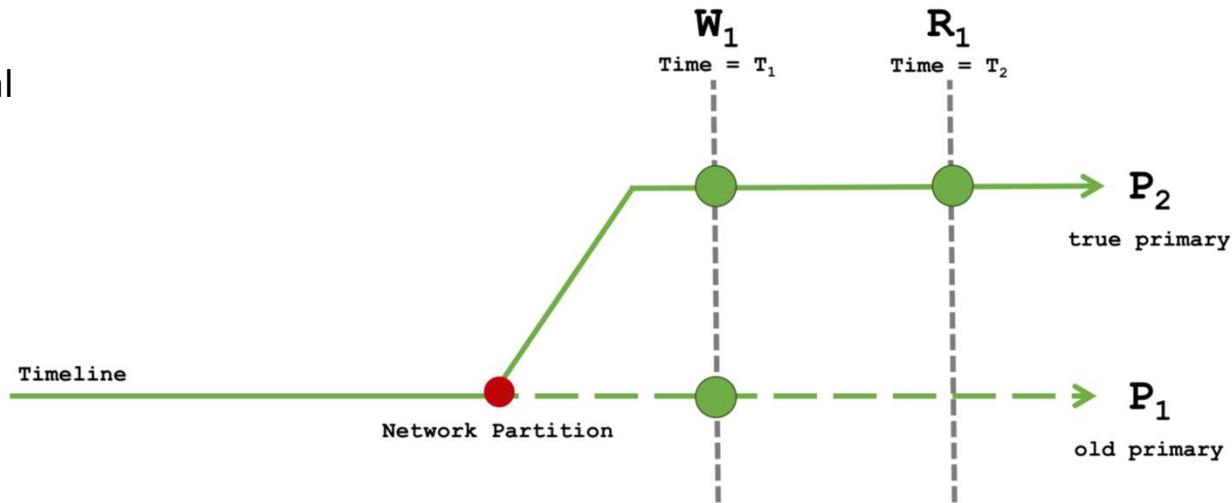
- W_1 will succeed only for P_2 and will not be accepted on P_1 after failure. Reads may not succeed to see the last write on P_1 . Slow durable writes and fast non-causal reads.
- *Example:* Review site where write should be durable if committed but reads don't need causal guarantee as long as it appears some time (eventual consistency).

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

Consistency scenarios - eventual consistency but no durability

read=local

write=1



- Same as previous scenario and not writes are also not durable and may be rolled back.
- *Example:* Real-time sensor data feed that needs fast writes to keep up with the rate and reads should get as much recent real-time data as possible. Data may be dropped on failures.

<https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3>

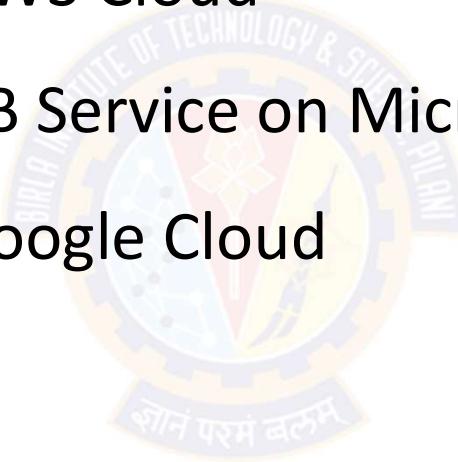
MongoDB – ACID Transactions

Can NoSQL databases be ACID-compliant?

- MongoDB is an ACID-compliant database.
- From MongoDB 4.0 onwards, there is support for ACID Transactions
✓ <https://www.mongodb.com/products/capabilities/transactions>
- Version 4.2 even brought distributed multi-document ACID transactions for even more flexibility - 2019.

MongoDB on Cloud

- MongoDB Atlas on AWS Cloud
- Automated MongoDB Service on Microsoft Azure
- MongoDB Atlas on Google Cloud



Import / Export data to / from MongoDB

Import data from a CSV file

Given a CSV file “sample.txt” in the D: drive, import the file into the MongoDB collection, “SampleJSON”. The collection is in the database “test”.

```
mongoimport --db test --collection SampleJSON --type csv --headerline --file d:\sample.txt
```



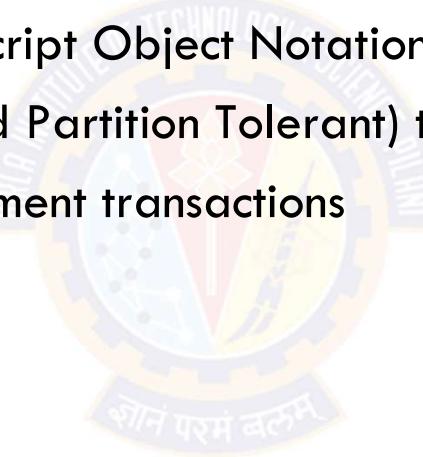
Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from “Customers” collection in the “test” database into a CSV file “Output.txt” in the D: drive.

```
mongoexport --db test --collection Customers --csv --fieldFile d:\fields.txt --out d:\output.txt
```

MongoDB in a nutshell

- MongoDB is a non-relational. Open source, Distributed database
- It stores data into JSON (Java Script Object Notation) documents
- It adheres to CP (Consistency and Partition Tolerant) traits of Brewer's CAP Theorem
- It has NO support for multi-statement transactions
- It supports embedded documents
- It practices automatic sharding



MongoDB Certification Trainings

Free MongoDB courses - MongoDB University -

<https://learn.mongodb.com/>



Handson with MongoDB

MongoDB Installation steps:

<https://edwinskyby.medium.com/installing-mongodb-in-linux-fedora-38-47e7be9f5e35>

1. To enable and start the MongoDB service run:

```
$ sudo systemctl enable mongod.service  
$ sudo systemctl start mongod.service
```

2. Check MongoDB's current status

```
$ sudo systemctl status mongod.service
```

3. Test MongoDB connection

Run Mongo shell to test the connection:

```
$ mongo
```

Then type >db

Mostly you will see test. a default database.

```
use test      - Switch to db test
```

All done start playing with the Mongo collection.

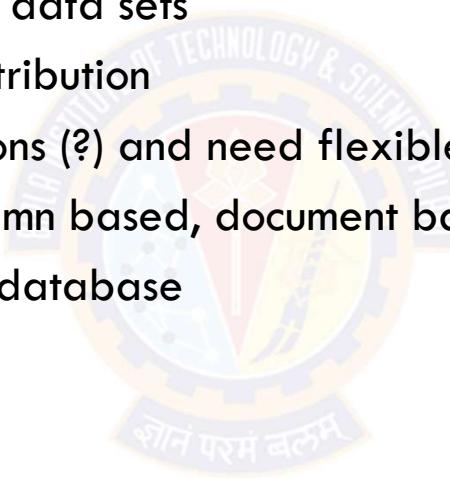
```
show dbs;
```

```
show collections;
```



Summary

- NoSQL databases are useful when
 - ✓ you have to deal with large data sets
 - ✓ may need geographical distribution
 - ✓ No need for ACID transactions (?) and need flexible consistency
- Choices between key-value, column based, document based, graph based data stores
- MongoDB is a typical document database





Next Session: **NoSQL Database - Cassandra**