Document Databases and **Mongo DB**

pm_jat @ daiict

Key Value Databases

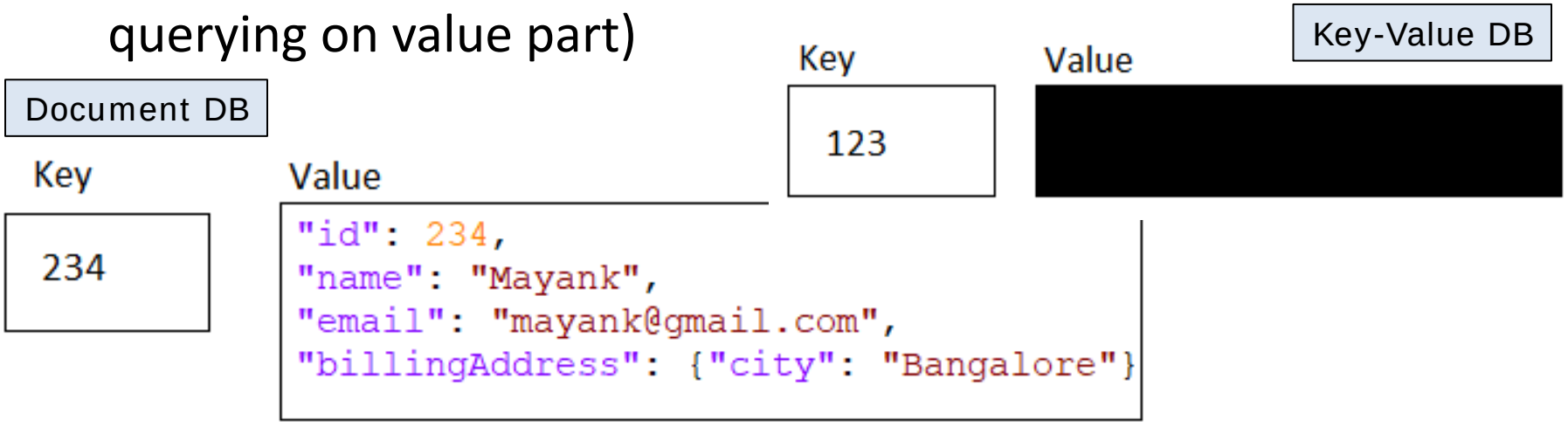
- Data Model
  - Key and Value
  - Value is opaque
  - How much “Schema Less” / Flexible Schema?
- Operations
  - GET
  - PUT

Dynamo DB as KV Database

- Also allows
  - Batch GET and Batch PUT
  - Querying “non-Key” attributes
    - Through Secondary Indexes
    - “QUERY” operation
    - “SCAN” operations
    - Some additional filtering, aggregation at “coordinator” (think of as master node of MR)
  - Note that Dynamo DB allows specifying only limited schema, i.e. attributes that are either part of Key or Index!
  - Peek into Dynamo DB implementation bit later!

Recall KV and Document Databases

- Recall Document databases are also built on top of “Key-Value” strategy and there is a thin boundary between KV and Document databases!
- A KV database does not know anything about value part, an does not perform ay operation on value.
- It just puts and get value as block. Whereas a document database is aware of value part and can operations (primarily querying on value part)



Document Databases

- Recall Document databases are also built on top of “Key-Value” strategy.
- Document databases use the concept of “Document”
- Document is an “Aggregate Object”, and typically is in the form of JSON, BSON, XML, etc.
- Table here shows correspondences of Document Databases with relational databases

RDB	Doc DB
Database/Schema	Database
Table	“Collection” of Documents
Row/Tuple	“Document”
Row ID	_id

“Document” in Document databases?

- Document is an “Aggregate Object”, and typically is in the form of JSON, BSON, XML, etc

Source: https://docs.mongodb.com/manual/introduction/

“Document” in Document databases?

```
{
  "order": {
    "orderNo":123,
    "customerId":1,
    "orderItems":[
      {
        "productId":27,
        "price": 325,
        "cat":"book",
        "productName": "NoSQL Distilled"
      },
      {
        "productId":19,
        "price": 550,
        "cat":"computer accessories",
        "productName": "Pen Drive : 128 GB"
      }
    ],
    "shippingAddress":[{"city":"Ahmedabad"}],
```

Document Databases

- In Document databases, Document is like a row in a Relational, and a collection of similar documents is called "**Collection**"
- Here database is a collection of "Document Collections"
- Also, Documents can be "nested" like a document can house a sub document or a collection of sub-documents.

Documents in Doc DBs

- Document is a “Aggregation” in the form of JSON, BSON, XML etc, and
- Document is a
  - **hierarchical tree data structures** which can consist of maps, collections, and scalar values
  - **self-describing** (attribute name, type, and value)
  - Documents stored in a collection are typically similar to each other but do not have to be exactly same in terms of “schema (attributes and types)”– **flexible schema**!

Popular Document DBs[1]

- Mongo DB (more towards “Consistency”)
- Couch DB (more towards “Availability”)
- Terrastore
- OrientDB
- RavenDB

Mongo DB

PostgreSQL	Mongo DB
Schema	Database
Table	Collection
Row	Document
OID	_id
FK	references

- Developed by 10gen (now MongoDB, Inc)
  - Founded in 2007
- A Mongo DB database is collection of “document collections”
- Document is stored as “BSON” (Binary JSON)
- Table here shows MongoDB and PostgreSQL correspondences.

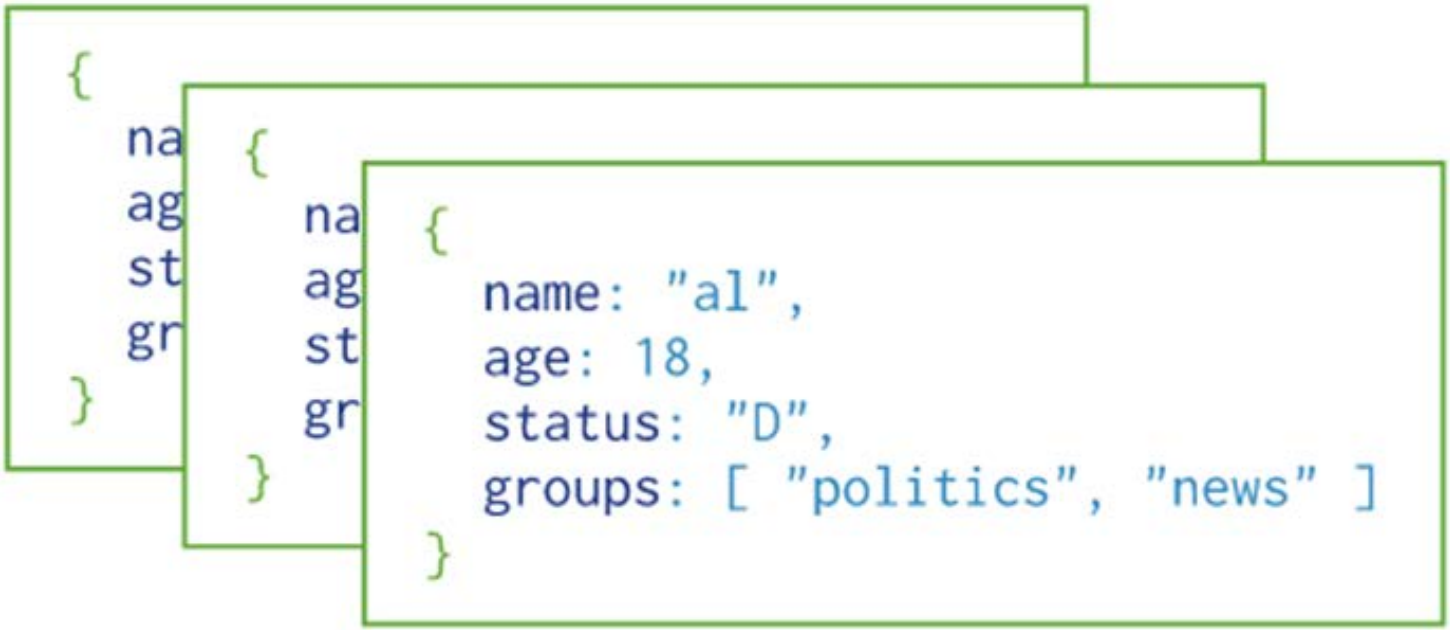
Mongo DB Key Features

- "Aggregated" data models ("Aggregation oriented")
- Full Index supports- can include keys from embedded documents and arrays (Note dynamo db did not allow this)
- Rich Query interface (through API)
- High Availability : Replicated data provides automatic failover
- Horizontal Scalability
  - Mongo DB provides horizontal scalability as part of its core functionality
  - Sharding distributes data across a cluster of machines based on "shard key".

Source: https://docs.mongodb.com/manual/introduction/

Mongo DB “Collection”

- MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
- Collection can be read as collection of “similar” document – not necessarily in terms of strict schema.



Source: https://docs.mongodb.com/manual/core/databases-and-collections/

Mongo DB Data Models

- Key-Value strategy
- Every document (Object) has a “Key”
- One of the major decision in no-sql database design is if related objects are “embedded” or are stored externally in separate collection or table.
  - Note: This is very much true in case of Dynamo DB as well!
- Mongo DB literature call them as

  (1) “Embedded, or “de-normalized” data models

  (2) Normalized Data Models

Source: https://docs.mongodb.com/manual/data-modeling/

Mongo DB Data Models

- **Embedded Data Models**
  - Related data are embedded in a single structure or document.
  - These schema are generally known as “de-normalized” models, and take advantage of MongoDB’s rich documents.
- **Normalized Data Models**
  - Normalized data models describe relationships using references between documents.

Source: https://docs.mongodb.com/manual/data-modeling/

Embedded Data Models

- When related objects are “stored” in a object. For example!

Source: https://docs.mongodb.com/manual/data-modeling/

Embedded Data Models

- Benefits
  - Certain “Use Case” Querying are more efficient; related data are easy to pull
- Downside
  - Querying is good of certain use cases where as very poor for other use cases (Recall discussion on other day, we shall get back to this bit later)
- Mongo DB Embedding Limitation: Maximum Object Size is 16MB!

Source: https://docs.mongodb.com/manual/data-modeling/

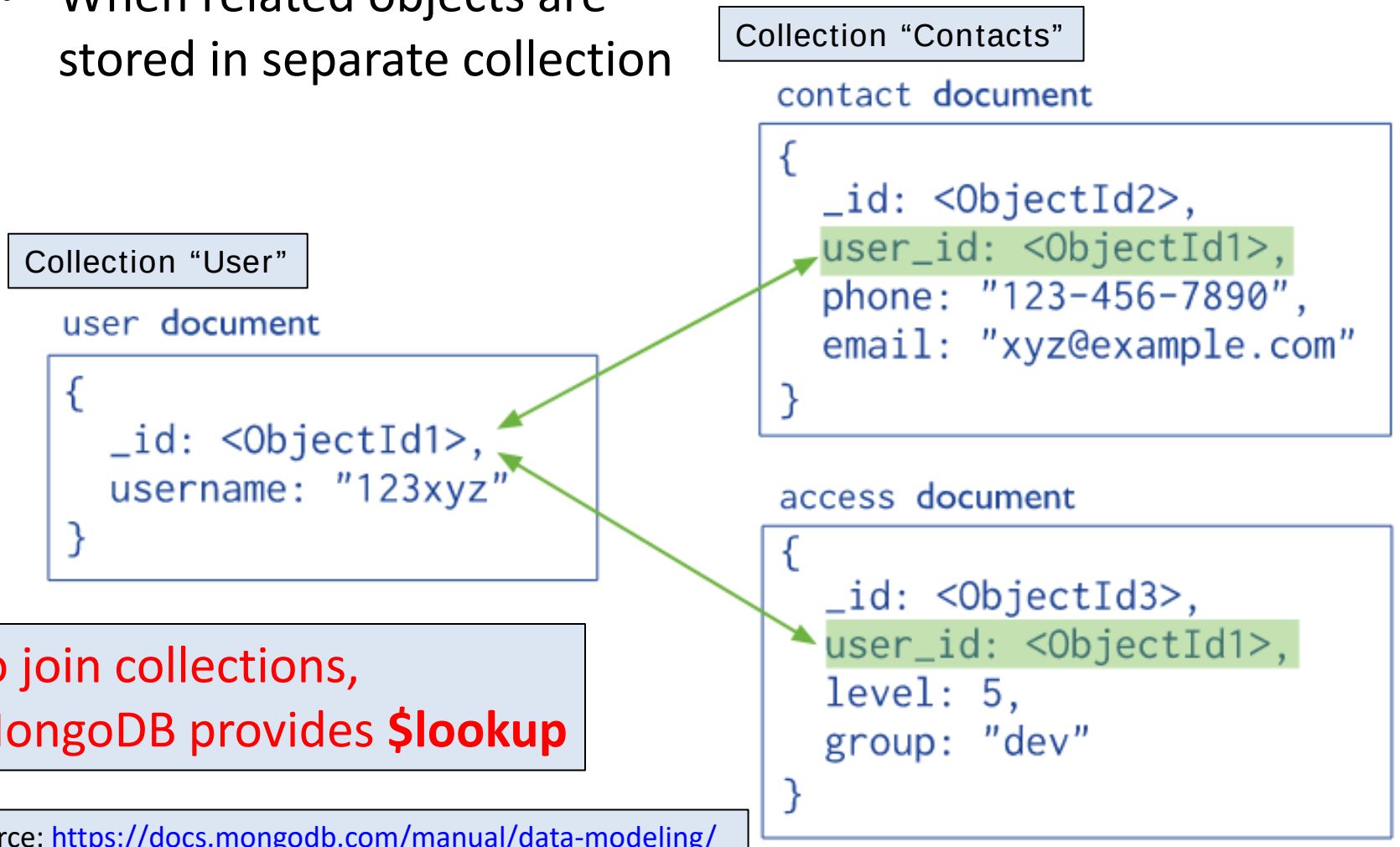
```
{  title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
         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

Another Example: Publisher documents (objects) are embedded in Book documents (objects)

```
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"
            }
```

Normalized Data Models

- When related objects are stored in separate collection



To join collections, MongoDB provides **$lookup**

Source: https://docs.mongodb.com/manual/data-modeling/

Normalized Data Models

- In embedding approach related objects are housed within a objects, however
- In case of normalized approach, relationships are represented in different ways-
  - One side storing list of references of on side objects
  - Many side storing references of one side objects
  - Or storing references in both sides

Source: https://docs.mongodb.com/manual/data-modeling/

Normalized Design Pattern #1

Reference List is stored in one side of relationship

Collection “Books”

```
{
    _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 Tri
    author: "Kristina Chodo
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English"
```

Collection “Publisher”

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [123456789, 234567890, ...]
}
```

Source: https://docs.mongodb.com/manual/data-modeling/

Normalized Design Pattern #2

References are stored in many side of the relationship

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

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

Source: https://docs.mongodb.com/manual/data-modeling/

Normalized Data Models

- Normalized data models are less biased towards “query load”? (Strength of relational databases)
- In general, normalized data models are used
  - when duplication of data outweighs the advantage of embedding.
  - to deal with varying query load
  - to represent more complex many-to-many relationships.
  - to model large hierarchical data sets.

Source: https://docs.mongodb.com/manual/data-modeling/

Schema and Databases

- No SQL databases pronounce to be “schema free”, “flexible schema”
- So what does it mean?
- Let us try understanding this:
  - What is Schema and What is Fixed Schema Databases?
  - What are the benefits of Schema
  - What is the “Cost of Schema”
  - Why do we want to have Schema free Databases?

Schema and Databases

- What is database schema?
- What is Fixed Schema Databases?

Schema and Databases

- What is database schema?
  - "Database structure", and "constraints", OR
  - "Table structure" and constraints
  - Schema is stored as part of database as "meta-data"
- Traditional **DBMS require us defining schema before any data can be put into it**.
- Database systems make sure that any data added to database comply the constraints defined in the schema.
- Validity of data is CHECKED before putting into database
- There is term around for this "**Load time data validation as per schema**"
- Such databases are also called as "Strict/Fixed schema" databases

Schema-less/Flexible Schema Databases

- What do we mean by Schema Less?
- A strictest understanding of this is no schema information at all; i.e. each element in a collection (row/item in a table , or a document in document collection) has its own schema.
- Flexible schema databases mean, we have some schema **defined at table/collection level**, and rest can be different for different elements in a collection
- We have seen what is flexible schema in Dynamo DB? At table level, we only have: Primary Key, and Indexes defined. Every item required to comply only this part, rest are specific to a data Items.

Schema-less/Flexible Schema Databases

- Why Schema-less/Flexible Schema is becoming important?
- In Big Data era: figuring out schema in advance is becoming impractical
- More and more column may get added with the time
- Some columns may be becoming irrelevant with the time
- Creating databases in “fixed schema” approach have problems of
  - Too many columns, and every row having values only for few of them.
  - Database becomes sparse with lots of null values

Problems with schema-less-ness

- Totally schema-less can be disastrous?
  - Two data objects having two names quantity attribute - qty for quantity; how do we reconcile this?
  - “3” and “three” as value for an attributes
- DBMS can not do any kind of validation for such anomalies in absence of schema?
- So, we require having a tradeoff between strict schema and schema less.
- Sometime schema is “**Implicit**” only. That is DBMS does not know anything about schema. Where as “application programmer” can assume some schema and access accordingly.
- Again, DBMS can not perform any kind of validation for implicit schema.
- Also DBMS performance of database operation gets affected in absence of schema.

MongoDB and Schema-less-ness

- MongoDB's collections, by default, does not require its documents to have the same schema, i.e. documents in a collection
  - Field names can different
  - data type for a field can differ across documents within a collection
- This means each document in a collection can have its own schema?
- However we can specify some partial schema at collection level? i.e. required fields for every document, their "global data type", and so on
- "Schema of a collection" (or document) can be altered later also?

MongoDB and Schema

- We can define certain (Schema) rules at Collection level.
- Mongo DB calls them “Document Validation Rules”.
  - This can be read as Rule that every document in a collection should comply.
- Document Validation rules can be specified at Collection Creation time or can be specified later through alter collection command.

References

[1] Chapter 9, NoSQL Distilled

[2] https://docs.mongodb.com/manual/