

XML: Features and Comparison



- Standard ways to specify XML document **schema**:
 - DTD, XML Schema (XSD), etc.
 - concept of Namespaces; XML editors (for given schema)
- Technologies for **parsing**: DOM, SAX
- **Many** associated **technologies**:
 - XPath, XQuery, XSLT (transformation)
- XML is **great** for **configurations**, **meta-data**, etc.
- **XML databases** are mature, **not** considered NoSQL
- Currently, **JSON** format **rules**:
 - **compact**, **easier** to write, has all features typically needed

MongoDB



- Initial release: 2009

- Written in C++
- Open-source
- Cross-platform

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

- JSON documents

- Basic **features**:

- High **performance** – many indexes
- High **availability** – replication + eventual consistency + automatic failover
- Automatic **scaling** – automatic sharding across the cluster
- **MapReduce** support

MongoDB: Terminology



RDBMS	MongoDB
database instance	database
schema	---
table	collection
row	document
rowid	_id

- each JSON **document**:
 - belongs to a **collection**
 - has a field **_id**
 - unique within the collection
- each collection:
 - belongs to a “**database**”

```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

Documents



- Use **JSON** for API **communication**
- Internally: **BSON**
 - **Binary** representation of JSON
 - For storage and inter-server communication
- Document has a **maximum size: 16MB** (in BSON)
 - Not to use too much RAM, bandwidth
 - GridFS tool can divide larger files into fragments

Document Fields



- Every **document** must have the field **_id**
 - Used as a **primary** key
 - **Unique** within the collection
 - **Immutable**
 - Any **type** other than an array
 - Can be **generated** automatically
- Restrictions on **field names**:
 - The field names **cannot** start with the **\$** character
 - Reserved for operators
 - The field names **cannot** contain the **.** character
 - Reserved for accessing sub-fields

Database Schema



- Documents have **flexible schema**
 - Collections do **not enforce** specific data structure
 - In practice, documents in a collection are similar
- Key **decision** of data modeling:
 - References vs. embedded documents
 - In other words: Where to draw lines between **aggregates**
 - Structure of data
 - Relationships between data

Schema: Embedded Docs



- Related data in a **single document** structure
 - Documents can have **subdocuments** (in a field or array)

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

Embedded sub-document

Embedded sub-document

Schema: Embedded Docs (2)

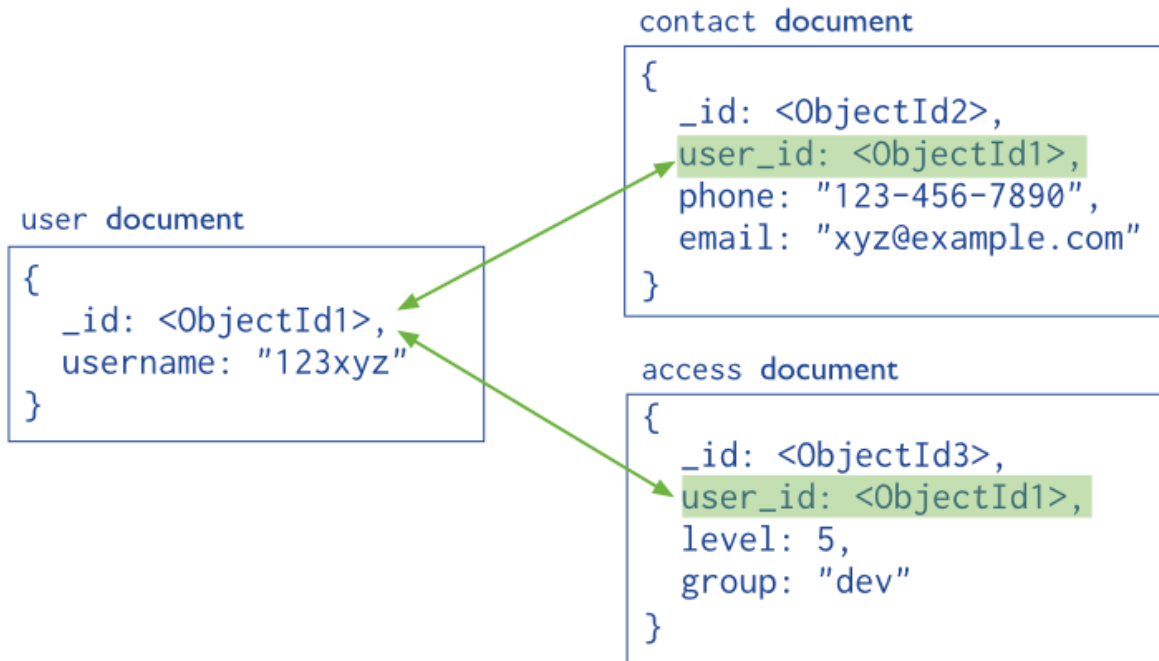


- **Denormalized** schema
- Main **advantage**:
Manipulate related data in a **single operation**
- **Use** this schema **when**:
 - **One-to-one** relationships: one doc “contains” the other
 - One-to-many: if children docs have **one parent** document
- **Disadvantages**:
 - Documents may **grow** significantly during the time
 - Impacts both read/write performance
 - Document must be **relocated** on disk if its **size exceeds** allocated space
 - May lead to data **fragmentation** on the disk

Schema: References



- Links/**references** from one document to another
- **Normalization** of the schema



Schema: References (2)



- More **flexibility** than embedding
- **Use** references:
 - When **embedding** would result in **duplication** of data
 - and only insignificant boost of read performance
 - To represent more **complex** many-to-many **relationships**
 - To model large hierarchical data sets
- **Disadvantages:**
 - Can require **more roundtrips** to the server
 - Documents are accessed one by one

Querying: Basics



- Mongo query language
- A MongoDB **query**:
 - Targets a specific **collection** of documents
 - Specifies **criteria** that identify the returned documents
 - May include a **projection** to **specify** returned **fields**
 - May impose limits, sort, orders, ...
- Basic query - all documents in the collection:

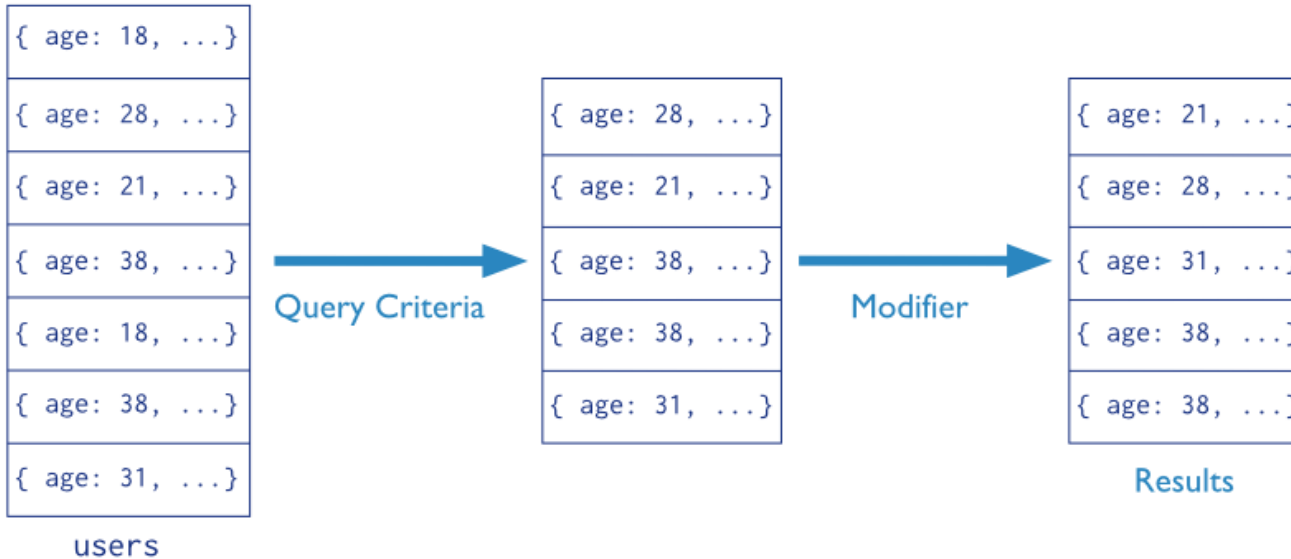
```
db.users.find()
```

```
db.users.find( {} )
```

Querying: Example



Collection Query Criteria Modifier
`db.users.find({ age: { $gt: 18 } }).sort({age: 1 })`



Querying: Selection



```
db.inventory.find( { type: "snacks" } )
```

- All documents from collection **inventory** where the **type** field has the value **snacks**

```
db.inventory.find( { type: { $in: [ 'food', 'snacks' ] } } )
```

- All **inventory** docs where the **type** field is either **food** or **snacks**

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

- All ... where the **type** field is **food** and the **price** is **less than 9.95**

Inserts



```
db.inventory.insertOne( { _id: 10, type:
"misc", item: "card", qty: 15 } )
```

- Inserts a document with three fields into collection **inventory**
 - User-specified **_id** field

```
db.inventory.insertOne( { type: "book", item:
"journal" } )
```

- The database generates **_id** field

```
$ db.inventory.find()
```

```
{ _id: ObjectId("58e209ecb3e168f1d3915300"),
type: "book", item: "journal" }
```

Updates



```
db.inventory.updateMany(  
  { type: "book", item : "journal" },  
  { $set: { qty: 10 } },  
  { upsert: true } )
```

- Finds all docs matching query

```
{ type: "book", item : "journal" }
```
- and sets the field

```
{ qty: 10 }
```
- `upsert: true`
 - if no document in the `inventory` collection matches
 - creates a new document (generated `_id`)
 - it contains fields `_id`, `type`, `item`, `qty`

MapReduce



```
collection "accesses":
{
  "user_id": <ObjectId>,
  "login_time": <time_the_user_entered_the_system>,
  "logout_time": <time_the_user_left_the_system>,
  "access_type": <type_of_the_access>
}
```

- How much time did **each user** spend logged in
 - Counting just accesses of type “regular”

```
db.accesses.mapReduce(
  function() { emit (this.user_id, this.logout_time - this.login_time); },
  function(key, values) { return Array.sum( values ); },
  {
    query: { access_type: "regular" },
    out: "access_times"
  }
)
```



In JavaScript



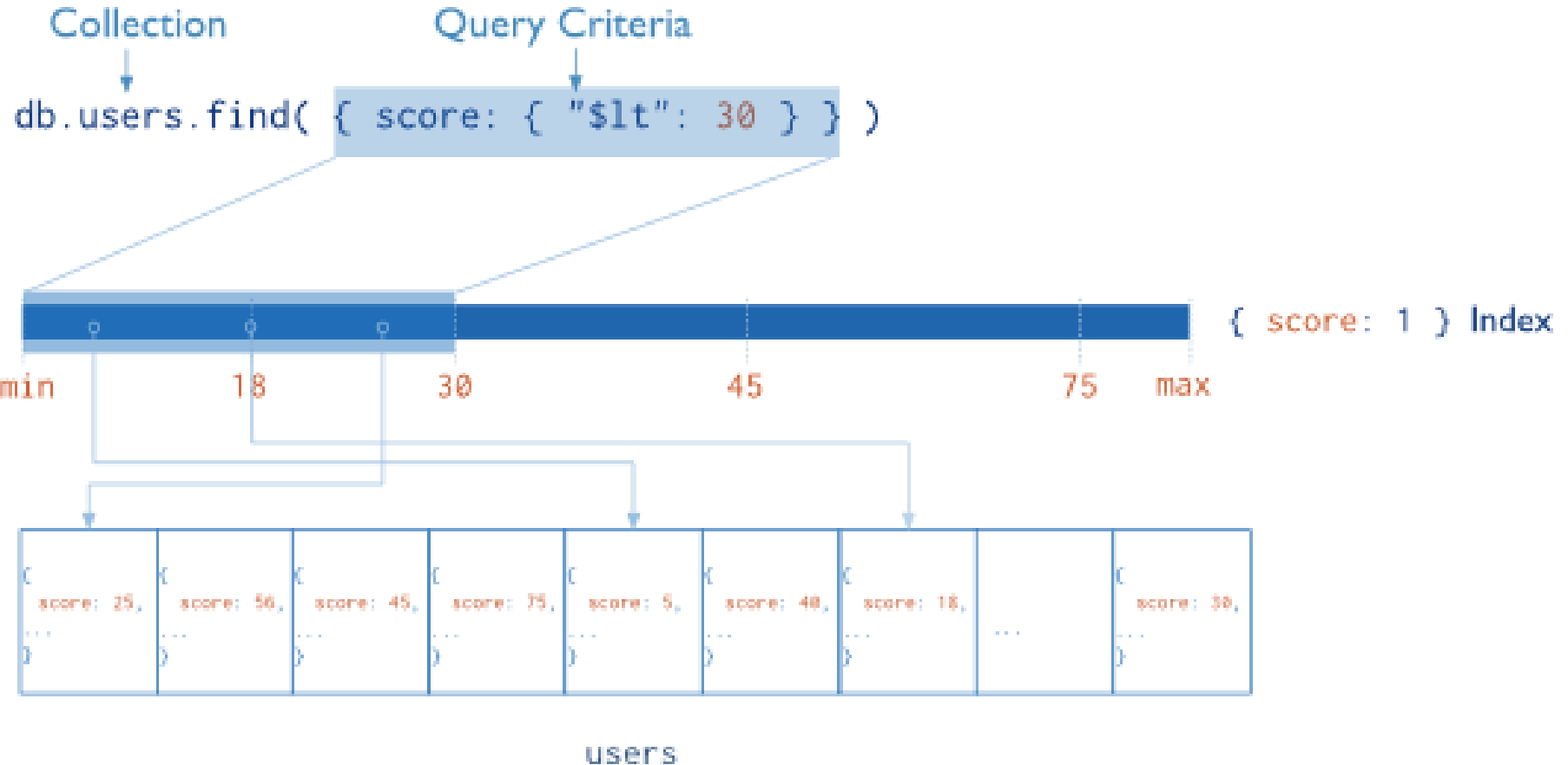
Part 2.2: MongoDB - Indexes

Indexes

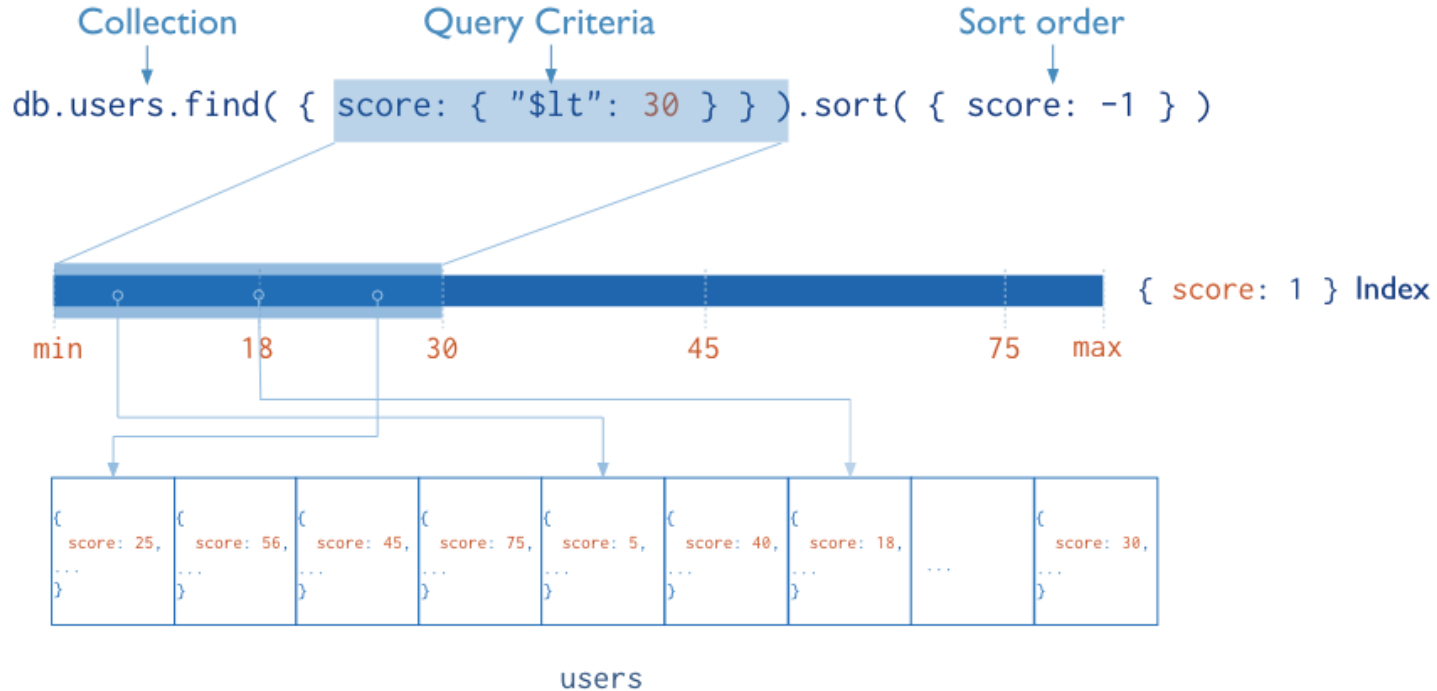


- **Indexes** are the key for MongoDB performance
 - **Without** indexes, MongoDB must **scan every** document in a collection to **select** matching documents
- **Indexes** store some fields in easily accessible form
 - Stores values of a specific field(s) ordered by the value
- Defined per **collection**
- Purpose:
 - To **speed up** common queries
 - To optimize **performance** of other specific operations

Indexes: Example of Use

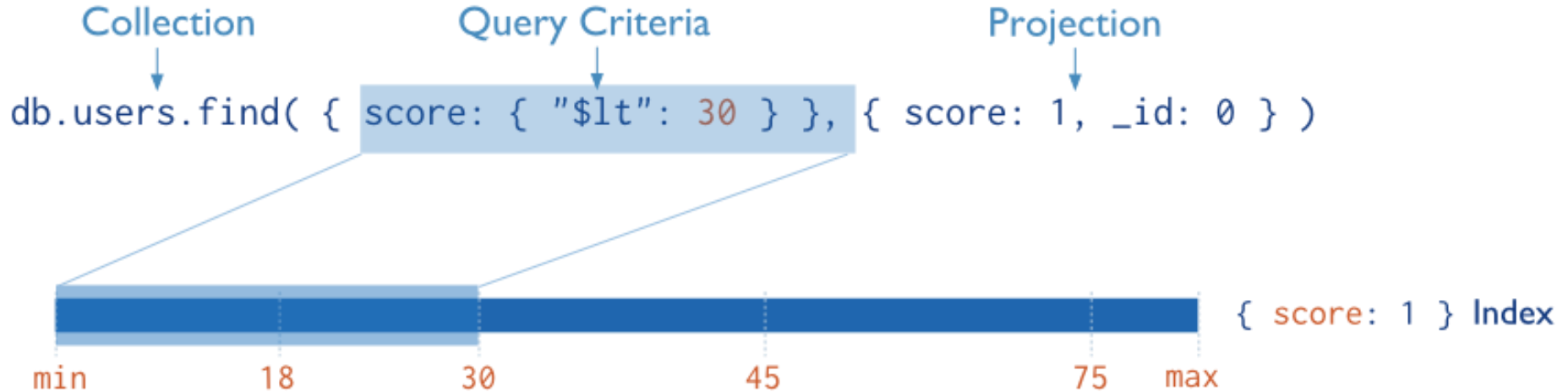


Indexes: Example of Use (2)



- The **index** can be **traversed** in order to return **sorted** results (**without** sorting)

Indexes: Example of Use (3)



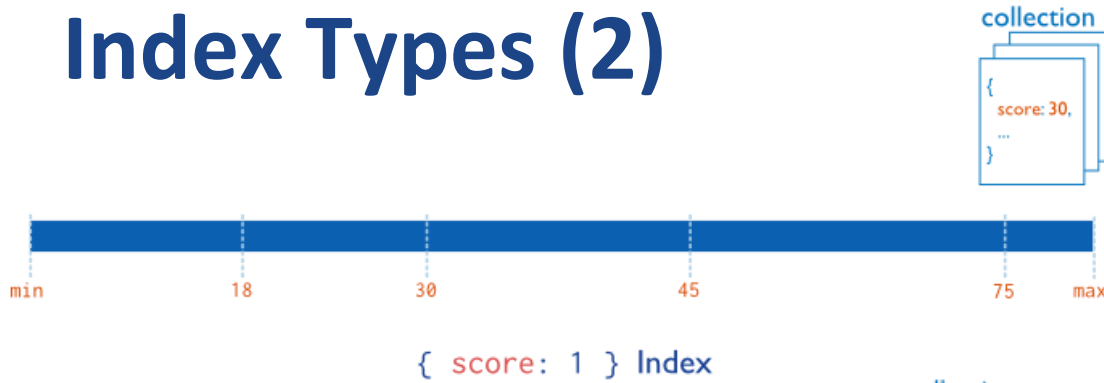
- MongoDB does **not** need to inspect data **outside** of the index to fulfill the query

Index Types

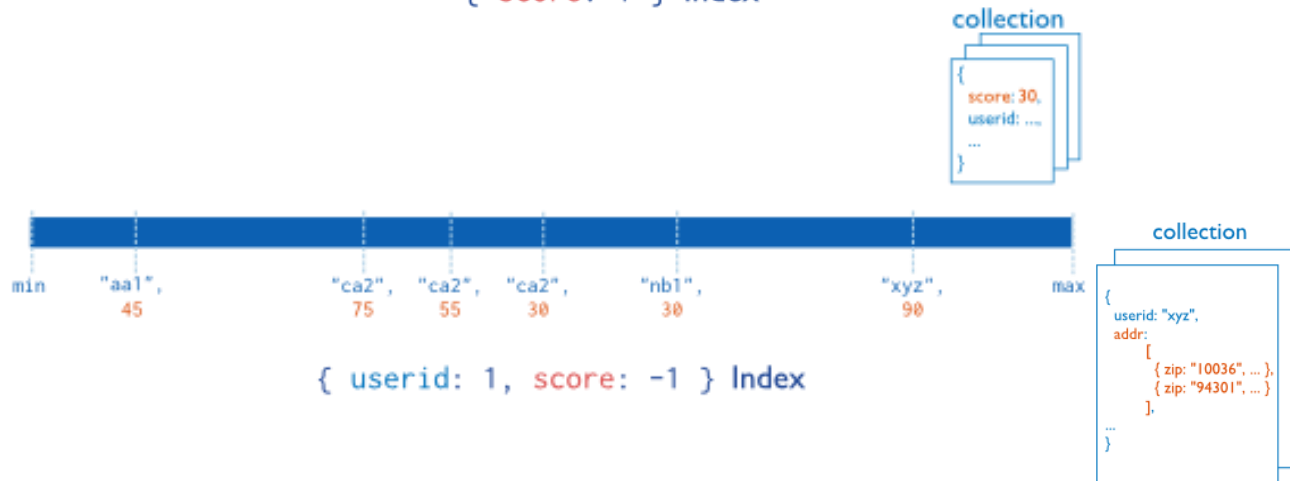


- **Default: `_id`**
 - Exists by default
 - If applications do not specify `_id` when inserting a doc, it is created.
 - Unique
- **Single Field**
 - **User-defined** indexes on a single field of a document
- **Compound**
 - User-defined indexes on **multiple** fields
- **Multikey index**
 - To index the content stored in **arrays**
 - Creates separate **index entry** for **each** array **element**

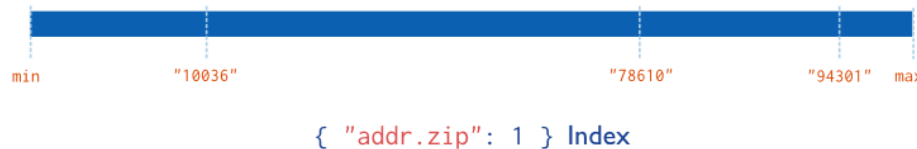
Index Types (2)



- Index on `score` field (ascending)



- Compound Index on `userid` (ascending) AND `score` field (descending)



- **Multikey** index on the `addr.zip` field

Index Types (3)



- **Ordered Index**
 - B⁺-Tree (see above)
- **Hashed Indexes**
 - **Fast** $O(1)$ indexes the hash of the value of a field
 - Only **equality** matches
- **Geospatial Index** ([operators](#) docs)
 - $2d$ indexes = use **planar geometry** when returning results
 - For data representing points on a two-dimensional plane
 - $2dsphere$ indexes = **spherical** (Earth-like WGS84) geometry
 - For data representing latitude, longitude
- **Text Indexes**
 - Searching for **string** content in a collection



Part 2.3: MongoDB - Behind the Scene

MongoDB: Behind the Scene



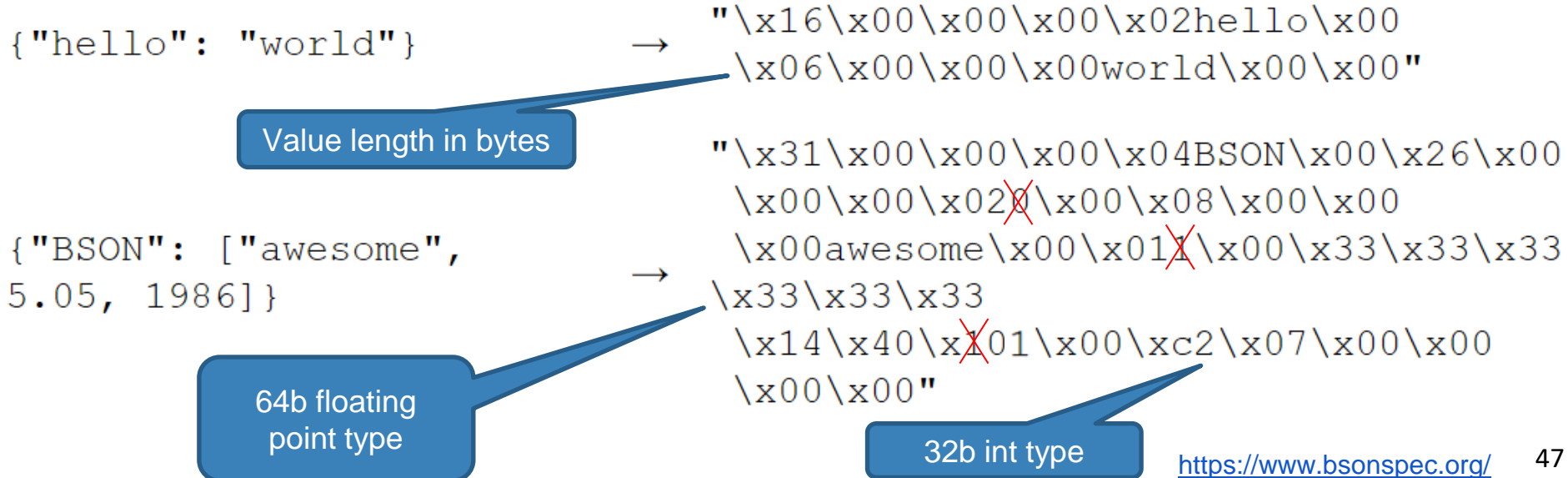
- **BSON** format
- **Distribution** models
 - Replication
 - Sharding
 - Balancing
- MapReduce
- Transactions
- Journaling

BSON (Binary JSON) Format



- **Binary**-encoded **serialization** of JSON documents
 - Representation of documents, arrays, JSON simple data types + other types (e.g., Date and BinData)

Data type
(string, here)



BSON: Basic Types



- `byte` – 1 byte (8-bits)
- `int32` – 4 bytes (32-bit signed integer)
- `int64` – 8 bytes (64-bit signed integer)
- `double` – 8 bytes (64-bit IEEE 754 floating point)

BSON Grammar



```
document ::= int32 e_list "\x00"
```

- BSON document
- `int32` = total number of **bytes** in document

```
e_list ::= element e_list | ""
```

- Sequence of elements

BSON Grammar (2)



```
element ::= "\x01" e_name double
         | "\x02" e_name string
         | "\x03" e_name document
         | "\x04" e_name document
         | "\x05" e_name binary
         | ...
```

Floating point
UTF-8 string
Embedded document
Array
Binary data
...

```
e_name ::= cstring
```

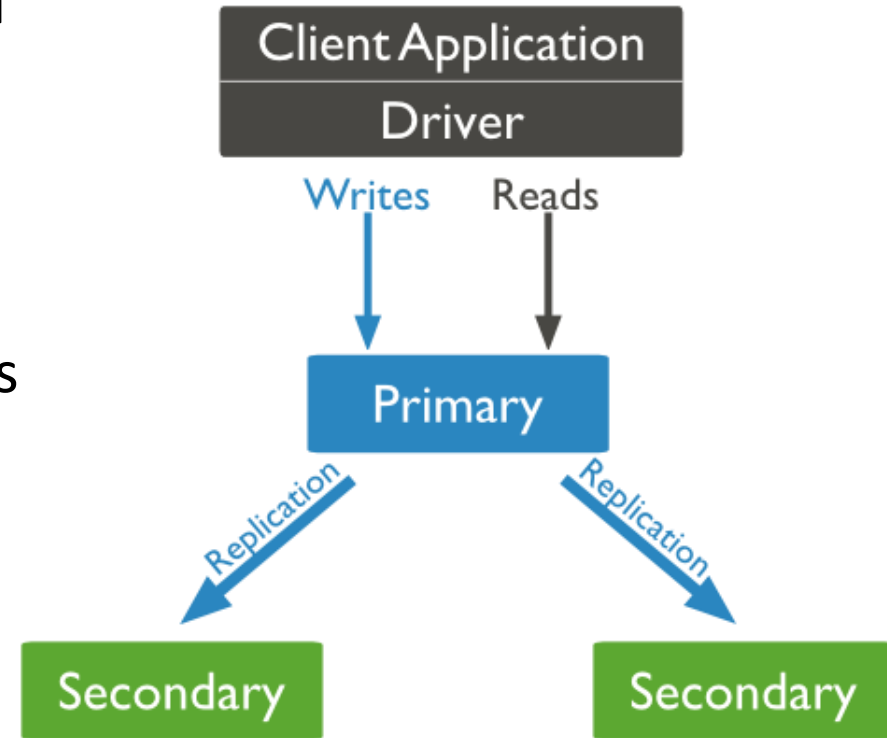
- Field **key**

```
cstring ::= (byte*) "\x00"
```

```
string ::= int32 (byte*) "\x00"
```

Data Replication

- Master/slave replication
- **Replica set** = a group of instances that host the **same data** set
 - **primary** (master) – handles all **write** operations
 - **secondaries** (slaves) – apply operations from the primary so that they have the same data set



Replication: Read & Write



- **Write operation:**

1. Write operation is applied on the **primary**
2. Operation is recorded to primary's **oplog** (operation log)
3. **Secondaries** replicate the **oplog** + **apply** the operations to their data sets

- **Read: All** replica set **members** can accept **reads**

- By **default**, application directs its reads to the primary
 - Guaranties the latest version of a document
 - **Decreases** read **throughput**
- Read **preference** mode can be **set**
 - See below

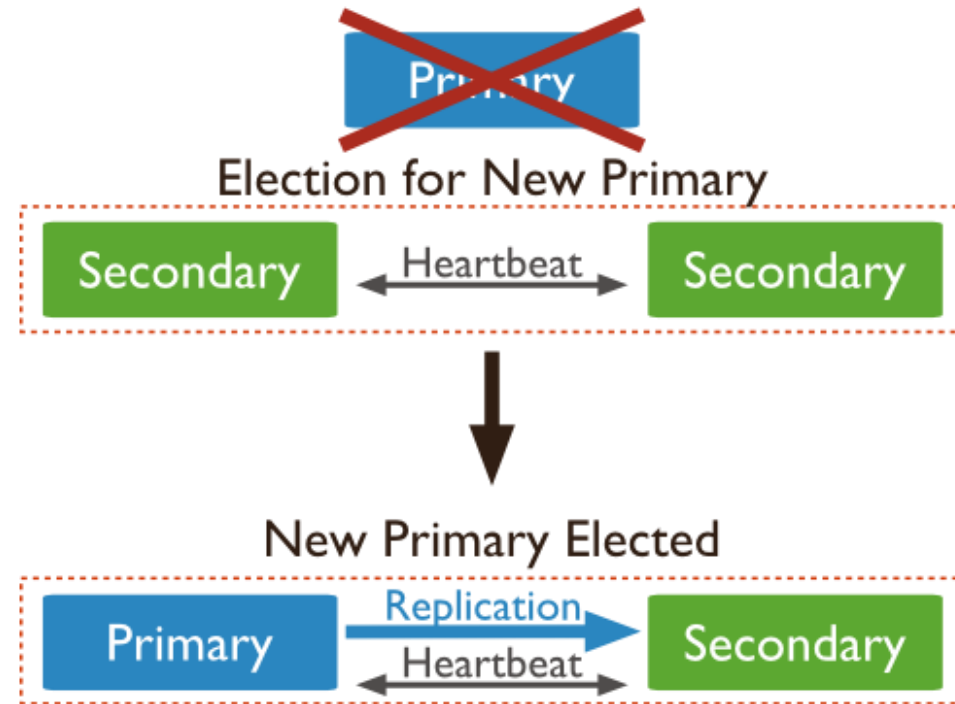
Replication: Read Modes



Read Preference Mode	Description
<code>primary</code>	operations read from the primary of the replica set
<code>primaryPreferred</code>	operations read from the primary , but if unavailable, operations read from secondary members
<code>secondary</code>	operations read from the secondary members
<code>secondaryPreferred</code>	operations read from secondary members, but if none is available, operations read from the primary
<code>nearest</code>	operations read from the nearest member (= shortest ping time) of the replica set

Replica Set Elections

- If the **primary** becomes **unavailable**, an election determines a **new primary**
 - Elections need some time
 - No primary => no writes



Replica Set: CAP

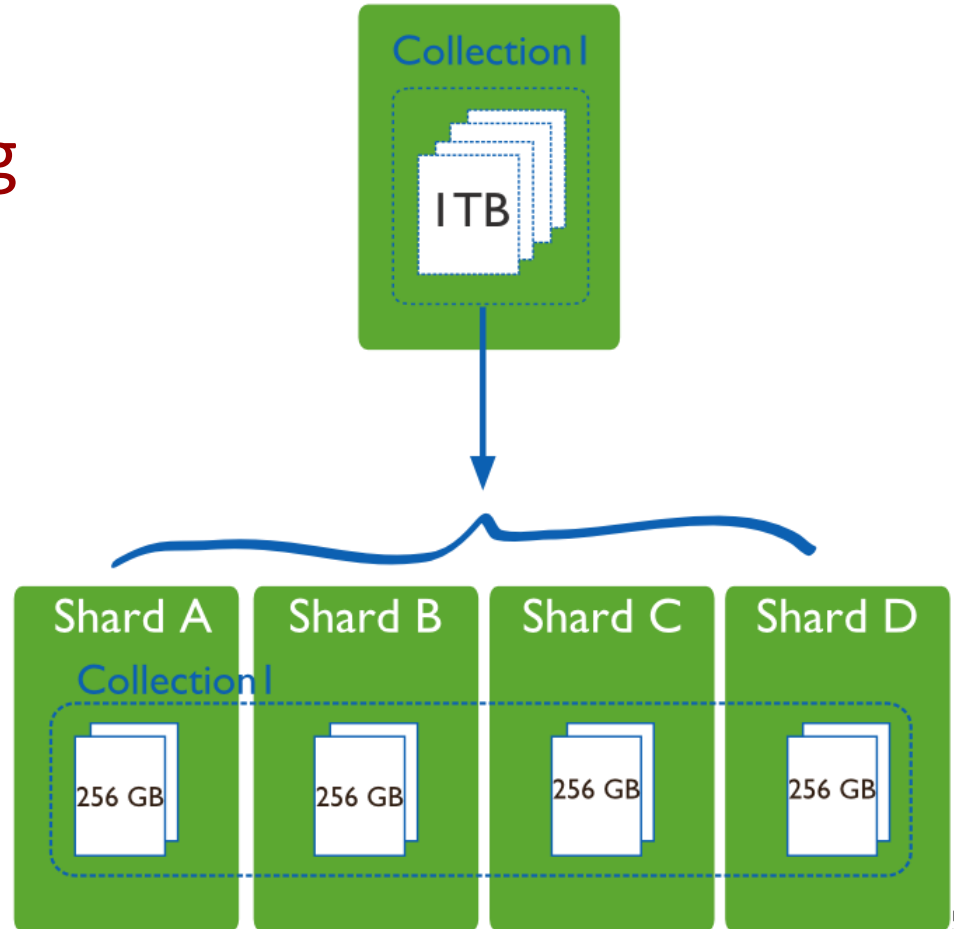


- Let us have **three** nodes in the **replica** set
 - Let's say that the **master** is **disconnected** from the other two
 - The distributed system is **partitioned**
 - The two slaves “think” that the **master failed**
 - Because they form a partition with **more than half** of the nodes
 - And elect a new master
 - The **master** finds out, that it is **alone**
 - Specifically, that can communicate with **less than half** of the nodes
 - And it steps down from being master (handles just reads)
- In case of just **two nodes** in RS
 - **Both** partitions will become **read-only**
 - Similar case can occur with any **even number of nodes** in RS
 - Therefore, we can always **add** an **arbiter** node to even-sized RS

Sharding



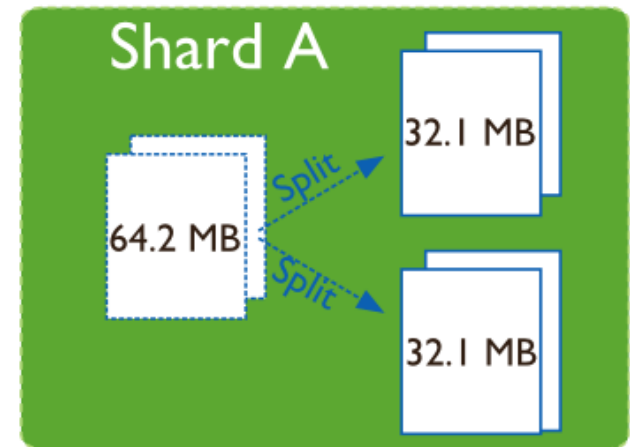
- MongoDB enables **collection partitioning** (sharding)



Collection Partitioning



- Mongo partitions collection's data by the **shard key**
 - shard key = **field(s)** that exist in **each document** in the collection, it is **indexed**
 - Since Mongo 4.2, the value is mutable
 - Since Mongo 4.4, the shard key can be missing -> NULL is then used
 - **Divided** into chunks, distributed across shards
 - **Range-based** partitioning
 - **Hash-based** partitioning
 - When a chunk grows **beyond** the size **limit**, it is **split**
 - Metadata change, **no data migration**
- Data **balancing**:
 - Background chunk migration

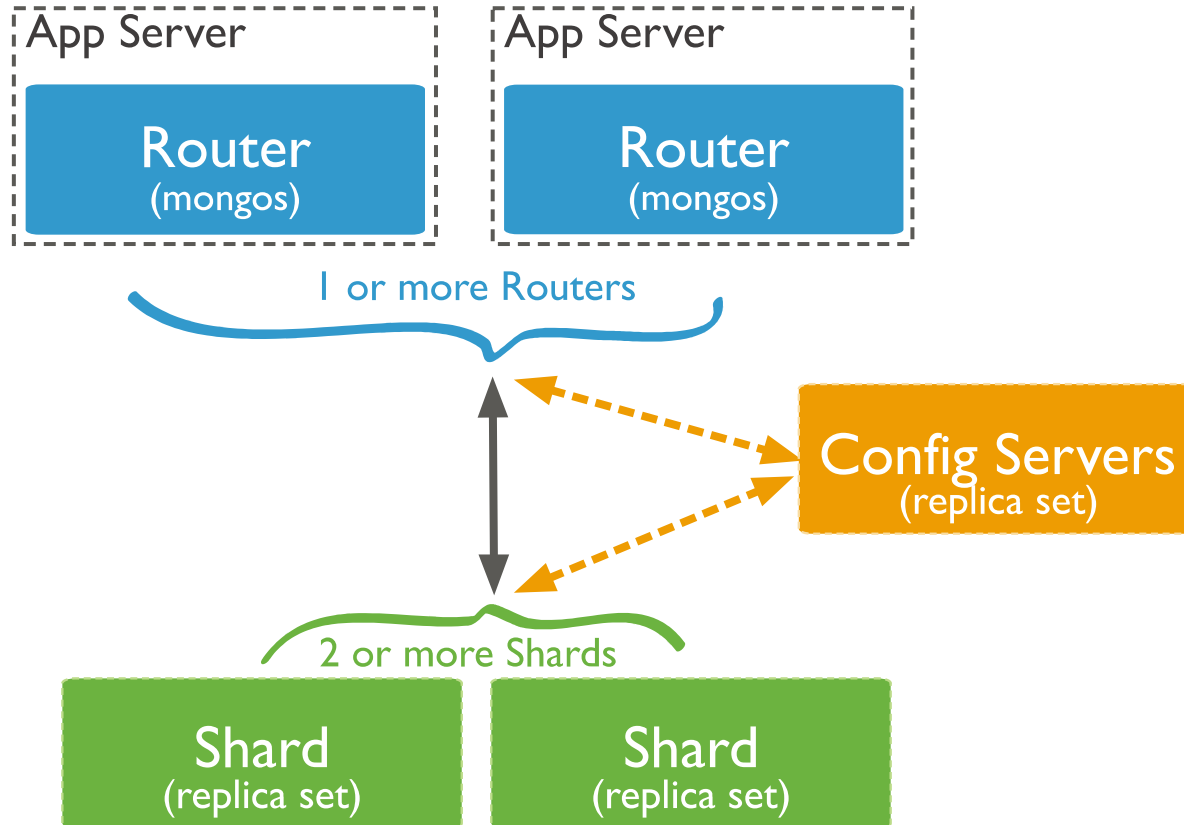


Sharding: Components



- MongoDB runs in **cluster** of different node types:
- **Shards** – store the data
 - Each **shard** is a replica set
 - Can be a **single node**
- **Mongos (Query routers)** – interface btw. client applications and sharded cluster
 - **Direct** operations **to** the **relevant** shard(s)
 - + **return** the result to the client
 - More than one => to divide the client request load
- **Config servers** – store the cluster's metadata
 - **Mapping** of the cluster's data set to the shards
 - Recommended number: 3

Sharding: Diagram



Journaling



- **Write** operations are applied **in memory and into a journal** before done in the data files (on disk)
 - To **restore** a consistent state after a **hard shutdown**
 - Can be switched on/off
- **Journal directory** – holds journal files
- **Journal file** = write-ahead **redo logs**
 - Append only file
 - **Deleted when** all the writes are **durable**
 - When size > 1GB of data, MongoDB creates a new file
 - The size can be modified
- **Clean shutdown** removes all journal files

Transactions



- Write ops: **atomic** at the level of **single document**
 - Including nested documents
 - Sufficient for many cases, but not all
 - When a write operation modifies **multiple** documents, **other** operations may **interleave**

\$isolated operator is deprecated

- **Transactions:** ([docs](#))

- **Isolation** of a write operation that affects multiple docs

```
db.foo.update( { field1 : 1 , $isolated : 1 }, { $inc : { field2 : 1 } } , { multi: true } )
```

- Two-phase commit

- Multi-document updates
- In a session (.start/endSession), do **.start/abort/commitTransaction**

Summary



- Concepts of text documents
 - JSON vs. XML
 - binary format of JSON
- MongoDB principles
 - data manipulation
 - replication
 - distribution
 - transactions

References

- I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.
- Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 192 p.
- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases
- MongoDB Manual: <http://docs.mongodb.org/manual/>