MongoDB

Document oriented database

- Document type: JSON
- First release: 2009
- Open source, free license
- Highly scalable
- Transaction support
- CP system (by default)

```
{
  "employeeID": 16004,
  "personal": {
    "firstname": "Natale",
    "lastname":  "Berget",
    "gender":    "Male",
    "birthdate": "1992-12-03"
  },
  units: [ "development",
           "facility" ],
  "salary": {
    "yearly": "22485"
  }
}
```

MongoDB

Data Model Hierarchy:

- Database Cluster (spanning several nodes)
- Database (set of collections)
- Collection (set of documents)
- Document (JSON object)
- Field (atomic or composite)

MongoDB

Value Data Type: the BSON types (JSON types + some more)

- String (UTF-8)
- Boolean (true/false)
- Array
- (Sub-)Document (BSON)
- Null
- Integer (32/64-bit)
- Decimal (128-bit floating point)
- Datetime (UTC)
- Binary (8-bit)
- ... and even more

See BSON specification at https://bsonspec.org/

MongoDB

How do I create databases and collections?

Just write into it!

```
> db
test

> use socialnetwork
switched to db socialnetwork

> db.people.insertOne( { "name": "Peter", "hobbies": ["piano", "yoga"],
                         "info": { "age": 27, "gender": "m" } } )
> db.people.updateOne({name:"Peter"}, {$set:{"info.favcolor":"red"}})

> db.people.find()
{ "_id" : ObjectId("629388c0fe954dc76f230f99"), "name" : "Peter",
  "hobbies" : [  "piano",  "yoga" ],
```

When inserting a document in a non-existing collection or database, these database objects are automatically created. Each document in MongoDB has the field `_id`. If not manually set (to a number, string, ...), its value is an automatically generated ObjectId. Like the key in Redis, the row-id in HBase, the `_id` field in MongoDB must be unique and it allows fast access. MongoDB also supports indexes over other fields, and hash or range partitioning - they call it sharding - over the values of an arbitrary field.

Queries in MongoDB

`db.collectionname.find( selection, projection )`

```
> db.people.find( { "name": "Peter" }, {"_id":1, "name":1} )
{ "_id" : ObjectId("60811e66216a217c24e2b7c5", "name" : "Peter" ) }
```

Complex queries: `$$and, $or, $not, $ne, $gt, $lt, ...`

Match subdocument fields: dot-notation ("info.age")

Implicit `$and`: Separate by comma

```
> db.people.find( { "name": "Peter", "info.age": 27 },
                  {"_id":1, "name":1} )
> db.people.find( { "$or": [{"name": "Peter"}, {"name": "Anna"}] } )
> db.people.find( { "$or": [{"name": "Peter"},
                            {"age.info": { "$gt": 28 } }] } )
```

Queries in MongoDB

Sorting and limits:

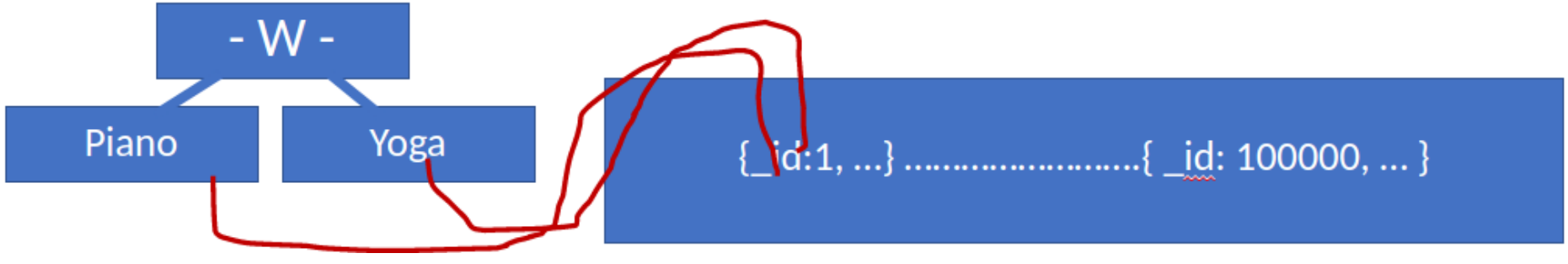
```
> db.people.find(null,{"name":1}).sort({"name":1,"info.age":-1})
[
  { _id: bacd3, name:  'Anna', info: { age: 25 } },
  { _id: baccf, name: 'Peter', info: { age: 27 } },
  { _id: bacd0, name: 'Peter', info: { age: 16 } },
  { _id: bacd2, name: 'Susan', info: { age: 32 } }
]

> db.people.find(null, {"name":1}).limit(2)
[
  { _id: baccf, name: 'Peter' },
  { _id: bacd0, name: 'Peter' }
]
```

Indexes in MongoDB

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

Multi-Key Indexes

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



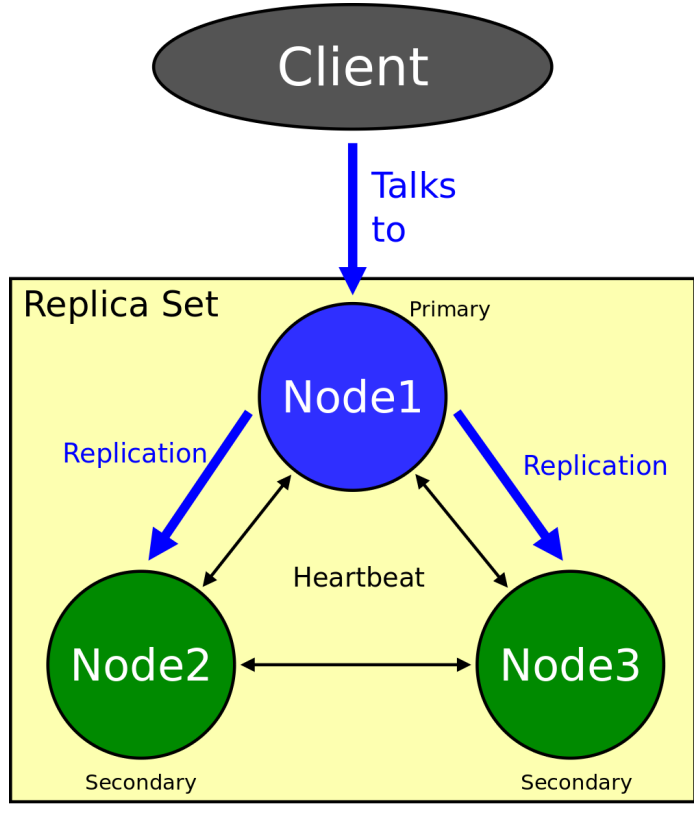
Like in SQL, indexes are automatically used when they can accelerate a query. E.g., the first index on this slide can be used for the query `db.people.find({"name":"Peter"})`, the second one for `db.people.find({"hobbies":"yoga"})`. As the field `"hobbies"` is an array, we call this index a multi-key index. Each document is referenced within the index multiple times - once for each of its array elements.

MongoDB

Mongodb Cluster

Basic building blocks: Replica Sets

- Data redundancy
- Primary/Secondary model
- High availability
- Automatic switchover



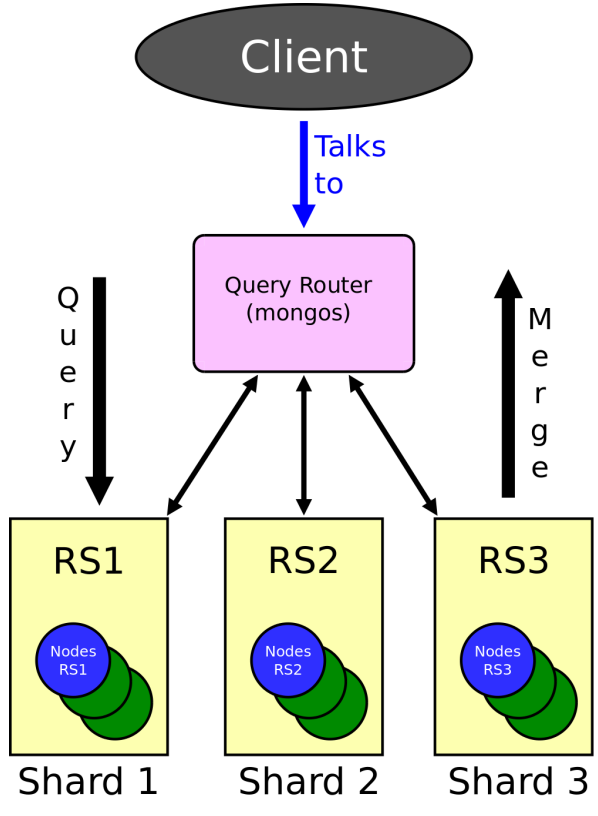

```
> rs.initiate()
> rs.add("node2:27019")
> rs.add("node3:27019")
```

MongoDB

Mongodb Cluster

Sharding/Partitioning

- Collection-based sharding
- Sharding key has to be specified
- Hash or Range partitioning
- Built from Replica Sets
- Query router (mongos)

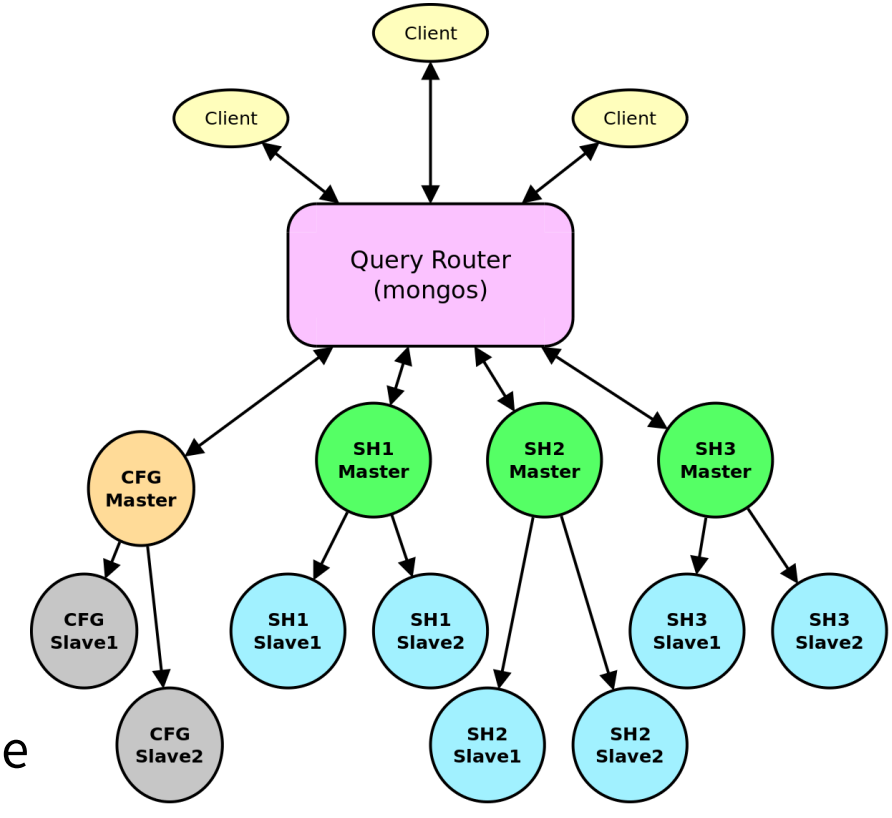



```
> sh.addShard("RS1/<nodes:ports>")
> sh.addShard("RS2/<nodes:ports>")
> sh.addShard("RS3/<nodes:ports>")
> sh.enableSharding("sn")
> sh.shardCollection("sn.ppl", {"name":1})
```

... or `{"name":"hashed"}` for hash partitioning

MongoDB

Mongodb Example Cluster

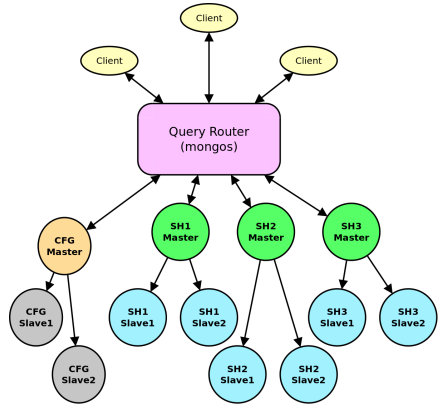


For high availability:

- At least 3 nodes per replica set
- One can be an arbiter
- Multiple "mongos" query router possible

MongoDB

Mongodb and Consistency



Multi-Document Transactions:

- Atomic operations
- Changes become visible after commit (Isolation)
- Read and Write concern adjustable per operation and transaction (Durability)

Read from secondaries:

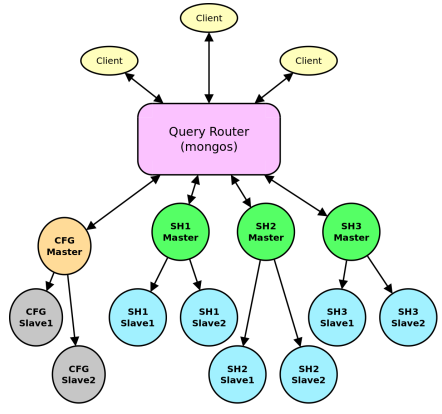
```
> db.getMongo().setReadPref("secondaryPreferred")
```

❗Caution: "Strong Consistency" no longer guaranteed!

MongoDB

Schema validation:

```
> db.createCollection('people', { validator:
{
  $jsonSchema: {
    required: [ "name", "info" ],
      properties: {
        name: { bsonType: "string" },
        info: {
          bsonType: "object",
          required: [ "birthdate" ],
          properties: {
            age: { bsonType: "int" },
            birthdate: { bsonType: "date" }
          }
        }
    }
  }
}
})
```



MongoDB

MongoDB **Map/Reduce**: Computations distributed amongst shards

Example: Sum up the ages by name

```
> map = function() {
  emit(this.name, this.info.age)
}
> reduce = function (k, v) {
  return Array.sum(v)
}
> db.people.mapReduce(map, reduce, {out: {inline:1}})
results: [ { _id: 'Susan', value: 25 },
           { _id: 'Peter', value: 43 },
           { _id:  'Anna', value: 23 } ]
```

❗Deprecated in favor of the aggregation pipeline

MongoDB Aggregation Pipeline

```
> db.people.aggregate([
 { "$match": { "hobbies": "yoga" } },
 { "$sort": { "name": 1 }},
 { "$limit": 10 },
 { "$project": { "_id": 1 }}
])
```

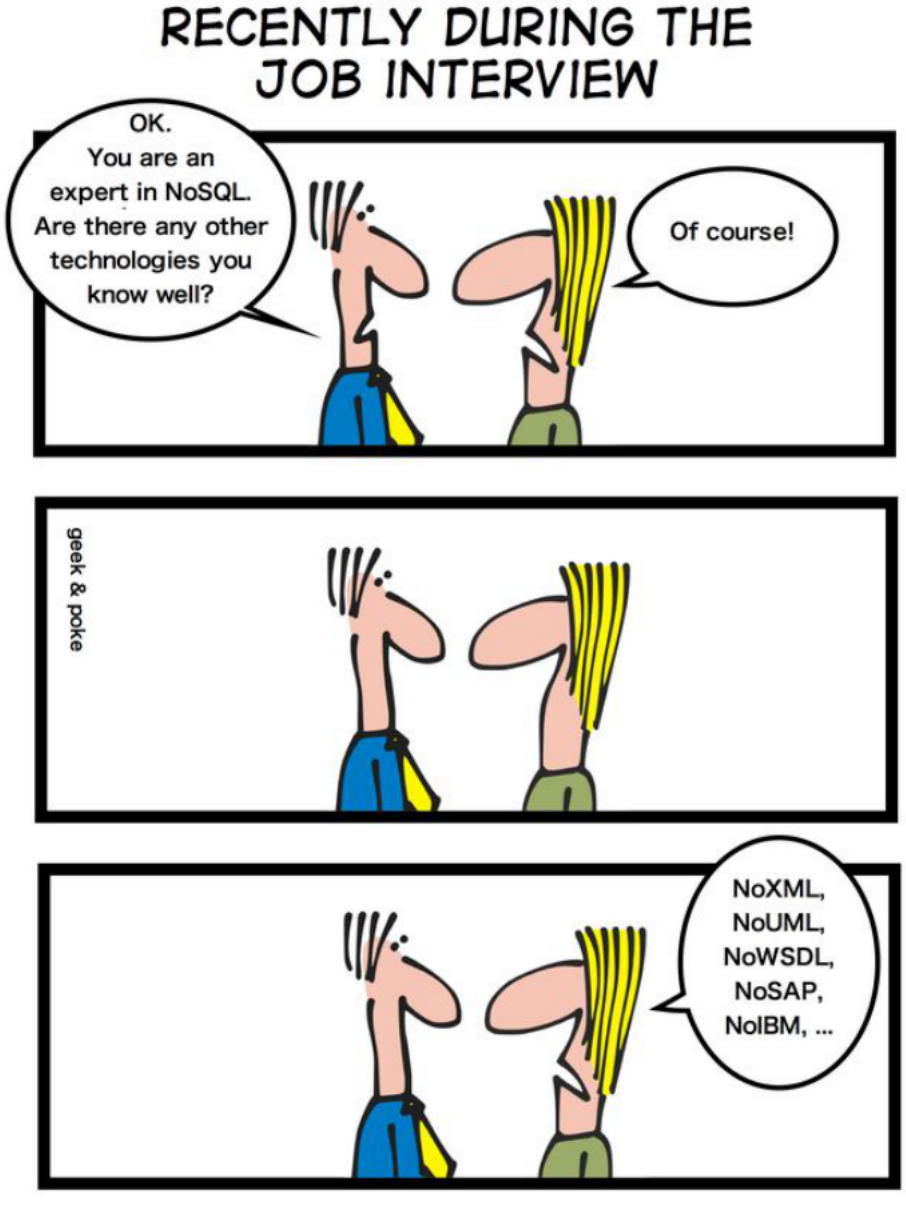
```
> db.people.aggregate([
 { "$unwind": "$hobbies" },
 { "$group": { "_id": "$hobbies", "names" : {"$push": "$name"} }}
])
```

```
{ "_id": "yoga", "names": ["Peter", "Jane"] }
{ "_id": "piano", "names": ["Peter"] }
```

The MongoDB aggregation pipeline allows for complex data-transformation queries. A pipeline is defined as a sequence of steps. The first query on this slide would also be possible with a `find` command. The second query shows further pipeline steps like `$unwind` for unwinding an array into its elements, and `$group` for grouping documents by a given field (here, grouping by hobby). `$push` is a so-called accumulator. It creates an array with all elements within a group. Other accumulators are `$sum`, `$avg`, `$max`, etc. A dollar symbol in a key (e.g. `$push`) is a reserved MongoDB command. A dollar in a value (e.g., `$name`) is an attribute dereference. In this case it references the value in the JSON field `"name"`.

Summary

- NoSQL = Not only SQL
- Key-Value Stores
  e.g., Redis: supports simple and complex value types (lists, sets, hashes, ...))
- Wide-Column Stores
  e.g., HBase: PUT/GET/SCAN API based on row-id (ranges); versioning
- Document Databases
  e.g. MongoDB: collections of JSON documents; indexes, aggregation pipeline
- Graph Databases
  e.g., Neo4J: property graphs, Cypher, Gremlin



Source: https://geek-and-poke.com