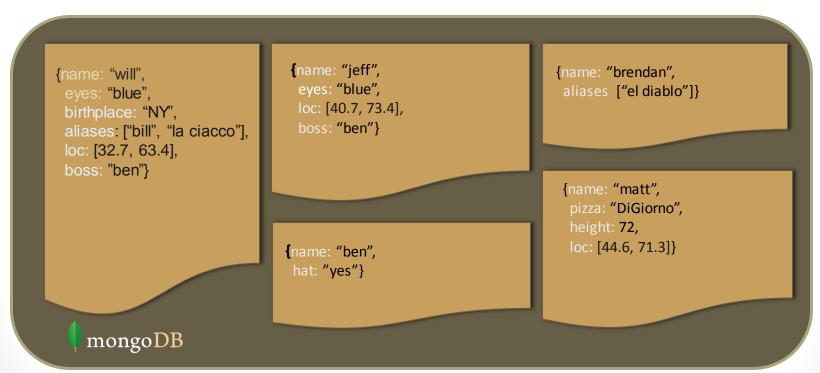


Functionality of MongoDB

- Dynamic schema
 - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
 - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
 - Addresses NULL data fields



CRUD operations

- Create
 - db.collection.insert(<document>)
 - db.collection.save(<document>)
 - db.collection.update(<query>, <update>, { upsert: true })
- Read
 - db.collection.find(<query>, <projection>)
 - db.collection.findOne(<query>, <projection>)
- Update
 - db.collection.update(<query>, <update>, <options>)
- Delete
 - db.collection.remove(<query>, <justOne>)

Collection specifies the collection or the 'table' to store the document

Create Operations

Db.collection specifies the collection or the 'table' to store the document

- db.collection_name.insert(<document>)
 - Omit the id field to have MongoDB generate a unique key
 - Example db.parts.insert({{type: "screwdriver", quantity: 15 })
 - db.parts.insert({_id:10, type: "hammer", quantity:1 })
- db.collection_name.update(<query>, <update>, { upsert: true })
 - Will update 1 or more records in a collection satisfying query
- db.collection_name.save(<document>)
 - Updates an existing record or creates a new record

Read Operations

- db.collection.find(<query>, <projection>).cursor modified
 - Provides functionality similar to the SELECT command
 - query> where condition, <projection> fields in result set
 - Example: var PartsCursor = db.parts.find({parts: "hammer"}).limit(5)
 - Has cursors to handle a result set
 - Can modify the query to impose limits, skips, and sort orders.
 - Can specify to return the 'top' number of records from the result set
- db.collection.findOne(<query>, <projection>)

Query Operators

Name	Description
\$eq	Matches value that are equal to a specified value
\$gt, \$gte	Matches values that are greater than (or equal to a specified value
\$lt, \$lte	Matches values less than or (equal to) a specified value
\$ne	Matches values that are not equal to a specified value
\$in	Matches any of the values specified in an array
\$nin	Matches none of the values specified in an array
\$or	Joins query clauses with a logical OR returns all
\$and	Join query clauses with a loginal AND
\$not	Inverts the effect of a query expression
\$nor	Join query clauses with a logical NOR
\$exists	Matches documents that have a specified field

Update Operations

- db.collection_name.insert(<document>)
 - Omit the _id field to have MongoDB generate a unique key
 - Example db.parts.insert({{type: "screwdriver", quantity: 15 })
 - db.parts.insert({_id:10, type: "hammer", quantity:1 })
- db.collection_name.save(<document>)
 - Updates an existing record or creates a new record
- db.collection_name.update(<query>, <update>, { upsert: true })
 - Will update 1 or more records in a collection satisfying query
- db.collection_name.findAndModify(<query>, <sort>,
 <update>,<new>, <fields>,<upsert>)
 - Modify existing record(s) retrieve old or new version of the record

Delete Operations

- db.collection_name.remove(<query>, <justone>)
 - Delete all records from a collection or matching a criterion
 - <justone> specifies to delete only 1 record matching the criterion
 - Example: db.parts.remove(type: /^h/ }) remove all parts starting with h
 - Db.parts.remove() delete all documents in the parts collections

CRUD examples

```
> db.user.insert({
    first: "John",
    last: "Doe",
    age: 39
})
```

```
> db.user.remove({
    "first": /^J/
})
```