
MONTANA
STATE UNIVERSITY

MongoDB

More NoSQL: Document Databases

MongoDB

- Like Redis, MongoDB is a *NoSQL* data store
- Unlike Redis, MongoDB is a *Document Database*
  - Stores JSON-like documents rather than offering various data types

Documents In MongoDB

- At right is a MongoDB document
- As you can see, it looks a lot like JSON
- Additionally, you can see that field values need not be a primitive type
  - Arrays
  - Nested Objects
  - etc.

Documents In MongoDB

- Advantages of this approach
  - Documents (i.e. objects) correspond to native data types in many programming languages
  - Embedded documents and arrays reduce need for expensive joins
  - Dynamic schema supports fluent polymorphism

Documents In MongoDB

- *“Documents correspond to native data types in many programming languages”*
  - To an extent
  - Obviously there is a direct mapping to Javascript, but what about other languages?
    - Yeah, but not as good a match
    - Static typing?

Documents In MongoDB

- *“Embedded documents and arrays reduce need for expensive joins”*
  - True, to an extent
  - But what about normalization?
  - What if I want to rename the “sports” group to “college sports”?

Documents In MongoDB

- “Dynamic schema supports fluent polymorphism”
  - Uhhhh, OK
  - Documents don’t need to meet any schema
  - So two documents in the same collection can look totally different…
    - Uhhhh, I guess that works for javascript...

Documents In MongoDB

- Mongo 3.2 added support for document validation
  - Allows you to enforce a schema via JSONSchema

    https://json-schema.org/

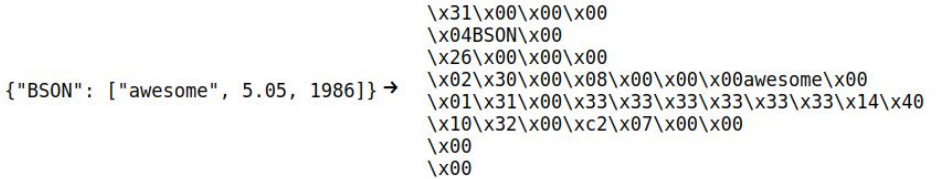
  - Side bar, guess who wrote this:

    http://jschema.org

MongoDB Concepts

- Database: a set of named *Collections*
  - As well as views and a few other things
- Collections: a set of Documents
  - Stored in BSON format

http://bsonspec.org/



Creating Collections

- One really nice feature of Mongo is how easy it is to create a database
- This will create a new database, *myNewDB* and then create a new collection and insert some data into it
  - Very liberating if you are used to DDL

```
use myNewDB

db.myNewCollection1.insertOne( { x: 1 } )
```

Collections

- Inserting multiple documents into a collection is similar, although not identical, to inserting rows in a relational database

Collections

- You can explicitly create a collection if you wish to specify configuration options for it
  - Note the validator option
  - You can also cap the size of the collection, etc.

```
db.createCollection( <name>,
   {
     capped: <boolean>,
     autoIndexId: <boolean>,
     size: <number>,
     max: <number>,
     storageEngine: <document>,
     validator: <document>,
     validationLevel: <string>,
     validationAction: <string>,
     indexOptionDefaults: <document>,
     viewOn: <string>,              // Added in MongoDB 3.4
     pipeline: <pipeline>,          // Added in MongoDB 3.4
     collation: <document>,         // Added in MongoDB 3.4
     writeConcern: <document>
   }
)
```

Views

- As with relational databases, you can create views in MongoDB
- Here <source> is a query that will define the view
  - We will discuss the pipeline later
- As with relational database, views are read only

```
db.createView(
  "<viewName>",
  "<source>",
  [<pipeline>],
  {
    "collation" : { <collation> }
  }
)
```

Documents

- In MongoDB documents you will always have an _id field, with is of type ObjectId
  - ObjectIds are small, *likely unique*, fast to generate, and ordered. ObjectId values are 12 bytes in length, consisting of:
    - a 4-byte timestamp value, representing the ObjectId's creation, measured in seconds since the Unix epoch
    - a 5-byte random value
    - a 3-byte incrementing counter, initialized to a random value

```
var mydoc = {
                _id: ObjectId("5099803df3f4948bd2f98391"),
                name: { first: "Alan", last: "Turing" },
                birth: new Date('Jun 23, 1912'),
                death: new Date('Jun 07, 1954'),
                contribs: [ "Turing machine", "Turing test", "Turingery" ],
                views : NumberLong(1250000)
              }
```

Documents

- This document also shows
  - nested objects (name)
  - Dates
  - Arrays
  - A NumberLong(64 bit integer)
    - By default the mongodb shell treats numbers as double precision floating point, like javascript
      - ¯_(ツ)_/¯

```
var mydoc = {
                _id: ObjectId("5099803df3f4948bd2f98391"),
                name: { first: "Alan", last: "Turing" },
                birth: new Date('Jun 23, 1912'),
                death: new Date('Jun 07, 1954'),
                contribs: [ "Turing machine", "Turing test", "Turingery" ],
                views : NumberLong(1250000)
              }
```

CRUD in Mongo

- Documents can be created with the *insertOne()* and *insertMany()* operations
  - This example inserts a single value into the example collection
  - Recall that the collection will automatically created if it does not exist
  - Note that you get back the object ID of the inserted document

CRUD in Mongo

- Querying in Mongo is usually done with the *find()* method
- Consider an inventory collection created with the following *insertMany()* statements
  - Properties - item, quantity, size & status

```
> db.inventory.insertMany([
...     { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
...     { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
...     { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
...     { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
...     { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
... ]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5f98480f663182f750282297"),
                ObjectId("5f98480f663182f750282298"),
                ObjectId("5f98480f663182f750282299"),
                ObjectId("5f98480f663182f75028229a"),
                ObjectId("5f98480f663182f75028229b")
        ]
}
```

CRUD in Mongo

- A basic query using the find method takes a JSON-like query specification
- This example finds all documents with the status “D”
- Equivalent to the SQL at right in a relational database

```
> db.inventory.find( { status: "D" } )
{ "_id" : ObjectId("5f98480f663182f750282299"), "
" }
{ "_id" : ObjectId("5f98480f663182f75028229a"), "
 "D" }
>
```

```
SELECT * FROM inventory WHERE status = "D"
```

CRUD in Mongo

- IN-style queries use the *$in* keyword
- This query returns all documents whose status is in the given list
- Equivalent to the given SQL

```
> db.inventory.find( { status: { $in: [ "A", "D" ] } } )
{ "_id" : ObjectId("5f98480f663182f750282297"), "item" : "j
" }
{ "_id" : ObjectId("5f98480f663182f750282298"), "item" : "n
"A" }
{ "_id" : ObjectId("5f98480f663182f750282299"), "item" : "p
" }
{ "_id" : ObjectId("5f98480f663182f75028229a"), "item" : "p
 "D" }
{ "_id" : ObjectId("5f98480f663182f75028229b"), "item" : "p
: "A" }
>
```

```sql
SELECT * FROM inventory WHERE status in ("A", "D")
```

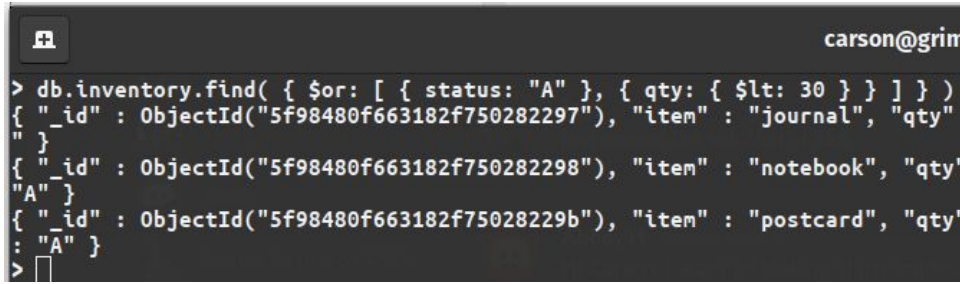
CRUD in Mongo

- AND queries are comma separated
- This query returns all documents with status “A” AND a quantity less than 30
  - Note the less than specification, using the *$lt* keyword
  - Many comparison keywords list this are available: *$eq, $gt, $gte, $in, $nin,* etc.


```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

CRUD in Mongo

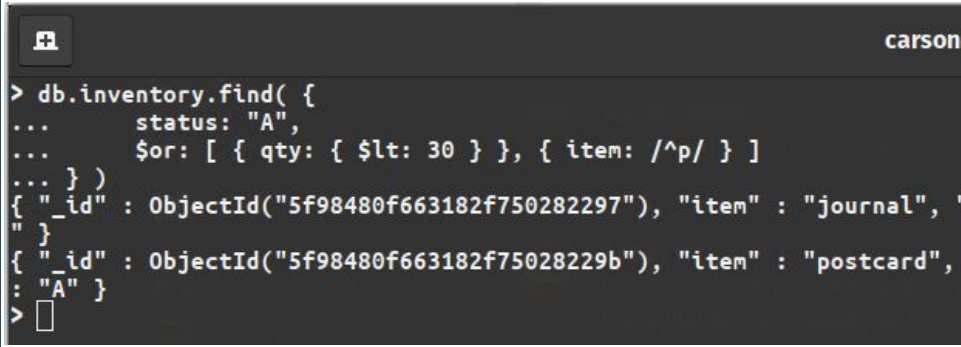
- OR queries use the *$or* keyword, which takes an array of conditions
- Here we are finding all inventory documents with status “A” or quantity less than 30




```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

CRUD in Mongo

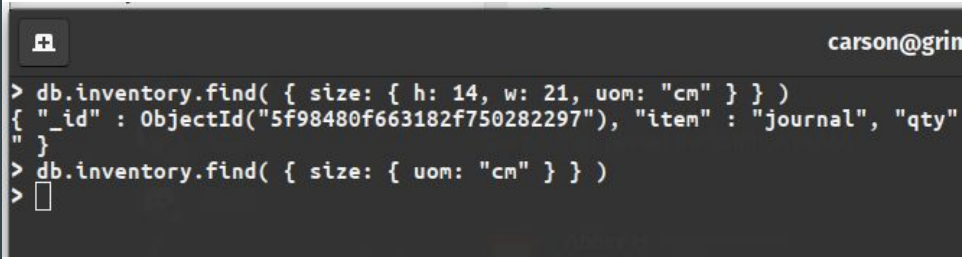
- ANDs and ORs can be combined with a nested query tree
- Use a standard comma separated JSON object for AND and the *$or:[]* syntax for ORs




```
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

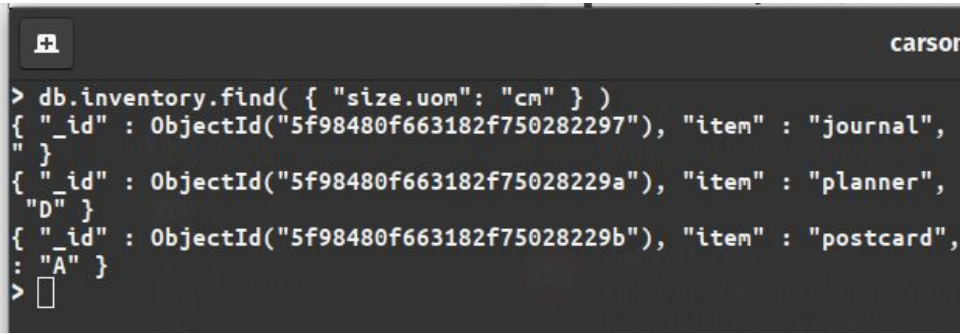
CRUD in Mongo

- What about querying embedded/nested data?
- Multiple syntaxes with slight differences
  - Obvious nested syntax requires an exact match on all fiends
  - Cannot omit any fields, or a document won't match
    - Note that second query does not match any documents!



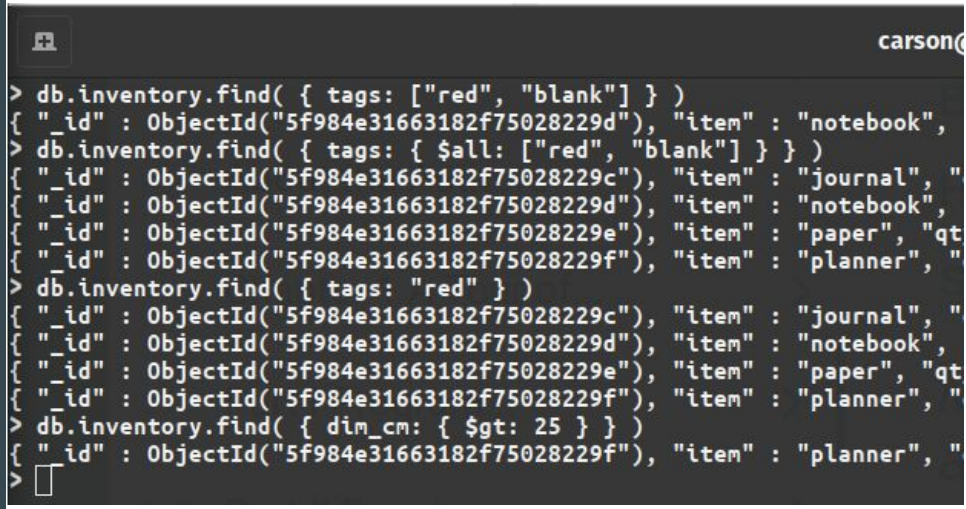
CRUD in Mongo

- More intuitive syntax is available if you use the dot syntax
- This syntax does *not* require a total match



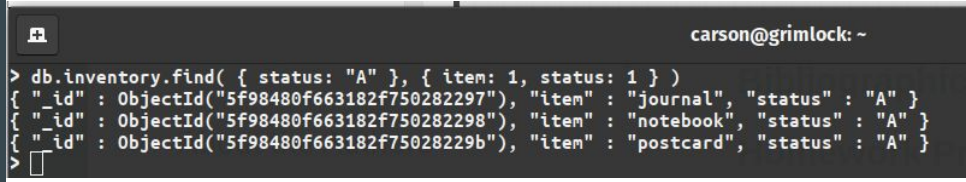
CRUD in Mongo

- Arrays can be queried in various ways
  - Exact match
  - Unordered match
  - Contains
  - Conditions
- Nested documents can be queried against via the same dot syntax used for fields



CRUD in Mongo

- Field Selection: to select specific fields from a document, you pass in a second JSON object
  - Passing a 1 for a field name indicates it is included
  - Passing 0 for a field name indicates all other *except* this field should be returned
  - Can use dot syntax to include or exclude embedded documents




```
SELECT _id, item, status from inventory WHERE status = "A"
```

CRUD in Mongo

- Updates are done with one of the following methods:
  - *db.collection.updateOne(<filter>, <update>, <options>)*
  - *db.collection.updateMany(<filter>, <update>, <options>)*
  - *db.collection.replaceOne(<filter>, <update>, <options>)*
  - *db.collection.update(<filter>, <update>, <options>)*

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

CRUD in Mongo

- *updateOne()* takes a condition and an update expression and updates the *first match*
  - Here we update the first inventory document whose item is “paper”
  - sets the embedded document *size uom* property to “cm”
  - Sets the status to “P”
  - Sets lastModified to the current date

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

CRUD in Mongo

- *updateMany()* takes a condition and an update expression and updates all matches
  - Here we update all inventory documents whose item is “paper”
  - sets the embedded document *size uom* property to “in”
  - Sets the status to “P”
  - Sets lastModified to the current date

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

CRUD in Mongo

- *replaceOne()* takes a condition and an update expression and *replaces* all matches
  - This will replace the entire document, not just update fields

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 },
)
```

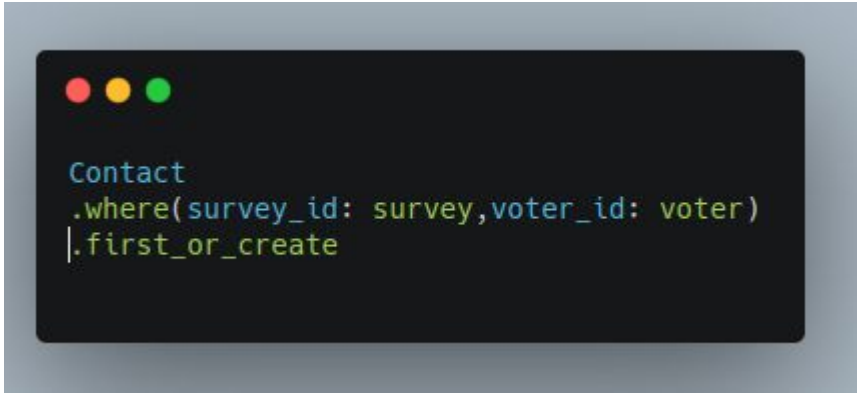
CRUD in Mongo

- Upsert Support
  - Any update operation can be converted into an *upsert* by including the {upsert : true} option
  - If no match, the document is inserted instead
  - Here, if no product with id 6 exists, the data will be inserted instead

```
db.products.updateMany(
    { _id: 6 },
    { $set: {price: 999} },
    { upsert: true}
)
```

CRUD in Mongo

- Upserts are very useful in many online systems
  - Rails ORM ActiveRecord supports a similar pattern with *first_or_create()*



CRUD in Mongo

- Delete operations
  - *db.collection.deleteOne(<filter>, <options>)*
  - *db.collection.deleteMany(<filter>, <options>)*
    - *<Tell Funny OR story here>*

```
db.inventory.deleteMany({ status : "A" })
```

Text Search

- Mongo supports broad, google-like text search out of the box
  - First example: general text search
  - Second example: exact match for “coffee shop”
  - Third excludes “coffee”
- Excellent functionality compared with most RDBMS
  - Especially a decade ago!

```
db.stores.find( { $text: { $search: "java coffee shop" } } )
```

```
db.stores.find( { $text: { $search: "\"coffee shop\"" } } )
```

```
db.stores.find( { $text: { $search: "java shop -coffee" } } )
```

Indexing

- Mongo supports the easy creation of indexes
  - Supports geospatial indexing, text indexing, multi-key indexing, etc.
- Index usage can be viewed via the *$indexStats* keyword

```
db.collection.createIndex( { name: -1 } )
```

Transactions

- Historically Mongo has not had a good reputation for data persistence
  - Arguments over whether this is FUD or not
    - *I would use an RDBMs for crucial data, myself*
- Mongo now has a transaction API
  - API is based on distributed transactions, pretty complex

```
# Step 2: Start a client session.
with client.start_session() as session:
    # Step 3: Use with_transaction to start a transact
    session.with_transaction(
        callback, read_concern=ReadConcern('local'),
        write_concern=wc_majority,
        read_preference=ReadPreference.PRIMARY)
```

When To Use Mongo?

- As much of a technology grump as I am, I think Mongo has a place in many systems
- Useful for:
  - Non-core data
  - Data streams
  - Flexible, early data modeling
  - Data that “doesn’t matter”
- I would still recommend RDBMS for core, “must be correct” data

MongoDB	RDBMS
Database	Database
Collection	Table
Document	Row
Field	Column

MongoDB

- MongoDB is a Document Database
  - In contrast with a *Relational Database*
  - Optimized for the storage and retrieval of documents
- We reviewed many of the core operations for Mongo
  - CRUD
  - Indexes
- A good SQL/Mongo mapping document
  https://docs.mongodb.com/manual/reference/sql-comparison/

- Next time we will talk about the aggregation pipeline in Mongo


MONTANA
STATE UNIVERSITY