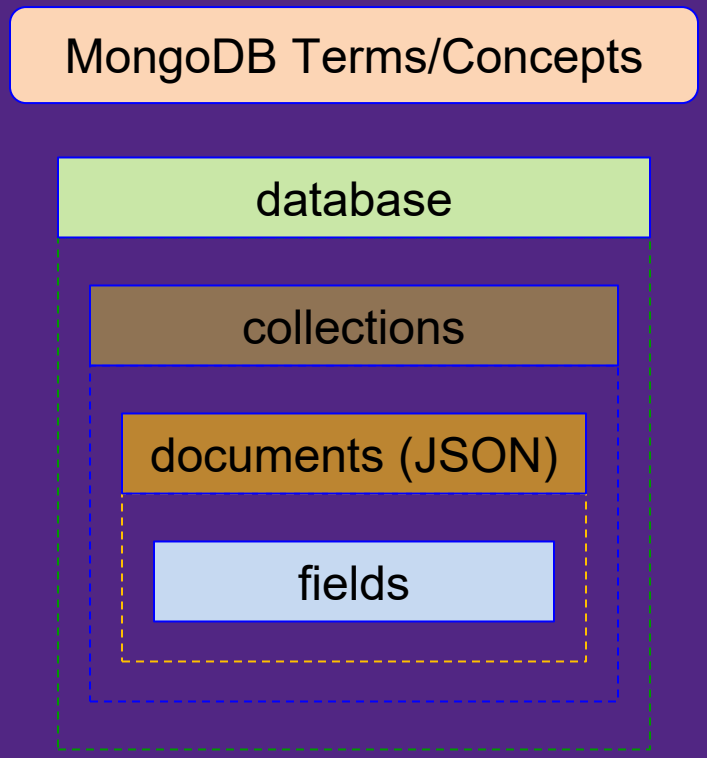
MongoDB and the Aggregation Pipeline

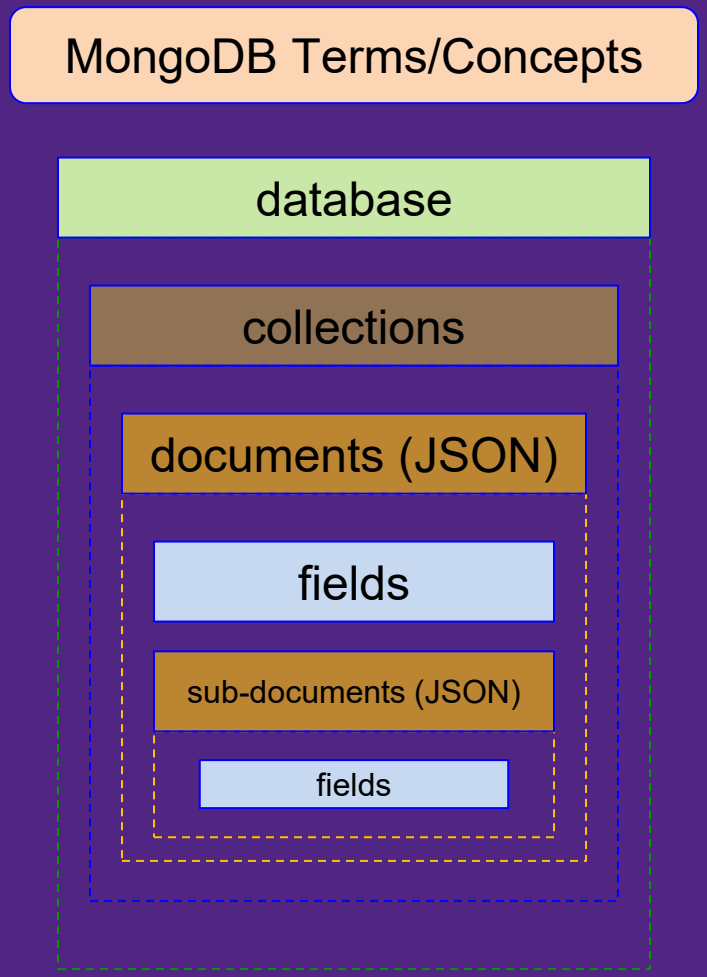
CS 4417B

The University of Western Ontario

MongoDB - Reminders

- Recall: Slides on NoSQL




MongoDB Terms/Concepts
database
collections
documents (JSON)
fields
sub-documents (JSON)
fields

JSON (JavaScript Object Notation)

- JavaScript arrays:
  - `var myarray = ["one", "two", "three"];`
  - Then `var[1] == "two"`

- JavaScript objects
  - ```
    var myobject =
    {
      firstName : "John",
      lastName : "Doe"
    };
    ```
  - Then `myobject.firstName == "John"`
    ```

JSON (Javascript Object Notation)

```
{
 "EXCERPT": {
  "STAGEDIR": "Thunder and lightning. Enter three Witches",
  "SPEECH": [
   { "SPEAKER": {
       "ROLE": "WITCH",
       "text": "First Witch"
     },
     "LINE": [
            "When shall we three meet again",
            "In thunder, lightning, or in rain?"
            ]
   },
   { "SPEAKER": {
       "ROLE": "WITCH",
       "text": "Second Witch"
     },
     "LINE": [
            "When the hurlyburly's done,",
            "When the battle's lost and won."
            ]
   }
  ]
 }
}
```

What is the value of `myjson.EXCERPT.SPEECH[1].LINE[0]`?

MongoDB: `myjson.EXCERPT.SPEECH.1.LINE.0`?

JSON (Javascript Object Notation)

```
{
 "EXCERPT": {
  "STAGEDIR": "Thunder and lightning. Enter three Witches",
  "SPEECH": [
   { "SPEAKER": {
       "ROLE: "WITCH",
       "text": "First Witch"
     },
     "LINE": [
            "When shall we three meet again",
            "In thunder, lightning, or in rain?"
            ]
   },
   { "SPEAKER": {
       "ROLE": "WITCH",
       "text": "Second Witch"
     },
     "LINE": [
            "When the hurlyburly's done,",
            "When the battle's lost and won."
            ]
   }
  ]
 }
}
```

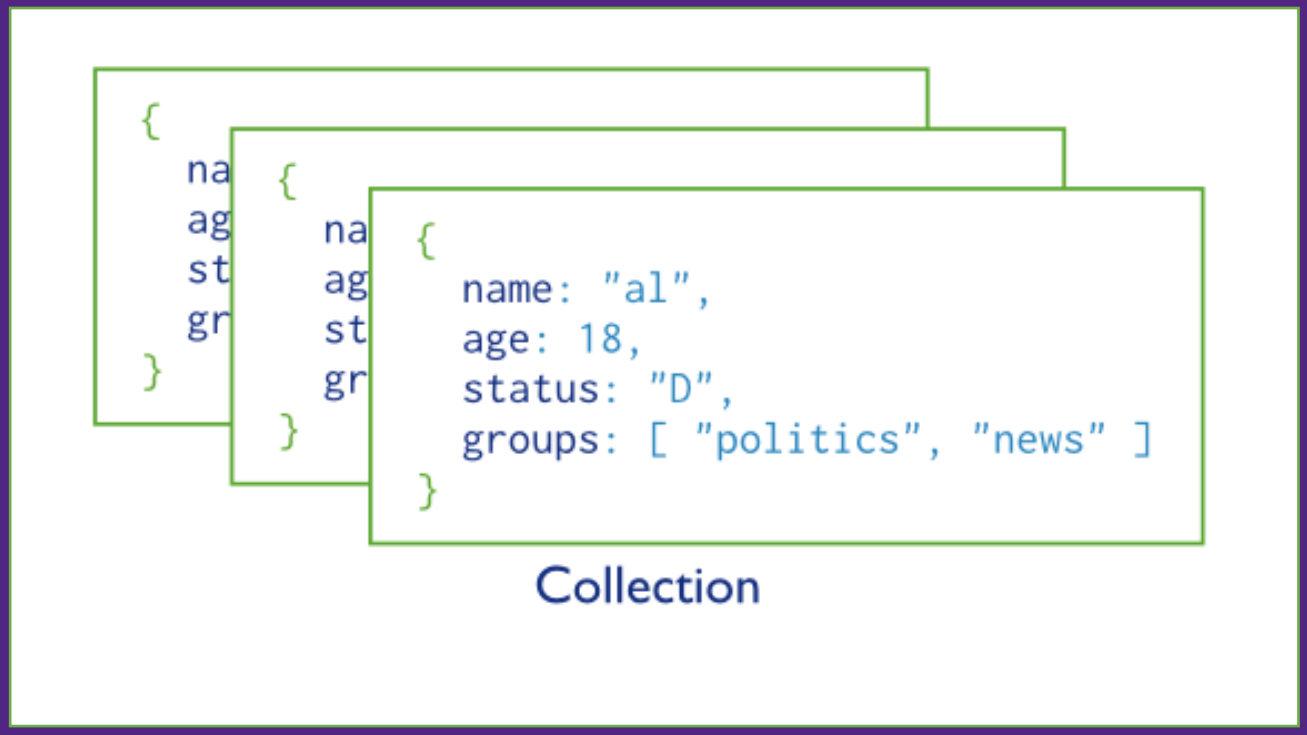
← How would we reference this value?

myjson.EXCERPT.SPEECH[0].SPEAKER.ROLE

MongoDB: myjson.EXCERPT.SPEECH.0.SPEAKER.ROLE

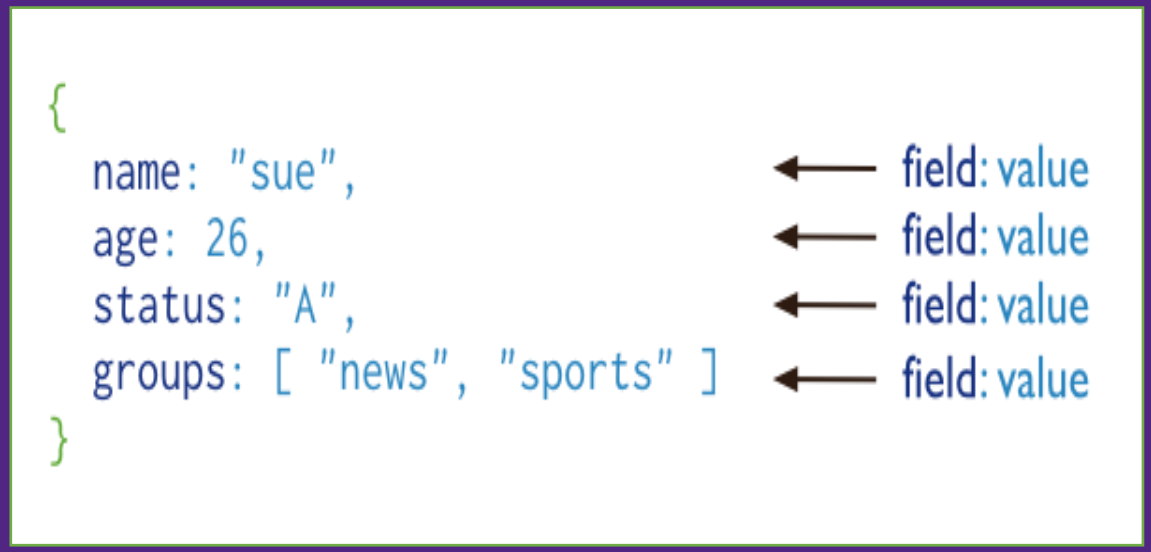
MongoDB Data Model

A collection includes documents.



MongoDB Data Model

Structure of a JSON-document:

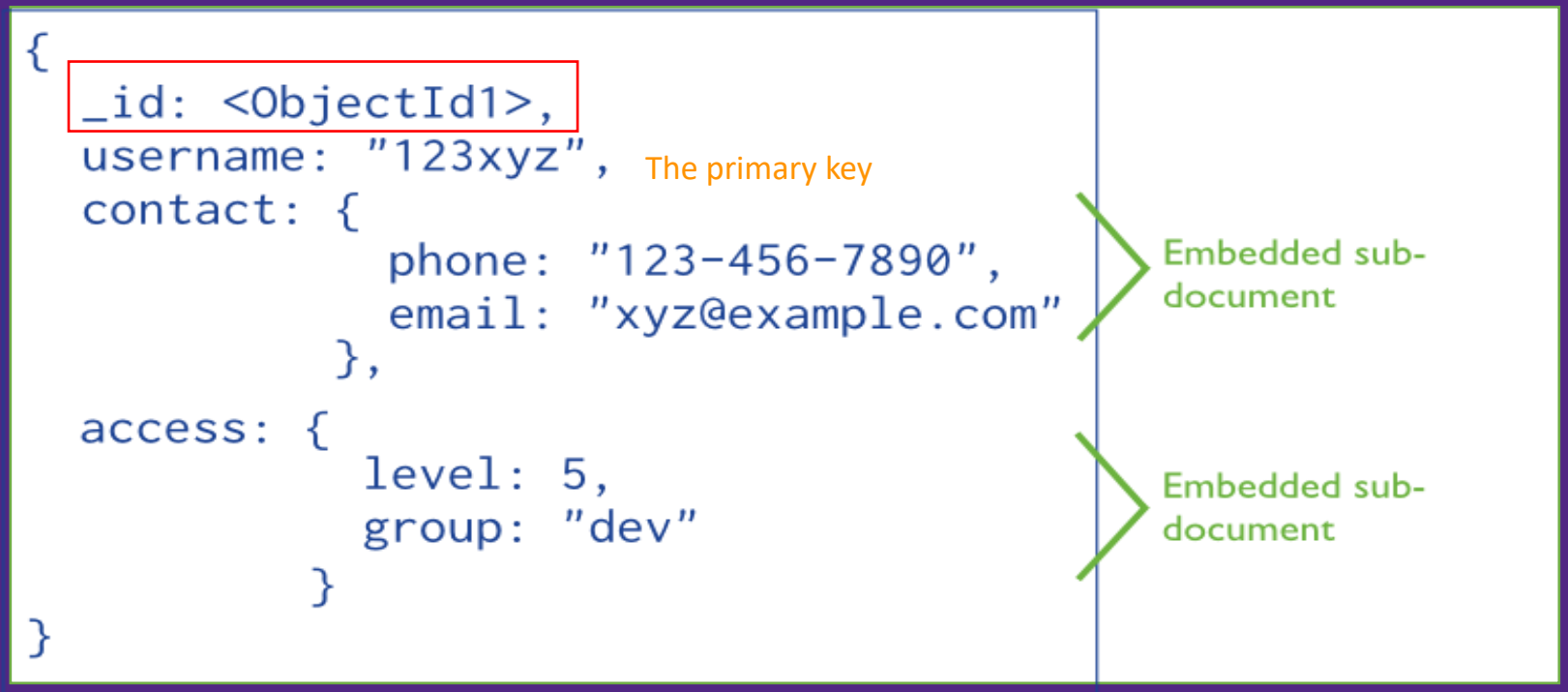


The value of field

- MongoDB data types
- Arrays
- Other documents

MongoDB Data Model

Embedded documents:

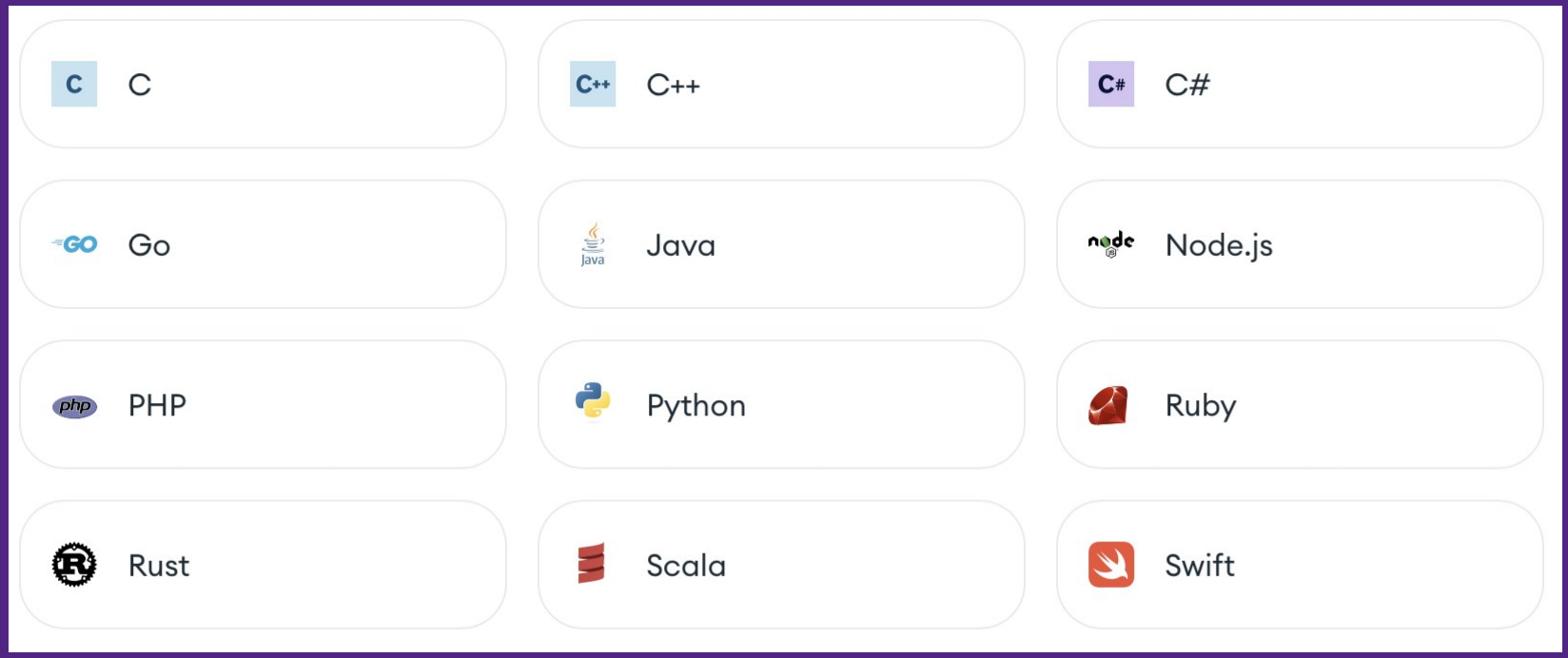


- Example of specification of access to a field in an embedded document
  - contact.email
- Embedded documents means we do not need complicated join tables

MongoDB Document Representation

- Extended JSON
- Like JSON except it allows specification of more data types with special "$" syntax, e.g.
  - Date
    - `"birthdate" : { "$date" : "2014-10-31T00:00:00Z" }`
  - Large numbers
    - `"thenumber" : { "$numberLong" : "571809347373758"}`
  - ObjectIDs
    - `"_id" : { "$oid" : "5aa588b2c4214f42f4c622dd" }`
- Other programs may not understand these! (Like ElasticSearch...)
- Regular strings and numbers don't need type information.
- Dates are stored internally as number of milliseconds since 1 January 1970
  - Class today started at 1710876600000

Interacting with MongoDB



- Also dedicated shell mongosh
- Syntax of commands differs for mongosh – watch out

Getting data into MongoDB

- mongoimport
  - JSON, CSV, TSV, file from mongoexport
  - can specify types for CSV/TSV, but not for JSON

- Either modify your JSON file to add type information, or fix type information after importing with updates (shown later)

Interacting with MongoDB

- Atlas UI
  - GUI

- mongosh
  - The MongoDB Shell, mongosh, is a fully functional JavaScript and Node.js 16.x REPL environment for interacting with MongoDB deployments. You can use the MongoDB Shell to test queries and operations directly with your database.
  - https://www.mongodb.com/docs/mongodb-shell/

Commands to get oriented

- show dbs
  admin 40.00 KiB
  config 48.00 KiB
  local 72.00 KiB
  tweetdb 120.89 MiB

- use tweetdb
  switched to db tweetdb

- db.tweets.findOne()

  {
  _id: ObjectId("5aa588b2c4214f42f4c622dd"),
  created_at: 'Mon Apr 17 02:49:03 +0000 2017',
  id: Long("853802458841317377"),
  id_str: '853802458841317377',
  text: '@allisonsimss Love you lots❤',
  …

MongoDB CRUD

- Create
- Read
- Update
- Delete
- HIGHLY RECOMMEND this tutorial:
- https://www.mongodb.com/docs/manual/tutorial/getting-started/
- (go through all 6 tabs from *1. Switch Database* to *6. Aggregate*)

Create

Insert: Insert a new user using the insert operation on the users collection.

SQL

```
INSERT INTO users                          <-- table
            ( name, age, status )          <-- columns
VALUES      ( "sue", 26, "A" )             <-- values/row
```

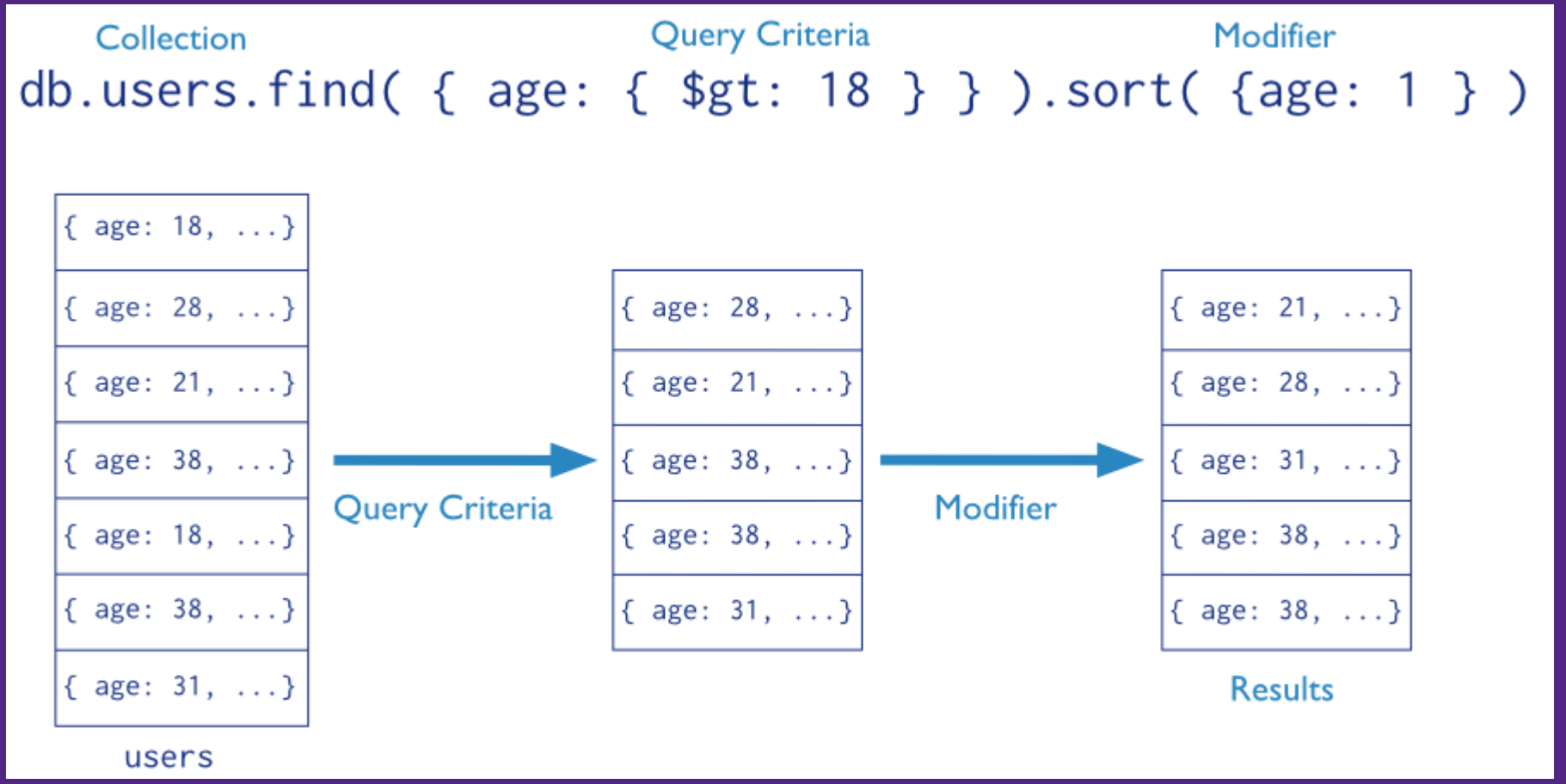
MongoDB

```
db.users.insert (                          <-- collection
   {
     name: "sue",                          <-- field: value
     age: 26,                              <-- field: value
     status: "A"                           <-- field: value
   }                                       } document
)
```

Every document needs an _id field – if not supplied, MongoDB generates one.

Read

Read: The find operation searches for the users of age greater than 18 and sorts by age. “1” is for ascending order and “-1” is for descending order



Read

Read: The find operation searches for the users of age greater than 18 and sorts by age. “1” is for ascending order and “-1” is for descending order

Collection Query Criteria Modifier

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

MongoDB uses the $ to identify *operators* on the data. In this case, $gt is the "Greater Than" operator, and selects all documents whose age is greater than 18.

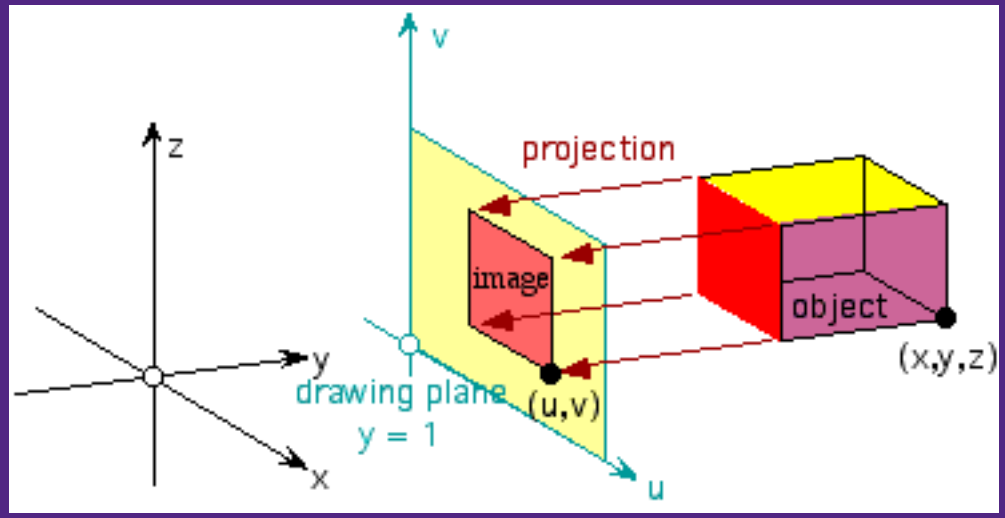
Operator Resources
https://www.mongodb.com/docs/manual/reference/operator/query/

Read – Text Search

- "MongoDB Atlas" – their hosted solution
  - Atlas Search is an embedded full-text search in MongoDB Atlas that gives you a seamless, scalable experience for building relevance-based app features. Built on Apache Lucene, Atlas Search eliminates the need to run a separate search system alongside your database.
- Self-hosted
  - Must create a "text index" for any fields to be searched
  - Seems like they basically made up their matching/scoring function.
- https://www.mongodb.com/docs/v6.0/core/index-text/
- https://ananya281294.medium.com/mongo-maths-676469e55f78

Projection?

- In this case, projection is "throwing away" dimensions.
- Think of each field as a dimension.



https://docs.mongodb.com/manual/tutorial/project-fields-from-query-results/

Projecting during query

- db.tweets.find(
  { in_reply_to_screen_name : “CityofLdnOnt” },
  { in_reply_to_screen_name: 1,
  “user.screen_name”: 1}
)

```
[
 {
  _id: ObjectId("5aa588e7c4214f42f4c63e5b"),
  in_reply_to_screen_name: 'CityofLdnOnt',
  user: { screen_name: 'late2game' }
 },
 {
  _id: ObjectId("5aa58910c4214f42f4c652f2"),
  in_reply_to_screen_name: 'CityofLdnOnt',
  user: { screen_name: 'late2game' }
 }
….
```

- https://www.mongodb.com/docs/manual/tutorial/project-fields-from-query-results/

Update

Update: Update the users of age greater than 18 by setting the status field to A

If the multi parameter is set to true then multiple documents are updated

SQL

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

```
db.users.update(                      <-- collection
   { age: { $gt: 18 } },              <-- update criteria
   { $set: { status: "A" } },         <-- update action
   { multi: true }                    <-- update option
)
```

The set operator will create a new field it it doesn't exist.

Updates to fix datatypes

```
db.tweets.update(
 { }, // Do not filter out documents
 [
  {          new field                                    old field
    $set: { createdDate: { $dateFromString: { dateString: "$created_at" } } }
  }
 ], // Create new field
 {multi: true} // Don't operate on just one document
)
```

$dateFromString is a MongoDB function

Delete

Delete: Delete the users with status equal to D.

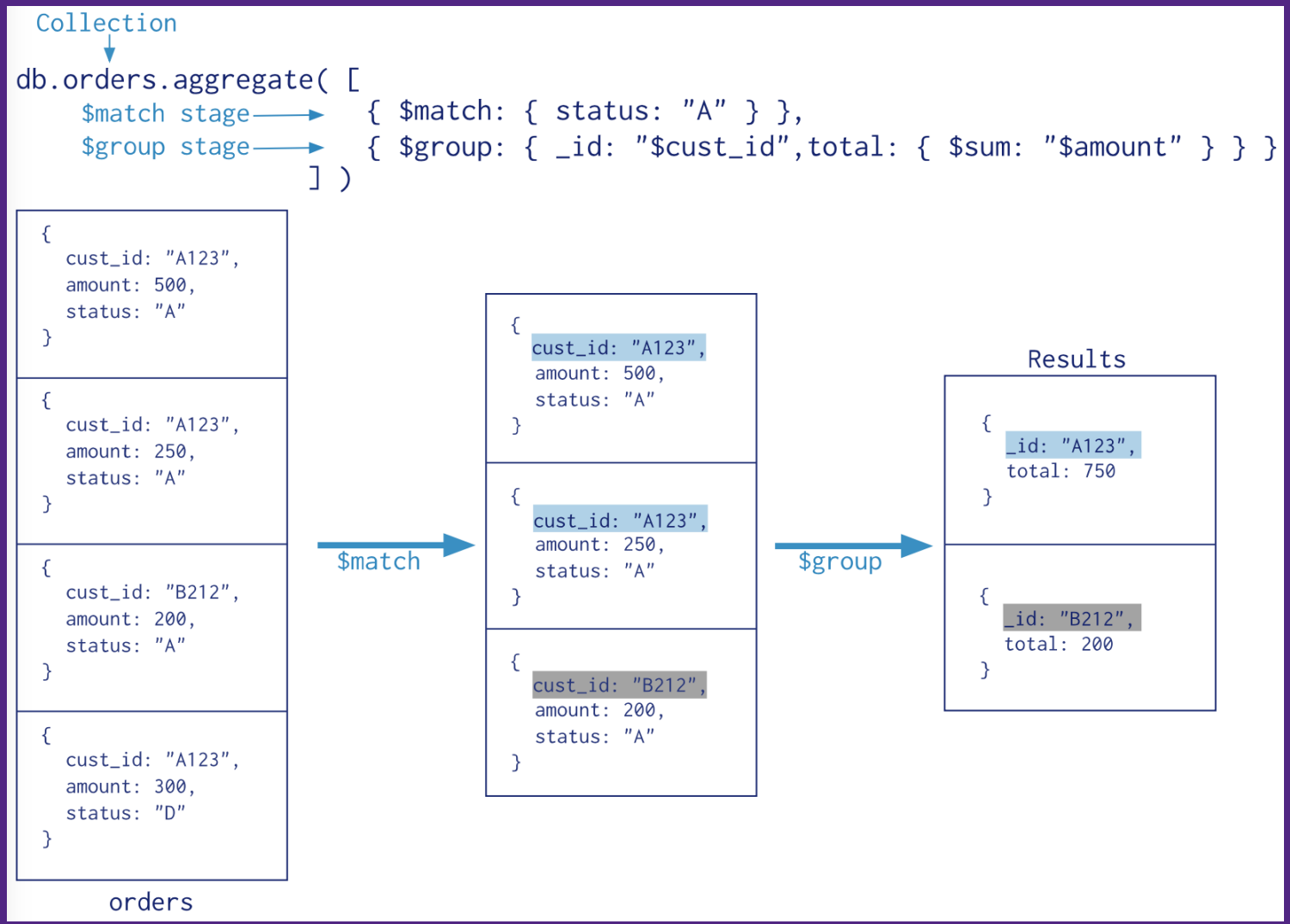
SQL

```
DELETE FROM users      <-- table
WHERE  status = 'D'    <-- delete criteria
```

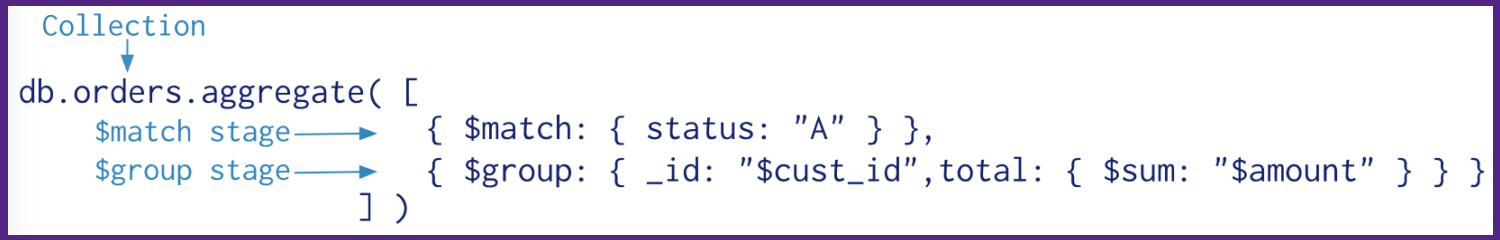
MongoDB

```
db.users.remove(       <-- collection
   { status: "D" }     <-- remove criteria
)
```

Aggregations



Aggregations



$match works basically like find

$group works by combining documents together **into new documents**

_id field defines the grouping

other fields are computed using groups of documents

Lots of operators available: `$sum`, `$max`, `$min`, ...

Note: when we want an operator to use a field, **prefix that field with a dollar sign.**

Above, "amount" is the field name in the documents.

Examples:

https://docs.mongodb.com/manual/tutorial/aggregation-with-user-preference-data/

https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/

Other Aggregation Pipeline Stages

- `$project`
  - Choose which fields to send along the pipeline. If directly examining output, can be helpful to add as a last step.
- `$bucket`
  - Group documents according to some criterion. E.g. group ages into <0, [0, 18), [18, 65), 65 and up.
- `$sort`
  - Sorts by given field
- $lookup
  - LEFT JOIN
- $unwind
  - Replicate document for each value in specified array
- Many many more!
- https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

Aggregations - Unwinding

EXAMPLE:

Given the following document from the `users` collection:

```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : ["golf", "racquetball"]
}
```

The `$unwind` operator would create the following documents:

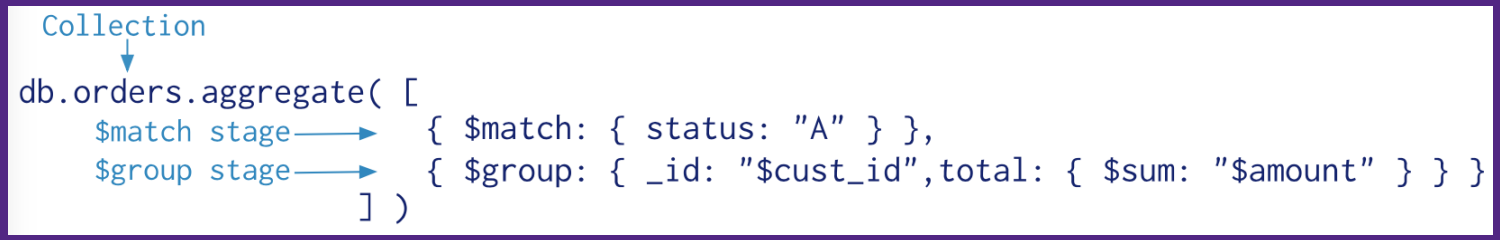
```
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : "golf"
}
{
  _id : "jane",
  joined : ISODate("2011-03-02"),
  likes : "racquetball"
}
```

After unwinding, how could I find the number of "likes" each sport has?

What if I have a 4 million users and everybody likes at least 2 sports?

Forget MongoDB for a moment: How could I do this without the unwinding step?

Aggregation efficiency



Think about the collection of documents that flows along the pipeline.

The sooner you can match and throw away documents, the better!

The MongoDB pipeline has query optimization:

https://www.mongodb.com/docs/manual/core/aggregation-pipeline-optimization/

MongoDB (mongosh) Syntax “Gotchas”

- Dollar signs
  - *field path*
    - Path to a field in the document. To specify a field path, use a string that prefixes the field name with a dollar sign ($).
      “$user.name”
  - Also used for operators, builtin functions, parameters,…
  - Except, apparently, when creating a new field???

- Quotation marks
  - { $user.name : “Arshin” } // *Error*
  - { “$user.name” : “Arshin” } // *OK*

- Array accesses

{ …

contribs: ["Turing machine", "Turing test", "Turingery"], …

}

Second element: “contribs.1”

A3 – MongoDB and MapReduce

- HIGHLY RECOMMEND this tutorial:
- https://www.mongodb.com/docs/manual/tutorial/getting-started/
- (go through all 6 tabs from *1. Switch Database* to *6. Aggregate*)

- https://www.mongodb.com/docs/manual/core/aggregation-pipeline/