MongoDB

1. Cursor
2. Aggregation Framework
    a. Aggregation Pipeline Stages
    b. Single Purpose Aggregation methods
    c. Updates with Aggregation Pipeline
    d. Aggregation Expressions
3. Store a JavaScript Function on the Server
4. Aggregation Pipeline as Alternative to Map-Reduce
5. Custom aggregation function or expression in JavaScript
6. Quiz on Aggregation

Cursor

- The Cursor is a MongoDB Collection of the document which is returned upon the find method execution.
- db.collection.find() function returns a cursor

Note: If a cursor inactive for 10 min then MongoDB server will automatically close that cursor.

Example: Cursor Object — Copy

```
var cursor = db.employees.find()
```

The cursor object has the following important methods:

Method	Description
cursor.count()	Returns the total number of documents referenced by a cursor.
cursor.forEach()	Iterates the cursor to apply a JavaScript function to each document from the cursor.
cursor.hasNext()	Returns true if a cursor can iterate further to return more documents.
cursor.isExhausted()	Returns true if the cursor is closed and there are no remaining objects in the batch.
cursor.itcount()	Counts the number of documents remaining in a cursor.
cursor.limit()	Specify the maximum number of documents the cursor will return.
cursor.map()	Applies a function to each document visited by the cursor and collects the return values from successive applications of the function into a Cursor object.
cursor.max()	Specifies the exclusive upper bound for a specific index in order to constrain the results of find().
cursor.min()	Specifies the inclusive lower bound for a specific index in order to constrain the results of find().
cursor.next()	Returns the next document from the result set.
cursor.pretty()	Display result in the readable format.
cursor.readConcern()	Specifies a level of isolation for read operations.
cursor.skip()	Skips the specified number of document for pagination.
cursor.sort()	Specifies the order in which the query returns matching documents.
cursor.toArray()	Returns an array that contains all the documents from a cursor.

(check practically - cursor-methods.js)

Aggregation Framework

Aggregation is a way of processing a large number of documents in a collection by means of passing them through different stages.

a. **Aggregation pipelines, which are the preferred method for performing aggregations.**

- An aggregation pipeline consists of one or more stages that process documents.
- Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.
- The documents that are output from a stage are passed to the next stage.
- All except the $out, $merge, and $geoNear stages can appear multiple times in a pipeline.
- allowDiskUse() allows MongoDB to use temporary files on disk to store data exceeding the 100 megabyte system memory limit while processing a blocking sort operation

Pipeline Stages

- **$project**
  This stage is used to select certain fields from a collection. We can also add, remove or reshape a key.
- **$match**
  It is used in filtering operation and it can reduce the number of documents that are given as input to the next stage.
- **$group**
  It groups all documents based on some keys.
- **$sort**
  It is used to sort all the documents. Sorting stage cannot use more than 100 MB of RAM
- **$skip**
  With this, it is possible to skip forward in the list of documents for a given amount of documents
- **$limit**
  This limits the number of documents to look at, by the given number starting from the current positions
- **$unwind**
  We can use unwind for all documents, that are using arrays.
- **$lookup**
  Performs a left outer join to an unsharded collection in the *same* database to filter in documents from the "joined" collection for processing. To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the "joined" collection. The $lookup stage passes these reshaped documents to the next stage.
- **$out & $merge**
  Takes the documents returned by the aggregation pipeline and writes them to a specified collection. The $out / $merge stage must be *the last stage* in the pipeline.

b. Single Purpose Aggregation Methods, which are simple but lack the capabilities of an aggregation pipeline.

- distinct(), count(), estimatedDocumentCount()

c. Updates with Aggregation Pipeline

- With the update operations, the aggregation pipeline can consist of the following stages: $addFields, $set $unset, $replaceRoot, $replaceWith

d. Aggregation Expressions

- There are many types of stages in the Aggregation Framework that don't allow expressions to be embedded. Examples of some of the most commonly used of these stages are:

  $match, $limit, $skip, $sort, $count, $lookup, $out

- There are Expressions like Arithmetic, Data, Array, Conditional, Boolean, Set, Text Search, String, Variable, Literal, Accumulator. (Will cover only accumulator)
- **Accumulator Operators**
  - $sum, $avg, $max, $min
    - both $group and $project stages.
  - $first, $last, $push, $addToSet
    - Available in $group stage only.

(check practically - Aggregation Pipeline Stages.js / Single Purpose Aggregation Methods.js / Updates with Aggregation Pipeline.js files for demo)

Store a JavaScript Function on the Server

Why we use SQL Stored procedures and How we achieve this in MongoDB?

In RDBMS (SQL Server),

- A stored procedure is SQL code stored on a database for repeated usage and to perform complex query operations. You can think of a stored procedure as equivalent to functions for databases.
- However, modern databases such as MongoDB have solved this problem by using a document model.

In NoSQL (MongoDB)

- Early versions of MongoDB tried to implement a concept similar to stored procedures as stored functions.
- There is a special system collection named system.js (as a collection under system folder) that can store JavaScript functions for reuse.
- These JavaScript functions could be stored on the server and later reused as part of a Map-Reduce API.
- However, these stored functions are now slowly being phased out.

Note - Do not store application logic in the database. There are performance limitations to running JavaScript inside of MongoDB. Application code also is typically most effective when it shares version control with the application itself.

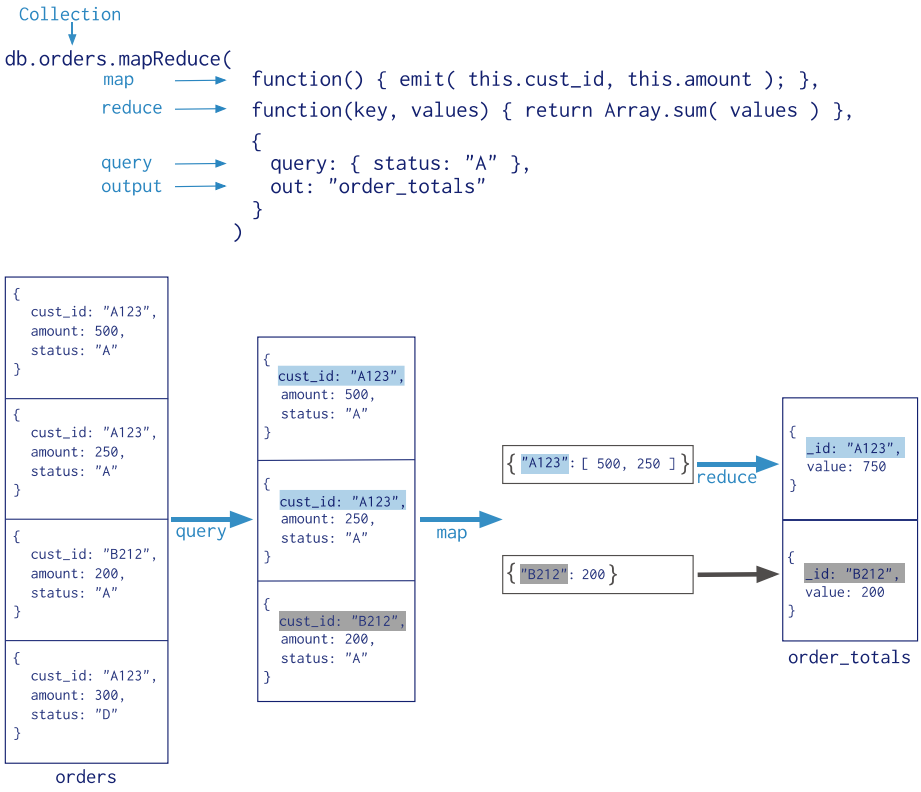
(check practically the StoredJSExample1.js / StoredJSExample2.js)

Aggregation Pipeline as Alternative to Map-Reduce

- An aggregation pipeline provides better performance and usability than a map-reduce operation.
- Various map-reduce expressions can be rewritten using aggregation pipeline operators, such as $group, $merge, and for custom functions Use $accumulator , $function operators to define custom aggregation expressions in JavaScript.

Example

The following map-reduce operation on the orders collection groups by the cust_id, and calculates the sum of the price for each cust_id:



- **Using mapReduce (deprecated)**

  (check practically using mapReduce for the above example - proc-1-storedJS-mapReduceFunction.js / proc-2-loadScriptsPerformMapReduce.js)

- ***Using* Aggregation *Pipeline as Alternative (Recommended)***
    - You can rewrite the operation into an aggregation pipeline without translating the map-reduce function to equivalent pipeline stages using $group, $merge

(check practically using aggregation pipeline for the above example without function - proc-3-mapReduceUsingAggregateWithoutFunc.js)

Custom aggregation $function or $expr in JavaScript

- Executing JavaScript inside an aggregation expression may decrease performance.
- Only use the $function operator if the provided pipeline operators cannot fulfill your application's needs.

- ***Using $function* Aggregation**

(check practically using aggregation pipeline for the above example : proc-4-mapReduceUsingAggregateWithFunc.js)

- **$expr in JavaScript**

(check practically different example - proc-5-customFunctionOrExprJS.js)

QUIZ on Aggregation

(check practically different example - 4-sql-to-mongodb-query.sql)

References -

https://www.mongodb.com/docs/manual/tutorial/store-javascript-function-on-server/
https://www.mongodb.com/features/stored-procedures
https://www.mongodb.com/docs/manual/reference/map-reduce-to-aggregation-pipeline/
https://www.mongodb.com/docs/manual/reference/aggregation-commands-comparison/
https://www.mongodb.com/docs/manual/reference/operator/aggregation/
https://www.mongodb.com/docs/manual/tutorial/update-documents-with-aggregation-pipeline/
https://www.mongodb.com/docs/manual/core/aggregation-pipeline-optimization/
https://www.marcelbelmont.com/nosql-workshop/docs/mongodb-aggregation.html
https://www.mongodb.com/docs/manual/reference/sql-comparison/