MongoDB

Scope: Aggregation

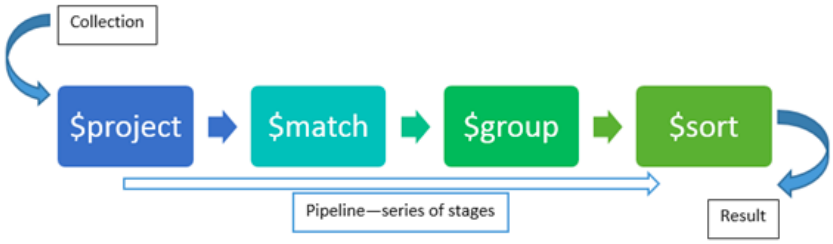
Ramu RC

1

Aggregation

2

Pipeline



3

Stages

- $project - select, reshape data
- $match - filter data
- $group - aggregate data
- $sort - sorts data
- $skip - skips data
- $limit - limit data
- $unwind - normalizes data

4

Example

collection

- { "_id" : "model_0", "brand_name" : "nokia", "phone_type" : "smart", "price" : 5378 }
- { "_id" : "model_1", "brand_name" : "nokia", "phone_type" : "smart", "price" : 7635 }
- { "_id" : "model_2", "brand_name" : "samsung", "phone_type" : "feature", "price" : 1305 }
- { "_id" : "model_3", "brand_name" : "samsung", "phone_type" : "smart", "price" : 9000 }
- { "_id" : "model_4", "brand_name" : "nokia", "phone_type" : "smart", "price" : 8931 }

$match

- { "_id" : "model_0", "brand_name" : "nokia", "phone_type" : "smart", "price" : 5378 }
- { "_id" : "model_1", "brand_name" : "nokia", "phone_type" : "smart", "price" : 7635 }
- { "_id" : "model_3", "brand_name" : "samsung", "phone_type" : "smart", "price" : 9000 }
- { "_id" : "model_4", "brand_name" : "nokia", "phone_type" : "smart", "price" : 8931 }

$group - result set

- { "_id" : "nokia", "total" : 21944 }
- { "_id" : "samsung", "total" : 9000 }

5

Σ AGGREGATION FRAMEWORK

Pipeline Stages

$project	Change the set of documents by modifying keys and values. This is a 1:1 mapping.
$match	This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage. This can be used for example if aggregation should only happen on a subset of the data.
$group	This does the actual aggregation and as we are grouping by one or more keys this can have a reducing effect on the amount of documents.
$sort	Sorting the documents one way or the other for the next stage. It should be noted that this might use a lot of memory. Thus if possible one should always try to reduce the amount of documents first.
$skip	With this it is possible to skip forward in the list of documents for a given amount of documents. This allows for example starting only from the 10th document. Typically this will be used together with "$sort" and especially together with "$limit".
$limit	This limits the amount of documents to look at by the given number starting from the current position.
$unwind	This is used to unwind document that are using arrays. When using an array the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

Comparison with SQL

WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM	$sum
COUNT	$sum
JOIN	$unwind

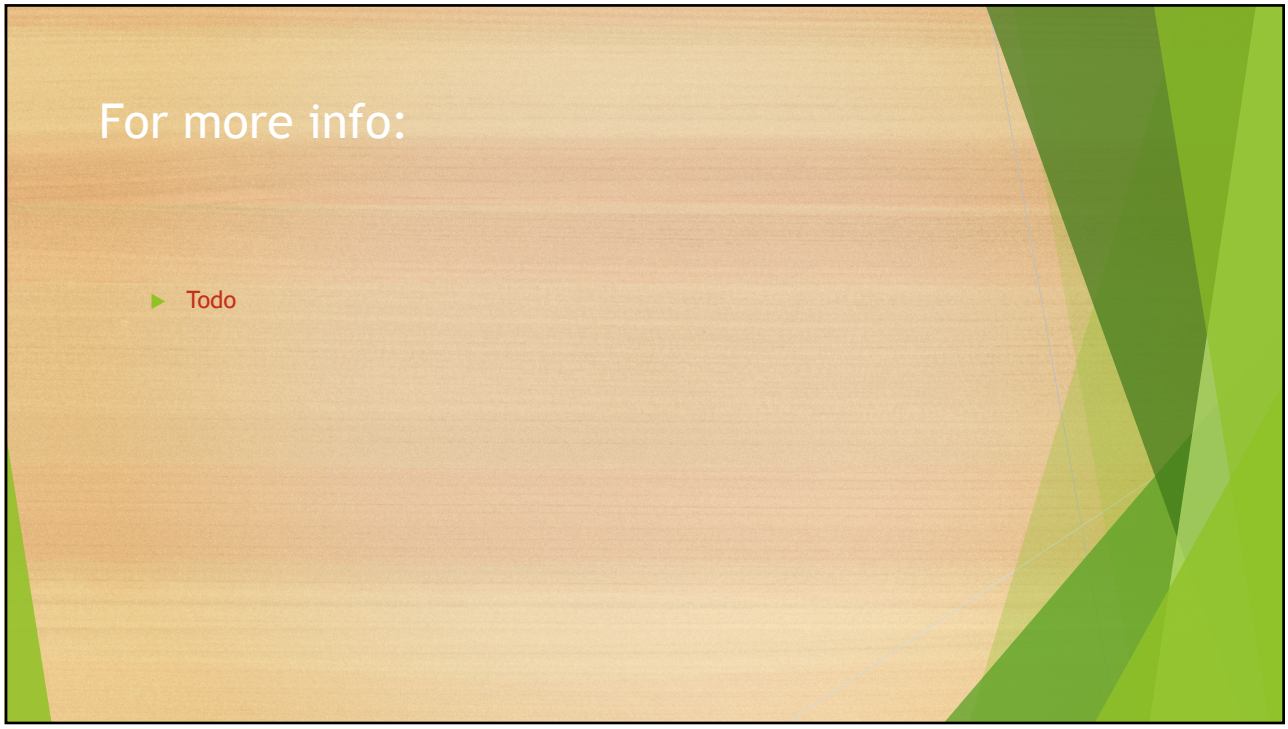
Aggregation Examples

`db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$sum : 1}}}])`	Counts the number of ships per operator, would be in SQL: `SELECT operator, count(*) FROM ships GROUP BY operator;`
`db.ships.aggregate([{$project : {_id : 0, operator : {$toLower : "$operator"}, crew : {"$multiply" : ["$crew",10]}}}])`	Combination of $project-stage and $group-stage.

Aggregation Expressions

$sum	Summing up values	`db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$sum : "$crew"}}}])`
$avg	Calculating average values	`db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$avg : "$crew"}}}])`
$min / $max	Finding min/max values	`db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$min : "$crew"}}}])`
$push	Pushing values to a result array	`db.ships.aggregate([{$group : {_id : "$operator", classes : {$push: "$class"}}}])`
$addToSet	Pushing values to a result array without duplicates	`db.ships.aggregate([{$group : {_id : "$operator", classes : {$addToSet : "$class"}}}])`
$first / $last	Getting the first / last document	`db.ships.aggregate([{$group : {_id : "$operator", last_class : {$last : "$class"}}}])`

6



7