OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование

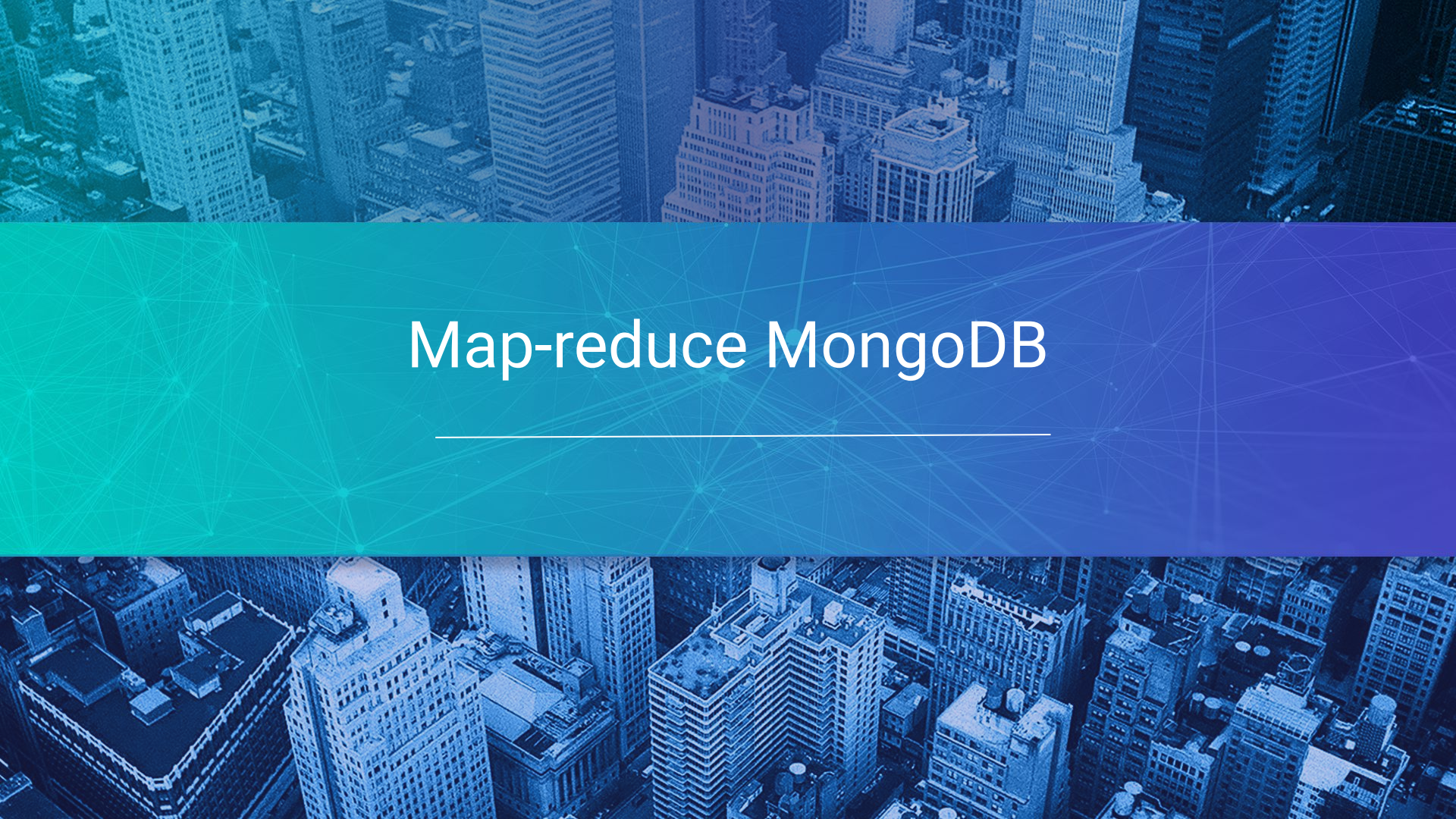
Меня хорошо видно && слышно?

Ставьте + , если все хорошо

Напишите в чат, если есть проблемы

заодно проверяем, включена ли запись занятия

Включил Юджин запись ли ты



Map-reduce MongoDB

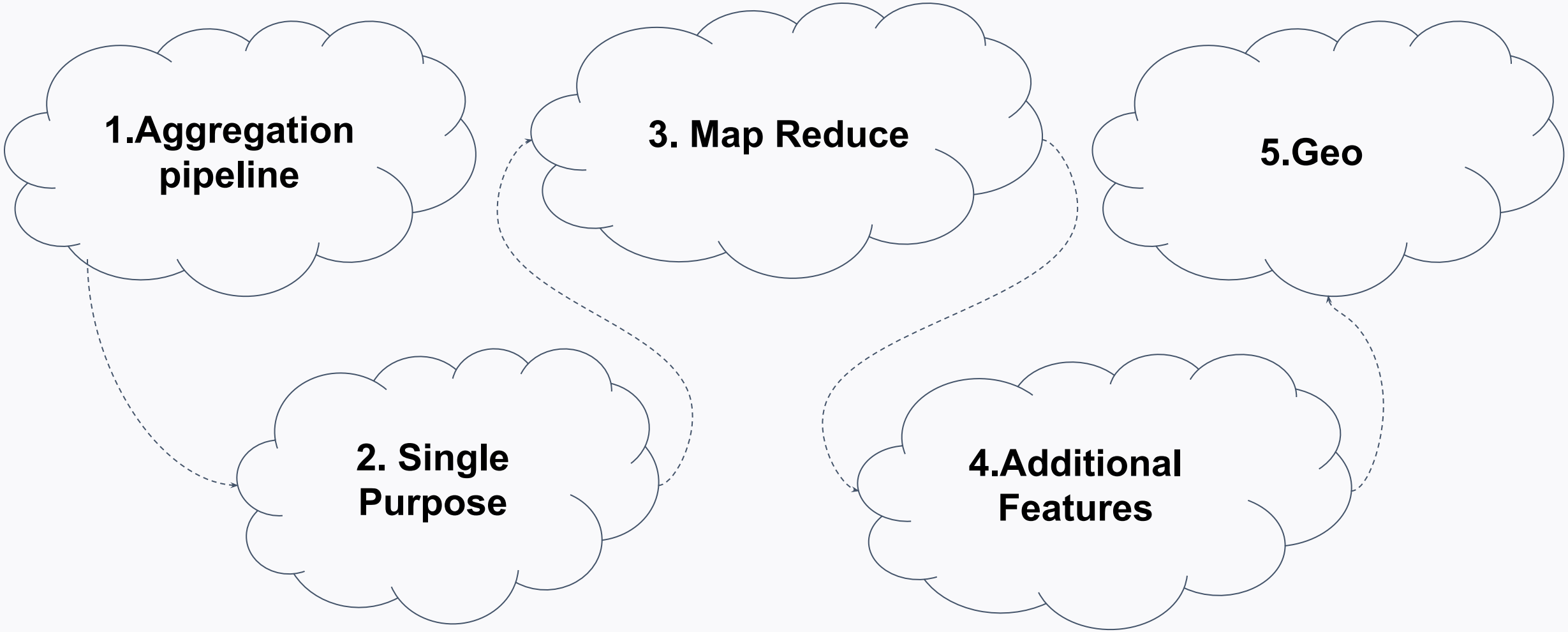
Правила вебинара

Активно участвуем

Задаем вопрос в чат

Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара

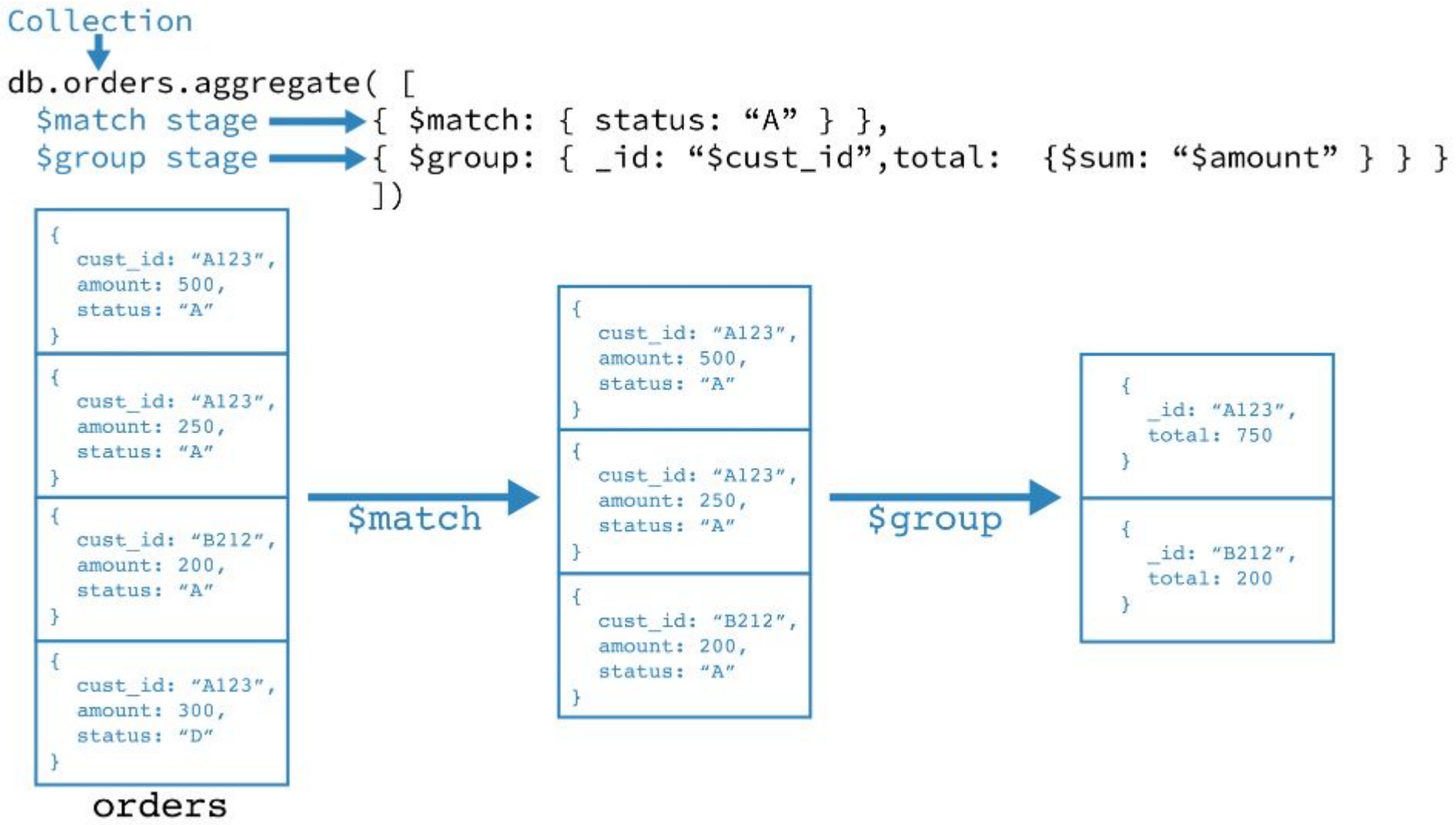



Aggregation pipeline

Aggregation Pipeline

Aggregation operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single purpose aggregation methods.

Aggregation Pipeline



Aggregation Pipeline Stages

db.collection.aggregate([{ <stage> }, ...])

https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference

Aggregation Pipeline

практика

Aggregation Pipeline

SQL Aggregation Terms and Corresponding MongoDB Aggregation Operators

SQL Term	MongoDB Operator
SELECT	$project
WHERE	$match
GROUP BY	$group
HAVING	$match
ORDER BY	$sort
LIMIT	$limit
SUM	$sum
COUNT	$count
JOIN	$lookup

Aggregation Pipeline

Ограничения:

- All except the $out, $merge, and $geoNear stages can appear multiple times in a pipeline.

- Pipeline stages have a limit of 100 MiB (100 * 1024 * 1024 bytes) of RAM
- https://docs.mongodb.com/manual/core/aggregation-pipeline-limits/

- you can set the allowDiskUse option in the aggregate() method

Aggregation Pipeline & Indexes

Pipeline Operators and Indexes

MongoDB's query planner analyzes an aggregation pipeline to determine whether indexes can be used to improve pipeline performance. For example, the following pipeline stages can take advantage of indexes:

$match

The $match stage can use an index to filter documents if it occurs at the beginning of a pipeline.

$sort

The $sort stage can use an index as long as it is not preceded by a $project, $unwind, or $group stage.

$group

The $group stage can sometimes use an index to find the first document in each group

See Optimization to Return the First Document of Each Group for an example.

$geoNear

The $geoNear pipeline operator takes advantage of a geospatial index. When using $geoNear, the $geoNear pipeline operation must appear as the first stage in an aggregation pipeline.

Aggregation pipeline optimization

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

Статья по более менее подробному разбору этапов пайплайна

https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/

Single Purpose Aggregation Operations

Single Purpose

db.collection.count()

db.collection.count(*query*, *options*)

OPtions:

limit	integer	Optional. The maximum number of documents to count.
skip	integer	Optional. The number of documents to skip before counting.
hint	string or document	Optional. An index name hint or specification for the query.
maxTimeMS	integer	Optional. The maximum amount of time to allow the query to run.
readConcern	string	Optional. Specifies the read concern. The default level is "local".

On a sharded cluster, db.collection.count() without a query predicate can result in an *inaccurate* count if orphaned documents exist or if a chunk migration is in progress.

db.orders.count({ ord_dt: { $gt: new Date('01/01/2012') } }

The query is equivalent to the following:

db.orders.find({ ord_dt: { $gt: new Date('01/01/2012') } }).count()

Single Purpose

db.collection.distinct()

db.collection.distinct(*field*, *query*, *options*)

Options:

- collation

пример

Single Purpose

db.collection.estimatedDocumentCount()

db.collection.estimatedDocumentCount(*options*)

- does not take a query filter and instead uses metadata to return the count for a collection.
- After an unclean shutdown, the count may be incorrect.
- Run validate on each collection on the mongod to restore the correct statistics after an unclean shutdown.


Map Reduce

Map Reduce

Map-Reduce

MongoDB also provides map-reduce operations to perform aggregation. In general, map-reduce operations have two phases: a *map* stage that processes each document and *emits* one or more objects for each input document, and *reduce* phase that combines the output of the map operation. Optionally, map-reduce can have a *finalize* stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results.

Map-reduce uses custom JavaScript functions to perform the map and reduce operations, as well as the optional *finalize* operation. While the custom JavaScript provide great flexibility compared to the aggregation pipeline, in general, map-reduce is less efficient and more complex than the aggregation pipeline.

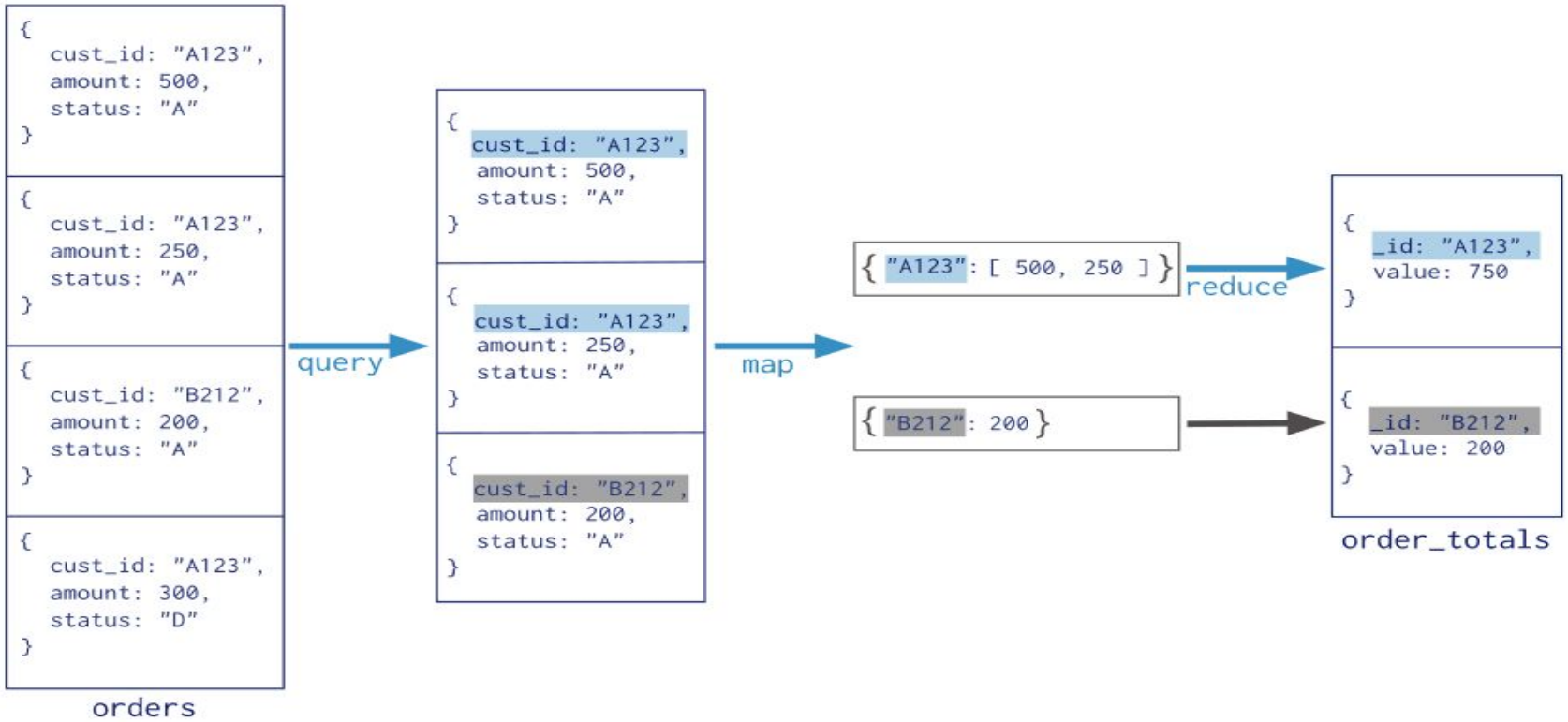
Map-reduce can operate on a sharded collection. Map-reduce operations can also output to a sharded collection. See Map-Reduce and Sharded Collections for details.

Note

Starting in MongoDB 2.4, certain mongo shell functions and properties are inaccessible in map-reduce operations. MongoDB 2.4 also provides support for multiple JavaScript operations to run at the same time. Before MongoDB 2.4, JavaScript code executed in a single thread, raising concurrency issues for map-reduce.

Map Reduce

```
Collection
    ↓
db.orders.mapReduce(
        map      ──→   function() { emit( this.cust_id, this.amount ); },
        reduce   ──→   function(key, values) { return Array.sum( values ) },
                       {
        query    ──→     query: { status: "A" },
        output   ──→     out: "order_totals"
                       }
                   )
```



Map Reduce

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction}, {   //reduce function
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

https://docs.mongodb.com/manual/reference/command/mapReduce/

Map Reduce

требования на реализацию функции **reduce**. Вот они:

1. Тип возвращаемого значения функции **reduce** должен совпадать с типом значения, которое выдается функцией **map** (второй параметр функции **emit**)
2. Должно выполняться равенство (ассоциативность):

   reduce(key, [A, reduce(key, [B, C])]) == reduce(key, [A, B, C])

3. Повторное применение операции **Reduce** к полученной паре <*ключ, значение*> не должно влиять на результат (идемпотентность)

   reduce(key, [reduce(key, valuesArray)]) == reduce(key, valuesArray)

4. Порядок значений, передаваемых функции **reduce**, не должен влиять на результат (коммутативность)

   reduce(key, [A, B]) == reduce(key, [B, A])

   https://docs.mongodb.com/manual/reference/command/mapReduce/#requirements-for-the-map-function

Map Reduce. Практика

```
{       name : "John",
        age : 23,
        interests : ["football", "IT", "cooking"]
}
```

На выходе мы хотим получить коллекцию такого типа:

```
{       key: "football",
        value: 1349
},
{       key: "MongoDB",
        value: 58
},
```

Map Reduce

Рассмотрим другую задачу. Предположим, мы хотим узнать среднее количество интересов у людей разных возрастов

??

Если внимательно посмотреть на **требования** к функции **reduce**, то становится ясно, что в ее рамках среднее арифметическое вычислить **не удастся**, так как эта математическая операция **не удовлетворяет 1 и 2 требованию:**

- **кроме несовпадения типов (int -> double),**
- **среднее средних не равно среднему**

Map Reduce

Чтобы в итоговой коллекции получить искомое среднее арифметическое, можно воспользоваться операцией **Finalize** — она применяется к финальной паре *<key, value>*, полученной после выполнения всех операций **Reduce** с ключом **key**:

```
function finalize(key, reducedValue) {
    return reducedValue.interests_count / reducedValue.count;
}
```

Map Reduce

Starting in version 4.2, MongoDB deprecates:

- The map-reduce option to *create* a new sharded collection as well as the use of the sharded option for map-reduce. To output to a sharded collection, create the sharded collection first. MongoDB 4.2 also deprecates the replacement of an existing sharded collection.
- The explicit specification of nonAtomic: false option.

Map Reduce

The **map function** has the following requirements:

- The map function should *not* access the database for any reason.
- The map function should be pure, or have *no* impact outside of the function (i.e. side effects.)
- A single emit can only hold half of MongoDB's maximum BSON document size.
- The map function may optionally call emit(key,value) any number of times to create an output document associating key with value.

Map Reduce

The **reduce function** exhibits the following behaviors:

- The reduce function should *not* access the database, even to perform read operations.
- The reduce function should *not* affect the outside system.
- MongoDB will **not** call the reduce function for a key that has only a single value. The values argument is an array whose elements are the value objects that are “mapped” to the key.
- MongoDB can invoke the reduce function more than once for the same key. In this case, the previous output from the reduce function for that key will become one of the input values to the next reduce function invocation for that key.
- The reduce function can access the variables defined in the scope parameter.
- The inputs to reduce must not be larger than half of MongoDB’s maximum BSON document size. This requirement may be violated when large documents are returned and then joined together in subsequent reduce steps.

Map Reduce

The **finalize function** has the following prototype:

```
function(key, reducedValue) {
  ...
  return modifiedObject;
}
```

The finalize function receives as its arguments a key value and the reducedValue from the reduce function. Be aware that:

- The finalize function should *not* access the database for any reason.
- The finalize function should be pure, or have *no* impact outside of the function (i.e. side effects.)
- The finalize function can access the variables defined in the scope parameter.

Map Reduce

Примеры написания функций

Простая методика построения фильтров товаров с помощью MongoDb и MapReduce

https://habr.com/ru/post/186572/

MongoDB Mapreduce Tutorial – Real-time Example & Commands

https://data-flair.training/blogs/mongodb-mapreduce/

Официальная документация с примерами

https://docs.mongodb.com/manual/reference/command/mapReduce/


Additional Features

Фичи

	aggregate / db.collection.aggregate()	mapReduce / db.collection.mapReduce()
Description	Designed with specific goals of improving performance and usability for aggregation tasks. Uses a “pipeline” approach where objects are transformed as they pass through a series of pipeline operators such as $group, $match, and $sort. See Aggregation Pipeline Operators for more information on the pipeline operators.	Implements the Map-Reduce aggregation for processing large data sets.

Фичи

	aggregate / db.collection.aggregate()	mapReduce / db.collection.mapReduce()
Key Features	Pipeline operators can be repeated as needed. Pipeline operators need not produce one output document for every input document. Can also generate new documents or filter out documents. With the addition of $merge in version 4.2, can create on-demand materialized views, where the content of the output collection can be updated incrementally the pipeline is run. $merge can incorporate results (insert new documents, merge documents, replace documents, keep existing documents, fail the operation, process documents with a custom update pipeline) into an existing collection.	In addition to grouping operations, can perform complex aggregation tasks as well as perform incremental aggregation on continuously growing datasets. See Map-Reduce Examples and Perform Incremental Map-Reduce.

Фичи

	aggregate / db.collection.aggregate()	mapReduce / db.collection.mapReduce()
Flexibility	Limited to the operators and expressions supported by the aggregation pipeline. However, can add computed fields, create new virtual sub-objects, and extract sub-fields into the top-level of results by using the $project pipeline operator. See $project for more information as well as Aggregation Pipeline Operators for more information on all the available pipeline operators.	Custom map, reduce and finalize JavaScript functions offer flexibility to aggregation logic. See mapReduce for details and restrictions on the functions.

Фичи

	aggregate / db.collection.aggregate()	mapReduce / db.collection.mapReduce()
Output Results	Returns results as a cursor. If the pipeline includes the $out stage or $merge stage, the cursor is empty. With $out, you can replace an existing output collection completely or output to a new collection. See $out for details. With $merge, you can output to a new or existing collection. For existing collections, you can specify how to incorporate the results into the output collection (insert new documents, merge documents, replace documents, keep existing documents, fail the operation, process documents with a custom update pipeline). See $merge for details.	Returns results in various options (inline, new collection, merge, replace, reduce). See mapReduce for details on the output options.

Geo

Geo

https://docs.mongodb.com/manual/reference/command/geoSearch/#dbcmd.geoSearch

The geoSearch command accepts a document that contains the following fields.

geoSearch	string	The collection to query.
search	document	Query to filter documents.
near	array	Coordinates of a point.
maxDistance	number	Optional. Maximum distance from the specified point.
limit	number	Optional. Maximum number of documents to return.
readConcern	document	Optional. Specifies the read concern

Geo

Limit

Unless specified otherwise, the geoSearch command limits results to 50 documents.

Sharded Clusters

geoSearch is not supported for sharded clusters.

Geo

```
db.runCommand({
   geoSearch : "places",
   near: [ -73.9667, 40.78 ],
   maxDistance : 6,
   search : { type : "restaurant" },
   limit : 30
})
```

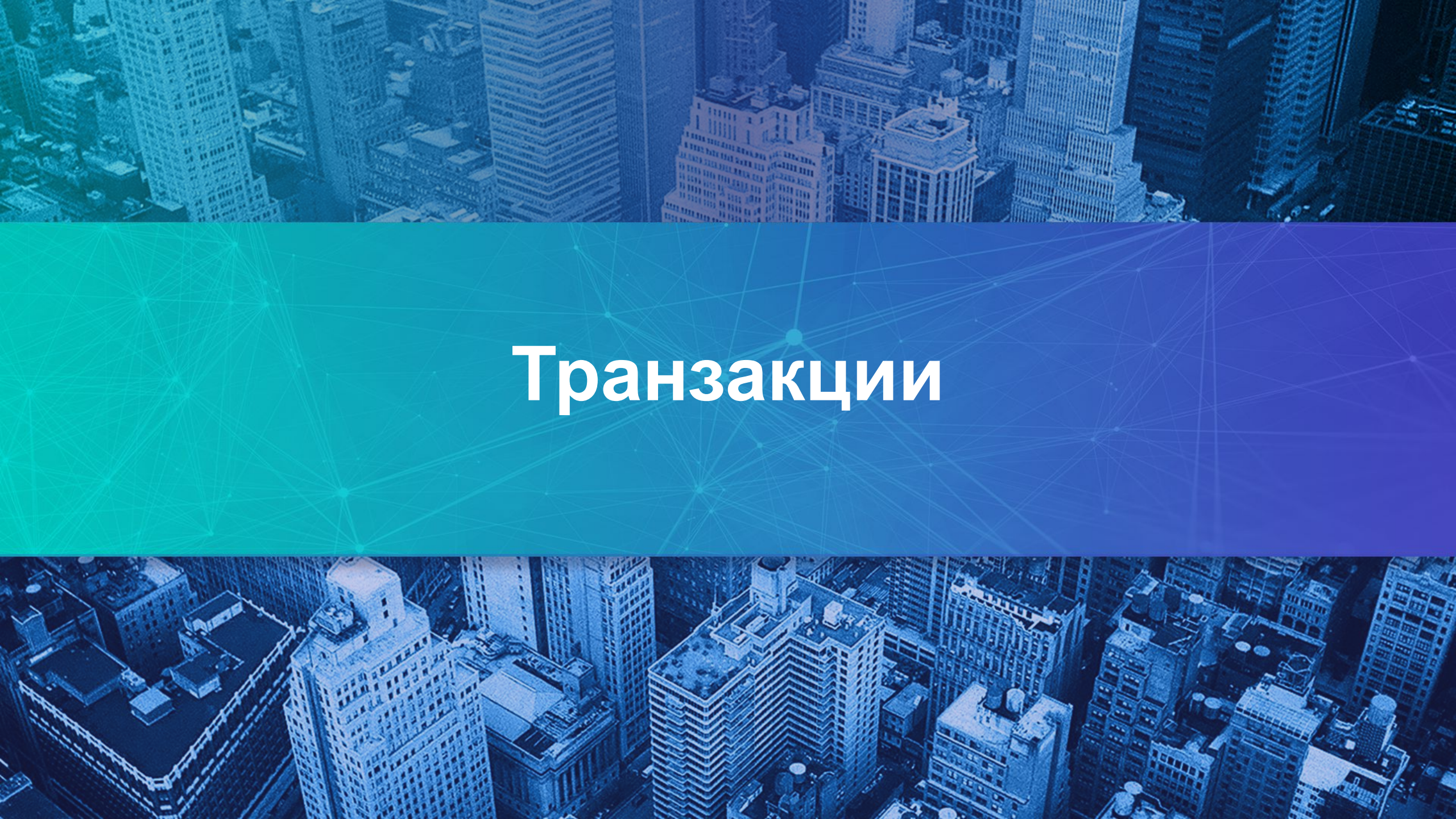
https://docs.mongodb.com/manual/tutorial/geospatial-tutorial/

Geo

Дипломный проект моего выпускника группы NoSQL 2020-09

Аспекты учета и поиска геоинформационных объектов с задействованием MongoDB

https://github.com/BorisPlus/mongodb_geo



Транзакции

Транзакции

https://docs.mongodb.com/manual/core/transactions/

In version 4.2, MongoDB introduces distributed transactions, which adds support for multi-document transactions on sharded clusters and incorporates the existing support for multi-document transactions on replica sets.

To use transactions on MongoDB 4.2 deployments (replica sets and sharded clusters), clients **must** use MongoDB drivers updated for MongoDB 4.2


View

Views

https://docs.mongodb.com/manual/core/views/

Materialized view

On-Demand Materialized Views

https://docs.mongodb.com/manual/core/materialized-views/


Triggers

Triggers

а вот нету их %)

Зато есть change Streams

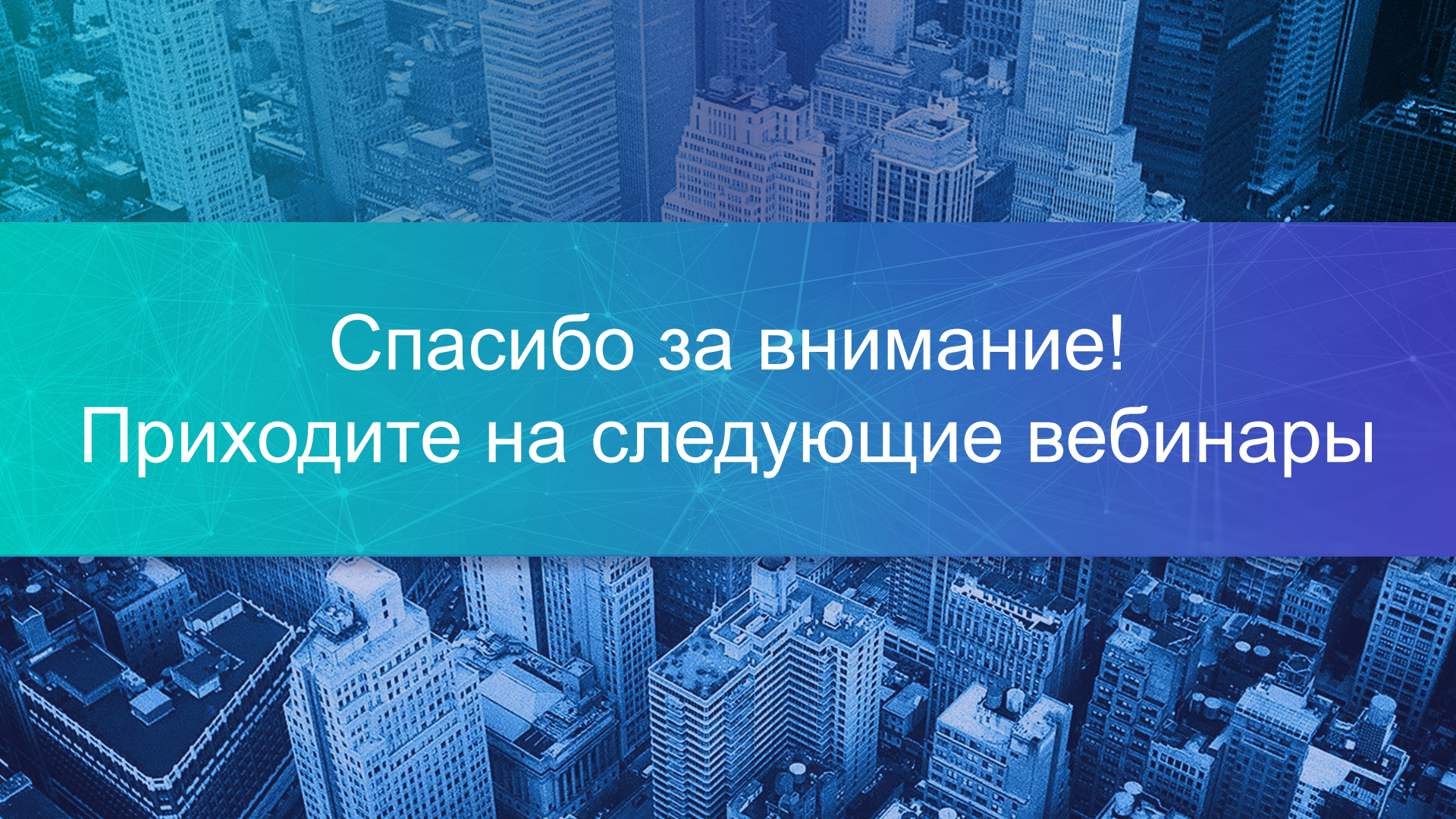
https://docs.mongodb.com/manual/changeStreams/


ДЗ

ДЗ

нет его %)

Заполните, пожалуйста,
опрос о занятии по ссылке в чате


Спасибо за внимание!
Приходите на следующие вебинары