

BD2 NOSQL - MONGO

SERGIO ÁLVAREZ

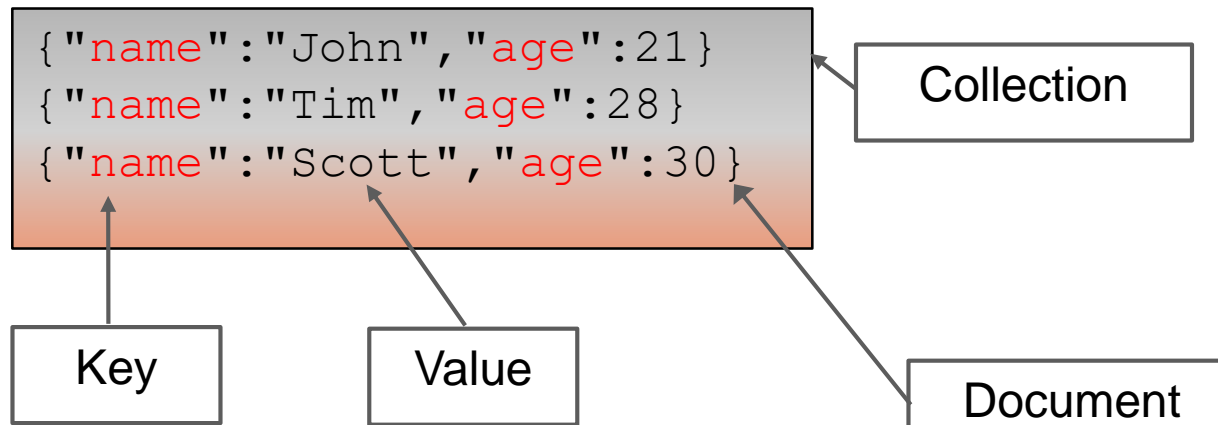
VERSIÓN 1.0

JSON

```
{  
  "Key": "Value"  
  "string": "John",  
  "number": 123.45,  
  "boolean": true,  
  "array": [ "a", "b", "c" ],  
  "object": { "str": "Miller", "num": 711 },  
  "value": NULL,  
  "date": ISODate("2013-10-01T00:33:14.000Z")  
}
```

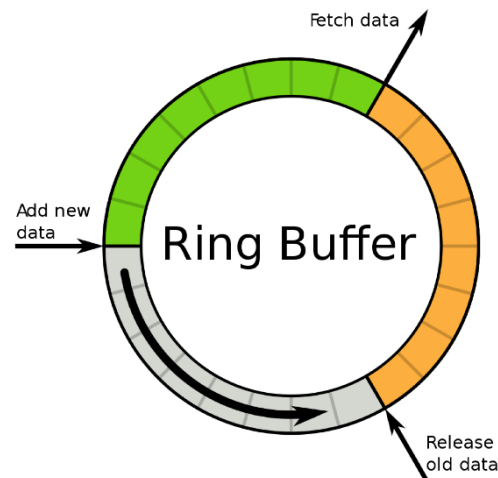
TRANSACCIONES Y CONVENCIONES

Name	Age
John	21
Tim	28
Scott	30




CAPPED COLLECTIONS

Una colección "capped" es un tipo especial de colección que tiene un tamaño fijo o un número fijo de elementos. Una vez que la colección está "llena", los elementos más viejos se eliminan cada vez que añadimos nuevos.



USE CUSTOM COLLATION

Field	Type	Description
locale	string	<p>The ICU locale. See Supported Languages and Locales for a list of supported locales.</p> <p>To specify simple binary comparison, specify <code>locale</code> value of <code>"simple"</code>.</p>
strength	integer	<p>Optional. The level of comparison to perform. Corresponds to ICU Comparison Levels . Possible values are:</p>

Se realiza un índice que no distingue entre mayúsculas y minúsculas al especificar una intercalación con una intensidad de 1 o 2. Puede crear un índice insensible a mayúsculas y minúsculas como este:

```
db.myCollection.createIndex({city: 1}, {collation: {locale: "en", strength: 2}});
```

VALIDACIONES

SCHEMA VALIDATION

validationAction	Description
"error"	Default Documents must pass validation before the write occurs. Otherwise, the write operation fails.
"warn"	Documents do not have to pass validation. If the document fails validation, the write operation logs the validation failure.
validationLevel	Description
"off"	No validation for inserts or updates.
"strict"	Default Apply validation rules to all inserts and all updates.
"moderate"	Apply validation rules to inserts and to updates on existing <i>valid</i> documents. Do not apply rules to updates on existing <i>invalid</i> documents.

SCHEMA VALIDATION

```
db.createCollection("students", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "name", "year", "major", "address" ],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        year: {
          bsonType: "int",
          minimum: 2017,
          maximum: 3017,
          description: "must be an integer in [ 2017, 3017 ] and is required"
        },
        major: {
          enum: [ "Math", "English", "Computer Science", "History", null ],
          description: "can only be one of the enum values and is required"
        },
        gpa: {
          bsonType: [ "double" ],
          description: "must be a double if the field exists"
        }
      }
    }
  },
  validationAction: "warn"
})
```

```
db.createCollection( "contacts",
  { validator: { $or:
    [
      { phone: { $type: "string" } },
      { email: { $regex: /@mongodb\.com$/ } },
      { status: { $in: [ "Unknown", "Incomplete" ] } }
    ]
  }
})
```


MOTORES Y VERSIÓN ENTERPRICE

STORAGE ENGINES

The [storage engine](#) is the component of the database that is responsible for managing how data is stored, both in memory and on disk. MongoDB supports multiple storage engines, as different engines perform better for specific workloads. Choosing the appropriate storage engine for your use case can significantly impact the performance of your applications.

NOTE:

Starting in version 4.2, MongoDB removes the deprecated [MMAPv1](#) storage engine.

► WiredTiger Storage Engine (*Default*)

[WiredTiger](#) is the default storage engine starting in MongoDB 3.2. It is well-suited for most workloads and is recommended for new deployments. WiredTiger provides a document-level concurrency model, checkpointing, and compression, among other features.

In MongoDB Enterprise, WiredTiger also supports [Encryption at Rest](#). See [Encrypted Storage Engine](#).

► In-Memory Storage Engine

[In-Memory Storage Engine](#) is available in [MongoDB Enterprise](#). Rather than storing documents on-disk, it retains them in-memory for more predictable data latencies.

Encrypted Storage Engine

New in version 3.2.

ENTERPRISE FEATURE:

[Available in MongoDB Enterprise only.](#)

IMPORTANT:

Available for the WiredTiger Storage Engine only.

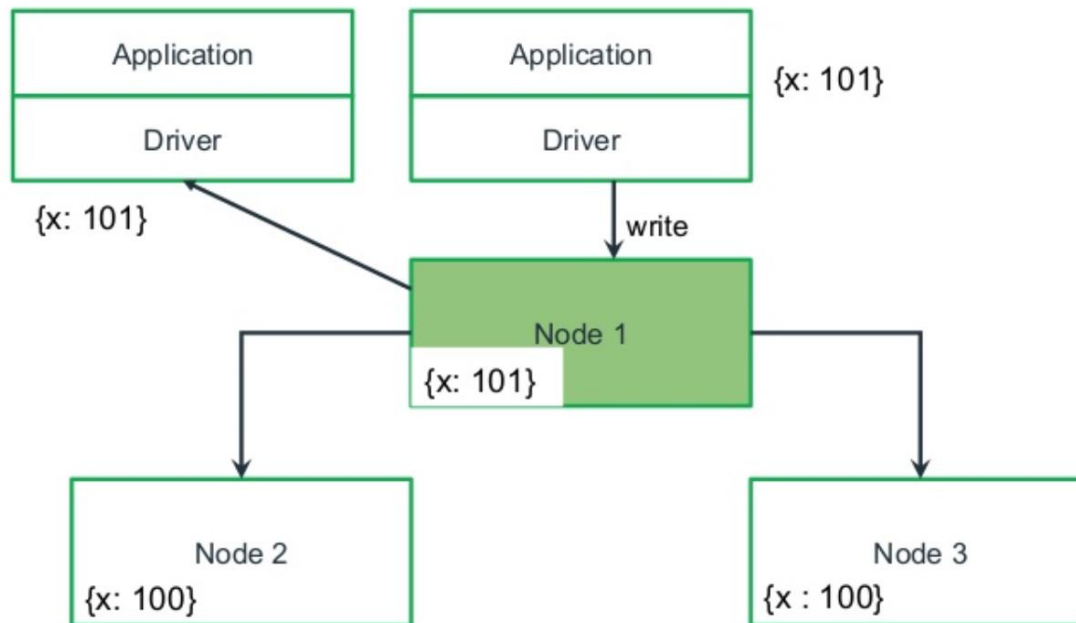
MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine. This feature allows MongoDB to encrypt data files such that only parties with the decryption key can decode and read the data.

NIVEL DEL CONSISTENCIA

LECTURA LOCAL

ReadConcern local

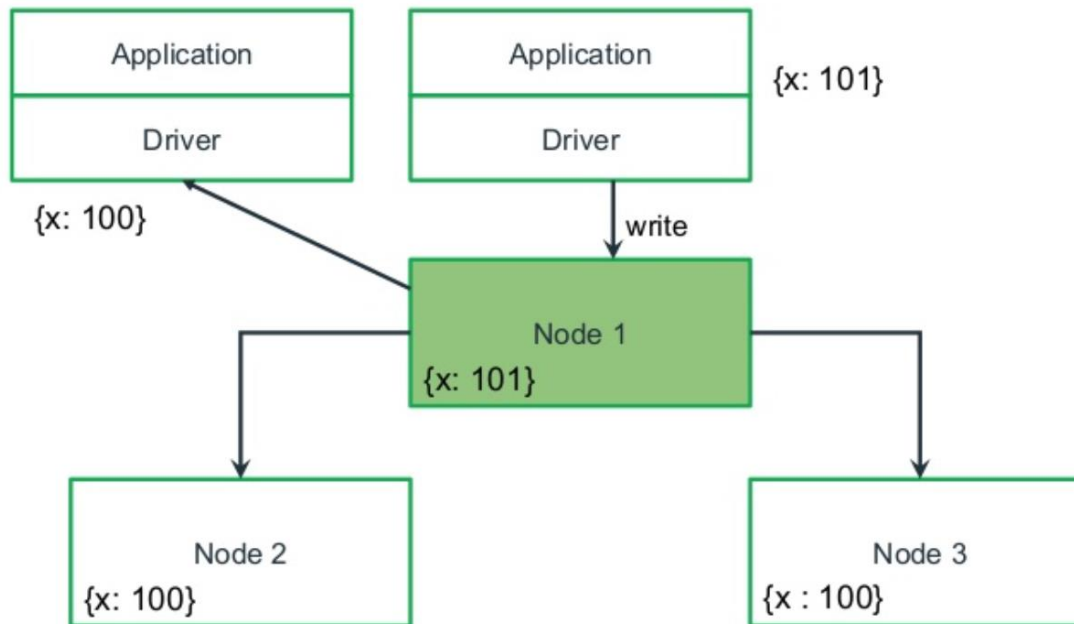
```
db.test.find({x : {$gt : 100}})
      .readConcern("local")
```



LECTURA MAYORÍA

ReadConcern: Majority

```
db.test.find({x : {$gt : 100}})
  .readConcern("majority")
```



ESCRITURAS

Write Concern

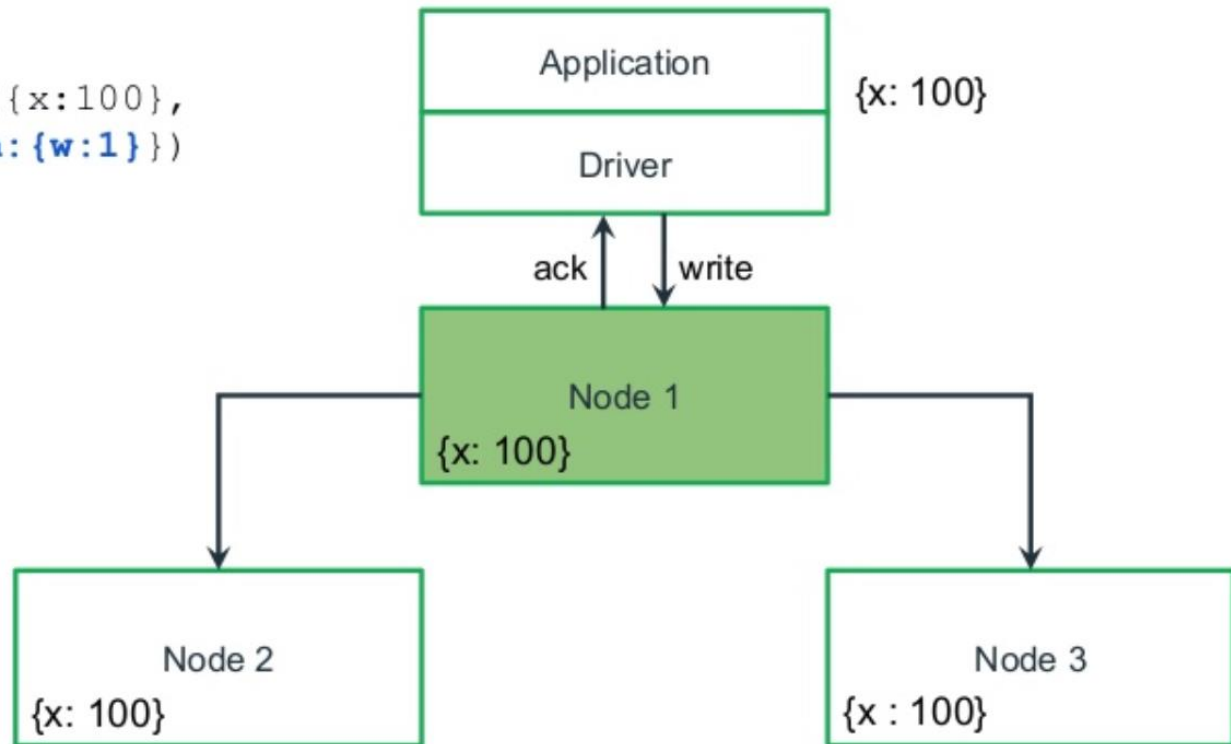
- Intelligent write receipt/confirmation
 - Specifies the the number of nodes that must have written the write to disk
 - Default number is 1

```
db.test.insert({x:100},{writeConcern:{w:2}})
```

ESCRITURA EN 1

Write Concern 1

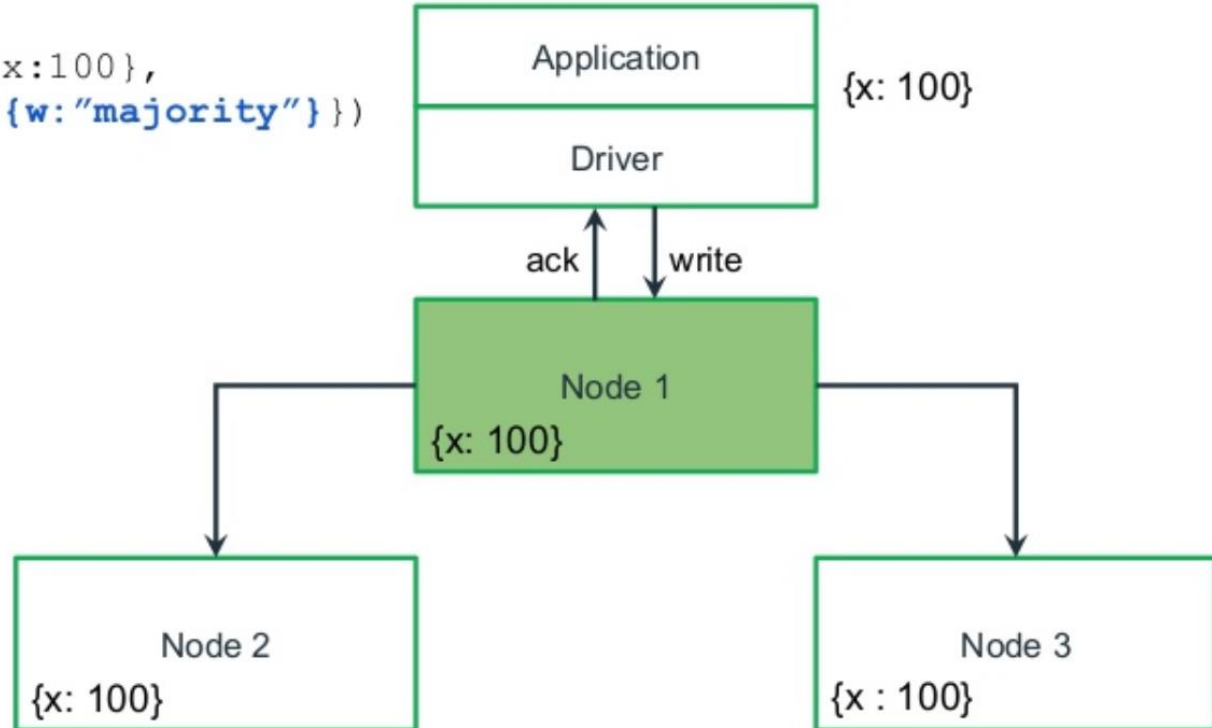
```
db.test.insert({x:100},  
  {writeConcern:{w:1}})
```



ESCRITURA EN MAYORIA

Write Concern Majority

```
db.test.insert({x:100},  
  {writeConcern:{w:"majority"}})
```



DOES MONGODB SUPPORT TRANSACTIONS?

Because a single document can contain related data that would otherwise be modeled across separate parent-child tables in a relational schema, MongoDB's atomic single-document operations already provide transaction semantics that meet the data integrity needs of the majority of applications. One or more fields may be written in a single operation, including updates to multiple sub-documents and elements of an array. The guarantees provided by MongoDB ensure complete isolation as a document is updated; any errors cause the operation to roll back so that clients receive a consistent view of the document.

However, for situations that require atomicity of reads and writes to multiple documents (in a single or multiple collections), MongoDB supports multi-document transactions:

- **In version 4.0**, MongoDB supports multi-document transactions on replica sets.
- **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.

TRANSACCIONES

```
client = MongoClient(uriString)
wc_majority = WriteConcern("majority", wtimeout=1000)

# Prereq: Create collections.
client.get_database(
    "mydb1", write_concern=wc_majority).foo.insert_one({'abc': 0})
client.get_database(
    "mydb2", write_concern=wc_majority).bar.insert_one({'xyz': 0})

# Step 1: Define the callback that specifies the sequence of operations to perform inside
def callback(session):
    collection_one = session.client.mydb1.foo
    collection_two = session.client.mydb2.bar

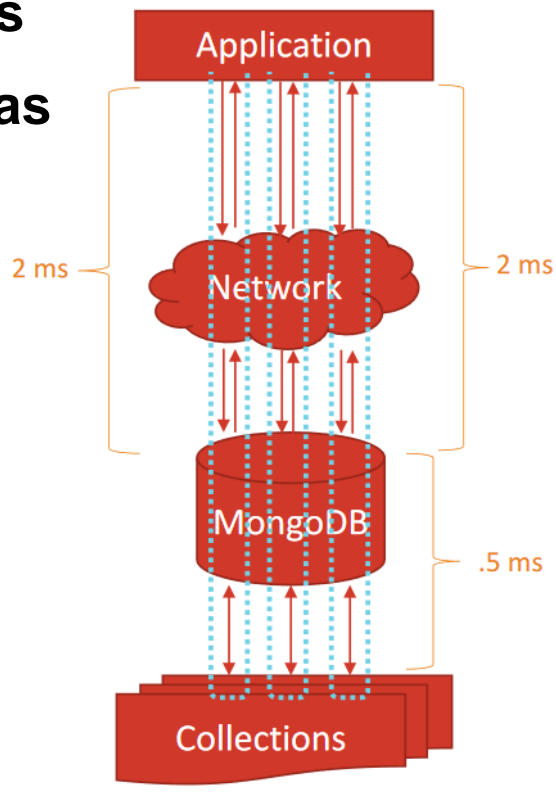
    # Important:: You must pass the session to the operations.
    collection_one.insert_one({'abc': 1}, session=session)
    collection_two.insert_one({'xyz': 999}, session=session)

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

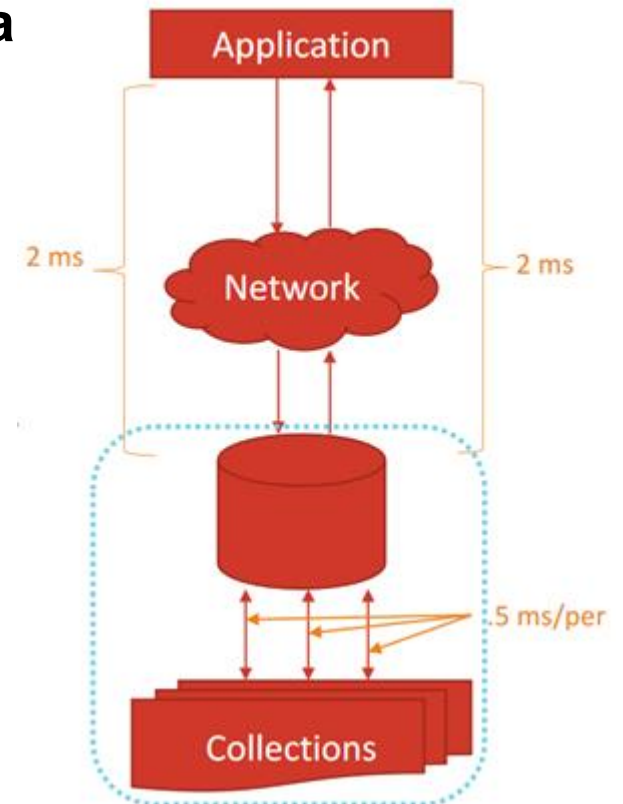
AGGREGATION-PIPELINE

PORQUE USAR PIPELINES

Múltiples Consultas



Consulta Pipeline



SQL TO AGGREGATION MAPPING CHART (1/2)

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	<u>\$match</u>
GROUP BY	<u>\$group</u>
HAVING	<u>\$match</u>
SELECT	<u>\$project</u>
ORDER BY	<u>\$sort</u>
LIMIT	<u>\$limit</u>
SUM()	<u>\$sum</u>
COUNT()	<u>\$sum</u> <u>\$sortByCount</u>
join	<u>\$lookup</u>
SELECT INTO NEW_TABLE	<u>\$out</u>
MERGE INTO TABLE	<u>\$merge</u> (Available starting in MongoDB 4.2)
UNION ALL	<u>\$unionWith</u> (Available starting in MongoDB 4.4)

SQL TO AGGREGATION MAPPING CHART (2/2)

<https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/>

```
{
  cust_id: "abc123",
  ord_date: ISODate("2012-11-02T17:04:11.102Z"),
  status: 'A',
  price: 50,
  items: [ { sku: "xxx", qty: 25, price: 1 },
           { sku: "yyy", qty: 25, price: 1 } ]
}
```

copy

SQL Example	MongoDB Example	Description
<pre>SELECT COUNT(*) AS count FROM orders</pre>	<pre>db.orders.aggregate([{ \$group: { _id: null, count: { \$sum: 1 } } }])</pre>	Count all records from orders

STAGE (1/2)

DESCRIPTION

\$group

Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.

\$lookup

Performs a left outer join to another collection in the *same* database to filter in documents from the “joined” collection for processing.

\$match

Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. \$match uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).

\$project

Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.

STAGE (2/2)

DESCRIPTION

\$sort

Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.

\$replaceWith

Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the `_id` field. Specify a document embedded in the input document to promote the embedded document to the top level.

\$unwind

Deconstructs an array field from the input documents to output a document for *each* element. Each output document replaces the array with an element value. For each input document, outputs *n* documents where *n* is the number of array elements and can be zero for an empty array.

CONSULTAR NÚMERO DE MATERIAS POR ESTUDIANTE

The screenshot shows the MongoDB Compass interface with the 'Aggregations' tab selected. The top bar includes 'Documents', 'Aggregations', 'Schema', and 'Explain P'. Below the bar, there are buttons for 'COLLATION', 'Untitled - Modified', 'SAVE', and a yellow button with a document icon. The main area is divided into two sections for aggregation stages.

Stage 1: \$group

Output after `$group`

```
1 /**
2  * Agrupar por el ID del Estudiante
3  * Sumar uno por cada registro
4  */
5 {
6   _id: "$student_id",
7   nro_materias: {
8     $sum: 1
9   }
10 }
```

The output preview shows: `_id: 10` and `nro_materias`.

Stage 2: \$sort

Output after `$sort`

```
1 /**
2  * Ordenar ascendentemente por el id
3  */
4 {
5   _id: 1
6 }
```

The output preview shows: `_id: 0` and `nro_materias`.

My Pipeline:

```
1 [{
2   $group: {
3     _id: "$student_id",
4     nro_materias: {
5       $sum: 1
6     }
7   }, {
8   $sort: {
9     _id: 1
10  }
11 }]
```

USO VALIDACIONES EN COLECCIONES

JSON SCHEMA

ejemplo_ciudades.personas

Documents Aggregations Schema Explain Plan Indexes Validation

Validation Action **ERROR** Validation Level **STRICT**

```
1 {
2   $jsonSchema: {
3     bsonType: 'object',
4     required: [
5       'nombre1',
6       'apellido1',
7       'ciudad_id_nace'
8     ],
9     properties: {
10      nombre1: {
11        bsonType: 'string',
12        pattern: '^[a-zA-Z]{3,}$',
13        description: 'El primer nombre de la requerido es requerido y es carcter de longitud minima 3'
14      },
15      nombre2: {
16        bsonType: 'string',
17        pattern: '^[a-zA-Z]{3,}$',
18        description: 'El segundo nombre es carcter de longitud minima 3'
19      },
20      ciudad_id_nace: {
21        bsonType: 'int',
22        description: 'ID ciudad id'
23      },
24      apellido1: {
25        bsonType: 'string',
26        pattern: '^[a-zA-Z]{3,}$',
27        description: 'El primer apellido de la requerido es requerido y es carcter de longitud minima 3'
28      },
29    }
30  }
31 }
```

Que hacer

Campos Mandatorios (obligatorios)

Especificación de un campo
Tipo de dato, Expresión regular y Documentación

INSERTAR DOCUMENTOS

Insert to Collection ejemplo_ciudades.personas

VIEW

```
1 {  
2   "nombre1": "pedro",  
3   "apellido1": "galindo"  
4 }
```

Document failed validation



```
required: [  
  'nombre1',  
  'apellido1',  
  'ciudad_id_nace'  
],  
properties: {  
  nombre1: {  
    bsonType: 'string',  
    pattern: '^[a-zA-Z]{3,}$',  
    description: 'El primer nom  
  },  
  nombre2: {  
    bsonType: 'string',  
    pattern: '^[a-zA-Z]{3,}$',  
    description: 'El segundo nom  
  },  
  ciudad_id_nace: {  
    bsonType: 'int',  
    description: 'ID ciudad id'  
  },  
}
```

Insert to Collection ejemplo_ciudades.personas

VIEW

```
1 {  
2   "nombre1": "pedro",  
3   "apellido1": "galindo",  
4   "ciudad_id_nace": 3,  
5   "telefono": 1234567  
6 }  
7
```



ejemplo_ciudades.personas

Documents

Aggregations

{"apellido1": "galindo"}



VIEW



```
_id: ObjectId("5f6a363e08f5fa1538a8edc5")  
nombre1: "pedro"  
apellido1: "galindo"  
ciudad_id_nace: 3  
telefono: 1234567
```

ANIDANDO VALIDACIONES

ejemplo_ciudades.personas

Documents	Aggregations
34	
35 ▾	ciudades_vive: {
36	bsonType: ['array'],
37 ▾	required: [
38	'ciudad_id',
39	'feinicio',
40	'fefin'
41],
42 ▾	properties: {
43 ▾	ciudad_id: {
44	bsonType: 'int',
45	description: 'ID ciudad'
46	},
47 ▾	feinicio: {
48	bsonType: 'date',
49	description: 'fecha inicio'
50	},
51 ▾	fefin: {
52	bsonType: 'date',
53	description: 'fecha fin'
54	}
55	}
56	}

A nivel global el “ARREGLO” ciudades_vive es opcional

Pero si se usa, se debe componen de forma obligatoria de 3 campos

Especificación de un campo
Tipo de dato y Documentación

OPTIMIZACIÓN

EXPLAIN PLAN (1/2)

ejemplo_ciudades.personas

The screenshot shows the MongoDB Explain Plan interface for the query `{ "_id": ObjectId("5f69fd92d30d110f940d3ba0") }`. The interface includes tabs for Documents, Aggregations, Schema, Explain Plan, and Indexes. The Explain Plan tab is active, showing a Query Performance Summary with the following metrics:

- Documents Returned: 1
- Index Keys Examined: 1
- Documents Examined: 1
- Actual Query Execution Time (ms): 0
- Sorted in Memory: no
- Query used the following index: `_id`

Below the summary, a section titled "IDHACK" shows the execution time as 0 ms and the number of documents returned as 1. A "DETAILS" button is visible at the bottom left.

Consulta

Uso Indice?

EXPLAIN PLAN (2/2)

ejemplo_ciudades.personas

Documents Aggregations Schema **Explain Plan** Indexes

FILTER `{ "apellido1": "perez", "nombre1": "pepe" }`

VIEW DETAILS AS **VISUAL TREE** RAW JSON

Query Performance Summary

Documents Returned: 1	Actual Query Execution Time (ms): 0
Index Keys Examined: 0	Sorted in Memory: no
Documents Examined: 3	⚠ No index available for this query.

COLLSCAN
nReturned: **1** Execution Time: **0** ms
Documents Examined: **3**
DETAILS

Consulta

Uso Indice?

ÍNDICES

ejemplo_ciudades.personas

DOCUMENTS 3 TOTAL SIZE 456B AVG. SIZE 152B INDEXES 1 TOTAL SIZE 36.0KB

Documents

Aggregations

Schema

Explain Plan

Indexes

Validation

CREATE INDEX

Name and Definition ^

Type

Properties

id

id ↑

REGULAR ⓘ

Create Index

Choose an index name

Configure the index definition

Select a field name

Select a t...



ADD ANOTHER FIELD

Options

- ☐ Build index in the background ⓘ
- ☐ Create unique index ⓘ
- ☐ Create TTL ⓘ
- ☐ Partial Filter Expression ⓘ
- ☐ Use Custom Collation ⓘ
- ☐ Wildcard Projection ⓘ

CANCEL

CREATE INDEX

ÍNDICE COMPUESTO

ejemplo_ciudades.personas

DocumentsAggregationsSchemaExplain Plan

FILTER { "apellido1": "perez", "nombre1": "pepe" }

VIEW DETAILS AS

VISUAL TREERAW JSON

Query Performance Summary

Documents Returned: 1

Index Keys Examined: 1

Documents Examined: 1

Actual Query Execution Time: 10ms

Sorted in Memory: no

Query used the following index:

apellido1 ↑ nombre1 ↑

Create Index

Choose an index name

personas_idx1

Configure the index definition

apellido11 (asc)

nombre11 (asc)

ADD ANOTHER FIELD

Options

CANCEL

CREATE INDEX

FILTER { "apellido1": "perez" }

SI

FILTER { "nombre1": "pepe" }

NO

34

VISTAS

SALVAR LA AGREGACIÓN COMO UNA VISTA

Clase20202.grades DOCUMENTS 280

Documents Aggregations Schema Explain Plan Indexes

COLLATION Untitled- Modified SAVE

Save pipeline as...
Create a view

\$unwind

```
1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for index.
4  * preserveNullAndEmptyArrays: Optional
5  *   toggle to unwind null and empty values.
6  */
7 {
8   path: "$scores",
9   preserveNullAndEmptyArrays: true
10 }
```

stage (Sample of 20 c

_id: ObjectId("50b59cd75bed76f46522")
student_id: 0
class_id: 2
scores: Object
 type: "exam"
 score: 57.92947112575566

\$group

```
1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4  */
5 {
6   _id: {"student_id": "$student_id",
7        "class_id": "$class_id"},
8 },
9 {
10  promedio_nota: {
11    $avg: "$scores.score"
12  }
13 }
```

Output after \$group stage (Sample of 20 dc

_id: Object
student_id: 29
class_id: 6
promedio_nota: 31.164434296429743

Create a View

Enter a View Name

grades_promedio_nota

CANCEL CREATE

Clase20202

- classes
- estudents
- grades
- grades_promedio_nota

EJEMPLO USO VISTA

Clase20202.grades_promedio_nota (view on: Clase20202.grades) [MODIFY SOURCE](#)

Documents Aggregations Schema Explain Plan Indexes Validation

COLLATION Untitled- Modified SAVE

```
1 /**
2  * from: The target collection.
3  * localField: The local join field.
4  * foreignField: The target join field.
5  * as: The name for the results.
6  * pipeline: The pipeline to run on the joined colle
7  * let: Optional variables to use in the pipeline fi
8  */
9 {
10   from: "estudents",
11   localField: "_id.student_id",
12   foreignField: "_id",
13   as: "estudiante"
14 }
```

Output after \$project stage (Sample of 20 documents)

```
1 /**
2  * specifications: The fields to
3  * include or exclude.
4  */
5 {
6   promedio_nota: 1,
7   "nombre": "$estudiante.name"
8 }
```

Output after \$sort stage (Sample of 20 documents)

```
1 /**
2  * Provide any number of field/order pairs.
3  */
4 {
5   "nombre": 1
6 }
```

Sample documents:

- `{ "_id": Object, "promedio_nota": 61.160541950542516, "estudiante": Array }`
- `{ "_id": Object, "promedio_nota": 66.95745545822932, "estudiante": Array }`
- `{ "_id": Object, "promedio_nota": 63.69108658226398, "nombre": Array, "nombre.0": "Rachell Aman" }`
- `{ "_id": Object, "promedio_nota": 55.98975485799118, "nombre": Array, "nombre.0": "Sanda Ryba" }`
- `{ "_id": Object, "promedio_nota": 36.571085790792736, "nombre": Array, "nombre.0": "Aleida Elsass" }`
- `{ "_id": Object, "promedio_nota": 32.12744601009303, "nombre": Array, "nombre.0": "Aleida Elsass" }`

FIN

sergalpe@gmail.com

