MongoDB

Cristian Buitrago

 @buty619

Acerca de mí

Cristian Buitrago

Software Engineer

- **Software Engineer** who loves working with Javascript and System Design .
- Anime Lover.
- **Education Enthusiast**.

Github @buty619

Código de conducta

Sé respetuoso, no hay malas preguntas o ideas.

Sé cordial y paciente.

Sé cuidadoso con tus palabras.

Puntos importantes

Identifícate en Zoom utilizando tu nombre y apellido.

Mantén tu micrófono apagado durante el transcurso de la sesión.

Utiliza el chat para hacer tus preguntas durante la sección de Q&A.

Procura enfocar tus preguntas al tema presentado.

Apaga tu cámara en caso de tener problemas con tu conexión.

Objetivos de la sesión

Al final de la sesión, serás capaz de:

- ➔ Entender los *aggregation* y sus tipos
- ➔ Conocer los tipo de relaciones en MongoDB
- ➔ Diseñar nuestro esquema
- ➔ Conocer los tipos de transacciones

Contenido del curso

1. **Aggregation**
2. **Data Models**
   Schema Validation, Design, Relationships
3. **Transaction & Atomicity**

Aggregation

La operación *aggregation* es una manera de procesar una **gran cantidad de documentos** dentro de una colección a través de diversas etapas conocidas como ***pipelines***.

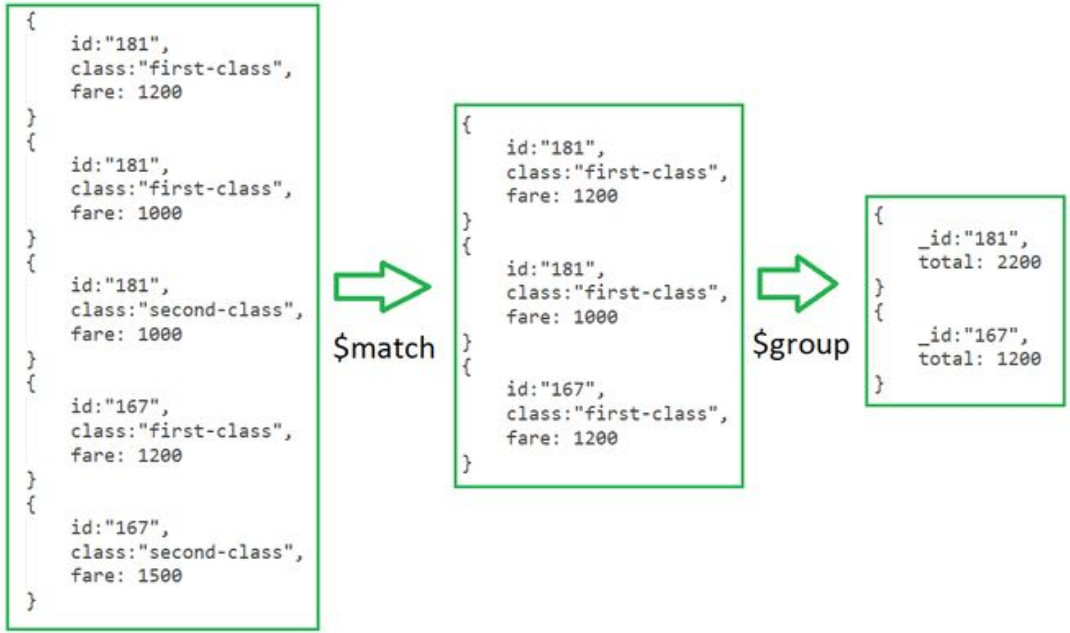
Una ***pipeline*** puede filtrar, ordenar, agrupar, remodelar y modificar los documentos que pasan dentro de la misma.

Ejemplo

Calcular los valores agregados para grupos de documentos.

Aggregation

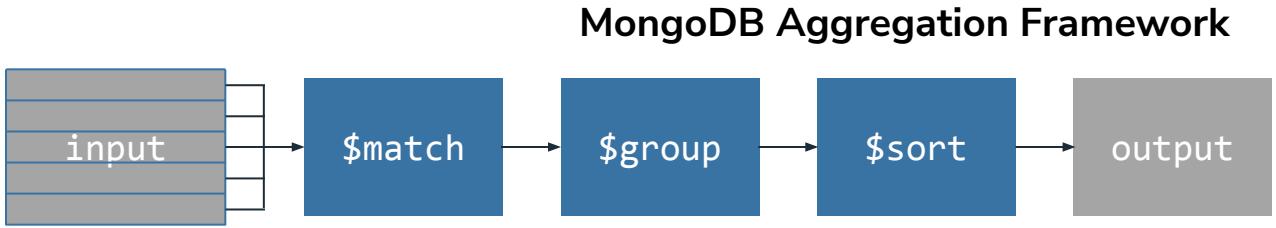
Procesamos múltiples documentos y regresamos resultados calculados.



Aggregation

Algunas acciones que se pueden realizar:

- Agrupar valores de multiples documentos.
- Realizar operaciones sobre los grupos de datos para regresar un solo resultado.
- Analizar los cambios de los datos con el paso del tiempo.



Aggregation

Para realizar operaciones de agregación, o *aggregation*, podemos usar:

Aggregation Pipelines

Aggregation Pipelines

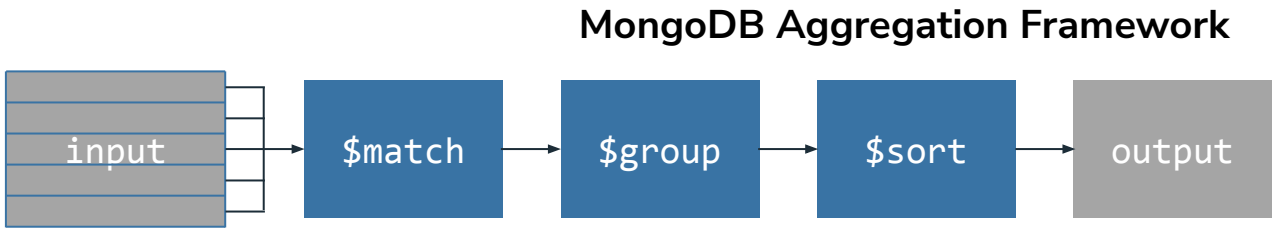
Single Purpose Aggregation Methods

Single Purpose Aggregation Methods

Aggregation Pipelines

Consiste en una o más etapas, o *stages*, para procesar los documentos. Tiene las siguientes características:

- Cada etapa realiza una operación con un *input* de documentos.
  Por ejemplo: Una etapa que puede filtrar, agrupar documentos y calcular valores.
- Los documentos con *output* (salidas) que vienen de una etapa, pasan a la siguiente etapa.
- Puede regresar resultados de grupos de documentos.
  Por ejemplo: Regresar el total, promedio, máximo y mínimo de valores.



```
db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by pizza size
   {
      $match: { size: "medium" }
   },

   // Stage 2: Group remaining documents by pizza name and calculate total quant
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   }

] )
```

Single Purpose Aggregation Methods

Agrega documentos de una sola colección.

Se usa cuando se necesita un simple acceso al documento.

- Por ejemplo, contar el número de documentos o encontrar todos los valores distintos en el documento.

Los métodos son simples pero **carecen de capacidades de agregación de una *pipeline*.**

```
db.collection.estimatedDocumentCount()
```

```
db.collection.count()
```

```
db.collection.distinct()
```

Contenido del curso

1. **Aggregation**
2. **Data Models**
   Schema Validation, Design, Relationships
3. **Transaction & Atomicity**

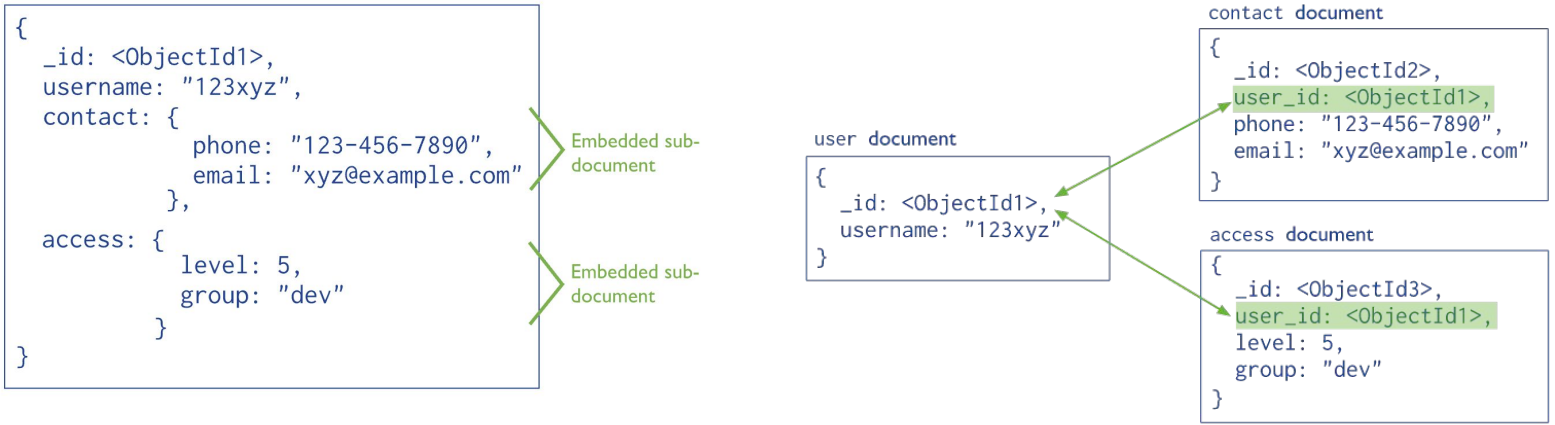
Data Models

Schema Validation, Design, Relationships

Data Models | Modelo de Datos

Recordemos:

- Embedded Data Model *(Modelo de Datos Embebido)*
- Normalized Data Model *(Modelo de Datos Normalizada)*



Relationships

Relationships

Una relación define cómo el sistema usa los datos. Existen tres tipos de relaciones definidas en un modelo de datos en MongoDB:

- *One-to-one (1 - 1)*
- *One-to-many (1 - n)*
- *Many-to-many (n - n)*

One to one

```
{
    "_id": "ObjectId('AAA')",
    "name": "Joe Karlsson",
    "company": "MongoDB",
    "twitter": "@JoeKarlsson1",
    "twitch": "joe_karlsson",
    "tiktok": "joekarlsson",
    "website": "joekarlsson.com"
}
```

One to many

```
{
    "_id": "ObjectId('AAA')",
    "name": "Joe Karlsson",
    "company": "MongoDB",
    "twitter": "@JoeKarlsson1",
    "twitch": "joe_karlsson",
    "tiktok": "joekarlsson",
    "website": "joekarlsson.com",
    "addresses": [
        { "street": "123 Sesame St", "city": "Anytown", "cc": "USA" },
        { "street": "123 Avenue Q",  "city": "New York", "cc": "USA" }
    ]
}
```

Many to many

Users

```
{
    "_id": ObjectID("AAF1"),
    "name": "Kate Monster",
    "tasks": [ObjectID("ADF9"), ObjectID("AE02"), ObjectID("AE73")]
}
```

Many to many

Tasks

```
{
    "_id": ObjectID("ADF9"),
    "description": "Write blog post about MongoDB schema design",
    "due_date": ISODate("2014-04-01"),
    "owners": [ObjectID("AAF1"), ObjectID("BB3G")]
}
```

Schema Validation

Schema Validation

Por default MongoDB, tiene un *flexible schema*.

En una colección, no necesariamente todos los documentos tienen los mismos campos o tipos de datos por default. Pero cuando estableces un esquema, puedes usar la herramienta **schema validation** que permite aplicar restricciones de estructura a los documentos.

Se hace sobre un **JSON Schema**

Schema Validation

```
db.createCollection("students", {
   validator: {$jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "gpa" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
            gender: {
               bsonType: "string",
               description: "must be a string and is not required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               exclusiveMaximum: false,
               description: "must be an integer in [ 2017, 2020 ] 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" ],
               minimum: 0,
               description: "must be a double and is required"
            }
         }

   }})
```

Schema Validation


```
db.students.insert({
   name: "James Karanja",
   year: NumberInt(2016),
   major: "History",
   gpa: NumberInt(3)
})
```

Schema Validation


```
WriteResult({
   "nInserted" : 0,
   "writeError" : {
      "code" : 121,
      "errmsg" : "Document failed validation"
   }
})
```

Schema Validation


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

Schema Design

Reglas generales para diseñar un esquema

- **Regla 1:** Dar preferencia a documentos embebidos a menos que haya una razón importante para *no* hacerlo.
- **Regla 2:** La necesidad de acceder a un objeto por sí solo es una razón importante para no ponerlo (*embed*).
- **Regla 3:** Evitar las uniones (*Joins*) y las búsquedas, pero no tengas miedo si puedes proporcionar un mejor diseño de esquema.
- **Regla 4:** Los arreglos (*arrays*) no deben crecer sin límite.
  - Si hay más de un par de cientos de documentos en el lado múltiple, no los incrustes (*embed*).
  - Si hay más de unos pocos miles de documentos en el lado de muchos, no uses una matriz de referencias de **ObjectID**. Las matrices de alta cardinalidad son una razón para no incrustar.
- **Regla 5:** Como siempre, la forma en que modelas los datos depende de los patrones de acceso a datos de la aplicación en particular. Estructura los datos para que coincidan con las formas en que la aplicación los consulta y actualiza.

https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/

Contenido del curso

1. **Aggregation**
2. **Data Models**
   Schema Validation, Design, Relationships
3. **Transaction & Atomicity**

Transaction & Atomicity

Atomicity

En **MongoDB**, una operación de escritura es **atómica** a nivel de un documento, incluso si la operación modifica varios documentos incrustados dentro de un solo documento.

¿Atomica?

Multi-Document Transactions

- Cuando haces una única operación de escritura, por ejemplo `db.collection.updateMany()`, modificas múltiples documentos. La modificación de cada documento es atómica, pero toda la operación no es atómica.
- Cuando realizas operaciones de escritura a múltiples documentos, ya sea a través de una sola operación de escritura o múltiples operaciones de escritura, otras operaciones pueden intercalarse.

Multi-Document Transactions

- En la **versión 4.0**, MongoDB admite transacciones de varios documentos en conjuntos de réplicas.
- En la **versión 4.2**, MongoDB presenta transacciones distribuidas. Estas agregan soporte para transacciones de múltiples documentos en clústeres fragmentados e incorpora el soporte existente para transacciones de múltiples documentos en conjuntos de réplicas.

Recap

Puntos a recordar:

- *Aggregation*
- *Data Models*
  - *Relationships*
  - *Flexible Schema*
  - *Schema Validation*
  - *Schema Design*
- *Atomicity*
- *Transactions*

Queremos conocer tu opinión

Gracias