Hibernate:

## Para que funcione:

El Beans, el PersistenceXML, el JpaUtil, la dependencia en el pom

## CRUD:

## Transaccionalidad:

Mongo:

## Para que funcione:

El MongoUtil, la dependencia en el pom

## CRUD:

Agregations:

Vamos a usar una database sol o porque son consultas get

tenemos varios tipos de busquedas

* van una tras la otra, son como una tuberia, puedes hacer una agregation tipo match y luego debajo una sort, se ordenarían los resultados del match, se llaman stages
* el match que es el común, para hacer un find normal
* los operadores empuiezan con un $ y es como un lenguaje:

$and [variable1:”hola”,variable2:”adios”,variable3: {$gt:10}, …]

* si quieres que te devuelva solo específicos valores, tendrías que hacer un agregation tipo porject, y el id se muestra por defecto:

{

 $match:[nombre:nacho]

 \_id:0, //no lo muestra

nombre:1 //si lo muestra, en este ejemplo acabaría mostrando solo el nombre.

}

* ha nombrado las agregaciones count y sort, simplemente indicas la variable que quieres contar o ordenar (en el caso de sort, variable y 1 si es ascendente, descendente ni idea)
* ha explicado el group, que ni idea, es como el groupBy pero nunca supe hacerlo
* hay que jugar con lo de hacer varios stages para llegar al resultado que desees, por ejemplo, primero las agrupo, luego las ordeno, luego saco la primera
* unwind, que abre el array y crea tantos documentos como elementos del array hayya esto te permite meterte dento del array para poder trabajar con objetos dentro de otros. pero no puedes hacer una stage de project para un objeto dentro de otro, por lo que ese valor lo tienes antes que sacar con otro tipo llamado addfields.
* queda el lookup que es para hacer join, te genera el objeto entero dentro del otro.
* en codigo, el export no puede hacerse porque cre un documento para cada cosa, hay que hacerlo como ella lo tiene el el github

Para los ejercicios.

Primero lo haces en mongo compass, y cuando sepas hacerlo, lo pasas a java pero si no puedes, y haces el export te vale la mitad de la nota.

tienes una clase para todas las agregations, llamada agregationrepository, teniendo un metodo para cada ejercicio, el nombre del metodo tiene que ser el enunciado, solo va a mirar el codigo no el compassttº

1. Aggregation Pipeline

Es una secuencia de etapas (stages) que procesan documentos. Cada etapa filtra, transforma o agrupa los datos, pasando el resultado a la siguiente etapa. Ejemplo de flujo:

$match → $sort → $project → $group

2. Etapas Comunes

a) $match

Filtra documentos, similar a un find.

Ejemplo: Buscar todas las ciudades en Massachusetts (MA) con población mayor a 15,000:

db.zipcodes.aggregate([

  {

    $match: {

      state: "MA",

      pop: { $gt: 15000 }

    }

  }

]);

b) $sort

Ordena los resultados.

* 1: Ascendente
* -1: Descendente

Ejemplo: Ordenar ciudades de MA por población (descendente):

db.zipcodes.aggregate([

  {

    $match: { state: "MA" }

  },

  {

    $sort: { pop: -1 }

  }

]);

c) $project

Selecciona campos específicos.

* 1: Incluir campo
* 0: Excluir campo

Ejemplo: Mostrar solo city y pop, excluyendo \_id:

db.zipcodes.aggregate([

  {

    $project: {

      \_id: 0,

      city: 1,

      pop: 1

    }

  }

]);

d) $count

Cuenta documentos.

Ejemplo: Contar ciudades en MA:

db.zipcodes.aggregate([

  {

    $match: { state: "MA" }

  },

  {

    $count: "total\_ciudades\_MA"

  }

]);

e) $unwind

Descompone un campo que es un array, creando un documento por cada elemento del array. Esto permite trabajar con objetos dentro de arrays.

Ejemplo: Si tienes un documento como este:

{

  "\_id": "01001",

  "city": "AGAWAM",

  "loc": [-72.622739, 42.070206],

  "pop": 15338,

  "state": "MA"

}

Y aplicas $unwind en loc, se generan dos documentos:

{ "\_id": "01001", "city": "AGAWAM", "loc": -72.622739, "pop": 15338, "state": "MA" }

{ "\_id": "01001", "city": "AGAWAM", "loc": 42.070206, "pop": 15338, "state": "MA" }

f) $addFields

Añade nuevos campos o modifica existentes. Es útil para trabajar con objetos anidados, ya que no puedes acceder directamente a ellos en un $project.

Ejemplo: Si tienes un array de objetos:

{

  "\_id": "01001",

  "city": "AGAWAM",

  "details": [

    { "type": "A", "value": 10 },

    { "type": "B", "value": 20 }

  ]

}

Y quieres añadir un campo que sea el doble de value:

db.collection.aggregate([

  {

    $unwind: "$details"

  },

  {

    $addFields: {

      "details.doubleValue": { $multiply: ["$details.value", 2] }

    }

  }

]);

Resultado:

{

  "\_id": "01001",

  "city": "AGAWAM",

  "details": { "type": "A", "value": 10, "doubleValue": 20 }

}

{

  "\_id": "01001",

  "city": "AGAWAM",

  "details": { "type": "B", "value": 20, "doubleValue": 40 }

}

Nota

$unwind es necesario para trabajar con arrays antes de usar $addFields o $project en objetos dentro de ellos.

$project no puede acceder directamente a objetos dentro de arrays sin antes usar $unwind.

3. $group (Agrupación)

Agrupa documentos por un campo y aplica operaciones (suma, promedio, etc.).

* \_id: Campo por el que se agrupa.
* Usa operadores como $sum, $avg, $max.

Ejemplo: Población total por estado:

db.zipcodes.aggregate([

  {

    $group: {

      \_id: "$state",

      total\_poblacion: { $sum: "$pop" }

    }

  }

]);

Resultado:

{ "\_id": "MA", "total\_poblacion": 522901 }

{ "\_id": "NY", "total\_poblacion": 873492 }

4. Combinando Etapas

Ejemplo:  
"Encontrar el estado con mayor población total".

db.zipcodes.aggregate([

  {

    $group: {

      \_id: "$state",

      total\_poblacion: { $sum: "$pop" }

    }

  },

  {

    $sort: { total\_poblacion: -1 }

  },

  {

    $limit: 1

  }

]);

Explicación:

1. Agrupa por estado y suma poblaciones.
2. Ordena de mayor a menor.
3. Limita al primer resultado (el más poblado).

Errores Comunes

* Sintaxis de operadores: Siempre usan $ (ej: $gt, $sum).
* $match incorrecto:  
  ❌ $match: [ { state: "MA" } ] → Correcto: $match: { state: "MA" }.

**EJERCICIOS PARA LA PRÁCTICA**:

**a. Get the oldest patient**

[

  {

    $sort:

      {

        birthDate: 1

      }

  },

  {

    $limit:  1

  }

]

**b. Get the name of the patient who has paid the most**

[

  {

    $unwind:

      {

        path: "$payments"

      }

  },

  {

    $addFields:

      {

        payment: "$payments.amount"

      }

  },

  {

    $group:

      {

        \_id: "$\_id",

        name: {

          $first: "$name"

        },

        totalPayments: {

          $sum: "$payments.amount"

        }

      }

  },

  {

    $sort:

      {

        totalPayments: -1

      }

  },

  {

    $project:

      {

        name: 1,

        totalPayments: 1

      }

  },

  {

    $limit: 1

  }

]