Aggregation Pipelines

Ejemplo básico

Crearemos una colección de muestra llamada estudiantes e insertamos algunos documentos con los siguientes campos:

```
{
  "name": "John Doe",
  "age": 25,
  "grade": 85,
  "subjects": ["Math", "English", "Science"]
}
```

Etapa $match

Utiliza $match para filtrar los estudiantes que tienen más de 20 años.

```
db.students.aggregate([
  { $match: { edad: { $gt: 20 } } }
])
```

Compruebe el resultado y asegúrese de que sólo se devuelven los estudiantes mayores de 20 años.

Etapa $project

Utilice $project para mostrar sólo los nombres y las calificaciones de los estudiantes.

```
db.students.aggregate([
  { $project: { nombre: 1, nota: 1 } }
])
```

Comprueba que sólo se muestran los nombres y las calificaciones.

Etapa $group

Utiliza $group para encontrar la nota media de todos los alumnos.

```
db.students.aggregate([
  { $group: { _id: null, averageGrade: { $avg: "$grado" } } }
])
```

Asegúrese de que la nota media se calcula correctamente.

Etapa $group con $sum

Utilizar $group y $sum para encontrar el número total de estudiantes de cada asignatura.

```
db.students.aggregate([
  { $suma "$asignaturas" },
  { $grupo: { _id: "$asignaturas", totalEstudiantes: { $suma: 1 } } }
])
```

Compruebe que el número total de alumnos de cada asignatura es correcto.

Etapa $unwind

Usa $unwind para obtener un documento separado para cada asignatura.

```
db.students.aggregate([
  { $unwind: "$asignaturas" }
])
```

Comprueba que cada asignatura es ahora un documento independiente.

Relaciones incrustadas

Modificamos la colección de estudiantes incrustando información sobre los cursos directamente en cada documento de estudiante.

```
{
  "name": "Jane Doe",
  "age": 24,
  "grade": 90,
  "courses": [
    { "subject": "Math", "instructor": "Professor Smith" },
    { "subject": "English", "instructor": "Professor Johnson" }
  ]
}
```

Ejemplo de consulta que encuentre todos los alumnos que están cursando la asignatura "Matemáticas".

```
db.students.find({ "courses.subject": "Math" })
```

Relaciones con referencias: Etapa $lookup

Crear una colección llamada cursos e insertar documentos

```
{
  "_id": ObjectId("5f91eb9ad7f3dca8caefe456"),
  "name": "Matemáticas",
```

```
  "instructor": "Profesor Smith"
}
```

Cree una colección llamada estudiantes e inserte algunos documentos:

```
{
  "_id": ObjectId("5f91eb9ad7f3dca8caefe123"),
  "name": "John Doe",
  "age": 20,
  "courses": [ObjectId("5f91eb9ad7f3dca8caefe456"),
ObjectId("5f91eb9ad7f3dca8caefe789")]
}
```

Utiliza la etapa $lookup para la búsqueda de coincidencias:

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "courses",
      foreignField: "_id",
      as: "courseInfo"
    }
  }
])
```

Añadir una etapa $project para mostrar sólo el nombre del estudiante y el instructor de cada curso.

```
db.students.aggregate([
  {
    $lookup: {
      from: "courses",
      localField: "courses",
      foreignField: "_id",
      as: "courseInfo"
    }
  },
  {
    $project: {
      name: 1,
      courseInstructors: "$courseInfo.instructor"
    }
  }
])
```

Ejercicio más complejo

Consideremos un escenario hipotético en el que se dispone de una colección de documentos que representan pedidos en un sistema de comercio electrónico. Cada documento de pedido puede contener información sobre los artículos comprados, el cliente y la fecha del pedido.

Para este ejercicio, vamos a suponer un esquema simplificado para un documento de pedido:

```
{
  "_id": ObjectId("..."),
  "order_date": ISODate("..."),
  "customer": {
    "name": "John Doe",
    "email": "john.doe@example.com",
    "address": "123 Main St, Cityville"
  },
  "items": [
    {"product": "ProductA", "quantity": 2, "price": 10},
    {"product": "ProductB", "quantity": 1, "price": 20},
  ]
}
```

Ahora, vamos a crear una Aggregation Pipeline compleja que realice las siguientes tareas:

- **unwind**: Cada documento de pedido tiene una array de items. Queremos desenrollar esta matriz para que cada item se convierta en un documento independiente.
- **Calcular el precio total de cada artículo:** Multiplicar la cantidad por el precio de cada artículo.
- **Agrupar por producto y calcular la cantidad total vendida y los ingresos:** Suma las cantidades y los precios totales de cada producto.
- **Ordenar** el resultado por ingresos totales en orden descendente.

```
db.orders.aggregate([
  {
    $unwind: "$items"
  },
  {
    $project: {
      "product": "$items.product",
      "quantity": "$items.quantity",
      "revenue": { $multiply: ["$items.quantity", "$items.price"] }
    }
```

```
  },
  {
    $group: {
      "_id": "$product",
      "total_quantity_sold": { $sum: "$quantity" },
      "total_revenue": { $sum: "$revenue" }
    }
  },
  {
    $sort: {
      "total_revenue": -1
    }
  }
])
```

Vamos a desglosar cada etapa:

- $**unwind**: Esta etapa deconstruye la matriz de elementos, creando un nuevo documento para cada elemento de la matriz.
- $**project**: Esta etapa remodela los documentos, extrayendo la información relevante sobre el producto, la cantidad y los ingresos (cantidad * precio).
- $**group**: Esta etapa agrupa los documentos por el producto, calculando la cantidad total vendida y los ingresos totales para cada producto.
- $**sort**: Finalmente, esta etapa ordena el resultado por ingresos totales en orden descendente

Suponed ahora que los productos están en otra collection

```
{
  "_id": ObjectId("..."),
  "name": "ProductA",
  "category": "Electronics"
}
```

Ahora, tenéis que realizar una Aggregation Pipeline que combine la información de pedidos y productos utilizando la etapa $lookup.

- Unwind el array items en la colección orders.
- Buscar la información del producto basándose en el nombre del producto en los artículos.
- Calcular el precio total de cada artículo.
- Agrupar por producto y calcular la cantidad total vendida, los ingresos e incluir los detalles del producto.
- Ordenar el resultado por ingresos totales en orden descendente.