Gestión/Almacenamiento De Información No Estructurada: PEC 2 Bases de datos documentales (MongoDB)

Sergio Soler Rocha
Universidad Nacional de Educación a Distancia

1. Ejercicio 1: Desplegar una infraestructura de MongoDB en la nube empleando MongoDB Atlas.

Para la realización del ejercicio hemos seguido los pasos descritos en el vídeo https://www.youtube.com/watch?v=97FfXEy1zas. Hemos creado la siguiente estructura en MongoDB Atlas:

- **Versión de MongoDB:** La instancia de MongoDB se ejecuta en la versión 7.0.7, lo que garantiza que se utilicen las últimas características y mejoras de rendimiento disponibles en MongoDB.
- **Región de Implementación:** La infraestructura de base de datos MongoDB está alojada en la región de AWS Paris (eu-west-3), lo que proporciona una baja latencia y un rápido acceso a los datos para los usuarios ubicados en esa región geográfica.
- **Tipo de Clúster:** El clúster de base de datos MongoDB se ha configurado como un conjunto de réplicas con 3 nodos, lo que garantiza la redundancia y la alta disponibilidad de los datos. Este tipo de configuración permite la tolerancia a fallos y la recuperación automática en caso de que un nodo falle.
- **Tier del Clúster:** La configuración del clúster se ha seleccionado como M0 Sandbox, lo que indica que se está utilizando una instancia de menor capacidad y recursos. Esto puede ser adecuado para aplicaciones de desarrollo y pruebas de bajo tráfico o pequeñas aplicaciones de producción.
- **Tamaño de los Datos:** El tamaño máximo de los datos almacenados en la base de datos MongoDB es de 512 MB, lo que indica que se está utilizando una cantidad moderada de almacenamiento.

El usuario creado para que el equipo docente se puede ver en la siguiente tabla:

Cuadro 1: Credenciales del usuario de lectura

Usuario	EquipoDocente
Contraseña	Gl0tk4.L1n34r
Acceso	`mongodb+srv://EquipoDocente:Gl0tk4.L1n34r@cluster0.gpalq2h.mongodb.net/`

Haciendo uso de MongoDB Compass confirmamos que se puede acceder al cluster tanto desde el usuario administrador tanto desde el de lectura.

2. Ejercicio 2: Diseñar un blog de noticias donde los usuarios registrados puedan publicar sus comentarios.

2.1. Diseño de la BD Documental

En la base de datos con nombre *Blog_de_noticias* las colecciones a crear son las siguientes:

a) **autores**: Guarda la información de los autores de las noticias.

- nombre (string)
- nombreDeUsuario (string, único)
- cuentaTwitter (string, único)
- descripción (string)
- direcciónPostal (opcional, objeto que contiene calle, número, puerta, CP, ciudad)
- teléfonos (array de strings)

b) **noticias**: Guarda los detalles de las noticias.

- título (string)
- cuerpo (string)
- fechaPublicación (fecha)
- autor (ObjectId, referencia a autores)
- etiquetas (array de strings, opcional)

c) **comentarios**: Guarda los comentarios hechos a las noticias.

- texto (string)
- usuarioAutor (ObjectId, referencia a autores)
- fechaCreación (fecha)
- noticia (ObjectId, referencia a noticias)

2.2. Creación de los índices

Para mejorar el rendimiento de las consultas más comunes en nuestras colecciones, se definen los siguientes índices:

- **autores**:

```
db.autores.createIndex({ nombreDeUsuario: 1 })
db.autores.createIndex({ cuentaTwitter: 1 })
db.autores.createIndex({ "direcciónPostal.CP": 1 })

```

- **noticias**:

```
db.noticias.createIndex({ autor: 1, fechaPublicación: -1 })

```

- **comentarios**:

```
db.comentarios.createIndex({ noticia: 1, fechaCreación: 1 })
db.comentarios.createIndex({ usuarioAutor: 1, fechaCreación: 1 })

```

2.3. Consultas más frecuentes

- **Consultas por nombre de usuario**
  Para encontrar un autor por su nombre de usuario, utilizamos el siguiente comando:

```
    db.autores.find({ "nombreDeUsuario": "nombre_usuario_deseado" });

```

  Reemplazamos `"nombre_usuario_deseado"` con el nombre de usuario específico que deseamos buscar.

- **Consultas por cuenta de Twitter**
  Para buscar un autor por su cuenta de Twitter, usamos el siguiente comando:

```
db.autores.find({ "cuentaTwitter": "@cuentaTwitter" });
```

Reemplazamos "@cuentaTwitter" con la cuenta de Twitter específica que queremos encontrar.

- **Agregaciones por código postal**
  Para contar el número de usuarios que tienen el mismo código postal, utilizamos el siguiente comando de agregación:

```
db.autores.aggregate([
{ $group: { _id: "$direcciónPostal.CP", numeroUsuarios: { $sum: 1 } } }
]);
```

Esto nos dará el número de usuarios por código postal.

- **Consultas de noticias de un usuario, ordenadas por fecha**
  Primero, necesitamos el ObjectId del usuario. Una vez que lo tengamos, buscamos todas las noticias publicadas por ese usuario, ordenadas de la más reciente a la más antigua:

```
db.noticias.find({ "autor": ObjectId("id_del_autor") }).sort({ "fechaPublicación": -1 });
```

Reemplazamos ïd_del_autor" con el ObjectId real del autor.

- **Número de comentarios por noticia**:

```
db.comentarios.aggregate([
{ $group: { _id: "$noticia", numeroComentarios: { $sum: 1 } } }
]);
```

- **Número de comentarios por día**:

```
db.comentarios.aggregate([
{ $group: { _id: { $dateToString: { format: "%Y-%m-%d", date: "$fechaCreación" } }, numeroComentarios: { $sum: 1 } } }
]);
```

- **Número de comentarios por usuario**:

```
db.comentarios.aggregate([
{ $group: { _id: "$usuarioAutor", numeroComentarios: { $sum: 1 } } }
]);
```

2.4. Inserción de Datos

- **autores**:

```
db.autores.insertMany([
{
  "nombre": "Juan Pérez",
  "nombreDeUsuario": "juanp",
  "cuentaTwitter": "@juanp",
  "descripción": "Periodista especializado en tecnología.",
  "direcciónPostal": {
    "calle": "Av. Libertador",
    "número": "1234",
    "puerta": "3A",
    "CP": "1010",
    "ciudad": "Buenos Aires"
  },
  "teléfonos": ["+541112345678", "+541198765432"]
},
{
  "nombre": "Ana Gómez",
  "nombreDeUsuario": "anag",
```

```
    "cuentaTwitter": "@anag",
    "descripción": "Periodista deportiva."
  }
  ]);

```

- **noticias**:

```
  db.noticias.insertMany([
  {
    "título": "Nueva tecnología en IA",
    "cuerpo": "Texto largo del cuerpo...",
    "fechaPublicación": new Date("2024-03-27T10:00:00Z"),
    "autor": ObjectId("autor_id_1"),
    "etiquetas": ["tecnología", "IA"]
  },
  {
    "título": "Avances en la lucha contra el cambio climático",
    "cuerpo": "Texto largo del cuerpo...",
    "fechaPublicación": new Date("2024-03-29T09:00:00Z"),
    "autor": ObjectId("autor_id_2"),
    "etiquetas": ["medio ambiente", "sostenibilidad"]
  }
  ]);

```

- **comentarios**:

```
  db.comentarios.insertMany([
  {
    "texto": "Muy interesante el artículo sobre IA.",
    "usuarioAutor": ObjectId("autor_id_1"),
    "fechaCreación": new Date("2024-03-27T12:00:00Z"),
    "noticia": ObjectId("noticia_id_1")
  },
  {
    "texto": "Excelente información sobre el cambio climático.",
      "usuarioAutor": ObjectId("autor_id_2"),
  "fechaCreación": new Date("2024-03-29T11:30:00Z"),
  "noticia": ObjectId("noticia_id_2")
  }
  ]);

```

3. Ejercicio 3: Sobre la colección webs se pide ejecutar las siguientes consulta y agregaciones.

- **Consulta 1**: Número de sesiones (documentos) en los que se ha usado el navegador Explorer.

  Para esta consulta, utilizamos `count` o `countDocuments` para contar el número de documentos que coinciden con el criterio de que el campo navegador sea 'Explorer'.

```
  db.webs.countDocuments({ "navegador": "Explorer" });

```

  output:

```
  20

```

- **Consulta 2**: Mostrar las URLs de las páginas webs que se han consultado.

  En esta consulta, queremos proyectar únicamente las URLs de las páginas visitadas. Utilizamos el operador $project con un pipeline de agregación para transformar los documentos y mostrar solo las URLs. Este comando desagrega el array 'pags' para luego proyectar solo las URLs. Se utiliza $unwind para separar cada elemento del array 'pags' en un documento individual, permitiendo así proyectar las URLs directamente.

```
db.webs.aggregate([
{ $unwind: "$pags" },
{ $project: { "url": "$pags.url", "_id": 0 } }
]);

```

output:

```
{
  url: 'http://www.nocheviejuna.es'
}

{
  url: 'http://www.uvasexpress.es'
}

...

```

- **Consulta 3**: Mostrar únicamente el campo navegador empleado por Herminia, cuyas en las sesiones que tengan visitas a páginas webs que hayan durado entre 300 segundos y 1000 segundos.

  Para esta consulta, primero filtramos los documentos por el nombre "Herminiaz luego filtramos las páginas visitadas por la duración especificada. Finalmente, proyectamos solo el campo navegador.

```
db.webs.aggregate([
{ $match: { "nombre": "Herminia" } },
{ $unwind: "$pags" },
{ $match: { "pags.segs": { $gte: 300, $lte: 1000 } } },
{ $project: { "navegador": 1, "_id": 0 } },
{ $group: { "_id": "$navegador" } }
]);

```

output:

```
{
  _id: 'Explorer'
}

```

- **Agregación 1**: Obtener el número de sesiones por cada navegador, ordenadas en orden ascendente. La salida mostrará, para cada navegador, un '_id' que contiene el nombre del navegador, y un campo "totalçon el número de sesiones de dicho navegador.

  Agrupamos los documentos de la colección por el campo navegador con $group. Para cada grupo, calculamos el número total de sesiones usando $sum: 1. El resultado es un conjunto de documentos donde cada documento representa un navegador, identificado por '_id', y el campo 'total' contiene el recuento de sesiones para ese navegador. Con $sort ordenamos los documentos resultantes del paso de agrupamiento en orden ascendente ('1') según el campo 'total'.

```
db.webs.aggregate([
{
  $group: {
    _id: "$navegador",
    total: { $sum: 1 }
  }
},
{
  $sort: { total: 1 }
}
]);

```

output:

```
    {
      _id: 'Chrome',
      total: 14
    }

    {
      _id: 'firefox',
      total: 17
    }

    {
      _id: 'Explorer',
      total: 20
    }

```

- **Agregación 2**: Obtener la persona con mayor número de sesiones totales en la colección. La salida mostrará para cada persona (sólo 1) un '_id' conteniendo el nombre de la persona y un campo "total" con el número de sesiones totales.

  Primero, agrupamos los documentos por el nombre de la persona para contar el número total de sesiones por persona. Luego, ordenamos los resultados en orden descendente por el número de sesiones y limitamos los resultados al documento superior.

```
    db.webs.aggregate([
    {
      $group: {
        _id: "$nombre",
        total: { $sum: 1 }
      }
    },
    {
      $sort: { total: -1 }
    },
    {
      $limit: 1
    }
    ]);

```

  output:

```
    {
      _id: 'Herminia',
      total: 20
    }

```

- **Agregación 3**: Para cada persona indicar el número de URLs visitadas durante más de 100 segundos, contando las repeticiones. La salida mostrará para cada persona un '_id' conteniendo el nombre de la persona y un campo 'num' con el número de URLs. Pista: utilizar la opción $unwind para devolver la lista de pags como documentos independientes.

  Utilizaremos el operador $unwind para "desenrollar.[el] array de páginas visitadas ('pags'). Después filtramos las visitas que duraron más de 100 segundos y contamos el número de estas por persona.

```
    db.webs.aggregate([
    {
      $unwind: "$pags"
    },
    {
      $match: {
        "pags.segs": { $gt: 100 }
      }
    },
    {
      $group: {
        _id: "$nombre",
        num: { $sum: 1 }
      }
    }
```

```
]);
```

output:

```
{
  _id: 'Herminia',
  num: 40
}

{
  _id: 'Bertoldo',
  num: 51
}

{
  _id: 'Aniceto',
  num: 35
}
```

- **Agregación 4**: Mostrar por cada URL, el tiempo de visita total, ordenando los registros de mayor a menor. Pista: utilizar la opción $unwind de aggregate.

  Con un enfoque similar al anterior, utilizamos $unwind para trabajar con cada página visitada de manera independiente. Después, agrupamos los resultados por URL y sumamos el tiempo de visita. Finalmente, ordenamos los resultados según el tiempo total de visita en orden descendente.

```
db.webs.aggregate([
{
  $unwind: "$pags"
},
{
  $group: {
    _id: "$pags.url",
    tiempoTotalVisita: { $sum: "$pags.segs"   }
  }
},
{
  $sort: { tiempoTotalVisita: -1 }
}
]);
```

output:

```
{
  _id: 'http://www.nocheviejuna.es',
  tiempoTotalVisita: 12000
}

{
  _id: 'http://www.nomiresatras.es',
  tiempoTotalVisita: 12000
}

{
  _id: 'http://www.quizaseahoy.es',
  tiempoTotalVisita: 8400
}

...
```

- **Agregación 5**: Calcular el total de sesiones con el navegador 'Explorer' en 2016. La salida mostrará el '_id' conteniendo el año (2016) y un campo 'total' con el número total de sesiones. Pista: utilizar la opción $match. Para obtener el año de una fecha se puede utilizar $year:'$fecha'.

  Utilizamos $match para filtrar por navegador y año, luego usamos $group para contar el número de sesiones.

```
db.webs.aggregate([
{
  $match: {
    "navegador": "Explorer",
    "fecha": {
      $gte: new Date("2016-01-01T00:00:00.000Z"),
      $lte: new Date("2016-12-31T23:59:59.999Z")
    }
  }
},
{
  $group: {
    _id: { $year: "$fecha" },
    total: { $sum: 1 }
  }
}
]);

```

output:

```
{
  _id: 2016,
  total: 10
}

```

- **Agregación 6**: Para cada persona calcular el tiempo acumulado de todas las URLs visitadas, en minutos. La salida mostrará un campo 'nombre' con el nombre de la persona, y un campo 'tiempomin' con el tiempo acumulado. La salida estará ordenada alfabéticamente por los nombres de los usuarios. Pista: utilizar la opción $unwind y $project para cambiar de segundos a minutos.

  Utilizaremos $unwind para descomponer el array de páginas visitadas, $group para agrupar por nombre y sumar los tiempos, y $project para convertir los segundos a minutos.

```
db.webs.aggregate([
{
  $unwind: "$pags"
},
{
  $group: {
    _id: "$nombre",
    tiempoTotalSegs: { $sum: "$pags.segs" }
  }
},
{
  $project: {
    _id: 0,
    nombre: "$_id",
    tiempomin: { $divide: ["$tiempoTotalSegs", 60] }
  }
},
{
  $sort: { nombre: 1 }
}
]);

```

output:

```
{
  nombre: 'Aniceto',
  tiempomin: 329.35
}

{
  nombre: 'Bertoldo',
  tiempomin: 263.3333333333333
}

{
  nombre: 'Herminia',
```

```
      tiempomin: 551.6666666666666
    }

```

4. Ejercicio 4: Proponer una base de datos documental de libre elección por parte del alumno.

Para este ejercicio, proponemos diseñar una base de datos documental para una aplicación de gestión de eventos. La aplicación permitirá a los usuarios crear eventos, registrarse en eventos existentes, y comentar sobre ellos. Este diseño abordará las necesidades de los usuarios de encontrar eventos de su interés, inscribirse en ellos, y participar en discusiones relacionadas.

4.1. Diseño de la BD Documental

a) **usuarios**: Almacena información sobre los usuarios registrados en la aplicación.

- nombre (string)
- email (string, único)
- contraseña (string)
- fechaRegistro (fecha)

b) **eventos**: Contiene detalles de los eventos creados por los usuarios.

- titulo (string)
- descripcion (string)
- fechaEvento (fecha)
- ubicacion (string)
- organizador (ObjectId, referencia a usuarios)
- asistentes (array de ObjectId, referencias a usuarios)

c) **comentarios**: Almacena comentarios realizados por los usuarios sobre los eventos.

- texto (string)
- autor (ObjectId, referencia a usuarios)
- evento (ObjectId, referencia a eventos)
- fechaCreacion (fecha)

4.2. Creación de los índices

Para optimizar las consultas más comunes:

- **usuarios**:

```
      db.usuarios.createIndex({ "email": 1 }, { unique: true });

```

- **eventos**:

```
      db.eventos.createIndex({ "fechaEvento": 1 });
      db.eventos.createIndex({ "organizador": 1 });

```

- **comentarios**:

```
      db.comentarios.createIndex({ "evento": 1, "fechaCreacion": -1 });

```

4.3. Inserción de Datos

- **usuarios**:

```
db.usuarios.insertMany([
  { "nombre": "Alice", "email": "alice@example.com", "contraseña": "password1", "fechaRegistro": ISODate("2023-03-01T10:00:00Z") },
  { "nombre": "Bob", "email": "bob@example.com", "contraseña": "password2", "fechaRegistro": ISODate("2023-03-02T11:00:00Z") }
  ]);

```

- **eventos**:

```
db.eventos.insertMany([
  { "titulo": "Concierto de Rock", "descripcion": "Evento de música rock.", "fechaEvento": ISODate("2023-05-15T20:00:00Z"), "ubicacion": "Ciudad Rock", "organizador": ObjectId("organizador_id_1"), "asistentes": [ObjectId("asistente_id_1")] },
  { "titulo": "Conferencia de Tecnología", "descripcion": "Discusión sobre las últimas tendencias en tecnología.", "fechaEvento": ISODate("2023-06-20T09:00:00Z"), "ubicacion": "Centro de Convenciones Tech", "organizador": ObjectId("organizador_id_2"), "asistentes": [] }
  ]);

```

- **comentarios**:

```
db.comentarios.insertMany([
  { "texto": "No puedo esperar por este evento!", "autor": ObjectId("autor_id_1"), "evento": ObjectId("evento_id_1"), "fechaCreacion": ISODate("2023-03-10T12:00:00Z") },
  { "texto": "Será increíble reunirnos.", "autor": ObjectId("autor_id_2"), "evento": ObjectId("evento_id_2"), "fechaCreacion": ISODate("2023-03-11T15:30:00Z") }
  ]);

```

4.4. Ejemplos de Consultas

- **Buscar eventos por fecha**
  Para buscar eventos en un rango de fechas específico, utilizamos el siguiente comando:

```
db.eventos.find({ "fechaEvento": { $gte: ISODate("2023-05-01T00:00:00Z"), $lte: ISODate("2023-06-01T00:00:00Z") } }).sort({ "fechaEvento": 1 });

```

  Esto buscará eventos que ocurran entre el 1 de mayo de 2023 y el 1 de junio de 2023, ordenándolos en orden ascendente por la fecha del evento.

- **Consultar los eventos organizados por un usuario específico**
  Para encontrar eventos organizados por un usuario específico, usamos el siguiente comando:

```
db.eventos.find({ "organizador": ObjectId("id_del_organizador") });

```

  Reemplazamos `id_del_organizador"` con el ObjectId del usuario organizador específico cuyos eventos queremos buscar.