IES Virgen del Carmen

JUNTA DE ANDALUCIA
CONSEJERÍA DE EDUCACIÓN

Departamento de Informática

Programación de bases de datos documentales

Juan Gualberto

Enero 2024

UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

INFORMÁTICA - IES Virgen del Carmen - JAÉN

Índice

1 Bases de datos documentales

Las bases de datos documentales son un tipo de sistema de gestión de bases de datos (SGBD) que se centra en el almacenamiento y recuperación de datos en formato de documentos. En lugar de organizar la información en tablas como en las bases de datos relacionales, las bases de datos documentales utilizan documentos, que pueden ser en formatos como JSON (JavaScript Object Notation) o BSON (Binary JSON). Cada documento puede contener datos estructurados y no estructurados, lo que proporciona flexibilidad en la representación de la información.

Mientras que las bases de datos relacionales están más recomendadas a casos en los que hay mucha rotación de datos, las documentales se recomiendan para los casos donde hay pocas o nulas actualizaciones.

Características comunes de las bases de datos documentales:

1. **Documentos:** La unidad básica de almacenamiento es el documento, que puede contener datos en formato clave-valor, matrices, objetos anidados, etc.
2. **Esquema dinámico:** A diferencia de las bases de datos relacionales, las bases de datos documentales permiten esquemas dinámicos, lo que significa que cada documento en la colección puede tener diferentes campos.
3. **Flexibilidad:** Son adecuadas para datos semiestructurados y no estructurados, lo que facilita el almacenamiento de información variada y cambiante.
4. **Escalabilidad horizontal:** Muchas bases de datos documentales están diseñadas para escalar horizontalmente, distribuyendo la carga de trabajo en varios servidores para manejar grandes volúmenes de datos y tráfico.
5. **Consultas eficientes:** Permiten realizar consultas eficientes utilizando índices en los campos clave.

Ejemplos de bases de datos documentales:

1. **MongoDB:** Es una de las bases de datos documentales más populares y ampliamente utilizadas. Almacena datos en formato BSON (una representación binaria de JSON) y permite esquemas flexibles.
2. **Firebase Firestore:** Es una base de datos documental en la nube proporcionada por Firebase, que es parte de Google Cloud Platform. Almacena datos en formato JSON y es especialmente popular para aplicaciones web y móviles.

2 Planteamiento del proyecto

El presente proyecto forma parte de una de las tareas de la asignatura de Acceso a Datos del Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma. Con este proyecto aprenderemos los contenidos del tema, en nuestro caso desarrollar una aplicación con NodeJS, Express, PUG y MongoDB.

2.1 Introducción a la tarea

Tenemos que implementar un sencillo sistema de gestión para un instituto operado por el personal de administración y servicios del mismo. Básicamente se trata de crear un CRUD para alumnos, otro CRUD para profesores, otro CRUD para asignaturas, un maestro-detalle para ver asignaciones de profesores a asignaturas y otro para matriculación de alumnos en asignaturas. Todo protegido con un login para que los datos no puedan ser abusados.

2.2 Casos de uso

Un caso de uso no es más que plasmar ejemplos de cómo los actores (usuarios de nuestro sistema) interactúan con nuestra aplicación. Esto nos ayudará a dividir el problema en cada una de sus partes. Un ejemplo del problema que nos atañe es:

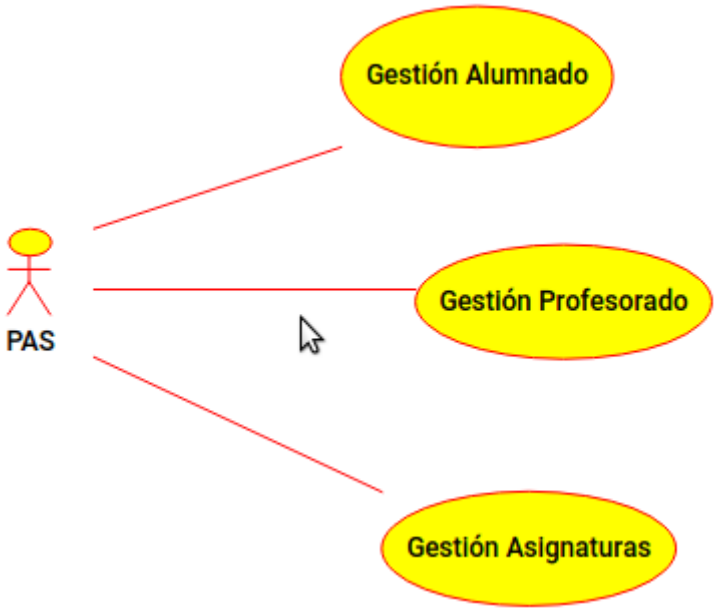

Figura 1: Diagrama de caso de uso de primer nivel

2.3 Diagrama entidad/relación

El diagrama entidad/relación es el paso previo al diseño de tablas de la base de datos y nunca debe faltar en la fase de diseño. No obstante, en la actualidad cada vez cogen más fuerza las bases de datos documentales que dan un giro de tuerca al modelo relacional tradicional. Si bien en un modelo relacional tradicional tendríamos el siguiente diagrama ER:

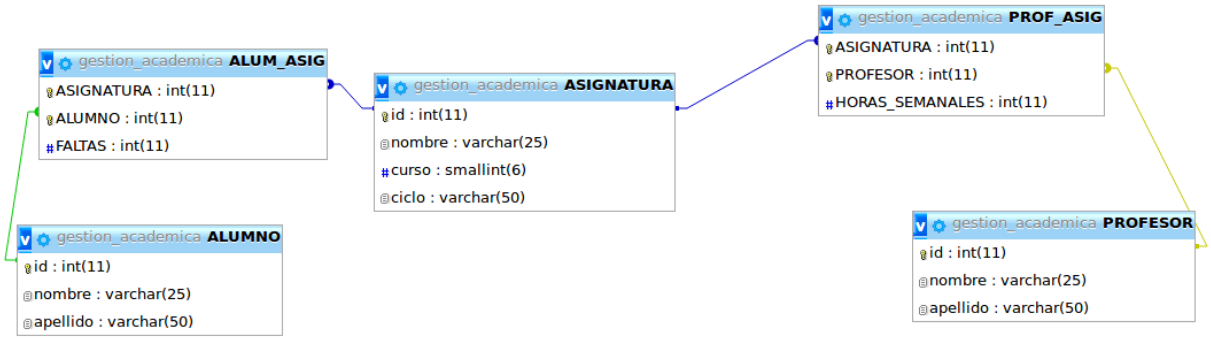

Figura 2: Diagrama entidad-relación

Con un enfoque documental podemos tener información duplicada. Siempre ponemos el ejemplo de la famosa plataforma de ventas por internet de la que no diremos el nombre. Cuando tú compras una serie de productos, se genera una factura o recibo de compra que puedes consultar en cualquier momento. Esa factura es una fotografía del precio de los productos en ese momento. Cuando miramos el precio de los mismos productos unos días después, probablemente habrán fluctuado. La información de los precios es redundante para todos los que compramos esos productos en ese momento, pero es más rápido para consultas tenerlo como un documento que tenerlo como una tabla.

En un modelo documental tendremos colecciones de documentos, como por ejemplo:

`alumno`:

```
{
  nombre: "Juan",
  apellido: "Sin miedo",
  email: "juan@sincorreo.com",
  telefono: 123456789
}
```

`profesor`:

```
{
  nombre: "Juan",
  apellido: "Sin miedo",
  email: "juan@sincorreo.com",
```

```
  departamento: "Informática y Comunicaciones"
}
```

asignatura:

```
{
  codigo: 486,
  nombre: "Acceso a Datos",
  curso: 2,
  ciclo: {
    grado: "Superior",
    denominacion: "Desarrollo de Aplicaciones Multiplataforma",
    normativa: "Real Decreto 405/2023"
  }
}
```

matrícula:

```
{
    asignatura: {
        codigo: 486,
        nombre: "Acceso a Datos",
        curso: 2,
        ciclo: {
            grado: "Superior",
            denominacion: "Desarrollo de Aplicaciones Multiplataforma",
            normativa: "Real Decreto 405/2023"
        }
    },
    alumnos: [
        {
            nombre: "Juan",
            apellido: "Sin miedo",
            email: "juan@sincorreo.com",
            telefono: 123456789
        },
        {
            nombre: "Pedro",
            apellido: "El Cruel",
            email: "pedro@sincorreo.com",
            telefono: 987654321
        }
    ]
}
```

2.4 Diagrama de clases

Otro diagrama fundamental en UML es el diagrama de clases. En el vemos los objetos que habrá en nuestra aplicación y cómo interactuarán entre ellos.

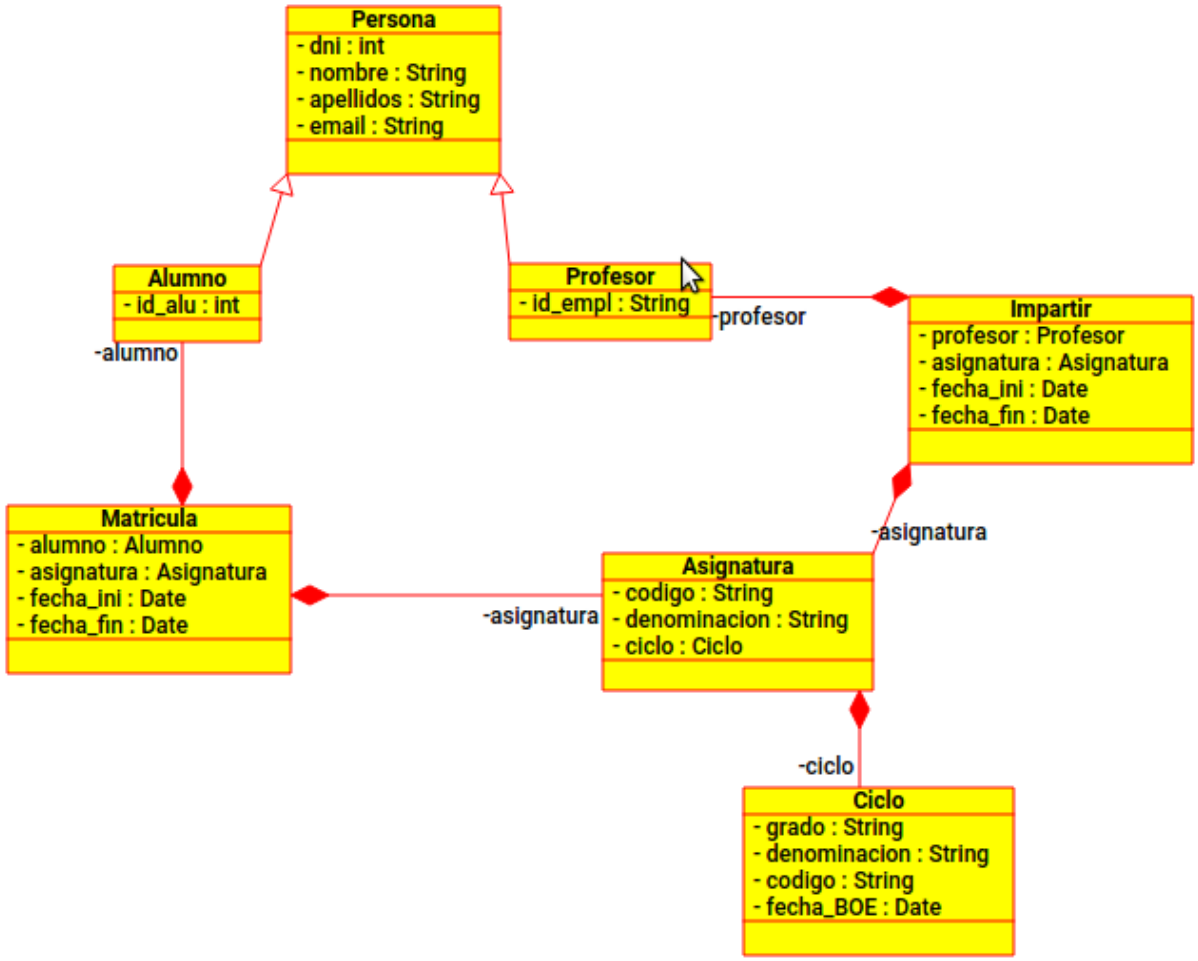

Figura 3: Diagrama UML de caso de uso donde se ve la relación entre todas las clases.

3 Instalación y configuración de MongoDB

Para crear la infraestructura de servicios necesaria, en vez de instalar MongoDB, y como siempre hacemos, vamos a usar un contenedor para el servidor MongoDB y otro contenedor para Mongo-Express (una interfaz para interactuar con la base de datos).

Creamos la carpeta `stack` y dentro de ésta el archivo `docker-compose.yml` con el siguiente contenido:

```yaml
# Use root/example as user/password credentials
version: '3.1'

services:

  mongo:
    image: mongo
    restart: 'no'
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 83uddjfp0cmMD
    ports:
      - 27017:27017

  mongo-express:
    image: mongo-express
    restart: 'no'
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_BASICAUTH_USERNAME: mongo
      ME_CONFIG_BASICAUTH_PASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_URL: mongodb://root:83uddjfp0cmMD@mongo:27017/
```

Aunque ya lo hemos visto anteriormente, este archivo `docker-compose.yml` lo utilizamos para definir y ejecutar servicios necesarios. Vamos a analizar las secciones y configuraciones de este archivo:

```yaml
version: '3.1'
```

Esta línea especifica la versión de la sintaxis de `docker-compose` que se está utilizando. En este caso, es la versión 3.1.

```yaml
services:
  mongo:
    image: mongo
    restart: 'no'
```

```
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 83uddjfp0cmMD
    ports:
      - 27017:27017
```

Aquí se define un servicio llamado `mongo`. Este servicio utiliza la imagen oficial de MongoDB (`mongo`). Algunas configuraciones clave incluyen:

- `restart: 'no'`: Esto indica que el contenedor no se reiniciará automáticamente a menos que se haga manualmente.
- `environment`: Establece variables de entorno necesarias para configurar la base de datos MongoDB. En este caso, se especifica el nombre de usuario (`root`) y la contraseña (`83uddjfp0cmMD`) para el usuario root de MongoDB.
- `ports`: Mapea el puerto 27017 del host al puerto 27017 del contenedor, lo que permite la comunicación con la instancia de MongoDB.

```
  mongo-express:
    image: mongo-express
    restart: 'no'
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_BASICAUTH_USERNAME: mongo
      ME_CONFIG_BASICAUTH_PASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_URL: mongodb://root:83uddjfp0cmMD@mongo:27017/
```

Aquí se define un segundo servicio llamado `mongo-express`. Este servicio utiliza la imagen de Mongo Express (`mongo-express`). Algunas configuraciones clave son similares al servicio `mongo`, pero específicas de Mongo Express:

- `restart: 'no'`: Al igual que en el servicio `mongo`, indica que el contenedor no se reiniciará automáticamente.
- `ports`: Mapea el puerto 8081 del host al puerto 8081 del contenedor, permitiendo acceder a la interfaz web de Mongo Express.
- `environment`: Configura las variables de entorno necesarias para la autenticación en MongoDB.
- `ME_CONFIG_MONGODB_URL`: Especifica la URL de conexión a la base de datos MongoDB. En este caso, utiliza el usuario root y la contraseña proporcionados en las variables de entorno anteriores.
- `ME_CONFIG_BASICAUTH_USERNAME` y `ME_CONFIG_BASICAUTH_PASSWORD` definen el usuario y contraseña para la interfaz Web.

Recuerda que una cosa es Express (interfaz Web) y otra el servicio Mongo (base de datos documental). **SIEMPRE** debemos de usar password fuertes incluso en desarrollo, por si en algún momento, por accidente, nuestros servicios estuviesen expuestos en Internet o a toda la Intranet de la empresa, ningún dato sensible sería accesible.

Para poner en marcha estos servicios, debes tener Docker y Docker Compose instalados. Luego, ejecuta el siguiente comando en el directorio donde se encuentra el archivo `docker-compose.yml`:

```
1 docker-compose up -d
```

Esto descargará las imágenes necesarias, creará y ejecutará los contenedores según la configuración proporcionada. Después de que los contenedores estén en funcionamiento, podrás acceder a MongoDB a través del puerto 27017 y a Mongo Express a través del puerto 8081 en tu máquina local.

3.1 Conexión interactiva a MongoDB

Aunque por lo general no usaremos MongoDB en modo interactivo, vamos a ver algunos ejemplos de cómo interactuar con la shell de mongo (mongosh):

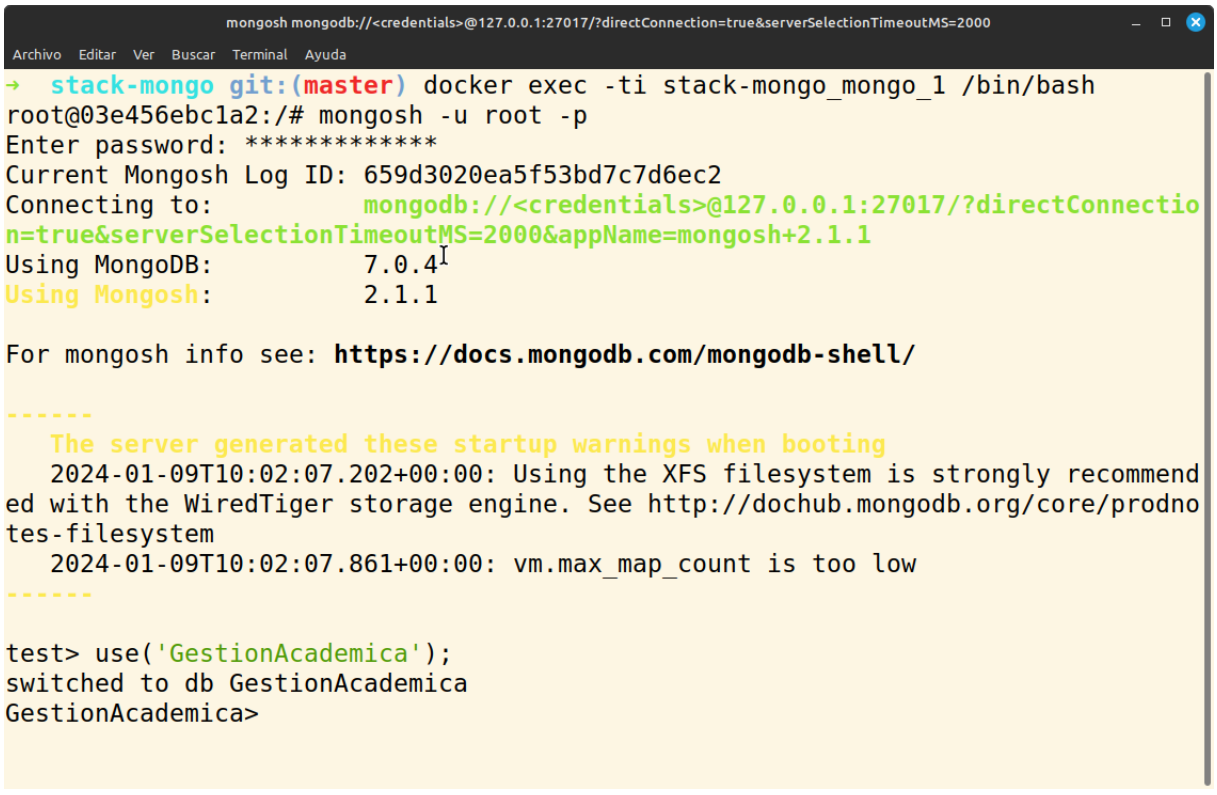

Figura 4: Ejemplo de conexión en modo interactivo.

1. **Iniciar el shell interactivo:** Abre tu terminal y ejecuta los siguientes comandos:
    1. `docker exec -ti stack-mongo_mongo_1 /bin/bash`: para abrir una terminal interactiva en el contenedor de nuestro servicio **mongo**.
    2. `mongosh -u root -p`: para ingresar al shell interactivo de MongoDB, la contraseña es *83uddjfp0cmMD*, como fijamos en el docker-compose.
    3. `use('GestionAcademica')`: desde la shell de mongo, indicamos qué base de datos queremos usar de esta manera.

2. **Crear un documento (Create):** Para insertar un nuevo documento en una colección llamada `alumnos`, puedes utilizar el siguiente comando:

```
db.alumnos.insertOne({ nombre: "Juan Perez", edad: 20, carrera: "
    Ingeniería Informática" })
```

3. **Leer documentos (Read):** Para recuperar todos los documentos de la colección `alumnos`, puedes usar el comando `find()`:

```
db.alumnos.find()
```

    Esto mostrará todos los documentos que representan a los alumnos.

4. **Actualizar un documento (Update):** Para actualizar un documento, puedes utilizar el comando `updateOne()`. Supongamos que Juan Pérez cambió su carrera a “Ingeniería Eléctrica”:

```
db.alumnos.updateOne({ nombre: "Juan Perez" }, { $set: { carrera:
    "Ingeniería Eléctrica" } })
```

    Esto actualiza el documento de Juan Pérez con la nueva información sobre su carrera.

5. **Eliminar un documento (Delete):** Para eliminar un documento, puedes utilizar el comando `deleteOne()`. Supongamos que Juan Pérez ya no es alumno:

```
db.alumnos.deleteOne({ nombre: "Juan Perez" })
```

    Esto eliminará el documento que representa a Juan Pérez de la colección.

Recuerda, para buscar un objeto en una colección usamos el método find. Así el formato para consultar la colección `profesor` sería:

```
db.profesor.find( { "nombre":"Juan"}).pretty();
```

Pero ¿y si quiero buscar profesore con asignaturas con más de 5 horas (inclusive)?

```
db.profesor.find({
    asignaturas:
        {
```

```
            $elemMatch: {
                    horas: { $gt: 4}
            }
        }
});
```

Tienes más ejemplos sobre la sintaxis de consulta en la Web de MongoDB: https://www.mongodb.com/docs/manual/tutorial/query-documents/.

3.2 Colección de ejemplo

Vamos a utilizar una colección ficticia llamada `estudiantes` que contiene información sobre estudiantes en una universidad. Aquí tienes algunos documentos de ejemplo en esta colección:

```
db.estudiantes.insertMany([
  { nombre: "Ana García", edad: 22, carrera: "Biología", semestre: 5, promedio: 8.5 },
  { nombre: "Carlos López", edad: 21, carrera: "Ingeniería Civil", semestre: 4, promedio: 7.2 },
  { nombre: "María Torres", edad: 20, carrera: "Psicología", semestre: 3, promedio: 9.0 },
  { nombre: "Juan Rodríguez", edad: 23, carrera: "Historia", semestre: 6, promedio: 7.8 },
  { nombre: "Elena Pérez", edad: 19, carrera: "Matemáticas", semestre: 2, promedio: 9.5 }
])
```

Ahora, puedes realizar diversas consultas utilizando el método `find()` de MongoDB. Aquí tienes algunos ejemplos:

1. **Recuperar todos los estudiantes:**

   ```
   db.estudiantes.find({})
   ```

2. **Filtrar estudiantes por carrera:**

   ```
   db.estudiantes.find({ carrera: "Biología" })
   ```

3. **Estudiantes mayores de 21 años:**

   ```
   db.estudiantes.find({ edad: { $gt: 21 } })
   ```

4. **Estudiantes con promedio mayor o igual a 8.0:**

   ```
   db.estudiantes.find({ promedio: { $gte: 8.0 } })
   ```

5. **Estudiantes de Psicología en el tercer semestre:**

```
db.estudiantes.find({ carrera: "Psicología", semestre: 3 })
```

6. **Ordenar estudiantes por promedio en orden descendente:**

```
db.estudiantes.find().sort({ promedio: -1 })
```

7. **Limitar la cantidad de resultados a 3:**

```
db.estudiantes.find().limit(3)
```

Estos son solo ejemplos básicos de consultas utilizando el método `find()` en MongoDB. La sintaxis puede variar según las necesidades específicas de tu aplicación, y MongoDB ofrece una amplia variedad de operadores y opciones para realizar consultas más avanzadas. Puedes consultar la documentación oficial de MongoDB para obtener más detalles sobre la sintaxis y los operadores de consulta: MongoDB Query Documents.

4 Gestión de sesiones (login)

Vamos a comenzar nuestro proyecto con el sistema de Login. Creamos una carpeta `backend` y ejecutamos estos comandos:

1. **Inicializar el Proyecto:** Crea un nuevo directorio y navega a él en la terminal. Luego, ejecuta el siguiente comando para inicializar un proyecto de Node.js:

```
npm init -y
```

   Esto creará un archivo **`package.json`** con la configuración predeterminada.

2. **Instalar Dependencias:** Instala las dependencias necesarias: Express, Mongoose para interactuar con MongoDB, Bcrypt para el cifrado de contraseñas y Pug para las plantillas.

```
npm install express express-session mongoose bcrypt pug
```

3. **Configurar la Aplicación y el middleware para la gestión de sesiones:** Crea un archivo `app.js` (o cualquier nombre que prefieras) y configura tu aplicación Express y Express-Session:

```
    const express = require('express');
    const mongoose = require('mongoose');
    const bcrypt = require('bcrypt');
    const app = express();
    const session = require('express-session');
    // para cargar configuración de la APP desde .env
    const dotenv = require('dotenv');
    // sistema de login y registro
    const authRoutes = require('./routes/auth');

    app.set('view engine', 'pug');
    app.use(express.urlencoded({ extended: true }));
    app.use(express.static('public'));

    // Configuración middleware express-session
    app.use(session({
        secret: 'unsupersecretoinconfesable',
        resave: true,
        saveUninitialized: false
      }));

    // Middleware para pasar información de sesión a las vistas
    app.use((req, res, next) => {
        res.locals.currentUser = req.session.user;
        next();
      });

    // cargamos configuración desde .env
    dotenv.config();
```

```
30
31      mongoose.connect(process.env.MONGO_URI);
```

Crea un archivo `.env` con este contenido:

```
1  MONGO_URI=mongodb://root:83uddjfp0cmMD@localhost:27017/
       GestionAcademica?authSource=admin
2  BACKEND_PORT=8000
```

Asegúrate de cambiar `'tuBaseDeDatos'` por el nombre de tu base de datos MongoDB.

4. **Definir el Modelo de Usuario:** Crea un archivo `models/User.js` para definir el modelo de usuario:

```
1  const mongoose = require('mongoose');
2
3  const userSchema = new mongoose.Schema({
4    username: { type: String, unique: true, required: true },
5    password: { type: String, required: true }
6  });
7
8  const User = mongoose.model('User', userSchema);
9
10 module.exports = User;
```

5. **Rutas y Controladores:** Crea un archivo `routes/auth.js` para manejar las rutas relacionadas con la autenticación:

```
1     const express = require('express');
2     const router = express.Router();
3     const bcrypt = require('bcrypt');
4     const User = require('../models/User');
5
6     router.get("/", (req, res) =>{
7         res.render('index');
8     });
9
10    router.get('/login', (req, res) => {
11        res.render('login');
12    });
13
14    router.post('/login', async (req, res) => {
15        const { username, password } = req.body;
16
17        const user = await User.findOne({ username });
18
19        if (user && bcrypt.compareSync(password, user.password)) {
20            req.session.user = user; // Almacenamos el usuario en
                  la sesión
21            res.send('Inicio de sesión exitoso');
```

```
        } else {
            res.send('Credenciales incorrectas');
        }
    });

    router.get('/register', (req, res) => {
        res.render('register');
    });

    router.post('/register', async (req, res) => {
        const { username, password } = req.body;

        const hashedPassword = bcrypt.hashSync(password, 10);

        const newUser = new User({ username, password:
            hashedPassword });

        try {
            await newUser.save();
            res.send('Usuario registrado exitosamente');
        } catch (error) {
            res.send('Error al registrar el usuario');
        }
    });

    router.get('/logout', (req, res) => {
        req.session.destroy();
        res.redirect('/auth');
      });

    module.exports = router;
```

6. **Vistas Pug:** Crea las vistas Pug en la carpeta `views`. Puedes tener archivos como `login.pug` y `register.pug`. Aquí tienes un ejemplo simple de `index.pug`:

```
    html
      head
        title Login y Registro
      body
        h1 Bienvenido, #{currentUser ? currentUser.username : '
            Invitado'}

        if !currentUser
          h2 Iniciar Sesión
          form(action='/auth/login', method='POST')
            label(for='username') Usuario:
            input(type='text', name='username', required)
            br
            label(for='password') Contraseña:
            input(type='password', name='password', required)
            br
```

```
            input(type='submit', value='Iniciar Sesión')

          h2 Registrarse
          form(action='/auth/register', method='POST')
            label(for='username') Usuario:
            input(type='text', name='username', required)
            br
            label(for='password') Contraseña:
            input(type='password', name='password', required)
            br
            input(type='submit', value='Registrarse')
        else
          a(href='/auth/logout') Cerrar Sesión
```

Puedes usar código de este `index` para la vista `login.pug` y `register.pug`.

7. **Configurar Rutas en la Aplicación Principal:** Modifica `app.js` para utilizar las rutas que hemos definido:

```
    // configuramos rutas

    app.use('/auth', authRoutes);

    app.use('/', (req, res) => {
        res.redirect('/auth');
    });

    app.listen(process.env.BACKEND_PORT, () => {
        console.log(`Servidor en funcionamiento en el puerto ${
            process.env.BACKEND_PORT}`);
    });
```

Aquí, las rutas de autenticación estarán bajo la ruta `/auth`.

8. **Ejecutar la Aplicación:** Ejecuta tu aplicación con:

```
node app.js
```

Visita `http://localhost:8000/auth/login` para ver la página de inicio de sesión.

5 CRUD Alumno

Un CRUD (acrónimo de Crear, Leer, Actualizar, Borrar) es un conjunto de operaciones básicas que podremos hacer contra un sistema de información.

A continuación vamos a hacer un ejemplo para un modelo de alumno con los datos: nombre, apellidos y email.

1. **Modificar el modelo `models/Alumno.js`:** Agrega un nuevo modelo para representar a los alumnos. Crea un archivo llamado `Alumno.js` dentro de la carpeta `models`.

```
const mongoose = require('mongoose');

const alumnoSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  apellidos: { type: String, required: true },
  email: { type: String, required: true, unique: true }
});

const Alumno = mongoose.model('Alumno', alumnoSchema);

module.exports = Alumno;
```

2. **Modificar el archivo `routes/alumnos.js`:** Crea un nuevo archivo llamado `alumnos.js` dentro de la carpeta `routes`. Este archivo manejará las rutas relacionadas con los alumnos.

```
const express = require('express');
const router = express.Router();
const Alumno = require('../models/Alumno');

// Obtener todos los alumnos
router.get('/alumnos', async (req, res) => {
  try {
    const alumnos = await Alumno.find();
    res.render('alumnos/index', { alumnos });
  } catch (error) {
    res.send('Error al obtener los alumnos');
  }
});

// Formulario para agregar un nuevo alumno
router.get('/alumnos/new', (req, res) => {
  res.render('alumnos/new');
});

// Agregar un nuevo alumno
router.post('/alumnos', async (req, res) => {
  const { nombre, apellidos, email } = req.body;

  const nuevoAlumno = new Alumno({ nombre, apellidos, email });

  try {
    await nuevoAlumno.save();
    res.redirect('/alumnos');
  } catch (error) {
    res.send('Error al agregar el alumno');
  }
});
```

```
33
34 // Mostrar detalles de un alumno
35 router.get('/alumnos/:id', async (req, res) => {
36   try {
37     const alumno = await Alumno.findById(req.params.id);
38     res.render('alumnos/show', { alumno });
39   } catch (error) {
40     res.send('Error al obtener los detalles del alumno');
41   }
42 });
43
44 // Formulario para editar un alumno
45 router.get('/alumnos/:id/edit', async (req, res) => {
46   try {
47     const alumno = await Alumno.findById(req.params.id);
48     res.render('alumnos/edit', { alumno });
49   } catch (error) {
50     res.send('Error al obtener los detalles del alumno para editar
         ');
51   }
52 });
53
54 // Actualizar un alumno
55 router.put('/alumnos/:id', async (req, res) => {
56   const { nombre, apellidos, email } = req.body;
57
58   try {
59     await Alumno.findByIdAndUpdate(req.params.id, { nombre,
         apellidos, email });
60     res.redirect('/alumnos');
61   } catch (error) {
62     res.send('Error al actualizar el alumno');
63   }
64 });
65
66 // Eliminar un alumno
67 router.delete('/alumnos/:id', async (req, res) => {
68   try {
69     await Alumno.findByIdAndRemove(req.params.id);
70     res.redirect('/alumnos');
71   } catch (error) {
72     res.send('Error al eliminar el alumno');
73   }
74 });
75
76 module.exports = router;
```

3. **Crear las vistas correspondientes:** Crea las carpetas y archivos de vistas necesarios. Aquí tienes los archivos necesarios:

   - `views/alumnos/index.pug`: Lista de todos los alumnos.

- views/alumnos/**new**.pug: Formulario para agregar un nuevo alumno.
- views/alumnos/show.pug: Detalles de un alumno específico.
- views/alumnos/edit.pug: Formulario para editar un alumno.

A continuación detallamos el contenido de la carpeta views/alumnos:

1. **index.pug:** Lista de todos los alumnos.

```
html
head
  title Lista de Alumnos
body
  h1 Lista de Alumnos
  ul
    each alumno in alumnos
      li
        a(href=`/alumnos/${alumno._id}`) #{alumno.nombre} #{
           alumno.apellidos} (#{alumno.email})
  br
  a(href='/alumnos/new') Agregar Nuevo Alumno
```

2. **new.pug:** Formulario para agregar un nuevo alumno.

```
html
head
  title Agregar Nuevo Alumno
body
  h1 Agregar Nuevo Alumno
  form(action='/alumnos', method='POST')
    label(for='nombre') Nombre:
    input(type='text', name='nombre', required)
    br
    label(for='apellidos') Apellidos:
    input(type='text', name='apellidos', required)
    br
    label(for='email') Email:
    input(type='email', name='email', required)
    br
    input(type='submit', value='Agregar Alumno')
  br
  a(href='/alumnos') Volver a la Lista de Alumnos
```

3. **show.pug:** Detalles de un alumno específico.

```
html
head
  title Detalles del Alumno
body
  h1 Detalles del Alumno
  p
```

```
    strong Nombre:
    span #{alumno.nombre}
  p
    strong Apellidos:
    span #{alumno.apellidos}
  p
    strong Email:
    span #{alumno.email}
  br
  a(href=`/alumnos/${alumno._id}/edit`) Editar Alumno
  form(action=`/alumnos/${alumno._id}?_method=DELETE`, method='POST')
    input(type='submit', value='Eliminar Alumno')
  br
  a(href='/alumnos') Volver a la Lista de Alumnos
```

4. **`edit.pug`:** Formulario para editar un alumno.

```
html
head
  title Editar Alumno
body
  h1 Editar Alumno
  form(action=`/alumnos/${alumno._id}?_method=PUT`, method='POST')
    label(for='nombre') Nombre:
    input(type='text', name='nombre', value=alumno.nombre, required)
    br
    label(for='apellidos') Apellidos:
    input(type='text', name='apellidos', value=alumno.apellidos, required)
    br
    label(for='email') Email:
    input(type='email', name='email', value=alumno.email, required)
    br
    input(type='submit', value='Guardar Cambios')
  br
  a(href=`/alumnos/${alumno._id}`) Cancelar
  br
  a(href='/alumnos') Volver a la Lista de Alumnos
```

6 Bibliografía

Bibliografía complementaria:

- Documnetación oficial de Mongo: https://docs.mongo.org.