Bases de datos II

Sistema Integral de Gestión de Hospitales

Aguero Julian - Galvan Adriel - Molins Cristian - Montemarini Gianfranco - Parlatore Matias - Sahonero Didier

Docente: Salas, Joaquin
Auxiliar: Mosquera, Diego

Universidad Argentina de la Empresa
Junio 2024

Índice

1. Introducción

Expondremos mediante el siguiente trabajo practico la justificación de la elección de nuestro modelo de Bases de datos SQL y noSQL que cumplirán dentro de un ámbito real, un producto final.

Se buscara aplicar los conocimientos aprendidos durante las clases de este cuatrimestre, trabajar en equipo y ver el alcance que tendrá cada base de datos para cada requerimiento.

1.1. Requerimientos

Operaciones CRUD(Crear, leer, actualizar, eliminar) sobre los siguientes aspectos de cada sub tipo.

1.1.1. Pacientes

- Datos personales
- Historial médico
- Contacto de emergencia

1.1.2. Médicos

- Especialidad
- Horario
- Disponibilidad

1.1.3. Citas

- Asignar una cita entre médico y paciente
- Cambiar el horario si este lo requiere

1.1.4. Hospitalización

- Registrar medicamento en cada encuentro y tratamiento
- Recuperar registros de hospitalización con fecha de ingreso y alta

1.1.5. Historial

- Recuperar mediante un archivo pdf o excel el historial de tratamientos medicos, citas recibias de algun paciente

2. Propuesta

Nuestra propuesta cociste en poder aplicar los conceptos vistos en clase, los conceptos vistos en la materia "Bases de datos 1 ƨ autodidacta.

2.1. Persistencia Políglota

La Persistencia Políglota se encuentra en la convivencia de estos dos tipos de bases de datos, su funcionamiento es independiente porque tienen distintos objetivos y se complementan al dar un producto y dan soluciones eficientes a distintas problemáticas. Tenemos por ejemplo la generación de reportes que se encargara MongoDB y la ABM de cada médico y/o paciente tendremos MySQL.

2.2. Base de datos Relacional - MySQL

Optamos por utilizar una base de datos relacional para la gestión de pacientes, médicos y citas. Consideramos que estos tres componentes deben tener una estructura clara y bien definida con relaciones entre las tablas (por ejemplo, un paciente puede tener múltiples citas y un médico puede atender a múltiples pacientes). Las bases de datos SQL son muy eficientes para manejar estas relaciones mediante el uso de claves foráneas.

Además, para gestionar datos críticos como los de un paciente, un médico o una cita en un hospital, creemos que no hay mejor opción que garantizar las propiedades A.C.I.D, las cuales tienen como base fundamental mantener la integridad y consistencia de la información.

Por último, las bases de datos relacionales ofrecen un rendimiento fiable y robusto para transacciones frecuentes y concurrencia de usuarios, asegurando que los datos sensibles y vitales estén protegidos. Esto es esencial para un entorno hospitalario donde cualquier error o inconsistencia en los datos podría tener consecuencias graves

2.3. Base de datos No Relacional - MongoDB

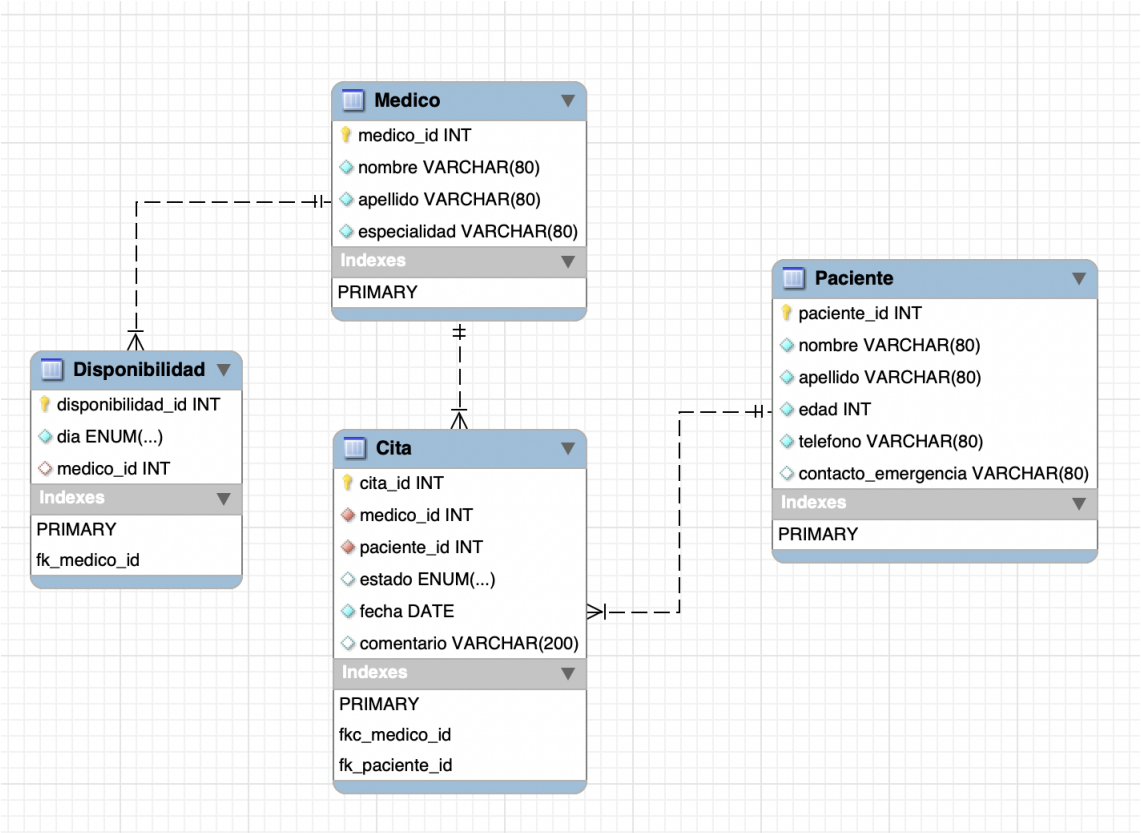
Para el resto de los requerimientos, decidimos que la mejor opción para almacenar los datos era una base de datos documental como lo es MongoDB.

Para la gestión de tratamientos, hospitalización y reportes médicos, MongoDB es ideal debido a su capacidad para manejar grandes volúmenes de datos de manera eficiente. Al utilizar un modelo de datos basado en documentos JSON, permite almacenar tratamientos médicos con detalles complejos en una estructura flexible y dinámica. Además, su escalabilidad horizontal garantiza un rendimiento óptimo incluso cuando la cantidad de datos de tratamientos y registros de hospitalización crece significativamente, asegurando la disponibilidad continua de la información crucial para la atención médica.

Por otro lado, consideramos a MongoDB una buena elección en cuanto al historial médico y comentarios, ya que permite adaptarse fácilmente a la naturaleza variable de estos datos gracias a su (ya mencionada anteriormente) flexibilidad. Esto facilita la inclusión de información diversa, como diagnósticos, tratamientos previos y observaciones, sin la necesidad de definir una estructura fija de antemano.

Finalmente, y hablando en lo que respecta a la gestión de usuarios y seguridad, MongoDB proporciona características integradas que facilitan el control de acceso y la protección de datos sensibles. Además, ofrece funcionalidades de seguridad, como autenticación, autorización basada en roles, auditoría y registro, que garantizan un control granular del acceso de los usuarios y la protección de datos confidenciales, haciendo que sea una opción sólida para aplicaciones que requieren una gestión segura y eficiente de usuarios y datos.

3. Diagrama Entidad Relación - DER



Médico Crearemos médicos con los datos personales y su especialidad.

Disponibilidad: Por cada médico que se cree se deberá cargar en esta tabla los días disponibles que tiene ese médico. Es importante aclarar que esta entidad se creo para mantener una coherencia en la información evitando redundancia en la tabla Médico, seguirá el siguiente criterio.

- Insertar médico con ID 1 disponibilidad día 'Lunes'
- Insertar médico con ID 1 disponibilidad día 'Martes'

Nos permite crear, modificar o eliminar los días de cada médico a nuestro gusto sin gran impacto en nuestro sistema.

Paciente Mantiene el mismo criterio de creación que la entidad Médico pero se agregan columnas como edad,teléfono y contacto de emergencia.

Cita A través de las FK de las tablas médico y paciente respectivamente se determinara una fecha donde se encontraran. Sus estados pueden ser "Programado.º Çancelado"manteniendo como "Programado"por defecto.

4. Queries de mySQL

Creacion de tablas de Medicos

```sql
CREATE TABLE Medico (
medico_id INT UNSIGNED AUTO_INCREMENT,
nombre VARCHAR(80) NOT NULL,
apellido VARCHAR(80) NOT NULL,
especialidad VARCHAR(80) NOT NULL,
CONSTRAINT pk_medico PRIMARY KEY (medico_id)
);
```

Creacion de tablas de Paciente

```sql
CREATE TABLE Paciente (
paciente_id INT UNSIGNED AUTO_INCREMENT,
nombre VARCHAR(80) NOT NULL,
apellido VARCHAR(80) NOT NULL,
edad INT UNSIGNED NOT NULL,
telefono varchar(80) NOT NULL,
contacto_emergencia varchar(80) ,
CONSTRAINT pk_paciente PRIMARY KEY (paciente_id)
);
```

Creacion de tablas de Cita

```sql
CREATE TABLE Cita (
cita_id INT UNSIGNED AUTO_INCREMENT,
medico_id INT UNSIGNED NOT NULL,
paciente_id INT UNSIGNED NOT NULL,
estado ENUM('Programado','Cancelado') DEFAULT 'Programado',
fecha DATE NOT NULL,
comentario VARCHAR(200), -- aca podriamos traer las condiciones "traer ropa holgada
    o venir en ayuno" por ejemplo, preguntar

CONSTRAINT pk_cita PRIMARY KEY(cita_id),
CONSTRAINT fkc_medico_id FOREIGN KEY(medico_id) REFERENCES Medico(medico_id),
CONSTRAINT fk_paciente_id FOREIGN KEY(paciente_id) REFERENCES Paciente(paciente_id)
);
```

Creacion de tablas de Disponibilidad

```sql
CREATE TABLE Disponibilidad (
disponibilidad_id INT UNSIGNED AUTO_INCREMENT,
dia ENUM('lunes','martes','miercoles','jueves','viernes','sabado','domingo') NOT
    NULL,
medico_id INT UNSIGNED,
CONSTRAINT pk_disponibilidad PRIMARY KEY(disponibilidad_id),-- Aunque no creo
    usarlo
CONSTRAINT fk_medico_id FOREIGN KEY(medico_id)REFERENCES Medico (medico_id)
);
```

5. Documento MongoDB

Creacion del documento Historial

```
//Diagnostico
{
    "id_paciente": "1",
    "id_medico": "3",
    "tipo": "diagnostico",
    "titulo": "Gripe",
    "descripcion": "El paciente presenta dolor de cabeza y tos",
    "comentarios": "El paciente no tenia fiebre"
}

//Tratameinto
{
    "id_paciente": "1",
    "id_medico": "3",
    "tipo": "tratamiento",
    "titulo": "Tratamiento para la gripe",
    "medicacion": [
        {
            "nombre": "Medicamento A",
            "dosis": "10mg",
            "frecuencia": "dos veces al día"
        }
    ],
    "procedimientos": [
        {
            "nombre": "Procedimiento A",
            "fecha": "2024-06-01",
            "notas": "El procedimiento fue exitoso."
        }
    ],
    "recomendaciones": "El paciente debe descansar una semana."
}
//Hospitalizacion
{
    "id_paciente": "1",
    "id_medico": "3",
    "tipo": "hospitalizacion",
    "titulo": "Cirujia de apendice",
    "fecha_ingreso": "2024-06-01",
    "fecha_salida": "2024-06-03",
    "ambulatorio": false,
    "medicacion": [
        {
            "nombre": "Medicamento A",
            "dosis": "10mg",
            "frecuencia": "dos veces al día"
        }
    ],
    "procedimientos": [
        {
            "nombre": "Procedimiento A",
            "fecha": "2024-06-01",
            "notas": "El procedimiento fue exitoso."
        }
    ],
    "recomendaciones": "El paciente debe descansar una semana."
}
```

6. Teorema de CAP

En el caso de mongoDB entra dentro del diagrama de Venn en el sector CP (Consistencia y Tolerancia a la partición). Esto quiere decir que a medida que nuestro sistema escale y de distribuya de en distintas áreas o sedes porque se trata de un hospital (una escalabilidad horizontal), se podrá obtener la mayor consistencia de los datos a costa de la disponibilidad de manera temporal.

6.1. Definiciones:

Consistencia: Cuando nos referimos a consistencia de los datos nos referimos a la información replicada en cada nodo a medida que este es solicitado desde nuestro cluster.

Disponibilidad: El sistema continua trabajando aunque algún cluster falle

Tolerancia a la partición: Que se mantenga la misma informacion en cada nodo.

7. Calculo de cluster bajo los parámetros N,R,W

7.1. Parámetros

N: Cantidad de cluster, en nuestro caso serán la cantidad de sedes que alberguen la información de nuestro pacientes y médicos.

R: Cantidad de replicas de nuestros cluster (N), cuanto mas grande este valor, afectaría nuestra performance.

W: Toma valores para cada operación de escritura o actualización.

7.2. Cuenta

N:3 Suponiendo que hay un nodo de nuestro producto en al menos tres provincias.

R:1 Una replica por cada cluster.

W:3

¿Verifica la el calculo? $R + W > N$

$$1 + 3 > 3 \text{ Verifica}$$

8. Seguridad y permisos

Establecemos distintos permisos y roles que permiten realizar distintas tareas a nuestro proyecto.

Rol de doctor

```
{
  "nombre": "Dr. Juan Pérez",
  "correo": "juan.perez@example.com",
  "rol": "medico",
  "permisos": ["crear_reportes", "editar_pacientes", "ver_citas"],
  "hash_contrasena": "hashed_password",
  "salt": "unique_salt"
}
```

Rol de administrativo

```
{
  "nombre": "Paula Méndez",
  "correo": "paula.mendez@example.com",
  "rol": "administrativo",
  "permisos": ["crear_pacientes", "ver_reportes", "gestionar_citas"],
  "hash_contrasena": "hashed_password",
  "salt": "unique_salt"
}
```

Con esto nos aseguramos que la información sensible u operaciones entre los índices sean afectadas solamente por personas con ciertos permisos de lectura o escritura.

9. Links

Repositorio en Github