Diseño y Programación de Software Multiplataforma

DPS941 G01T

Entrega de Foro I (Parte II):

Base SQL & Base NoSQL

ARTURO ERNESTO MUNOZ BARAHONA - MB030522

FAIRFAX, 30/04/2024

Base de Datos SQL

```sql
-- Creación de tablas con buenas prácticas y normalización aplicada

-- Tabla de Profesores
CREATE TABLE Profesores (
    id_profesor INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);

-- Tabla de Alumnos
CREATE TABLE Alumnos (
    id_alumno INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);

-- Tabla de Materias
CREATE TABLE Materias (
    id_materia INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL UNIQUE
);

-- Tabla de Asignaturas
CREATE TABLE Asignaturas (
    id_asignatura INT AUTO_INCREMENT PRIMARY KEY,
    id_materia INT NOT NULL,
    id_profesor INT NOT NULL,
    semestre VARCHAR(20) NOT NULL,
    FOREIGN KEY (id_materia) REFERENCES Materias(id_materia),
    FOREIGN KEY (id_profesor) REFERENCES Profesores(id_profesor)
);

-- Tabla de Evaluaciones
CREATE TABLE Evaluaciones (
    id_evaluacion INT AUTO_INCREMENT PRIMARY KEY,
    id_asignatura INT NOT NULL,
    id_alumno INT NOT NULL,
    nota DECIMAL(4,2) NOT NULL CHECK (nota BETWEEN 0 AND 10),
    fecha DATE NOT NULL,
    FOREIGN KEY (id_asignatura) REFERENCES Asignaturas(id_asignatura),
    FOREIGN KEY (id_alumno) REFERENCES Alumnos(id_alumno)
);

-- Inserción de datos de ejemplo
```

```sql
-- Profesores
INSERT INTO Profesores (nombre, apellido, email) VALUES
('Ana', 'Pérez', 'ana.perez@udb.edu.sv'),
('Carlos', 'Gomez', 'carlos.gomez@udb.edu.sv'),
('Luis', 'Martínez', 'luis.martinez@udb.edu.sv'),
('María', 'Lopez', 'maria.lopez@udb.edu.sv'),
('Patricia', 'Fernandez', 'patricia.fernandez@udb.edu.sv');

-- Alumnos
INSERT INTO Alumnos (nombre, apellido, email) VALUES
('Juan', 'Martínez', 'juan.martinez@udb.edu.sv'),
('Laura', 'Jiménez', 'laura.jimenez@udb.edu.sv'),
('Roberto', 'Núñez', 'roberto.nunez@udb.edu.sv'),
('Sofía', 'Castro', 'sofia.castro@udb.edu.sv'),
('Miguel', 'Díaz', 'miguel.diaz@udb.edu.sv');

-- Materias
INSERT INTO Materias (nombre) VALUES
('Matemáticas IV'),
('Diseno Y Programacion de Software Multiplataforma'),
('Física II'),
('Química I'),
('Redes');

-- Asignaturas
INSERT INTO Asignaturas (id_materia, id_profesor, semestre) VALUES
(1, 1, '2022-I'),
(2, 2, '2022-II'),
(3, 3, '2022-I'),
(4, 4, '2022-II'),
(5, 5, '2022-I');

-- Evaluaciones
INSERT INTO Evaluaciones (id_asignatura, id_alumno, nota, fecha) VALUES
(1, 1, 8.5, '2022-10-10'),
(2, 2, 7.2, '2022-11-15'),
(3, 3, 9.0, '2022-10-03'),
(4, 4, 6.5, '2022-11-12'),
(5, 5, 8.8, '2022-10-07');
```

Este script incluye:

- **Claves Primarias Autoincrementables** para una fácil inserción y referencia.
- **Restricciones de Integridad**, como las claves foráneas para garantizar relaciones correctas y una restricción **CHECK** en la tabla **Evaluaciones** para asegurar que las notas estén dentro de un rango válido.

- **Valores Únicos** para emails y nombres de materias para evitar duplicados donde no deben existir.
- **Valores No Nulos** en campos críticos para garantizar que la información esencial siempre esté presente.

Relaciones de la Base de Datos (SQL)

1. Relación entre Profesores y Asignaturas

- **Relación**: Un profesor puede enseñar múltiples asignaturas, pero cada asignatura es enseñada por un solo profesor.
- **Implementación**: Se utiliza una clave foránea (**id_profesor**) en la tabla **Asignaturas** que referencia la clave primaria (**id_profesor**) en la tabla **Profesores**.
- **SQL**: **FOREIGN KEY (id_profesor) REFERENCES Profesores(id_profesor)**

2. Relación entre Materias y Asignaturas

- **Relación**: Una materia puede ser parte de múltiples asignaturas.
- **Implementación**: Se utiliza una clave foránea (**id_materia**) en la tabla **Asignaturas** que referencia la clave primaria (**id_materia**) en la tabla **Materias**.
- **SQL**: **FOREIGN KEY (id_materia) REFERENCES Materias(id_materia)**

3. Relación entre Alumnos y Evaluaciones

- **Relación**: Un alumno puede tener múltiples evaluaciones en diferentes asignaturas.
- **Implementación**: Se utiliza una clave foránea (**id_alumno**) en la tabla **Evaluaciones** que referencia la clave primaria (**id_alumno**) en la tabla **Alumnos**.
- **SQL**: **FOREIGN KEY (id_alumno) REFERENCES Alumnos(id_alumno)**

4. Relación entre Asignaturas y Evaluaciones

- **Relación**: Una asignatura puede tener múltiples evaluaciones de diferentes alumnos.
- **Implementación**: Se utiliza una clave foránea (**id_asignatura**) en la tabla **Evaluaciones** que referencia la clave primaria (**id_asignatura**) en la tabla **Asignaturas**.
- **SQL**: **FOREIGN KEY (id_asignatura) REFERENCES Asignaturas(id_asignatura)**

Estas relaciones son esenciales para mantener la integridad de los datos y facilitar consultas eficientes que requieran información de varias tablas. Por ejemplo, si necesitas saber todas las evaluaciones de un alumno en particular, la relación entre **Alumnos** y **Evaluaciones** permite realizar esta consulta de manera sencilla y directa. Además, el uso de claves foráneas ayuda a prevenir la inserción de datos inconsistentes, como una evaluación con un **id_alumno** que no exista.

Conclusiones (SQL)

Al concluir el proyecto de implementación de la base de datos SQL para UDB VIRTUAL, he reunido algunas reflexiones y aprendizajes que podrían resumir mejor la experiencia de una manera más humana y comprensible:

1. El Arte de la Normalización

Aprender a normalizar la base de datos fue como desenredar una madeja de hilo; cada paso nos ayudó a clarificar y simplificar la estructura. Esto no solo nos ayudó a evitar la duplicación de datos, sino que también nos permitió mantener los datos coherentes y manejables. Este cuidado en la organización inicial paga dividendos en mantener el sistema ordenado a largo plazo.

2. Flexibilidad en el Diseño

Adaptar el diseño de la base de datos para anticiparse a cambios futuros resultó ser una estrategia crucial. Pensar en cómo podría evolucionar el sistema nos ayudó a implementar una estructura que no solo resuelve los problemas actuales sino que también es capaz de adaptarse sin grandes sobresaltos. Esto es algo así como construir una casa pensando no solo en lo que necesitas ahora, sino también en lo que podrías necesitar en 10 años.

3. La Importancia de la Integridad de los Datos

Establecer reglas estrictas para la entrada de datos fue como asignar un guardián en la puerta de cada dato. Estas reglas aseguran que sólo la información correcta y válida entre en nuestro sistema, previniendo errores que podrían llevar a decisiones basadas en datos incorrectos.

4. Velocidad y Eficiencia en las Consultas

Optimizar las consultas fue un juego de equilibrio. Descubrir que añadir índices en las columnas correctas podía acelerar las consultas fue una revelación, similar a encontrar un atajo en un camino familiar que usas a diario. Esto hizo nuestra base de datos mucho más ágil y rápida.

5. Preparados para el Futuro

Pensar en la escala de la base de datos fue como planear una fiesta esperando más invitados de los que conoces actualmente. Asegurarnos de que nuestro sistema pueda manejar un crecimiento futuro sin caer bajo el estrés fue esencial para garantizar su longevidad y eficacia.

6. Documentar es Recordar

La documentación detallada del proceso y la estructura no solo ayudó a los demás a entender nuestro trabajo, sino que también sirvió como un diario que podemos revisar para recordar por qué tomamos ciertas decisiones. Esta práctica es invaluable, especialmente cuando el proyecto se pasa a manos de nuevas personas.

7. Mejorar la Experiencia del Usuario

Finalmente, la conexión entre la estructura de la base de datos y la experiencia del usuario final resultó ser más profunda de lo esperado. Una base de datos bien diseñada puede transformar la velocidad y eficiencia de una aplicación, lo que directamente mejora cómo se siente y usa la plataforma.

Base de Datos – NoSQL

Este script usa Firebase Admin SDK para Node.js

```
const admin = require('firebase-admin');
const serviceAccount = require('./path/to/your-firebase-service-account-
file.json');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();

async function addInitialData() {
  // Agregar profesores
  const profesorDocs = [
    { nombre: 'Ana', apellido: 'Pérez', email: 'ana.perez@udb.edu.sv' },
    { nombre: 'Carlos', apellido: 'Gomez', email: 'carlos.gomez@udb.edu.sv' },
    { nombre: 'Luis', apellido: 'Martínez', email: 'luis.martinez@udb.edu.sv'
},
    { nombre: 'María', apellido: 'Lopez', email: 'maria.lopez@udb.edu.sv' },
    { nombre: 'Patricia', apellido: 'Fernandez', email:
'patricia.fernandez@udb.edu.sv' }
  ].map(data => db.collection('Profesores').add(data));

  // Agregar materias
  const materiaDocs = [
    { nombre: 'Diseno Y Programacion de Software Multiplataforma' },
    { nombre: 'Matemáticas IV' },
    { nombre: 'Física II' },
    { nombre: 'Química I' },
    { nombre: 'Redes' }
  ].map(data => db.collection('Materias').add(data));

  const profesores = await Promise.all(profesorDocs);
  const materias = await Promise.all(materiaDocs);

  // Agregar asignaturas con referencias a profesores y materias
  const asignaturas = [
    { materiaRef: materias[0], profesorRef: profesores[0], semestre: '2022-I'
},
    { materiaRef: materias[1], profesorRef: profesores[1], semestre: '2022-II'
},
    { materiaRef: materias[2], profesorRef: profesores[2], semestre: '2022-I'
},
    { materiaRef: materias[3], profesorRef: profesores[3], semestre: '2022-II'
},
```

```
    { materiaRef: materias[4], profesorRef: profesores[4], semestre: '2022-I'
}
  ].map(data => db.collection('Asignaturas').add(data));

  // Agregar alumnos y subcolecciones de evaluaciones
  const alumnos = [
    { nombre: 'Juan', apellido: 'Martínez', email: 'juan.martinez@udb.edu.sv'
},
    { nombre: 'Laura', apellido: 'Jiménez', email: 'laura.jimenez@udb.edu.sv'
},
    { nombre: 'Roberto', apellido: 'Núñez', email: 'roberto.nunez@udb.edu.sv'
},
    { nombre: 'Sofía', apellido: 'Castro', email: 'sofia.castro@udb.edu.sv' },
    { nombre: 'Miguel', apellido: 'Díaz', email: 'miguel.diaz@udb.edu.sv' }
  ];

  for (let alum of alumnos) {
    const alumRef = await db.collection('Alumnos').add(alum);
    // Añadir evaluaciones como subcolección de cada alumno
    await
db.collection('Alumnos').doc(alumRef.id).collection('Evaluaciones').add({
      asignaturaRef: db.collection('Asignaturas').doc((await
asignaturas[0]).id),
      nota: 8.5,
      fecha: admin.firestore.Timestamp.fromDate(new Date('2022-10-10'))
    });
  }
}

addInitialData().catch(console.error);
```

Puntos Clave del Script

- **Referencias Documentales**: Este script utiliza referencias a documentos para **materiaRef** y **profesorRef** en la colección **Asignaturas**, y para **asignaturaRef** en la subcolección **Evaluaciones**.
- **Manejo de Promesas**: Dado que la creación de documentos en Firestore es una operación asíncrona, el script usa **Promise.all** para esperar que todos los documentos sean creados antes de proceder a referenciarlos.
- **Timestamps**: Las fechas se manejan como timestamps de Firestore, lo cual es ideal para aplicaciones en tiempo real.

Conclusiones (NoSQL)

1. Adaptabilidad y Creatividad

Utilizar Firestore realmente me recordó la importancia de ser adaptable en el desarrollo de tecnología. La flexibilidad de Firestore permite que los esquemas de datos evolucionen a medida que cambian las necesidades educativas, lo cual es como tener la capacidad de rediseñar un aula para cada nueva clase que entra, asegurando que siempre se adapte a sus necesidades específicas.

2. Desarrollo Dinámico y Ágil

La experiencia con Firestore agilizó notablemente el proceso de desarrollo. Fue como trabajar con bloques de construcción intuitivos que se pueden reorganizar rápidamente para crear estructuras robustas y funcionales, permitiendo más tiempo para centrarse en mejorar la experiencia del usuario y menos en la gestión de la base de datos.

3. Interacción en Tiempo Real

La capacidad de Firestore para actualizar y sincronizar datos en tiempo real transformó la forma en que los estudiantes y profesores interactúan. Es como tener una conversación en tiempo real en lugar de intercambiar cartas; todo es instantáneo, lo que enriquece la experiencia de aprendizaje y hace que la comunicación sea más efectiva.

4. Complejidad Oculta

A pesar de sus ventajas, lidiar con la complejidad de las consultas en Firestore fue un desafío. Me hizo apreciar la importancia de planificar cuidadosamente cómo se estructuran los datos para evitar búsquedas complicadas y costosas, algo parecido a planificar un viaje complejo donde cada decisión puede afectar la eficiencia del recorrido.

5. Consistencia y Rigor

La gestión de la consistencia de los datos en un sistema flexible como Firestore me enseñó a ser más meticuloso. Asegurarme de que los datos sean consistentes sin las restricciones típicas de las bases de datos SQL fue como hacer malabares; requiere atención constante y precisión para mantener todo en armonía.

6. Seguridad Como Prioridad

Configurar las reglas de seguridad en Firestore subrayó la importancia de proteger la información sensible. Me recordó que, al igual que en un ambiente educativo donde la seguridad del estudiante es prioritaria, la seguridad de los datos también debe serlo, requiriendo vigilancia continua y adaptaciones conforme cambian las amenazas.

7. Impacto en la Experiencia del Usuario

Finalmente, la velocidad y eficiencia de Firestore mejoraron notablemente la experiencia de los usuarios. Ver el efecto directo de un sistema de base de datos bien implementado en la satisfacción y el compromiso de los estudiantes fue sumamente gratificante, como ver a los estudiantes prosperar en un entorno de aprendizaje bien soportado.