

Universidad Don Bosco  
Escuela De Computación



Docente: Alexander Alberto Sigüenza Campos

FACULTAD DE INGENIERÍA

Materia: Diseño y Programación de Software Multiplataforma

Actividad: Foro 1.

Integrantes: Jordan Ismael Zelaya Ramírez – ZR170168.

Fecha de entrega: 29 de octubre del 2023.

## **¿Cuáles son las diferencias fundamentales entre bases de datos SQL y NoSQL?**

Las bases de datos SQL (Structured Query Language) y NoSQL (Not Only SQL) son dos enfoques diferentes para almacenar y gestionar datos. Estos serían los puntos

### **Modelo de datos:**

- SQL: Las bases de datos SQL siguen un modelo de datos relacional. Utilizan tablas con filas y columnas para organizar la información, lo que facilita las consultas y la relación entre los datos.
- NoSQL: Las bases de datos NoSQL utilizan diversos modelos de datos, como documentos, gráficos, columnas o clave-valor. Esto brinda flexibilidad para almacenar datos no estructurados o semiestructurados.

### **Estructura y esquema:**

- SQL: Tienen un esquema fijo y definido, lo que significa que la estructura de la base de datos se establece antes de ingresar los datos.
- NoSQL: Son flexibles en cuanto al esquema, lo que permite agregar campos y estructuras de datos según las necesidades cambiantes.

### **Escalabilidad:**

- SQL: La escalabilidad vertical (mejora de recursos en un servidor único) es común en las bases de datos SQL, lo que puede ser costoso y limitado en términos de escalabilidad horizontal (agregar más servidores).
- NoSQL: Suelen ser más fáciles de escalar horizontalmente, lo que es beneficioso para aplicaciones web con grandes cantidades de datos y tráfico.

### **Transacciones:**

- SQL: Son conocidas por garantizar la integridad de los datos mediante transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad). Esto es esencial en aplicaciones financieras y críticas.
- NoSQL: Suelen seguir el modelo BASE (Básicamente Disponible, Suave, Eventualmente Consistente), que puede ser más flexible pero no garantiza la misma consistencia en todo momento.

### **Consultas y flexibilidad:**

- SQL: Son ideales para aplicaciones donde se requieren consultas complejas y estructuradas, ya que SQL es excelente para JOIN y agregaciones.
- NoSQL: Son más adecuadas cuando se necesita flexibilidad y se prioriza la velocidad de lectura/escritura sobre consultas complejas.

### **Ejemplos de uso:**

- SQL: Se utilizan en aplicaciones como sistemas de gestión de bases de datos tradicionales, aplicaciones empresariales y sistemas financieros.
- NoSQL: Son comunes en aplicaciones web, redes sociales, sistemas de recomendación y big data.

## **¿Cuáles son las diferencias específicas entre Cloud Firestore y Realtime Database?**

### **Modelo de datos:**

- Cloud Firestore: Utiliza un modelo de base de datos NoSQL de documentos que almacenan datos en colecciones. Cada documento es un objeto JSON, lo que facilita la organización de datos jerárquica y anidada.
- Realtime Database: Se basa en un modelo de datos JSON en tiempo real. Los datos se almacenan en un solo árbol JSON, lo que facilita la sincronización en tiempo real.

### **Escalabilidad:**

- Cloud Firestore: Ofrece una escalabilidad más sencilla y eficiente, lo que significa que es fácil de escalar horizontalmente para manejar grandes volúmenes de datos y tráfico.
- Realtime Database: Si bien es escalable, la escalabilidad horizontal puede ser más desafiante debido a su estructura de árbol único.

### **Consultas:**

- Cloud Firestore: Permite consultas más avanzadas y flexibles, incluyendo consultas compuestas, filtrado y ordenación. Esto facilita la recuperación de datos específicos.
- Realtime Database: Ofrece menos flexibilidad en las consultas. La filtración y el orden se limitan a una sola propiedad a la vez.

### **Sincronización en tiempo real:**

- Cloud Firestore: Proporciona sincronización en tiempo real de manera nativa. Esto significa que los cambios en los datos se reflejan automáticamente en todos los dispositivos conectados sin necesidad de configuración adicional.
- Realtime Database: También ofrece sincronización en tiempo real, pero puede requerir más trabajo para implementarla en comparación con Firestore.

### **Facturación:**

- Cloud Firestore: Tiene una estructura de facturación más granular, lo que puede ser más predecible en términos de costos. Pagas por lecturas, escrituras y almacenamiento.
- Realtime Database: Su facturación se basa en el ancho de banda y la cantidad de datos transferidos, lo que puede ser menos predecible.

### **Soporte para transacciones:**

- Cloud Firestore: Ofrece soporte para transacciones, lo que permite operaciones atómicas en varios documentos, garantizando la integridad de los datos.
- Realtime Database: También admite transacciones, pero se limita a un nodo específico en la base de datos.

### **Casos de uso:**

- Cloud Firestore: Es una excelente elección para aplicaciones modernas que requieren estructuras de datos más complejas y consultas flexibles.
- Realtime Database: A menudo se utiliza en aplicaciones en tiempo real, como chats y juegos, donde la sincronización instantánea es fundamental.
- 

**¿Cuál de estas bases de datos consideran que sería la mejor opción para implementar en una aplicación desarrollada en React Native?**

Depende del caso de uso, los recursos con los que se cuentan y el tiempo para poder entregar el proyecto. En este caso, hablando en concreto de la temática que se ha asignado para esta proyecto de catedra, yo me inclinaría por una base de datos SQL, pues es más fácil de implementar, dado el pequeño tamaño del proyecto, y porque permite consultas complejas que puedan llegar a surgir a medida que el cliente vea la utilidad del proyecto en sus empleados que serían los doctores, y sus clientes que serían los ancianos.

## Parte II

### Base de datos SQL:

```
CREATE DATABASE UDB_BECARIOS;
-- Seleccionar la base de datos
USE UDB_BECARIOS;
-- Creación de la tabla Usuarios
CREATE TABLE Usuarios (
    UserID INT PRIMARY KEY,
    Nombre VARCHAR(50),
    Apellido VARCHAR(50),
    FechaNacimiento DATE,
    Email VARCHAR(100),
    Contraseña VARCHAR(50)
);

-- Creación de la tabla Becas
CREATE TABLE Becas (
    BecaID INT PRIMARY KEY,
    NombreBeca VARCHAR(100),
    Descripcion TEXT,
    Monto DECIMAL(10, 2)
);

-- Creación de la tabla Cursos
CREATE TABLE Cursos (
    CursoID INT PRIMARY KEY,
    NombreCurso VARCHAR(100),
    Descripcion TEXT,
    Creditos INT
);

-- Creación de la tabla Notas
CREATE TABLE Notas (
    NotaID INT PRIMARY KEY,
    UserID INT,
    CursoID INT,
    Nota DECIMAL(4, 2),
    FOREIGN KEY (UserID) REFERENCES Usuarios(UserID),
```

```

FOREIGN KEY (CursoID) REFERENCES Cursos(CursoID)
);

```

```

-- Creación de la tabla BecasUsuarios (para gestionar las becas de los usuarios)
CREATE TABLE BecasUsuarios (
BecaUsuarioID INT PRIMARY KEY,
UserID INT,
BecaID INT,
FOREIGN KEY (UserID) REFERENCES Usuarios(UserID),
FOREIGN KEY (BecaID) REFERENCES Becas(BecaID)
);

```

## Base de datos No-SQL(FireStore):

Firestore Screenshot - Becas Collection:

- Document ID: Iv0Y4VuOSL6eJNTLDApZ
- Fields:
  - Descripcion: ""
  - Monto: 0
  - NombreBecas: ""

Firestore Screenshot - BecasUsuarios Collection:

- Document ID: hKhjX6xubdrQ1JCdHU1n
- Fields:
  - BecaID: ""
  - UserID: ""

Firestore Screenshot - Cursos Collection:

- Document ID: Aid2TJ3TBXhg1g1Rn8Gr
- Fields:
  - Creditos: 0
  - Descripcion: ""
  - NombreCursos: ""

## Conclusiones finales

En este caso concreto, para la base de datos SQL creada en MySQL, se puede notar como primer punto que está normalizada, es decir, que evita la redundancia. Por ejemplo en la tabla Notas, hay referencias hacia el ID del usuario, el cual está establecido como una clave foránea hacia la clave principal de la tabla usuario, que es su respectivo identificador, así mismo para el resto de tablas según la necesidad, esto permite que al momento de hacer consultas complejas, usando el JOIN sea más fácil crear nuevas tablas o vistas que permitan visualizar lo que realmente se quiere en ese momento. Esto es muy útil para negocios que están planeando usar sus datos a la larga para minería de datos, y con ello encontrar patrones de compra que les permitan elevar sus ventas. En este caso, si la universidad quisiera encontrar patrones entre becas, notas y los estudiantes, sería mucho más fácil llevarlo a cabo si se usa SQL.

Por otro lado, para el uso del No SQL, no existen claves foráneas, cada colección (equivalente a una tabla) puede tener datos que sean redundantes o bien que puedan relacionarse con otra tabla, pero a la hora de extraer y ejecutar transacciones complejas, no podremos realizarlo, por lo que si la universidad quiere

ocupar esa base de datos de FireStore únicamente para el ingreso y visualización de datos de manera rápida y eficaz, esta sería la mejor opción. Estas serían las diferencias en el uso e implementación de cada una.