UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3088 – Base de datos I

Sección 20

Ing. Bacilio Bolaños

Proyecto No. 2

Leonardo Dufrey Mejía Mejía, 23648

María José Girón Isidro, 23559

Dulce Rebeca Ambrosio Jiménez, 23114

Paula De León, 23202

GUATEMALA, 11 de marzo de 2025

Proyecto No. 2

Introducción

Durante el desarrollo del proyecto, nos enfocamos en implementar concurrencia usando distintos niveles de aislamiento para entender cómo afectan el rendimiento y la consistencia de los datos. Elegimos Go como lenguaje principal por su facilidad con las goroutines y el buen manejo de recursos, lo que nos permitió simular múltiples usuarios sin complicaciones.

Enlace de github: https://github.com/Dulce2004/BD1-Proyecto2.git

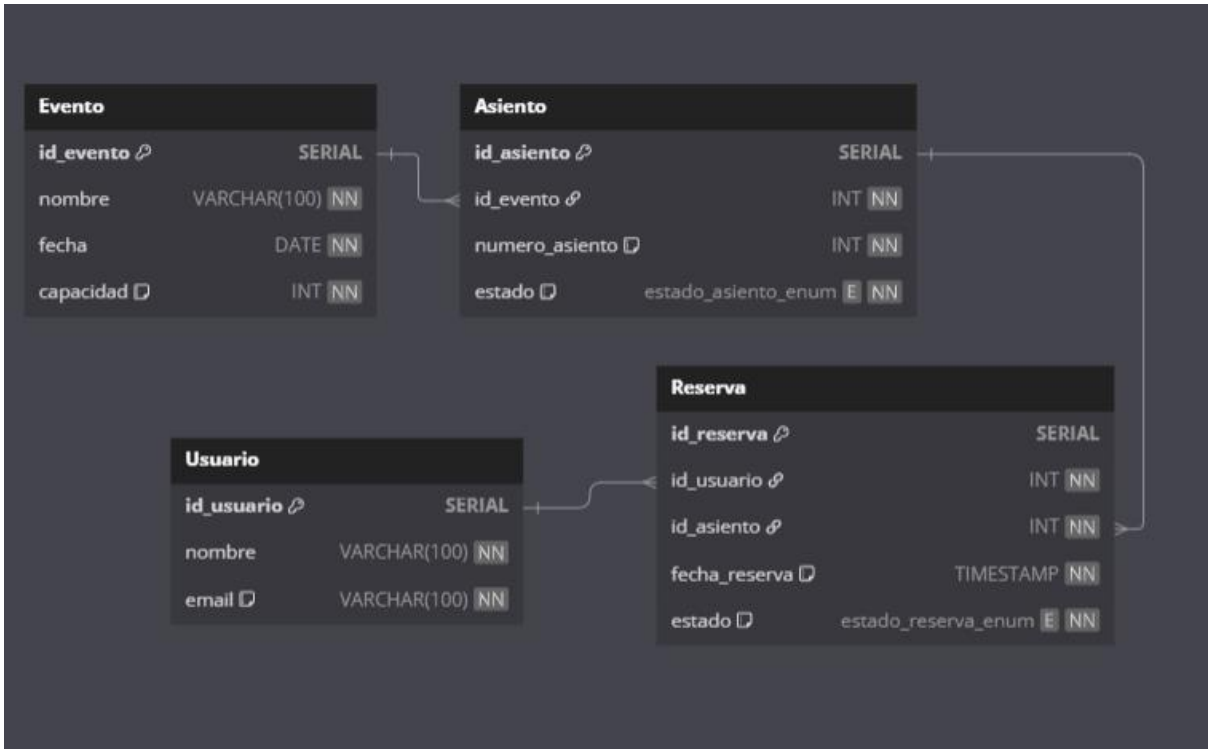

Figura 1. Diagrama de Entidad-Relación

Utilizamos dos enum, como recomendación del catedrático, no obstante, en el diagrama no se ven visibles.

Usuarios Concurrentes	Nivel de Aislamiento	Reservas Exitosas	Reservas Fallidas	Tiempo promedio (ms)
5	READ COMMITTED	1	4	23.4
5	REPEATABLE READ	1	4	8.8
5	SERIALIZABLE	1	4	8.2
10	READ COMMITTED	1	9	30.2
10	REPEATABLE READ	1	9	46.6
10	SERIALIZABLE	1	9	39.7
20	READ COMMITTED	1	19	75.2
20	REPEATABLE READ	1	19	47.05
20	SERIALIZABLE	1	19	78.15
30	READ COMMITTED	1	29	81.53
30	REPEATABLE READ	1	29	100.67
30	SERIALIZABLE	1	29	118.6

Figura No. 2 - Tabla de tiempos

Conclusiones Generales

Una de las cosas más importantes al trabajar con concurrencia fue asegurarnos de que solo un usuario pudiera reservar un asiento sin que se generaran errores o inconsistencias. Por eso, decidimos enfocarnos en un solo asiento para poder implementar mejor la lógica y ver cómo se comportaba en los distintos niveles de aislamiento.

Nos dimos cuenta de que, dependiendo del nivel de aislamiento, los resultados pueden cambiar bastante. Read Committed es más rápido, pero menos seguro, mientras que Serializable garantiza más seguridad, aunque tenga más errores y demore más. Entonces, al final, la elección depende del tipo de aplicación que se esté desarrollando.

- **¿Cuál fue el mayor reto al implementar la concurrencia?**

Uno de los mayores retos fue ejecutar las transacciones, la mayor dificultad que presentamos fue que un solo usuario se asignara un asiento de forma segura, por eso mismo implementamos que solo se pudiera asignar un id de asiento al cual se le iba a implementar la concurrencia porque se nos complicó agregar más parqueos y debido a que no era un requisito se decidió dejarlo de esa forma.

- **¿Qué problemas de bloqueo encontraron?**

En los niveles como repeatable o serializable, aunque en realidad notamos que los tiempos algunas veces también en Read Committed, pero se mantenían unos bloqueos con mayor tiempo que otros sobre los datos que se habían leído y había algunos códigos de error que nos salieron como los de serialización.

- **¿Cuál fue el nivel de aislamiento más eficiente?**

Depende a que es lo que se busca, debido a que read comitted, tiene el menor tiempo por usuario, pero existen mayor inconsistencia por lo que esto puede variar bastante como lo mencione anteriormente. Repeatable Read tiene menos conflictos y Serializable tiene la mayor seguridad, pero los mayores tiempos por usuario además de tener errores de transacciones. Entonces depende ya que Read Committed es el más eficiente, pero para la seguridad sería Serializable.

- **¿Qué ventajas y desventajas tuvo el lenguaje seleccionado?**

Ventajas:

Terminamos utilizando go debido a la fácil implementación con su función llamado goroutines sobre otros lenguajes debido a que estas ya son nativas y nos permitía enforcarnos en otros aspectos además de tener una sync nativa.

Se consumen pocos recursos por lo que se pueden simular más usuarios, además de una fácil integración de librerías externas como postgresql y el driver. Además de su tipado estático y manejo de errores que son fáciles de manejar en go, una curva de aprendizaje baja para que todos los miembros del equipo aportaran sus ideas y lograran comprender que estaba sucediendo.

Desventajas:

Los errores se tienen que tratar manualmente lo que genera un poco más de código.

Se requiere un manejo explícito de contexto y rollback.

Tal vez existen algunas mejoras en las abstracciones si lo comparamos con Python o Java, pero no creo que sean tan notorios.

Recomendaciones

Usar Read Committed para proyectos donde se priorice el tiempo y el rendimiento, pero siempre evaluar que es lo que se necesita para el programa para implementar el mejor, porque para un sistema por ejemplo bancario sería recomendable serializable con una lógica diferente por su seguridad.