UNIVERSIDAD DEL VALLE DE GUATEMALA

Base de Datos 1

Sección 40

Ing. Bacilio Alexander Bolaños

“Proyecto 2”

“Reservas con concurrencia y ACID”

Andrés Mazariegos Escobar

Diego José López Campos

June Herrera Watanabe

13 de abril de 2025 Ciudad de Guatemala

Resultados Obtenidos

Cuadro 1: Resultados Comparativos de Simulación de Concurrencia

Usuarios	Nivel de Aislamiento	Reservas Exitosas	Reservas Fallidas	Tiempo Promedio (ms)
5	READ COMMITTED	5	0	23
10	REPEATABLE READ	9	1	35
20	SERIALIZABLE	2	18	34
30	SERIALIZABLE	2	28	40

Conclusiones sobre Manejo de Concurrencia en DBs

1. Al trabajar con transacciones concurrentes, cada operación de reserva de un asiento (o cualquier recurso compartido) puede bloquear temporalmente ciertas filas de la base de datos (en nuestro caso, la fila correspondiente a cada asiento). Estos bloqueos se producen especialmente cuando se utilizan sentencias como SELECT ... FOR UPDATE, que evitan que otras transacciones modifiquen los mismos registros al mismo tiempo.

2. Niveles de Aislamiento:
    - READ COMMITTED permitió un mayor número de reservas exitosas con un mínimo de bloqueos a la vez que mantuvo una buena eficiencia en tiempo de respuesta.

    - REPEATABLE READ agregó restricciones adicionales de lectura que redujeron ligeramente el rendimiento, pero aun así mantuvo niveles aceptables de éxito.

    - SERIALIZABLE es el nivel más estricto: garantiza la mayor consistencia, pero sacrifica la tasa de éxito cuando hay muchos accesos simultáneos, generando más bloqueos y abortos de transacciones.

3. Con 5 y 10 usuarios concurrentes, las reservas exitosas fueron considerablemente altas, mostrando que el sistema maneja relativamente bien la concurrencia a baja escala.

4. Con 20 y 30 usuarios (especialmente usando SERIALIZABLE), se evidenció un mayor número de fallas por bloqueos o por abortos de transacciones, lo que deja ver la complejidad al aumentar el nivel de aislamiento y la cantidad de operaciones simultáneas.

Análisis y Reflexión

1. ¿Cuál fue el mayor reto al implementar la concurrencia?

El mayor reto consistió en coordinar y entender adecuadamente el manejo de hilos (goroutines) y transacciones en el lenguaje Go, especialmente considerando que no se tenía mucha experiencia con la concurrencia ni con la gestión de niveles de aislamiento en bases de datos. Fue necesario comprender cómo interactúan las transacciones cuando varias goroutines intentan reservar simultáneamente el mismo recurso (asiento) y cómo evitar inconsistencias o bloqueos prolongados.

2. ¿Qué problemas de bloqueo encontraron?

Se presentaron escenarios de bloqueo (locks) al utilizar sentencias FOR UPDATE, donde varias transacciones competían por el mismo asiento.
Bajo niveles de aislamiento más estrictos (SERIALIZABLE), hubo mayor posibilidad de que las transacciones se anularan entre sí al detectar posibles inconsistencias, incrementando el número de reservas fallidas.

3. ¿Cuál fue el nivel de aislamiento más eficiente?

De acuerdo con los resultados, READ COMMITTED resultó ser el más eficiente en términos de la combinación de altas reservas exitosas y bajos tiempos de respuesta. Aunque no garantiza la misma robustez que SERIALIZABLE, en escenarios de alta concurrencia logró minimizar los bloqueos y producir mayor cantidad de reservas exitosas.

4. ¿Qué ventajas y desventajas tuvo el lenguaje Go?

<u>Ventajas:</u>

- Facilidad de uso de goroutines y canales: Go provee un modelo de concurrencia bastante sencillo de implementar en comparación con otros lenguajes, lo cual permite escribir código concurrente de manera más natural.

- Alto rendimiento: El compilador y runtime de Go ofrecen muy buen desempeño, haciendo que las aplicaciones sean rápidas y eficientes en el uso de recursos.

<u>Desventajas:</u>

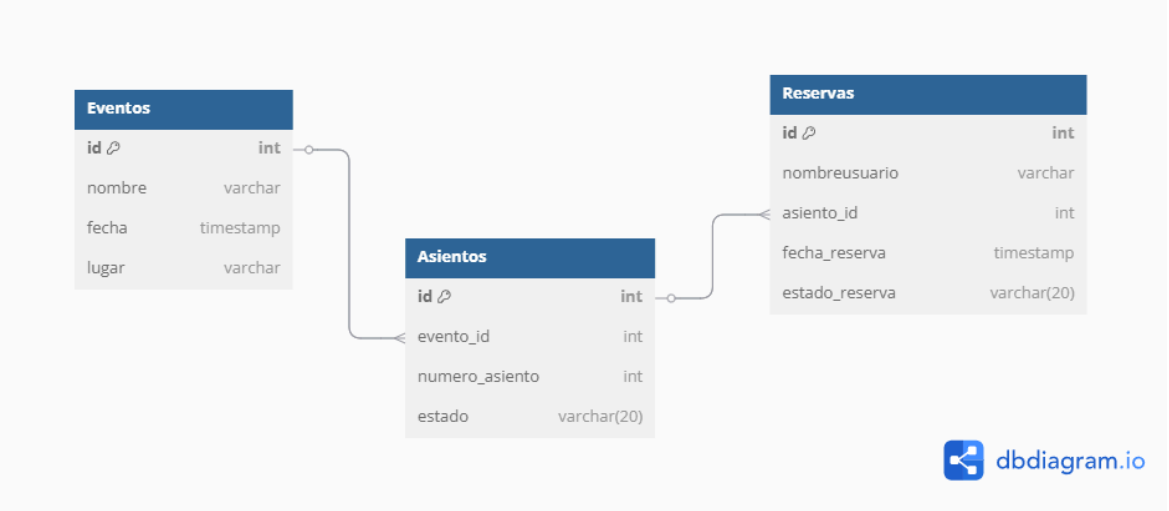
- Curva de aprendizaje inicial: Para quienes no están familiarizados con Go, su sintaxis minimalista y su manejo explícito de errores pueden resultar poco intuitivos al principio.

- Escasa experiencia previa del equipo: En nuestro caso particular, se desconocían muchos aspectos de Go y de la gestión de concurrencia (goroutines, canales, etc.), lo cual hizo más difícil la implementación.

- Manejo de errores manual: No existe un mecanismo de excepciones como en otros lenguajes; las funciones retornan errores que deben manejarse explícitamente, lo que puede resultar tedioso para quienes no estén habituados.

Anexos

Gráfica 1. Diagrama ER del modelo de reservas.



Enlace 1. Enlace para el repositorio de GitHub

https://github.com/skemono/Booking-Cocurrency