UNIVERSIDAD DEL VALLE DE GUATEMALA

Bases de datos

Bases de datos: Proyecto 2

Angel Esquit #23221

Javier España #23361

Roberto Barreda #23354

Guatemala, 11 de Abril de 2025

Fase 1

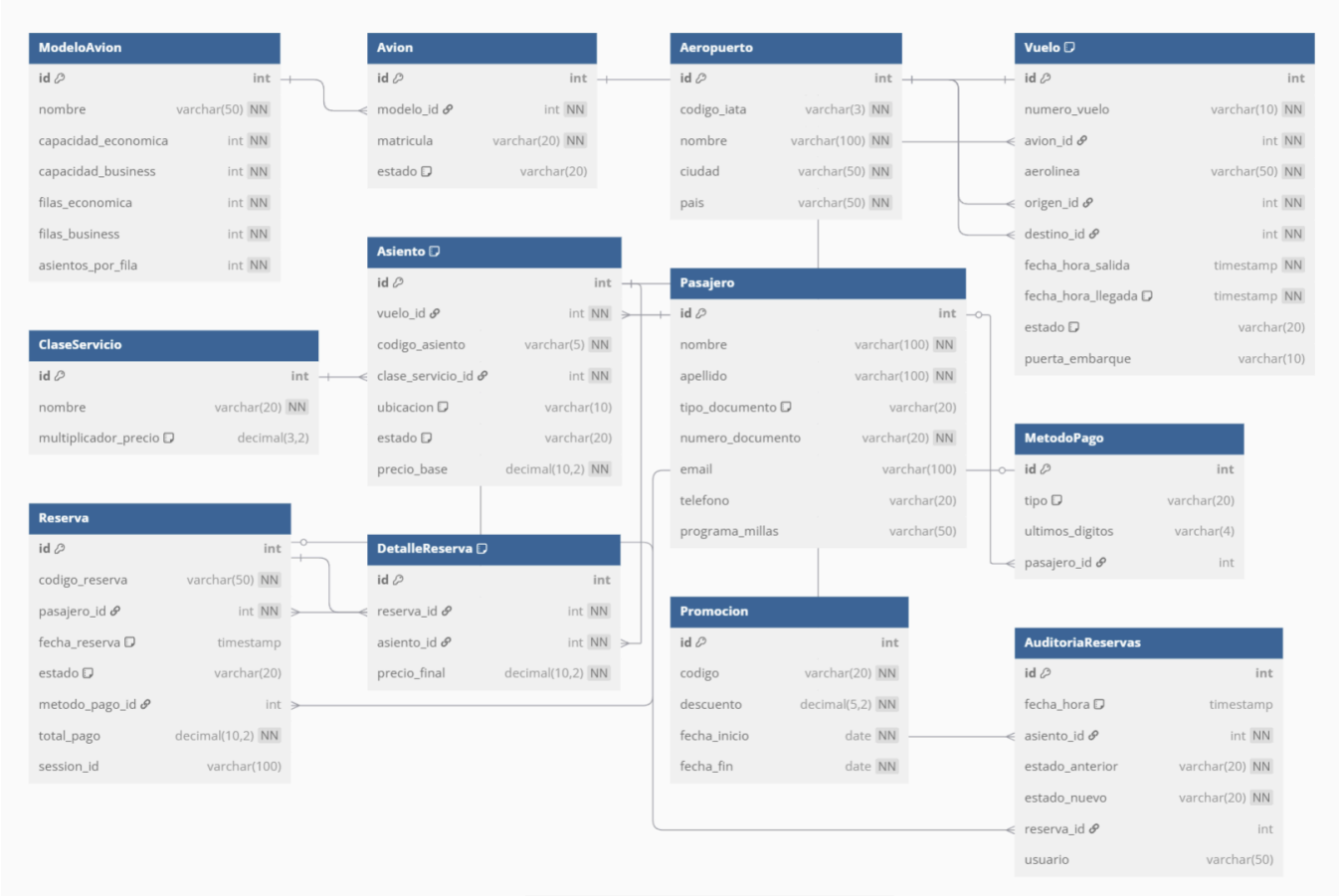

Ilustración 1. Diagrama Entidad-Relación.

Atención: La imágen está en el repositorio para verla con mayor calidad.

Fase 2

En el repositorio está data.sql dentro de la carpeta sql_scripts

Fase 3

El manual de uso se encuentra en el README.md

Fase 4

El objetivo de esta fase fue evaluar el comportamiento del sistema de reservas bajo diferentes niveles de aislamiento y concurrencia, utilizando la base de datos diseñada en la fase uno.

Hicimos una simulación del escenario de múltiples usuarios compitiendo por reservar el mismo asiento en un vuelo.

Concurrencia	Nivel Aislamiento	Exitos	Fallos	Deadlocks	Tiempo Avg (ms)	Estado Final
5	READ COMMITTED	5	0	0	303.39	disponible
5	REPEATABLE READ	5	0	0	357.78	disponible
5	SERIALIZABLE	5	0	0	574.3	disponible
10	READ COMMITTED	10	0	0	329.51	disponible
10	REPEATABLE READ	10	0	0	407.2	disponible
10	SERIALIZABLE	9	1	0	536.35	disponible
20	READ COMMITTED	16	4	0	395.43	disponible
20	REPEATABLE READ	17	3	0	458.03	disponible
20	SERIALIZABLE	16	4	0	542.59	disponible
30	READ COMMITTED	21	9	0	464.64	disponible
30	REPEATABLE READ	28	2	0	600.06	disponible
30	SERIALIZABLE	22	8	0	649.43	disponible

Cuadro 1: Resultados comparativos de simulación de concurrencia

Análisis de resultados:

a. Niveles de Aislamiento
   - READ COMMITTED:
     - Mayor eficiencia en tiempo (303.39ms para 5 usuarios).
     - En alta concurrencia, 21 éxitos con 30 usuarios pero 9 fallos por condiciones de carrera.

   - REPEATABLE READ:

    - Balance óptimo: 28 éxitos con 30 usuarios y solo 2 fallos.
    - Tiempo aceptable de 600.06ms para la consistencia lograda.

- SERIALIZABLE:
    - Sin inconsistencias, pero rendimiento degradado (649.43ms para 30 usuarios).
    - 1-8 fallos debido a timeouts por bloqueos prolongados.

b. Concurrencia
    - A mayor concurrencia, aumentan los fallos (9 fallos con 30 usuarios en READ COMMITTED).
    - REPEATABLE READ mostró mejor escalabilidad: 28 éxitos contra 22 en SERIALIZABLE, manejando la misma cantidad de usuarios.

c. Deadlocks
    - No se registraron deadlocks gracias al uso de FOR UPDATE en consultas y transacciones cortas.

Problemas Identificados

1. Contención de recursos: En SERIALIZABLE, múltiples transacciones bloquearon filas, causando esperas y fallos.

2. Tiempos de respuesta: SERIALIZABLE fue casi dos veces más lento que READ COMMITTED (574.30ms contra 303.39ms con 5 usuarios).

3. Fallos en alta concurrencia: READ COMMITTED permitió condiciones de carrera (4 fallos con 20 usuarios).

Fase 5

Análisis y reflexiones

La concurrencia en una base de datos es un aspecto importante para asegurar la integridad de datos en un sistema de múltiples usuarios. Se evidenció que los niveles de aislamiento son fundamentales para evitar problemas como las inconsistencias de datos, lecturas sucias y no repetibles. Los resultados mostraron claramente cómo los niveles más estrictos de aislamiento, como SERIALIZABLE, impactan negativamente el rendimiento

(649.43ms para 30 usuarios) debido al aumento de bloqueos y conflictos, aunque garantizan consistencia total. Por otro lado, READ COMMITTED demostró ser el más rápido (303.39ms para 5 usuarios), pero con fallos en alta concurrencia (9 fallos con 30 usuarios) por condiciones de carrera. REPEATABLE READ emergió como el equilibrio ideal, manteniendo 28/30 éxitos con tiempos aceptables (600.06ms).

La programación del sistema fue buena y eficiente por el uso de Go, ya que no se presentaron problemas de deadlocks gracias al uso de transacciones cortas y el bloqueo explícito de filas con FOR UPDATE. Esta estrategia permitió manejar la concurrencia de manera efectiva, aunque en SERIALIZABLE se observaron 8 fallos por timeouts debido a bloqueos prolongados. El diseño cuidadoso de las transacciones garantizó escenarios realistas, donde REPEATABLE READ mostró mejor escalabilidad que SERIALIZABLE en alta carga (28 contra 22 éxitos con 30 usuarios).

Aunque no hubo deadlocks durante las pruebas, el análisis de los resultados reveló varios desafíos importantes en el manejo de la concurrencia. En los niveles de aislamiento más estrictos como REPEATABLE READ y SERIALIZABLE, se observó que los bloqueos extendidos impactaron significativamente los tiempos de respuesta, como se evidencia en la diferencia de 600.06ms frente a 407.20ms al comparar estos niveles con 10 usuarios concurrentes. Por otro lado, el nivel READ COMMITTED mostró sus limitaciones en escenarios de alta demanda, donde la contención de recursos se hizo evidente con 9 fallos registrados al simular 30 usuarios, demostrando así su vulnerabilidad en cuanto a consistencia de datos. Cabe destacar que, si bien Go demostró ser eficiente en el manejo general de la concurrencia, los resultados mostraron variaciones considerables en el rendimiento dependiendo del nivel de aislamiento configurado, lo que subraya la importancia de seleccionar cuidadosamente esta configuración según los requisitos específicos de cada operación.

El uso de Go en el proyecto demostró ser una elección acertada gracias a sus características nativas para manejar concurrencia, particularmente mediante el uso de canales y goroutines que facilitaron significativamente la simulación de múltiples usuarios compitiendo por recursos. La eficiencia del lenguaje se vio reforzada por el paquete database/sql, que permitió establecer conexiones robustas con PostgreSQL y manejar transacciones de forma efectiva. Sin embargo, esta elección tecnológica también presentó desafíos importantes: el manejo explícito de errores, característica inherente de Go, aumentó la complejidad al trabajar con transacciones anidadas y múltiples puntos de fallo potenciales. Además, la integración con PostgreSQL requirió una configuración cuidadosa y un conocimiento profundo de los niveles de aislamiento, lo que añadió una capa adicional de complejidad al desarrollo inicial del proyecto. A pesar de estos retos, las capacidades nativas de Go para concurrencia y su sintaxis clara terminaron por compensar estas dificultades, permitiendo una implementación eficiente del sistema de reservas concurrentes.

Conclusiones

1. REPEATABLE READ ofrece el mejor balance entre rendimiento y consistencia para sistemas con alta concurrencia.
2. READ COMMITTED es adecuado para operaciones donde la velocidad es prioritaria, aunque con riesgos de condiciones de carrera.
3. SERIALIZABLE debe reservarse para casos que exijan consistencia absoluta, a pesar de su impacto en rendimiento.
4. El bloqueo explícito y transacciones cortas son esenciales para prevenir conflictos, como se comprobó con cero deadlocks.
5. Las simulaciones realistas validaron el diseño de la base de datos y revelaron cuellos de botella, como la contención en SERIALIZABLE.

Anexos

Enlace al repositorio de github: https://github.com/bar23354/DB-Proyecto-2.git