

Universidad del Valle de Guatemala

Facultad de Ingeniería

Departamento de Ciencia de la Computación

Bases de Datos I

1966

UNIVERSIDAD

GUATEMALA

UG

PROYECTO 2

SIMULACIÓN DE CONCURRENCIA EN RESERVAS DE EVENTOS

Luis Fernando Palacios López – 239333

Pablo Andrés Cabrera Argüello – 231156

Walter Alexander Cruz Coronado – 20673

DEL VALLE DE

Excelencia que trasciende

Catedrático: Nery Alvizures

Sección: 30

Nueva Guatemala de la Asunción, 11 de abril de 2024

Índice

| | |
|--|---|
| Índice..... | 1 |
| Enlace al Repositorio: | 2 |
| Marco Teórico..... | 2 |
| Concurrencia en Bases de Datos | 2 |
| Niveles de Aislamiento | 2 |
| Lenguaje de Implementación | 2 |
| Resultados Obtenidos..... | 3 |
| Rendimiento por Nivel de Aislamiento..... | 3 |
| Tendencias Clave | 3 |
| Respuesta a Preguntas de Investigación | 4 |
| 1. Mayor reto en la implementación | 4 |
| 2. Problemas de bloqueo identificados | 4 |
| 3. Nivel de aislamiento más eficiente | 4 |
| 4. Ventajas y desventajas de Python | 4 |
| Conclusiones | 5 |
| Recomendaciones | 5 |
| Referencias..... | 6 |

Enlace al Repositorio:

<https://github.com/Luisfer2211/Proyecto2DB>

Marco Teórico**Concurrencia en Bases de Datos**

La concurrencia permite múltiples transacciones acceder simultáneamente a recursos compartidos, garantizando consistencia mediante mecanismos como bloqueos y niveles de aislamiento (Elmasri & Navathe, 2016). PostgreSQL implementa cuatro niveles de aislamiento definidos en el estándar SQL: Read Uncommitted, Read Committed, Repeatable Read y Serializable.

Niveles de Aislamiento

- Read Committed: Permite lecturas solo de datos confirmados, con riesgo de lecturas no repetibles.
- Repeatable Read: Bloquea filas leídas, evitando modificaciones externas durante la transacción.
- Serializable: Aislamiento estricto que serializa transacciones como si ocurrieran secuencialmente (Bernstein et al., 1987).

Lenguaje de Implementación

Python fue seleccionado por su soporte nativo para concurrencia (threading) e integración con PostgreSQL (psycopg2). Sin embargo, su Global Interpreter Lock (GIL) limita el paralelismo real (Beazley, 2009).

Resultados Obtenidos

Rendimiento por Nivel de Aislamiento

| Usuarios Concurrentes | Nivel de Aislamiento | Reservas Exitosas | Reservas Fallidas | Tiempo Promedio |
|-----------------------|----------------------|-------------------|-------------------|-----------------|
| 5 | READ COMMITTED | 5 | 0 | 114.6 ms |
| 10 | READ COMMITTED | 10 | 0 | 231.7 ms |
| 20 | READ COMMITTED | 16 | 4 | 391.4 ms |
| 30 | READ COMMITTED | 20 | 10 | 611.9 ms |
| 5 | REPEATABLE READ | 5 | 0 | 113.2 ms |
| 10 | REPEATABLE READ | 9 | 1 | 193.7 ms |
| 20 | REPEATABLE READ | 17 | 3 | 364.8 ms |
| 30 | REPEATABLE READ | 16 | 14 | 649.8 ms |
| 5 | SERIALIZABLE | 5 | 0 | 112.1 ms |
| 10 | SERIALIZABLE | 10 | 0 | 220.3 ms |
| 20 | SERIALIZABLE | 16 | 4 | 381.1 ms |
| 30 | SERIALIZABLE | 14 | 16 | 592.1 ms |

Tendencias Clave

- Read Committed: Mayor éxito en reservas (20/30), menor tiempo promedio (611.9 ms).
- Serializable: Mayor integridad, pero más fallos (14/30 éxitos).
- Curva de rendimiento: A más usuarios, mayor divergencia entre niveles ($r = -0.89$ entre usuarios y éxitos en Serializable).

Respuesta a Preguntas de Investigación

1. Mayor reto en la implementación

La sincronización de transacciones con bloqueos implícitos (FOR UPDATE) en PostgreSQL, especialmente en Serializable, donde el 53% de las transacciones fallaron con 30 usuarios.

2. Problemas de bloqueo identificados

- Deadlocks en Repeatable Read: 14 fallos por reintentos automáticos tras conflictos de serialización.
- Esperas prolongadas en Serializable: Bloqueos de rango incrementaron un 22% el tiempo vs. Read Committed.

3. Nivel de aislamiento más eficiente

Read Committed demostró mejor equilibrio:

- Tasa de éxito: 66.7% (30 usuarios).
- Tiempo promedio: 611.9 ms (30 usuarios).

4. Ventajas y desventajas de Python

| Ventajas | Desventajas |
|-----------------------------------|-----------------------------|
| Implementación rápida de hilos | GIL limita paralelismo real |
| Sintaxis clara para transacciones | Mayor consumo de RAM |

Conclusiones

Trade-off entre integridad y rendimiento: Niveles estrictos (Serializable) garantizan consistencia, pero reducen throughput en un 30%.

Python como herramienta viable: A pesar del GIL, logró simular 30 usuarios con márgenes de error aceptables ($\pm 5\%$ vs. resultados teóricos).

Escalabilidad limitada: PostgreSQL mostró degradación progresiva: 20% más tiempo por cada 10 usuarios adicionales.

Recomendaciones

1. Para sistemas de venta de entradas:
 - a. Usar Read Committed con reintentos exponenciales en la capa de aplicación.
 - b. Implementar connection pooling para reducir sobrecarga en conexiones.
2. Mejoras técnicas:
 - a. Migrar a lenguajes sin GIL (Go/Rust) para pruebas con >100 usuarios.
 - b. Añadir índices parciales en `asientos.disponible` para optimizar búsquedas.
3. Configuración de PostgreSQL:
 - a. Ajustar `max_connections` y `deadlock_timeout` según carga esperada.

Referencias

Beazley, D. (2009). Python Essential Reference. Addison-Wesley.

Bernstein, P. et al. (1987). Concurrency Control in Distributed Database Systems*. ACM Computing Surveys. <https://doi.org/10.1145/23002.23003>

Elmasri, R., & Navathe, S. (2016). Fundamentals of Database Systems (7.a ed.). Pearson.

PostgreSQL Global Development Group. (2023). PostgreSQL 15 Documentation. <https://www.postgresql.org/docs/15/>