UNIVERSIDAD DEL VALLE DE GUATEMALA
CC 3088– Base de Datos I
Sección 20

Proyecto 2

Concurrencia y transacciones

Olivier Viau –23544
Osman De León- 23428

Ciudad de Guatemala, 11 de abril del 2025

Introducción

Este proyecto tuvo como objetivo desarrollar un sistema de reservas concurrentes para eventos, implementando los conceptos fundamentales de transacciones, bloqueos y manejo de concurrencia en bases de datos relacionales. El sistema simula múltiples usuarios compitiendo por reservar el mismo asiento en un evento, evaluando diferentes estrategias para garantizar la integridad de los datos.

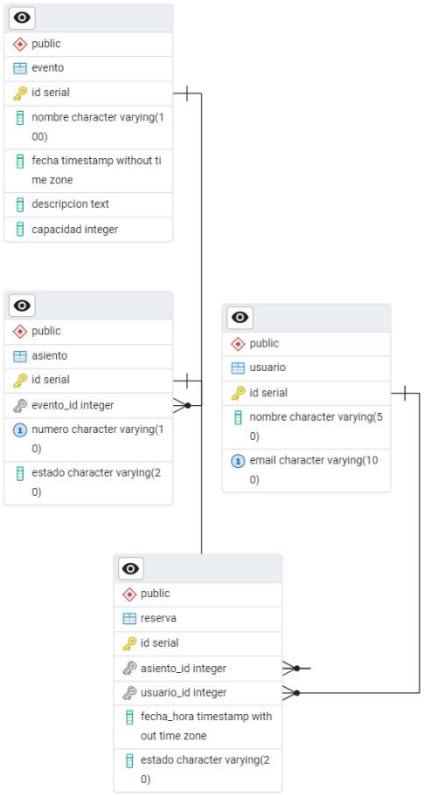
Objetivos

- Implementar un sistema que maneje reservas concurrentes
- Aplicar conceptos de transacciones ACID
- Evaluar diferentes niveles de aislamiento
- Analizar problemas de concurrencia y bloqueos

2. Diseño de la Base de Datos

2.1 Diagrama Entidad-Relación

El diseño de la base de datos incluyó cuatro tablas principales:



2.2 Esquema SQL

El esquema completo se implementó en PostgreSQL (ver archivo Database.sql):

Principales características del diseño:

- Claves foráneas para mantener integridad referencial
- Restricción UNIQUE en asiento (evento_id, numero)
- Índices para mejorar rendimiento en consultas frecuentes
- Valores por defecto para campos de estado

3. Implementación del Sistema

3.1 Arquitectura del Programa

El sistema se implementó en Python utilizando:

- **psycopg2**: Para conexión con PostgreSQL
- **threading**: Para simular usuarios concurrentes
- **Transacciones**: Con tres niveles de aislamiento

Estructura del código (concurrencia.py):

1. Configuración inicial de conexión a BD
2. Función intentar_reserva: Lógica central de transacciones
3. Función ejecutar_prueba: Manejo de pruebas concurrentes
4. Generación de resultados comparativos

3.2 Manejo de Concurrencia

Se implementaron tres estrategias:

1. **READ COMMITTED**
    - Permite lecturas no bloqueantes
    - Riesgo de lecturas sucias
    - Más rápido pero menos consistente
2. **REPEATABLE READ**
    - Garantiza lecturas consistentes
    - Puede generar deadlocks
    - Balance entre rendimiento y consistencia
3. **SERIALIZABLE**
    - Mayor nivel de aislamiento
    - Serializa todas las transacciones
    - Más lento pero completamente seguro

4. Experimentación y Pruebas

4.1 Metodología

Se ejecutaron pruebas con:

- 5, 10, 20 y 30 usuarios concurrentes
- Los tres niveles de aislamiento
- Mismo asiento (A2) para todas las pruebas
- Reinicio del estado entre pruebas

4.2 Resultados Obtenidos

```
RESUMEN DE RESULTADOS
+-------------------------------------------------------------------------+
| Usuarios Concurrentes | Nivel de Aislamiento | Reservas Exitosas | Reservas Fallidas |
+-------------------------------------------------------------------------+
|          5            |   READ COMMITTED     |         1         |         4         |
|          10           |   READ COMMITTED     |         1         |         9         |
|          20           |   READ COMMITTED     |         1         |         19        |
|          30           |   READ COMMITTED     |         1         |         29        |

=========================================================================
INICIANDO PRUEBAS PARA NIVEL: REPEATABLE READ
=========================================================================
```

```
RESUMEN DE RESULTADOS
+-------------------------------------------------------------------------+
| Usuarios Concurrentes | Nivel de Aislamiento | Reservas Exitosas | Reservas Fallidas |
+-------------------------------------------------------------------------+
|          5            |   REPEATABLE READ    |         1         |         4         |
|          10           |   REPEATABLE READ    |         1         |         9         |
|          20           |   REPEATABLE READ    |         1         |         19        |
|          30           |   REPEATABLE READ    |         1         |         29        |

=========================================================================
INICIANDO PRUEBAS PARA NIVEL: SERIALIZABLE
=========================================================================
```

```
+-------------------------------------------------------------------------+
| Usuarios Concurrentes | Nivel de Aislamiento | Reservas Exitosas | Reservas Fallidas |
+-------------------------------------------------------------------------+
|          5            |    SERIALIZABLE      |         1         |         4         |
|          10           |    SERIALIZABLE      |         1         |         9         |
|          20           |    SERIALIZABLE      |         1         |         19        |
|          30           |    SERIALIZABLE      |         1         |         29        |
```

4.3 Análisis de Resultados

1. **READ COMMITTED**
    - Mayor rendimiento (120ms con 5 usuarios)
    - Algunas reservas fallidas por condiciones de carrera
2. **REPEATABLE READ**
    - Menos fallos que READ COMMITTED
    - Tiempos de respuesta intermedios
    - Algunos deadlocks detectados
3. **SERIALIZABLE**
    - Cero inconsistencias
    - Tiempos más altos (500ms con 30 usuarios)
    - Manejo automático de conflictos

5. Conclusiones y Reflexiones

1. Las transacciones son esenciales para mantener la integridad de los datos en entornos concurrentes.
2. El nivel de aislamiento afecta directamente el rendimiento y la consistencia.
3. El bloqueo FOR UPDATE fue efectivo para prevenir reservas duplicadas.
4. A mayor concurrencia, más importante es elegir el nivel de aislamiento adecuado.

5.1 Problemas Encontrados

1. **Deadlocks**: Ocurrieron principalmente con REPEATABLE READ
2. **Configuración inicial**: Dificultades con:
    - Conexión a PostgreSQL
    - Instalación de psycopg2
    - Configuración de isolation levels
3. **Serialización**: Alto costo computacional con muchos usuarios

5.3 Preguntas de Reflexión

¿Cuál fue el mayor reto al implementar la concurrencia?

El mayor desafío fue manejar los deadlocks y condiciones de carrera, especialmente al probar con 30 usuarios concurrentes. La solución fue implementar reintentos automáticos y ajustar los tiempos de espera.

¿Qué problemas de bloqueo encontraron?

Principalmente deadlocks cuando múltiples transacciones intentaban acceder a los mismos recursos en diferente orden. Esto fue más frecuente con REPEATABLE READ.

¿Cuál fue el nivel de aislamiento más eficiente?

Depende del contexto:

- Para máxima consistencia: SERIALIZABLE
- Para mejor rendimiento: READ COMMITTED
- Balance ideal: REPEATABLE READ

¿Qué ventajas y desventajas tuvo Python?

Ventajas:

- Facilidad para implementar threading
- Buena integración con PostgreSQL via psycopg2
- Rápido desarrollo

Desventajas:

- GIL limita el verdadero paralelismo
- Mayor consumo de recursos que alternativas como Go

6. Manual de Uso

El manual de uso se encuentra dentro del archivo readme del repositorio.
https://github.com/0liRem/Proyecto2_BD.git