Understanding Transactions and Locking in PostgreSQL: Un análisis práctico y teórico

KEVIN DENILSON CAX COC, Universidad Mariano Gálvez de Guatemala, Guatemala

1. Resumen

PostgreSQL implementa un modelo robusto de control de concurrencia basado en *multiversion concurrency control* (MVCC) combinado con mecanismos de bloqueo implícitos y explícitos. Este artículo presenta un análisis técnico de la gestión de transacciones, los niveles de aislamiento y el comportamiento de los bloqueos en PostgreSQL. Mediante fundamentos teóricos y experimentos reproducibles en SQL, se analizan escenarios de conflicto, *deadlocks* y el funcionamiento de *Serializable Snapshot Isolation* (SSI). El estudio demuestra cómo la configuración del nivel de aislamiento impacta directamente en la consistencia, el rendimiento y la escalabilidad del sistema de base de datos.

2. Introducción

Los sistemas modernos de bases de datos deben equilibrar consistencia, disponibilidad y rendimiento en entornos altamente concurrentes. PostgreSQL aborda este desafío mediante el uso de MVCC y un administrador de transacciones sofisticado que evita los bloqueos innecesarios entre lectores y escritores.

Las transacciones garantizan las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Sin embargo, cuando múltiples transacciones se ejecutan simultáneamente, pueden surgir conflictos que comprometan la integridad lógica del sistema. Por ello, los mecanismos de aislamiento son fundamentales para prevenir anomalías como lecturas sucias, lecturas no repetibles, lecturas fantasma y conflictos de escritura.

El objetivo de este trabajo es analizar de forma teórica y práctica cómo PostgreSQL implementa los niveles de aislamiento y los mecanismos de bloqueo, haciendo énfasis en sus diferencias observables mediante experimentos reproducibles.

3. Fundamentos teóricos

3.1. Propiedades ACID

PostgreSQL cumple con el modelo ACID, que constituye la base de la confiabilidad transaccional en bases de datos relacionales:

- **Atomicidad:** Todas las operaciones de una transacción se completan o ninguna se aplica. El comando `ROLLBACK` revierte todos los cambios pendientes.
- **Consistencia:** La base de datos permanece en un estado válido tras cada transacción, respetando restricciones e invariantes definidos.
- **Aislamiento:** Las transacciones concurrentes no interfieren incorrectamente; el resultado debe ser equivalente a una ejecución serial.
- **Durabilidad:** Los cambios confirmados con `COMMIT` persisten incluso ante fallos del sistema gracias al *Write-Ahead Log* (WAL).

Author's Contact Information: Kevin Denilson Cax Coc, Universidad Mariano Gálvez de Guatemala, Guatemala, Guatemala, kcaxc@miumg.edu.gt.

3.2. Control de Concurrencia MVCC

MVCC (*Multiversion Concurrency Control*) permite que múltiples transacciones lean datos sin bloquearse entre sí. En lugar de sobrescribir filas, PostgreSQL crea nuevas versiones durante las actualizaciones. Cada fila contiene los campos `xmin` y `xmax` que determinan su visibilidad según el *snapshot* de cada transacción.

Las versiones obsoletas se eliminan periódicamente mediante el proceso `VACUUM`, que recupera el espacio físico y actualiza los mapas de visibilidad. Una configuración inadecuada de `autovacuum` puede provocar *table bloat* y degradación del rendimiento en entornos de alta escritura.

3.3. Niveles de Aislamiento

El estándar SQL define cuatro niveles de aislamiento; PostgreSQL implementa tres de forma efectiva y garantiza propiedades más fuertes que las exigidas por el estándar. La Tabla 1 resume las anomalías prevenidas por cada nivel.

Cuadro 1. Niveles de aislamiento y anomalías en PostgreSQL

Nivel	Lectura sucia	No repetible	Fantasma
Read Committed	Imposible	Posible	Posible
Repeatable Read	Imposible	Imposible	Imposible
Serializable	Imposible	Imposible	Imposible

* PostgreSQL previene lecturas fantasma en Repeatable Read mediante MVCC.

Desde un punto de vista formal, una ejecución es serializable si el grafo de dependencias entre transacciones (grafo de precedencia) no contiene ciclos. PostgreSQL implementa *Serializable Snapshot Isolation* (SSI): las transacciones operan sobre snapshots consistentes, pero el sistema monitorea dependencias lectura-escritura (*rw-dependencies*). Si se detecta una estructura peligrosa que podría generar un ciclo, el sistema aborta una transacción para preservar la serialización.

3.4. Mecanismos de Bloqueo

PostgreSQL utiliza dos tipos de bloqueos: implícitos y explícitos.

Los **bloqueos implícitos** son adquiridos automáticamente por el motor durante operaciones DML (`INSERT`, `UPDATE`, `DELETE`). Por ejemplo, una sentencia `UPDATE` adquiere un bloqueo `ROW EXCLUSIVE` sobre la tabla y un bloqueo a nivel de fila sobre las tuplas modificadas.

Los **bloqueos explícitos** se solicitan mediante el comando `LOCK TABLE`, que acepta distintos modos según la granularidad requerida: `ACCESS SHARE`, `ROW EXCLUSIVE`, `SHARE ROW EXCLUSIVE` y `ACCESS EXCLUSIVE`, entre otros. La opción `NOWAIT` indica que la transacción debe fallar inmediatamente si no puede adquirir el bloqueo en lugar de esperar.

4. Ejemplos prácticos

Todos los experimentos se ejecutaron sobre la siguiente tabla base:

Listing 1. Tabla de pruebas

```
CREATE TABLE cuentas (
    id     SERIAL PRIMARY KEY,
    nombre VARCHAR(50),
```

```
    saldo  NUMERIC(12,2)
);
INSERT INTO cuentas (nombre, saldo) VALUES
    ('Alice', 1000.00),
    ('Bob',   2000.00),
    ('Carol',  500.00);
```

4.1. Uso de SAVEPOINT

`SAVEPOINT` permite crear puntos de control intermedios dentro de una transacción. `ROLLBACK TO SAVEPOINT` revierte únicamente las operaciones posteriores al punto, conservando las anteriores.

Listing 2. Transacción con SAVEPOINT

```
BEGIN;
  UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

  SAVEPOINT sp1;  -- punto tras operacion valida

  -- Operacion erronea (saldo negativo)
  UPDATE cuentas SET saldo = saldo - 5000 WHERE id = 1;

  -- Revertir solo la operacion erronea
  ROLLBACK TO SAVEPOINT sp1;

  -- Verificar: saldo de Alice = 900.00
  SELECT nombre, saldo FROM cuentas WHERE id = 1;
COMMIT;
```

El resultado observable es que Alice conserva un saldo de $900.00; la operación errónea queda descartada sin cancelar la transacción completa.

4.2. Bloqueo explícito con NOWAIT

La opción `NOWAIT` evita que una transacción quede suspendida indefinidamente cuando otra sesión retiene un bloqueo incompatible.

Listing 3. LOCK con NOWAIT — Sesión A

```
-- Sesion A: adquiere el bloqueo
BEGIN;
LOCK TABLE cuentas IN SHARE ROW EXCLUSIVE MODE;
-- Simula trabajo prolongado; no hace COMMIT aun
```

Listing 4. LOCK con NOWAIT — Sesión B

```
-- Sesion B: intenta el mismo bloqueo con NOWAIT
BEGIN;
```

```
LOCK TABLE cuentas IN SHARE ROW EXCLUSIVE MODE NOWAIT;
-- Resultado esperado:
-- ERROR: could not obtain lock on relation "cuentas"
ROLLBACK;
```

Sin `NOWAIT`, la Sesión B habría permanecido bloqueada hasta que la Sesión A liberara el lock mediante `COMMIT` o `ROLLBACK`. `NOWAIT` permite a la aplicación implementar lógica de reintento o notificación de error de forma controlada.

4.3. READ COMMITTED versus SERIALIZABLE

READ COMMITTED genera un nuevo snapshot por cada sentencia `SELECT`, lo que permite que dos lecturas dentro de la misma transacción devuelvan valores distintos si otra sesión confirmó cambios entre medias (lectura no repetible).

Listing 5. Lectura no repetible en READ COMMITTED

```
-- Sesion A
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT saldo FROM cuentas WHERE id = 1;
-- Resultado: 1000.00

  -- (Sesion B ejecuta y confirma):
  -- UPDATE cuentas SET saldo = 9999 WHERE id = 1; COMMIT;

SELECT saldo FROM cuentas WHERE id = 1;
-- Resultado: 9999.00  <- valor modificado por B
COMMIT;
```

SERIALIZABLE detecta dependencias peligrosas y aborta transacciones que romperían la serialización:

Listing 6. Conflicto de serialización

```
-- Sesion A
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT SUM(saldo) FROM cuentas;          -- lee suma total
UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

-- (Sesion B concurrente hace su propio SELECT y UPDATE y hace COMMIT)

COMMIT;
-- ERROR: could not serialize access due to concurrent update
```

4.4. REPEATABLE READ

En este nivel, el snapshot se fija al inicio de la transacción, garantizando que lecturas repetidas devuelvan siempre el mismo resultado.

Listing 7. Snapshot estable en REPEATABLE READ

```
-- Sesion A
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT saldo FROM cuentas WHERE id = 1;
-- Resultado: 1000.00

  -- (Sesion B: UPDATE cuentas SET saldo = 9999 WHERE id = 1; COMMIT;)

SELECT saldo FROM cuentas WHERE id = 1;
-- Resultado: 1000.00  <- mismo valor; snapshot no cambia
COMMIT;
```

4.5. Simulación de Deadlock

Un *deadlock* ocurre cuando dos transacciones se esperan mutuamente. PostgreSQL lo detecta automáticamente y aborta una de ellas.

Listing 8. Deadlock entre dos sesiones

```
-- Sesion A: bloquea fila id=1
BEGIN;
UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

-- Sesion B: bloquea fila id=2
BEGIN;
UPDATE cuentas SET saldo = saldo - 50 WHERE id = 2;

-- Sesion A intenta bloquear fila id=2 -> espera a B
UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;

-- Sesion B intenta bloquear fila id=1 -> ciclo detectado
UPDATE cuentas SET saldo = saldo + 50 WHERE id = 1;
-- ERROR: deadlock detected
-- DETAIL: Process X waits for ShareLock on transaction Y;
--         blocked by process Z.
```

PostgreSQL aborta la transacción que provocó el ciclo y la otra puede continuar. La transacción abortada debe reiniciarse desde el `BEGIN`.

5. Discusión

5.1. Ventajas y riesgos por nivel de aislamiento

READ COMMITTED ofrece alto *throughput* porque cada sentencia trabaja con datos frescos. Es adecuado para aplicaciones con lecturas frecuentes y baja contención de escritura. Su principal riesgo es la lectura no repetible: si la lógica de negocio asume que dos lecturas de la misma fila dentro de una transacción son idénticas, puede tomar decisiones erróneas.

REPEATABLE READ estabiliza el snapshot durante toda la transacción, lo que es ideal para reportes y consultas analíticas que requieren coherencia. Sin embargo, una transacción larga con este nivel acumula versiones de fila que VACUUM no puede recolectar, incrementando el uso de disco.

SERIALIZABLE garantiza correctitud formal mediante SSI. A diferencia del bloqueo estricto en dos fases, PostgreSQL detecta dinámicamente estructuras peligrosas en el grafo de dependencias, lo que mejora la concurrencia respecto a modelos tradicionales. El costo es un incremento en la tasa de abortos bajo alta contención; las aplicaciones deben implementar lógica de reintento robusta.

5.2. Impacto en rendimiento y concurrencia

El uso excesivo de `LOCK TABLE ... IN ACCESS EXCLUSIVE MODE` anula las ventajas de MVCC al serializar completamente el acceso, degradando el paralelismo. Se recomienda preferir bloqueos a nivel de fila (implícitos mediante DML) sobre bloqueos de tabla.

La opción `NOWAIT` es útil en arquitecturas que requieren *fail-fast*: en lugar de acumular conexiones bloqueadas que saturan el *pool*, la aplicación recibe un error inmediato y puede reintentar o escalar el evento.

Finalmente, MVCC requiere una configuración adecuada de `autovacuum` para evitar acumulación de versiones obsoletas. En entornos OLTP con alta frecuencia de escritura, parámetros como `autovacuum_vacuum_cost_delay` y `autovacuum_vacuum_scale_factor` deben ajustarse para mantener la salud de las tablas.

6. Conclusiones

PostgreSQL combina MVCC, bloqueo selectivo y detección dinámica de conflictos para ofrecer un modelo sólido de concurrencia. Los experimentos realizados confirman que:

- `SAVEPOINT` permite recuperación parcial dentro de una transacción sin cancelarla por completo, facilitando flujos de negocio complejos.
- `LOCK ... NOWAIT` es una herramienta práctica para evitar esperas indefinidas y diseñar aplicaciones resistentes a fallos.
- **READ COMMITTED** proporciona un equilibrio adecuado para la mayoría de aplicaciones empresariales, pero exige cuidado cuando la lógica requiere lecturas estables.
- **REPEATABLE READ** es ideal para procesos analíticos de larga duración que necesitan vistas coherentes de los datos.
- **SERIALIZABLE** es indispensable cuando se requieren garantías formales de consistencia, a costa de una mayor tasa de abortos que la aplicación debe manejar.

El conocimiento profundo del comportamiento de bloqueos, niveles de aislamiento y detección de *deadlocks* es esencial para diseñar sistemas de bases de datos robustos, escalables y confiables en entornos de producción.

7. Referencias

PostgreSQL Global Development Group. *LOCK — lock a table*. Documentación oficial de PostgreSQL, 2024. https://www.postgresql.org/docs/current/sql-lock.html

PostgreSQL Global Development Group. *Transaction Isolation*. Documentación oficial de PostgreSQL, 2024. https://www.postgresql.org/docs/current/transaction-iso.html

PostgreSQL Global Development Group. *Multiversion Concurrency Control*. Documentación oficial de PostgreSQL, 2024. https://www.postgresql.org/docs/current/mvcc.html

M. J. Cahill, U. Röhm, and A. D. Fekete. *Serializable isolation for snapshot databases*. ACM Transactions on Database Systems, 34(4):1–42, 2009. https://doi.org/10.1145/1620585.1620587

PostgreSQL Global Development Group. *Routine Vacuuming*. Documentación oficial de PostgreSQL, 2024. https://www.postgresql.org/docs/current/routine-vacuuming.html

Anexo

El código completo utilizado para este trabajo se encuentra disponible públicamente en el siguiente repositorio:

- **Repositorio GitHub:** https://github.com/KevinCax/Transactions-DB-II.git

El repositorio incluye los scripts SQL correspondientes a:

- Creación y eliminación de la tabla `cuentas`.
- Inserción de datos iniciales para pruebas controladas.
- Uso de transacciones con `BEGIN` y `COMMIT`.
- Implementación de `SAVEPOINT` y `ROLLBACK TO SAVEPOINT`.
- Bloqueos explícitos con `LOCK TABLE` en diferentes modos.
- Pruebas bajo los niveles de aislamiento:
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- Simulación de lecturas no repetibles.
- Restauración de estados base para garantizar reproducibilidad.