INFORME

Optimización y Performance Engineering en PostgreSQL

Proyecto EAFITShop – Bases de Datos Avanzadas

Estudiantes: Daniel Arcila

Juan Simón Ospina

Sebastián Durán

Repositorio del proyecto:

https://github.com/darcilas1/bd-avanzadas-proyecto1-eafitshop.git

1. Introducción

El rendimiento de una base de datos relacional depende del diseño físico, la calidad de las consultas, la indexación, la configuración interna del motor y el manejo de concurrencia. En este proyecto se desarrolló un análisis integral de performance sobre PostgreSQL utilizando un dataset de alto volumen que simula un entorno empresarial tipo e-commerce.

2. Objetivo del Trabajo

Diagnosticar y optimizar el rendimiento del sistema EAFITShop aplicando técnicas reales de ingeniería de performance, comparando métricas antes y después de cada intervención mediante EXPLAIN (ANALYZE, BUFFERS).

3. Descripción del Caso

El sistema EAFITShop contiene tablas orders, order_item, payment, product y customer con millones de registros. Se analizaron seis consultas representativas que incluyen agregaciones masivas, joins de alta cardinalidad, filtros temporales, búsquedas por cliente y procesamiento concurrente.

4. Situación Real Empresarial Similar

Un caso empresarial real comparable al desarrollado en el proyecto EAFITShop se presenta en plataformas de comercio electrónico de gran escala como Mercado Libre y Shopify, las cuales manejan millones de transacciones diarias y utilizan PostgreSQL como uno de sus motores principales de almacenamiento transaccional.

Caso documentado: Shopify y el escalamiento de PostgreSQL

Shopify ha documentado públicamente desafíos relacionados con el crecimiento acelerado de tablas de órdenes, consultas analíticas sobre millones de registros, agregaciones masivas, alta concurrencia en procesamiento de pagos y problemas de spill a disco cuando la configuración de memoria no es adecuada.

Entre los problemas identificados se encontraban:

1. Full table scans en tablas de órdenes históricas.

2. Agregaciones que saturaban memoria (work_mem insuficiente).

3. Bloqueos bajo alta concurrencia en procesamiento de pedidos.

4. Consultas no sargables que impedían el uso de índices.

Las soluciones implementadas incluyeron:

• Creación de índices compuestos sobre columnas de filtrado frecuente.

• Implementación de particionamiento por rango temporal.

• Ajuste de parámetros como shared_buffers, work_mem y effective_cache_size.

• Reescritura de consultas para eliminar funciones sobre columnas indexadas.

• Uso de mecanismos de control de concurrencia para evitar condiciones de carrera.

Relación con el Proyecto EAFITShop

El proyecto desarrollado reproduce experimentalmente estos mismos patrones: se observaron Sequential Scans en consultas de órdenes, spill a disco en agregaciones masivas, problemas de sargabilidad con date_trunc() y condiciones de carrera en procesamiento concurrente. Las soluciones aplicadas —indexación estratégica, índices parciales, particionamiento, reescritura de queries y uso de FOR UPDATE SKIP LOCKED— reflejan prácticas reales utilizadas en entornos empresariales.

Esto demuestra que el laboratorio no es un ejercicio aislado, sino una simulación técnica de desafíos reales en sistemas de alta carga transaccional, donde la optimización debe abordarse desde un enfoque integral que combine diseño físico, tuning del motor y control de concurrencia.

4. Ambiente Tecnológico

Infraestructura: AWS EC2 t2.large

Motor: PostgreSQL ejecutado en contenedor Docker

Dataset: Big (millones de registros)

Herramientas: EXPLAIN (ANALYZE, BUFFERS), pg_stat_activity, pg_settings

5. Diagnóstico Inicial (Baseline)

Q1 – 3221 ms: Parallel Seq Scan + Hash Join. Alto I/O.

Q2 – 16745 ms: HashAggregate con spill a disco (~101MB).

Q3 – 1312 ms: Seq Scan + Sort + Limit.

Q5 – 1632 ms: Uso de date_trunc() impide uso de índices.

Q6 – 6255 ms: Join con alta cardinalidad antes de filtrado efectivo.

6. Optimización por Índices

Q1: 3221 ms → 1735 ms (↓46%) mediante Index Only Scan.

Q3: 1312 ms → 0.048 ms (↓99.996%) con índice compuesto.

Q6: 6255 ms → 4322 ms (↓31%) con índice parcial.

Q2: Sin mejora significativa debido a baja selectividad.

Conclusión: La indexación estratégica reduce drásticamente el I/O y elimina operaciones costosas.

7. Particionamiento por Rango

Se particionó la tabla orders por año. Se observó partition pruning efectivo en consultas con filtro temporal, pero sin mejora cuando la consulta no incluye la clave de partición.

8. Reescritura de Queries

La reescritura de la Query 5 eliminando date_trunc() permitió volver la condición sargable. Resultado: 1281 ms → 457 ms (~2.8x más rápido).

9. Concurrencia

Baseline: race condition y error 23505 (duplicate key).

FOR UPDATE: garantiza consistencia pero genera espera.

FOR UPDATE SKIP LOCKED: consistencia sin contención y mayor throughput.

10. Performance Tuning

Parámetros ajustados a nivel sesión:

SET work_mem = '64MB';

SET effective_cache_size = '6GB';

Mejoras observadas en Q1 (7.7%), Q3 (20.9%) y Q6 (10.9%).

El tuning mejora decisiones del planner y reduce spill, pero no reemplaza rediseño estructural.

11. Análisis Comparativo Global

TIEMPO TOMADO EN MILISEGUNDOS (ms)

	INDICES		PARTICIONAMIENTO		REESCRITURA DE QUERIES	
	Sin Optimizar	Optimizado	Sin Optimizar	Optimizado	Sin Optimizar	Optimizado
Q1	3835	**1735**	1735	**2048**	2048	**2048**
Q2	17585	**16644**	16644	**16624**	16624	**16624**
Q3	1744.5	**0.048**	0.048	**1155**	1155	**1155**
Q4	3.138	**1.643**	1.643	**2.183**	2.183	**2.183**
Q5	1687.5	**2493**	2493	**1281**	1281	**457**
Q6	7206	**4322**	4322	**4425**	4425	**4249**

CONCURRENCIA		
Métrica	**Sin Optimizar**	**Optimizado**
Sesiones exitosas (5 concurrentes)	1/5	5/5
Errores por duplicidad	4	0
Transacciones abortadas	4	0
Espera por lock	No control	No
Consistencia	Race condition	Garantizada

PERFORMANCE TUNNING			
Query	**Antes**	**Despues**	**% Mejora**
Q1	3814.650	3521.206	+7.69%
Q2	17630.707	17895.374	-1.50% (empeoró)
Q3	1835.204	1451.509	+20.91%
Q4	1.640	2.422	- 47.68% (empeoró)
Q5	349.393	333.963	+4.42%
Q6	5264.089	4689.423	+10.92%

12. Líneas Futuras de Optimización

Persistir parámetros en postgresql.conf.

Evaluar shared_buffers, random_page_cost y maintenance_work_mem.

Explorar materialized views y paralelismo avanzado.

Realizar pruebas de carga con múltiples corridas y promedios.

13. Declaración de Uso de ChatGPT

ChatGPT fue utilizado como herramienta de apoyo para interpretación conceptual y estructuración del informe. También fue utilizada para tomar ideas de cómo implementar la concurrencia y el performance tunning. Las mediciones y validaciones fueron realizadas directamente en el entorno PostgreSQL del laboratorio.

14. Conclusión General

La optimización en PostgreSQL requiere un enfoque integral que combine diseño físico, sargabilidad, indexación estratégica, control de concurrencia y tuning de memoria. El proyecto demostró mejoras significativas mediante intervenciones medibles y basadas en evidencia.