Índices y Estrategias de Optimización de Consultas - Día 2

Change Status

pending 40 min

Learning Objectives

1. Seleccionar índices apropiados para diferentes patrones de consulta analítica
2. Interpretar planes de ejecución para identificar cuellos de botella
3. Implementar estrategias de particionamiento efectivas para datasets grandes

Practical exercise to apply the concepts learned.

Ejercicio: Optimización completa de consultas analíticas en base de datos dimensional

Configuración de base de datos y carga de datos de ejemplo:

```
-- Crear esquema dimensional optimizado
CREATE TABLE dim_tiempo (
    id SERIAL PRIMARY KEY,
    fecha DATE UNIQUE,
    año INTEGER,
    mes INTEGER,
    trimestre INTEGER,
    dia_semana VARCHAR(10)
);

CREATE TABLE dim_cliente (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    segmento VARCHAR(20),
    region VARCHAR(50)
);

CREATE TABLE hechos_ventas (
    id SERIAL PRIMARY KEY,
    id_tiempo INTEGER REFERENCES dim_tiempo(id),
    id_cliente INTEGER REFERENCES dim_cliente(id),
    total_venta DECIMAL(10,2),
    cantidad INTEGER,
    margen DECIMAL(5,2)
);

-- Generar datos de ejemplo (100,000 ventas)
INSERT INTO dim_tiempo (fecha, año, mes, trimestre, dia_semana)
SELECT
    fecha,
    EXTRACT(YEAR FROM fecha),
    EXTRACT(MONTH FROM fecha),
    EXTRACT(QUARTER FROM fecha),
    TO_CHAR(fecha, 'Day')
FROM generate_series('2023-01-01'::date, '2024-12-31'::date, '1 day') as fecha;

-- Insertar datos de ventas (simulado)
-- Nota: En producción usar COPY o INSERT masivo
```

Análisis de consultas sin optimización:

```
-- Consulta analítica típica SIN optimización
EXPLAIN ANALYZE
SELECT
    dt.año,
    dt.trimestre,
    dc.segmento,
    COUNT(*) as num_ventas,
    SUM(hv.total_venta) as ventas_total,
    AVG(hv.margen) as margen_promedio
FROM hechos_ventas hv
JOIN dim_tiempo dt ON hv.id_tiempo = dt.id
JOIN dim_cliente dc ON hv.id_cliente = dc.id
WHERE dt.año = 2024
  AND dc.segmento IN ('VIP', 'Premium')
  AND hv.total_venta > 100
GROUP BY dt.año, dt.trimestre, dc.segmento
ORDER BY dt.año, dt.trimestre, SUM(hv.total_venta) DESC;

-- Resultado típico SIN índices:
```

```
-- Execution time: ~5000ms
-- Plan: Sequential Scan on hechos_ventas (cost=10000.00..50000.00 rows=50000)
--       Hash Join, Nested Loop, etc.
```

Implementación de índices estratégicos:

```
-- Crear índices para optimizar la consulta analítica
CREATE INDEX idx_hechos_tiempo ON hechos_ventas(id_tiempo);
CREATE INDEX idx_hechos_cliente ON hechos_ventas(id_cliente);
CREATE INDEX idx_tiempo_año ON dim_tiempo(año);
CREATE INDEX idx_cliente_segmento ON dim_cliente(segmento);
CREATE INDEX idx_hechos_venta_total ON hechos_ventas(total_venta);

-- Índice compuesto para consulta específica
CREATE INDEX idx_hechos_analisis ON hechos_ventas(id_tiempo, id_cliente, total_venta);

-- Verificar que los índices existen
SELECT schemaname, tablename, indexname, indexdef
FROM pg_indexes
WHERE tablename IN ('hechos_ventas', 'dim_tiempo', 'dim_cliente')
ORDER BY tablename, indexname;
```

Análisis de consulta optimizada:

```
-- Re-ejecutar consulta CON optimización
EXPLAIN ANALYZE
SELECT
    dt.año,
    dt.trimestre,
    dc.segmento,
    COUNT(*) as num_ventas,
    SUM(hv.total_venta) as ventas_total,
    AVG(hv.margen) as margen_promedio
FROM hechos_ventas hv
JOIN dim_tiempo dt ON hv.id_tiempo = dt.id
JOIN dim_cliente dc ON hv.id_cliente = dc.id
WHERE dt.año = 2024
  AND dc.segmento IN ('VIP', 'Premium')
  AND hv.total_venta > 100
GROUP BY dt.año, dt.trimestre, dc.segmento
ORDER BY dt.año, dt.trimestre, SUM(hv.total_venta) DESC;

-- Resultado esperado CON índices:
-- Execution time: ~50ms (100x más rápido)
-- Plan: Index Scan, Bitmap Index Scan, Hash Join optimizado
```

Implementación de particionamiento para escalabilidad:

```
-- Particionamiento por rangos para datos históricos
-- (Requiere recrear tabla - en producción planificar cuidadosamente)

-- Estrategia de particionamiento propuesta:
-- 1. Particionar hechos_ventas por id_tiempo (rangos mensuales)
-- 2. Subparticionar por hash de id_cliente para distribución uniforme
-- 3. Mantener particiones de los últimos 24 meses activas
-- 4. Archivar particiones más antiguas a storage económico

-- Script de particionamiento (PostgreSQL)
DO $$
DECLARE
    fecha_inicio DATE := '2023-01-01';
    fecha_fin DATE := '2024-12-31';
    mes_actual DATE;
BEGIN
    -- Crear particiones mensuales
    mes_actual := fecha_inicio;
    WHILE mes_actual <= fecha_fin LOOP
        EXECUTE format('
            CREATE TABLE IF NOT EXISTS hechos_ventas_y%sm%m PARTITION OF hechos_ventas
            FOR VALUES FROM (%L) TO (%L)',
            EXTRACT(YEAR FROM mes_actual),
            EXTRACT(MONTH FROM mes_actual),
            mes_actual,
            mes_actual + INTERVAL '1 month'
        );
        mes_actual := mes_actual + INTERVAL '1 month';
    END LOOP;
END $$;

-- Verificar particiones creadas
SELECT tablename, pg_size_pretty(pg_total_relation_size(tablename))
FROM pg_tables
WHERE tablename LIKE 'hechos_ventas_y%'
ORDER BY tablename;
```

Comparación de performance y recomendaciones:

```
-- Crear vista materializada para consultas muy frecuentes
CREATE MATERIALIZED VIEW mv_ventas_mensuales AS
SELECT
    dt.año,
    dt.mes,
    dc.segmento,
    COUNT(*) as num_ventas,
    SUM(hv.total_venta) as ventas_total,
    AVG(hv.margen) as margen_promedio
FROM hechos_ventas hv
JOIN dim_tiempo dt ON hv.id_tiempo = dt.id
JOIN dim_cliente dc ON hv.id_cliente = dc.id
GROUP BY dt.año, dt.mes, dc.segmento;

-- Índice en vista materializada
CREATE INDEX idx_mv_mensual ON mv_ventas_mensuales(año, mes, segmento);

-- Comparación de performance:
-- Consulta directa: ~50ms (con índices)
-- Vista materializada: ~5ms (precalculada)
-- Beneficio: 10x más rápido para consultas repetitivas

-- Recomendaciones de mantenimiento:
-- 1. Reindexar índices mensualmente: REINDEX INDEX CONCURRENTLY index_name;
-- 2. Actualizar estadísticas: ANALYZE hechos_ventas;
-- 3. Monitorear uso de índices: SELECT * FROM pg_stat_user_indexes;
-- 4. Refrescar vistas materializadas: REFRESH MATERIALIZED VIEW CONCURRENTLY mv_ventas_mensuales;
```

Verificación: Explica cómo los índices y el particionamiento transforman una consulta que podría tardar minutos en una que se ejecuta en milisegundos, y describe escenarios donde cada tipo de índice sería más apropiado.

Requerimientos:

PostgreSQL o MySQL con permisos para crear índices
Dataset dimensional para pruebas
pgAdmin o DBeaver para análisis visual de planes
Conocimientos intermedios de SQL