PostgreSQL Performance Tuning: A Comprehensive Guide

Table of Contents

Query Optimization

- **EXPLAIN ANALYZE**

```
-- Basic EXPLAIN ANALYZE
EXPLAIN ANALYZE
SELECT * FROM orders
WHERE order_date >= '2025-01-01'
AND customer_id IN (
   SELECT id FROM customers WHERE country = 'USA'
);

-- Show buffers and timing information
EXPLAIN (ANALYZE, BUFFERS, TIMING)
SELECT * FROM orders
WHERE order_date >= '2025-01-01';
```

- **Query Optimization Techniques**

```
-- Use EXISTS instead of IN for better performance
-- Before
SELECT * FROM orders
WHERE customer_id IN (SELECT id FROM customers WHERE country = 'USA');

-- After
SELECT * FROM orders o
WHERE EXISTS (
   SELECT 1 FROM customers c
   WHERE c.id = o.customer_id
   AND c.country = 'USA'
);
```

```
-- Use JOIN instead of correlated subqueries
-- Before
SELECT *,
   (SELECT COUNT(*) FROM order_items oi WHERE oi.order_id = o.id)
FROM orders o;

-- After
SELECT o.*, COUNT(oi.id)
FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
GROUP BY o.id;
```

Indexing Strategies

- **Index Types**

- **-- B-tree index (default)**

CREATE INDEX idx_orders_date ON orders(order_date);

- **-- Partial index**

CREATE INDEX idx_orders_status ON orders(status)
WHERE status IN ('pending', 'processing');

- **-- Multi-column index**

CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);

- **-- Expression index**

CREATE INDEX idx_lower_email ON customers(LOWER(email));

- **-- BRIN index for sequential data**

CREATE INDEX idx_orders_date_brin ON orders USING BRIN(order_date);

- **Index Maintenance**

```
-- Find unused indexes
SELECT
   schemaname || '.' || tablename as table_name,
   indexname,
   idx_scan,
   idx_tup_read,
   idx_tup_fetch
FROM pg_stat_user_indexes
WHERE idx_scan = 0
AND schemaname NOT IN ('pg_catalog', 'pg_toast')
ORDER BY pg_relation_size(indexrelid) DESC;

-- Reindex table
REINDEX TABLE orders;

-- Concurrent reindex (no lock)
CREATE INDEX CONCURRENTLY idx_new_index ON orders(column_name);
DROP INDEX CONCURRENTLY idx_old_index;
```

Configuration Tuning

● Memory Settings

```
# postgresql.conf

# Memory Configuration
shared_buffers = 2GB         # 25% of RAM for dedicated servers
work_mem = 16MB              # Depends on max_connections
maintenance_work_mem = 256MB  # For maintenance operations
effective_cache_size = 6GB    # 75% of RAM for dedicated servers

# Query Planning
random_page_cost = 1.1       # For SSD storage
effective_io_concurrency = 200 # For SSD storage
```

● Connection Settings

```
# postgresql.conf

# Connection Settings
max_connections = 100
superuser_reserved_connections = 3

# Statement Timeout
statement_timeout = '1min'
lock_timeout = '10s'
idle_in_transaction_session_timeout = '1min'
```

Memory Management

● Vacuum Settings

```
# postgresql.conf

# Autovacuum Configuration
autovacuum = on
autovacuum_vacuum_scale_factor = 0.1
autovacuum_analyze_scale_factor = 0.05
autovacuum_vacuum_cost_delay = 2ms
autovacuum_vacuum_cost_limit = 200
ini
```

Buffer Cache Management

```
-- Check buffer cache hit ratio
SELECT
   sum(heap_blks_read) as heap_read,
    sum(heap_blks_hit)  as heap_hit,
    sum(heap_blks_hit) / (sum(heap_blks_hit) + sum(heap_blks_read))::float as ratio
FROM pg_statio_user_tables;

-- Find tables with low cache hit ratio
SELECT
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit,
    heap_blks_hit::float / (heap_blks_read + heap_blks_hit) as hit_ratio
FROM pg_statio_user_tables
WHERE heap_blks_read + heap_blks_hit > 0
ORDER BY hit_ratio ASC;
```

Monitoring and Analysis

- ### Performance Monitoring

```
-- Monitor active queries
SELECT
    pid,
    age(clock_timestamp(), query_start) as duration,
    usename,
    query
FROM pg_stat_activity
WHERE state != 'idle'
AND query NOT ILIKE '%pg_stat_activity%'
ORDER BY duration DESC;

-- Find slow queries
SELECT
    substring(query, 1, 50) as short_query,
    round(total_time::numeric, 2) as total_time,
    calls,
    round(mean_time::numeric, 2) as mean_time,
    round((100 * total_time / sum(total_time::numeric) over ())::numeric, 2) as percentage
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 10;
```

- ### Table Statistics

```
-- Table size and bloat
SELECT
    schemaname,
    tablename,
    pg_size_pretty(pg_total_relation_size(schemaname || '.' || tablename)) as total_size,
    pg_size_pretty(pg_table_size(schemaname || '.' || tablename)) as table_size,
    pg_size_pretty(pg_indexes_size(schemaname || '.' || tablename)) as index_size,
    pg_size_pretty(
        pg_total_relation_size(schemaname || '.' || tablename) -
        pg_table_size(schemaname || '.' || tablename)
    ) as bloat_size
FROM pg_tables
WHERE schemaname NOT IN ('pg_catalog', 'information_schema')
ORDER BY pg_total_relation_size(schemaname || '.' || tablename) DESC;
```
sql

Maintenance Operations

- Regular Maintenance Tasks

```
-- Analyze tables
ANALYZE VERBOSE;

-- Update table statistics
ANALYZE VERBOSE mytable;

-- VACUUM tables
VACUUM (VERBOSE, ANALYZE) mytable;

-- Reindex database
REINDEX DATABASE mydb;
```
sql

- **Maintenance Schedule**

```
-- Create maintenance function
CREATE OR REPLACE FUNCTION perform_maintenance()
RETURNS void AS $$
BEGIN
    -- Vacuum analyze all tables
    VACUUM (ANALYZE, VERBOSE);

    -- Update statistics
    ANALYZE VERBOSE;

    -- Reindex specific tables if needed
    REINDEX TABLE frequently_updated_table;
END;
$$ LANGUAGE plpgsql;

-- Schedule maintenance (using cron or similar)
SELECT perform_maintenance();
```
sql

Performance Tuning Checklist

- **Query Optimization**

1. Use EXPLAIN ANALYZE
2. Optimize JOIN operations
3. Use appropriate subquery types

- **Indexing**

1. Create necessary indexes
2. Remove unused indexes
3. Use appropriate index types

- **Configuration**

1. Optimize memory settings
2. Configure autovacuum
3. Set appropriate timeouts

- **Monitoring**

1. Track slow queries
2. Monitor cache hit ratios
3. Check for bloat

- Maintenance

1. Regular VACUUM
2. Update statistics
3. Reindex when needed