50 Essential PostgreSQL Queries by Category: A Complete Guide

Table of Contents

Database Administration

Database and Table Information

-

```
● - List all databases

SELECT datname, datdba, encoding, datcollate, datctype
FROM pg_database;

● -- List all schemas

SELECT schema_name, schema_owner
FROM information_schema.schemata;

● -- List tables with sizes

SELECT
   schemaname as schema_name,
   relname as table_name,
   pg_size_pretty(pg_total_relation_size(relid)) as total_size,
   pg_size_pretty(pg_table_size(relid)) as table_size,
   pg_size_pretty(pg_indexes_size(relid)) as index_size
FROM pg_catalog.pg_statio_user_tables
ORDER BY pg_total_relation_size(relid) DESC;

● -- Show table column information

SELECT
   column_name,
   data_type,
   character_maximum_length,
   column_default,
   is_nullable
FROM information_schema.columns
WHERE table_schema = 'public'
AND table_name = 'your_table';
```

-

- - List all indexes

```sql
SELECT
    schemaname,
    tablename,
    indexname,
    indexdef
FROM pg_indexes
WHERE schemaname = 'public'
ORDER BY tablename, indexname;
```

sql

- **Backup and Maintenance**

```sql
-- Analyze table statistics
ANALYZE VERBOSE your_table;

-- Vacuum table
VACUUM (VERBOSE, ANALYZE) your_table;

-- Reindex table
REINDEX TABLE your_table;

-- Show last vacuum and analyze time
SELECT
    relname as table_name,
    last_vacuum,
    last_autovacuum,
    last_analyze,
    last_autoanalyze
FROM pg_stat_user_tables;

-- Show dead tuples and autovacuum status
SELECT
    schemaname,
    relname,
    n_dead_tup,
    n_live_tup,
    n_mod_since_analyze
FROM pg_stat_user_tables;
```

sql

Performance Monitoring

- **Query Performance**

```
-- Show slow queries
SELECT
   pid,
   age(clock_timestamp(), query_start) as duration,
   query
FROM pg_stat_activity
WHERE state != 'idle'
ORDER BY duration DESC;

-- Identify queries with high total time
SELECT
   query,
   calls,
   total_time,
   rows,
   mean_time
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 10;

-- Find queries with most block I/O
SELECT
   query,
   shared_blks_hit,
   shared_blks_read,
   shared_blks_written
FROM pg_stat_statements
ORDER BY shared_blks_read + shared_blks_written DESC
LIMIT 10;

-- Show table cache hit ratios
SELECT
   relname as table_name,
   heap_blks_read as blocks_read,
   heap_blks_hit as blocks_hit,
   CASE WHEN heap_blks_hit + heap_blks_read = 0
      THEN 0
      ELSE round(heap_blks_hit::numeric / (heap_blks_hit + heap_blks_read), 4)
   END as cache_hit_ratio
FROM pg_statio_user_tables
ORDER BY cache_hit_ratio DESC;

-- Index usage statistics
SELECT
   schemaname,
   relname,
   indexrelname,
   idx_scan,
   idx_tup_read,
   idx_tup_fetch
FROM pg_stat_user_indexes
ORDER BY idx_scan DESC;
```

Security Management

User and Role Management

```
-- List all roles
SELECT
   rolname,
   rolsuper,
   rolcreaterole,
   rolcreatedb,
   rolcanlogin
FROM pg_roles;

-- Show role permissions
SELECT
   grantee, table_schema, table_name, privilege_type
FROM information_schema.role_table_grants
WHERE grantee = 'your_role';

-- List active sessions by user
SELECT
   usename,
   count(*) as session_count
FROM pg_stat_activity
GROUP BY usename;

-- Show connection limits per role
SELECT
   rolname,
   rolconnlimit
FROM pg_roles
WHERE rolconnlimit <> -1;

-- Audit role memberships
SELECT
   pg_get_userbyid(member) as member,
   pg_get_userbyid(roleid) as role
FROM pg_auth_members;
```

Data Manipulation

Advanced Queries

```
-- Common Table Expression (CTE)
WITH ranked_orders AS (
   SELECT
      customer_id,
      order_date,
```

```sql
    total_amount,
    RANK() OVER (PARTITION BY customer_id ORDER BY total_amount DESC) as rank
  FROM orders
)
SELECT * FROM ranked_orders WHERE rank = 1;

-- Window Functions
SELECT
  department,
  employee_name,
  salary,
  AVG(salary) OVER (PARTITION BY department) as dept_avg,
  salary - AVG(salary) OVER (PARTITION BY department) as diff_from_avg
FROM employees;

-- JSON Operations
SELECT
  id,
  data->>'name' as name,
  (data->>'age')::int as age,
  jsonb_array_elements(data->'hobbies') as hobby
FROM users
WHERE (data->>'age')::int > 25;

-- Full Text Search
SELECT
  title,
  ts_rank_cd(to_tsvector('english', content), query) as rank
FROM articles, plainto_tsquery('english', 'your search terms') query
WHERE to_tsvector('english', content) @@ query
ORDER BY rank DESC;

-- Recursive Queries
WITH RECURSIVE subordinates AS (
  -- Base case
  SELECT employee_id, manager_id, name, 1 as level
  FROM employees
  WHERE employee_id = 1

  UNION ALL

  -- Recursive case
  SELECT e.employee_id, e.manager_id, e.name, s.level + 1
  FROM employees e
  INNER JOIN subordinates s ON s.employee_id = e.manager_id
)
SELECT * FROM subordinates;
```

Advanced Features

Partitioning and Inheritance

```sql
-- Create partitioned table
CREATE TABLE measurements (
```

```
    city_id        int not null,
    logdate        date not null,
    peaktemp       int,
    unitsales      int
) PARTITION BY RANGE (logdate);

-- Create partition
CREATE TABLE measurements_y2025m01 PARTITION OF measurements
    FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

-- Show partitions
SELECT
    parent.relname as table_name,
    child.relname as partition_name,
    pg_get_expr(child.relpartbound, child.oid) as partition_expression
FROM pg_inherits
JOIN pg_class parent ON pg_inherits.inhparent = parent.oid
JOIN pg_class child ON pg_inherits.inhrelid = child.oid
WHERE parent.relname = 'measurements';
```

Materialized Views

```
-- Create materialized view

CREATE MATERIALIZED VIEW sales_summary AS
SELECT
    date_trunc('month', order_date) as month,
    product_category,
    SUM(amount) as total_sales,
    COUNT(*) as order_count
FROM orders
GROUP BY 1, 2
WITH DATA;

-- Refresh materialized view

REFRESH MATERIALIZED VIEW CONCURRENTLY sales_summary;

-- Show materialized view status

SELECT
    schemaname,
    matviewname,
    matviewowner,
    ispopulated
FROM pg_matviews;
```

Extensions and Custom Functions

```
-- List available extensions
SELECT * FROM pg_available_extensions;

-- Create custom function
CREATE OR REPLACE FUNCTION calculate_age(birthdate date)
```

```
RETURNS integer AS $$
BEGIN
   RETURN extract(year from age(current_date, birthdate));
END;
$$ LANGUAGE plpgsql;

-- Show function definitions
SELECT
   p.proname as function_name,
   pg_get_functiondef(p.oid) as definition
FROM pg_proc p
JOIN pg_namespace n ON p.pronamespace = n.oid
WHERE n.nspname = 'public';
```

Best Practices

Always use appropriate indexes

Create indexes for frequently queried columns

1. Monitor index usage
2. Remove unused indexes
3. Regular maintenance

- **Schedule VACUUM and ANALYZE**

1. Monitor table bloat
2. Keep statistics up to date
3. Query optimization

- **Use EXPLAIN ANALYZE**

1. Avoid SELECT *
2. Use appropriate JOIN types
3. Security

- **Regular security audits**

1. Principle of least privilege
2. Strong password policies