

[PostgreSQL Advanced] (CheatSheet)

1. Advanced Database Operations

- **Clone a database:** `CREATE DATABASE new_db WITH TEMPLATE old_db OWNER db_owner;`
- **Drop all tables in a database:** `DO $$ DECLARE r RECORD; BEGIN FOR r IN (SELECT tablename FROM pg_tables WHERE schemaname = current_schema()) LOOP EXECUTE 'DROP TABLE IF EXISTS ' || quote_ident(r.tablename) || ' CASCADE'; END LOOP; END $$;`
- **Rename a database:** `ALTER DATABASE old_db RENAME TO new_db;`

2. Advanced Table Operations

- **Partition a table:** `CREATE TABLE tablename_part PARTITION OF tablename FOR VALUES FROM (min_value) TO (max_value);`
- **Convert a table to unlogged:** `ALTER TABLE tablename SET UNLOGGED;`
- **Move a table to another schema:** `ALTER TABLE old_schema.tablename SET SCHEMA new_schema;`
- **Change a column collation:** `ALTER TABLE tablename ALTER COLUMN column_name TYPE VARCHAR(255) COLLATE "C";`
- **Copy a table structure only:** `CREATE TABLE new_table (LIKE old_table INCLUDING ALL);`

3. Advanced Data Manipulation

- **Insert multiple rows:** `INSERT INTO tablename (column1, column2) VALUES (value1, value2), (value3, value4), (value5, value6);`
- **Insert if not exists:** `INSERT INTO tablename (column1, column2) SELECT value1, value2 WHERE NOT EXISTS (SELECT 1 FROM tablename WHERE condition);`
- **Insert or update:** `INSERT INTO tablename (column1, column2) VALUES (value1, value2) ON CONFLICT (column1) DO UPDATE SET column2 = EXCLUDED.column2;`
- **Batch update:** `UPDATE tablename SET column = CASE WHEN condition1 THEN value1 WHEN condition2 THEN value2 ELSE column END WHERE id IN (id1, id2, id3);`
- **Delete all rows and reset identity:** `TRUNCATE TABLE tablename RESTART IDENTITY;`

4. Advanced Subqueries and CTEs

- **CTE for recursive queries:** `WITH RECURSIVE cte_name AS (SELECT column1, column2 FROM table WHERE condition UNION ALL SELECT column1, column2 FROM table JOIN cte_name ON table.column = cte_name.column) SELECT * FROM cte_name;`
- **Use of subquery in JOIN:** `SELECT * FROM table1 JOIN (SELECT column FROM table2 WHERE condition) AS subquery ON table1.column = subquery.column;`
- **Correlated subquery:** `SELECT * FROM table1 WHERE column1 = (SELECT MAX(column2) FROM table2 WHERE table2.foreign_key = table1.key);`

5. Advanced User and Permissions Management

- **Create a user with specific privileges:** `CREATE USER username WITH PASSWORD 'password'; GRANT SELECT, INSERT ON ALL TABLES IN SCHEMA schema_name TO username;`
- **Clone a user with privileges:** `CREATE USER newuser WITH PASSWORD 'password'; GRANT ALL PRIVILEGES ON DATABASE dbname TO newuser;`
- **Show users with specific privileges:** `SELECT grantee FROM information_schema.role_table_grants WHERE table_schema = 'public' AND privilege_type = 'SELECT';`

6. Advanced Backup and Recovery

- **Incremental backup using pg_basebackup:** `pg_basebackup -D /path/to/backup -Fp -Xs -P -v;`
- **Restore specific table from backup:** `pg_restore -d dbname -t tablename backupfile.dump;`
- **Hot backup with pg_basebackup:** `pg_basebackup -D /data/backups -Ft -z -P -x;`

7. Advanced Performance and Maintenance

- **Partition pruning:** `ALTER TABLE tablename DETACH PARTITION partition_name;`
- **Query profiling with EXPLAIN ANALYZE:** `EXPLAIN ANALYZE SELECT * FROM tablename;`
- **Defragment table with VACUUM FULL:** `VACUUM FULL tablename;`

8. Advanced String and Text Functions

- **Extract part of string:** `SELECT substring(column FROM 'pattern') FROM tablename;`
- **Find and replace using REGEXP:** `SELECT regexp_replace(column, 'pattern', 'replacement') FROM tablename;`
- **Pattern matching:** `SELECT * FROM tablename WHERE column ~ 'pattern';`

9. Advanced Numeric and Date Functions

- **Calculate age from date:** `SELECT age(birthdate) FROM tablename;`
- **Generate a sequence of numbers:** `SELECT generate_series(1, 10);`
- **Format number with commas:** `SELECT to_char(column, 'FM999,999,999') FROM tablename;`

10. Advanced Conditional Expressions

- **Nested CASE statements:** `SELECT CASE WHEN condition1 THEN result1 WHEN condition2 THEN result2 ELSE default_result END FROM tablename;`
- **Conditional aggregation:** `SELECT SUM(CASE WHEN condition THEN column ELSE 0 END) FROM tablename;`
- **CASE in ORDER BY:** `SELECT * FROM tablename ORDER BY CASE WHEN condition1 THEN column1 WHEN condition2 THEN column2 ELSE column3 END;`

11. Advanced Join Techniques

- **Join with USING clause:** `SELECT * FROM table1 JOIN table2 USING (common_column);`
- **Anti Join (not in):** `SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column WHERE table2.column IS NULL;`
- **Semi Join (exists):** `SELECT * FROM table1 WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.column = table2.column);`

12. Advanced Set Operations

- **Intersect using INTERSECT:** `SELECT column FROM table1 INTERSECT SELECT column FROM table2;`
- **Except using EXCEPT:** `SELECT column FROM table1 EXCEPT SELECT column FROM table2;`

13. Advanced Security

- **Generate UUID:** `SELECT gen_random_uuid();`

- **Encrypt a column:** `UPDATE tablename SET column = pgp_sym_encrypt(column, 'key');`
- **Decrypt a column:** `SELECT pgp_sym_decrypt(column, 'key') FROM tablename;`

14. Advanced Transaction Control

- **Savepoint and rollback to savepoint:** `SAVEPOINT savepoint_name; ROLLBACK TO SAVEPOINT savepoint_name;`
- **Set transaction isolation level:** `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`
- **Show transaction isolation level:** `SHOW TRANSACTION ISOLATION LEVEL;`

15. Advanced Administration

- **Change user password:** `ALTER USER username WITH PASSWORD 'newpassword';`
- **Check PostgreSQL server uptime:** `SELECT date_trunc('second', current_timestamp - pg_postmaster_start_time()) AS uptime;`
- **Monitor slow queries:** `SELECT * FROM pg_stat_activity WHERE state = 'active' AND query_start < current_timestamp - interval '5 minutes';`

16. Advanced Debugging and Profiling

- **Log all queries for debugging:** `ALTER SYSTEM SET log_statement = 'all';`
- **Show current locks:** `SELECT * FROM pg_locks;`
- **Analyze query cost:** `EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM tablename WHERE column = 'value';`

17. Advanced Index and Performance Optimization

- **Add composite index:** `CREATE INDEX idx_name ON tablename (column1, column2);`
- **Drop all indexes in a table:** `DO $$ DECLARE r RECORD; BEGIN FOR r IN (SELECT indexname FROM pg_indexes WHERE tablename = 'tablename') LOOP EXECUTE 'DROP INDEX ' || quote_ident(r.indexname); END LOOP; END $$;`
- **Check index usage:** `SELECT * FROM pg_stat_user_indexes WHERE relname = 'tablename';`

18. Advanced Numeric and Date Functions

- **Add minutes to a time:** `SELECT column + interval '15 minutes' FROM tablename;`

- **Round to nearest integer:** `SELECT round(column) FROM tablename;`
- **Find the difference in months:** `SELECT date_part('month', age(date1, date2)) FROM tablename;`

19. Advanced Aggregate Functions Beyond Basics

- **Median of a column:** `SELECT percentile_cont(0.5) WITHIN GROUP (ORDER BY column) FROM tablename;`
- **Percentile calculation:** `SELECT percentile_cont(0.9) WITHIN GROUP (ORDER BY column) FROM tablename;`
- **Mode of a column:** `SELECT mode() WITHIN GROUP (ORDER BY column) FROM tablename;`

20. Advanced Database Maintenance and Inspection

- **Check for corrupted tables:** `SELECT * FROM pg_catalog.pg_tables WHERE schemaname = 'public' AND tablename NOT IN (SELECT relname FROM pg_stat_user_tables);`
- **Repair corrupted table:** `REINDEX TABLE tablename;`
- **Inspect table space usage:** `SELECT pg_size_pretty(pg_total_relation_size('tablename')) AS size;`

21. Advanced Information Schema Usage

- **Get column statistics:** `SELECT * FROM information_schema.columns WHERE table_name = 'tablename';`
- **List foreign keys in a schema:** `SELECT conname AS constraint_name, conrelid::regclass AS table_from, conkey AS fk_cols, confrelid::regclass AS table_to, confkey AS pk_cols FROM pg_constraint WHERE contype = 'f' AND connamespace = 'schema_name'::regnamespace;`
- **Get table size:** `SELECT relname AS table_name, pg_size_pretty(pg_total_relation_size(relid)) AS total_size FROM pg_catalog.pg_statio_user_tables ORDER BY pg_total_relation_size(relid) DESC;`

22. Advanced Performance Schema for Diagnostics

- **Track wait events:** `SELECT * FROM pg_stat_activity WHERE wait_event IS NOT NULL;`
- **Check IO statistics:** `SELECT * FROM pg_stat_io WHERE relname = 'tablename';`

- **Monitor query execution time:** `SELECT query, total_exec_time, mean_exec_time FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 10;`

23. Advanced Query Optimization Techniques

- **Use index hints:** `SELECT * FROM tablename /*+ IndexScan(tablename index_name) */ WHERE column = 'value';`
- **Force index usage:** `SET enable_seqscan = OFF; SELECT * FROM tablename WHERE column = 'value'; SET enable_seqscan = ON;`
- **Optimize query plan:** `SET enable_nestloop = OFF; EXPLAIN ANALYZE SELECT * FROM tablename WHERE column = 'value'; SET enable_nestloop = ON;`

24. Advanced Data Export and Import

- **Export data to CSV:** `COPY (SELECT * FROM tablename) TO '/path/to/file.csv' WITH CSV HEADER;`
- **Import data from CSV:** `COPY tablename FROM '/path/to/file.csv' WITH CSV HEADER;`
- **Export specific columns:** `COPY (SELECT column1, column2 FROM tablename) TO '/path/to/file.csv' WITH CSV HEADER;`

25. Advanced JSON Functions

- **Extract JSON value:** `SELECT json_column->>'key' FROM tablename;`
- **Update JSON value:** `UPDATE tablename SET json_column = jsonb_set(json_column, '{key}', '"new_value"') WHERE condition;`
- **Merge JSON values:** `SELECT jsonb_concat(json_column1, json_column2) FROM tablename;`

26. Advanced Spatial Data Operations

- **Create spatial index:** `CREATE INDEX idx_name ON tablename USING GIST (spatial_column);`
- **Select within radius:** `SELECT * FROM tablename WHERE ST_DWithin(spatial_column::geography, ST_MakePoint(lng, lat)::geography, radius);`
- **Find nearest point:** `SELECT * FROM tablename ORDER BY spatial_column <-> ST_SetSRID(ST_MakePoint(lng, lat), 4326) LIMIT 1;`

27. Advanced Full-Text Search

- **Full-text search with relevance:** `SELECT *, ts_rank_cd(to_tsvector('english', column), to_tsquery('search_term')) AS relevance FROM tablename WHERE to_tsvector('english', column) @@ to_tsquery('search_term');`
- **Boolean mode search:** `SELECT * FROM tablename WHERE to_tsvector('english', column) @@ plainto_tsquery('term1 & !term2');`
- **Phrase search:** `SELECT * FROM tablename WHERE to_tsvector('english', column) @@ phraseto_tsquery('phrase');`

28. Advanced View Management

- **Create a view with joins:** `CREATE VIEW viewname AS SELECT a.column1, b.column2 FROM table1 a JOIN table2 b ON a.common_column = b.common_column;`
- **Update a view:** `CREATE OR REPLACE VIEW viewname AS SELECT column1, column2 FROM tablename;`
- **Drop a view:** `DROP VIEW viewname;`

29. Advanced Stored Procedures and Functions

- **Create a stored procedure:** `CREATE OR REPLACE PROCEDURE proc_name(param1 INT, OUT param2 VARCHAR) LANGUAGE plpgsql AS $$ BEGIN SELECT column INTO param2 FROM tablename WHERE id = param1; END $$;`
- **Call a stored procedure:** `CALL proc_name(1, param2);`
- **Create a stored function:** `CREATE OR REPLACE FUNCTION func_name(param1 INT) RETURNS VARCHAR AS $$ BEGIN RETURN (SELECT column FROM tablename WHERE id = param1); END $$ LANGUAGE plpgsql;`

30. Advanced Triggers

- **Create a trigger for update:** `CREATE TRIGGER trigger_name BEFORE UPDATE ON tablename FOR EACH ROW EXECUTE FUNCTION trigger_function();`
- **Create a trigger for insert:** `CREATE TRIGGER trigger_name BEFORE INSERT ON tablename FOR EACH ROW EXECUTE FUNCTION trigger_function();`
- **Drop a trigger:** `DROP TRIGGER trigger_name ON tablename;`

31. Advanced Replication Management

- **Setup replication user:** `CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'password';`
- **Show replication status:** `SELECT * FROM pg_stat_replication;`

- **Start replication:** `SELECT pg_start_backup('label');`
- **Stop replication:** `SELECT pg_stop_backup();`

32. Advanced Event Scheduler

- **Create an event:** `CREATE OR REPLACE FUNCTION event_function() RETURNS void AS $$ BEGIN UPDATE tablename SET column = value WHERE condition; END $$ LANGUAGE plpgsql; CREATE EXTENSION pg_cron; SELECT cron.schedule('0 0 * * *', 'CALL event_function()');`
- **Show events:** `SELECT * FROM cron.job;`
- **Drop an event:** `SELECT cron.unschedule(job_id) FROM cron.job WHERE job_name = 'event_function';`

33. Advanced Schema Management

- **Copy schema:** `CREATE SCHEMA new_schema; ALTER SCHEMA old_schema RENAME TO new_schema;`
- **Rename schema:** `ALTER SCHEMA old_schema RENAME TO new_schema;`
- **Export schema structure:** `pg_dump -s -U username dbname > schema.sql;`

34. Advanced Audit and Logging

- **Enable general query log:** `ALTER SYSTEM SET log_statement = 'all';`
- **Show slow queries:** `SELECT * FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 10;`
- **Enable binary logging:** `ALTER SYSTEM SET wal_level = 'logical';`

35. Advanced Partition Management

- **Reorganize partition:** `ALTER TABLE tablename DETACH PARTITION partition_name;`
- **Merge partitions:** `ALTER TABLE tablename ATTACH PARTITION partition_name FOR VALUES FROM (min_value) TO (max_value);`
- **Drop partition:** `ALTER TABLE tablename DROP PARTITION partition_name;`

36. Advanced Data Masking and Anonymization

- **Mask sensitive data:** `UPDATE tablename SET column = LEFT(column, 3) || '***' || RIGHT(column, 3) WHERE condition;`
- **Anonymize data with random values:** `UPDATE tablename SET column = md5(random()::text) WHERE condition;`

37. Advanced Data Migration

- **Migrate data between tables:** `INSERT INTO new_table (column1, column2) SELECT column1, column2 FROM old_table;`
- **Copy data between servers:** `pg_dump -h source_host -U username dbname | psql -h destination_host -U username dbname;`
- **Transform data during migration:** `INSERT INTO new_table (column1, column2) SELECT column1, UPPER(column2) FROM old_table;`

38. Advanced Security Management

- **Change user host:** `ALTER ROLE username WITH LOGIN;`
- **Disable a user account:** `ALTER USER username NOLOGIN;`
- **Enable a user account:** `ALTER USER username LOGIN;`

39. Advanced Performance Analysis

- **Show buffer pool status:** `SELECT * FROM pg_buffercache;`
- **Analyze query cache:** `SELECT * FROM pg_stat_database WHERE datname = 'dbname';`
- **Analyze index usage:** `SELECT * FROM pg_stat_user_indexes WHERE relname = 'tablename';`

40. Advanced Storage Engine Management

- **Convert table storage engine:** `ALTER TABLE tablename SET (storage_parameter = 'value');`
- **Check storage engine status:** `SELECT * FROM pg_statio_user_tables WHERE relname = 'tablename';`
- **Enable/disable storage engine:** `ALTER TABLE tablename SET (autovacuum_enabled = false);`

41. Advanced Geographic Data Management

- **Insert spatial data:** `INSERT INTO tablename (spatial_column) VALUES (ST_GeomFromText('POINT(lng lat)', 4326));`
- **Query spatial data:** `SELECT ST_AsText(spatial_column) FROM tablename WHERE condition;`
- **Spatial distance calculation:** `SELECT ST_Distance_Sphere(ST_MakePoint(lng1, lat1), ST_MakePoint(lng2, lat2)) FROM tablename;`

42. Advanced JSON Operations

- **Merge JSON arrays:** `SELECT jsonb_agg(column) FROM tablename;`
- **Remove JSON key:** `UPDATE tablename SET json_column = jsonb - 'key' WHERE condition;`
- **Extract multiple JSON values:** `SELECT json_column->>'key1',
json_column->>'key2' FROM tablename;`