PostgreSQL Zero → Advanced Cheatsheet

Based on **PostgreSQL 18.x official documentation**.

---

1. What is PostgreSQL?

- **PostgreSQL (Postgres)** = Database Management System (DBMS)
- **SQL** = Query language used to communicate with PostgreSQL
- Object-relational, ACID compliant, extensible, open-source

---

2. Architecture (High Level)

- Client (psql / app)
- Postmaster (main server)
- Backend processes (one per connection)
- Shared memory (buffers, WAL)
- Data directory (tables, indexes, WAL)

---

3. Installation & Access

```
psql -U postgres
```

- Connects to PostgreSQL server using **psql client**

---

4. Database & Schema

```
CREATE DATABASE app_db; -- DDL
\c app_db
```

- **DDL (Data Definition Language)** → changes structure

Schemas:

```
CREATE SCHEMA auth;
SET search_path TO auth;
```

5. Data Types (Core)

- Numeric: `int`, `bigint`, `numeric`, `serial`
- Text: `text`, `varchar(n)`
- Boolean: `boolean`
- Time: `date`, `timestamp`, `timestamptz`
- JSON: `json`, `jsonb`
- UUID: `uuid`
- Arrays: `int[]`, `text[]`

---

6. Table Creation

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  password TEXT NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

- PRIMARY KEY = uniqueness + index - SERIAL = auto-increment (sequence)

---

7. Constraints

- `PRIMARY KEY`
- `UNIQUE`
- `NOT NULL`
- `CHECK`
- `FOREIGN KEY`

```
CHECK (char_length(password) > 8)
```

---

8. Insert / Update / Delete (DML)

```
INSERT INTO users(email, password) VALUES ('a@b.com','secret');
UPDATE users SET password='x' WHERE id=1;
DELETE FROM users WHERE id=1;
```

- **DML (Data Manipulation Language)** → changes data

---

9. SELECT Fundamentals

```
SELECT id, email FROM users WHERE id > 10 ORDER BY id DESC LIMIT 5;
```

Order of execution: 1. FROM 2. WHERE 3. SELECT 4. ORDER BY 5. LIMIT

---

10. Joins

```
SELECT u.email, p.title
FROM users u
JOIN posts p ON p.user_id = u.id;
```

Types: - INNER - LEFT - RIGHT - FULL

---

11. Aggregates

```
SELECT user_id, COUNT(*) FROM posts GROUP BY user_id HAVING COUNT(*) > 5;
```

Functions: - `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`

---

12. Indexes

```
CREATE INDEX idx_users_email ON users(email);
```

Types: - B-Tree (default) - Hash - GIN (JSON, arrays) - GiST (geo, ranges)

Partial index:

```
CREATE INDEX idx_active ON users(email) WHERE active = true;
```

---

13. Transactions (TCL)

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id=1;
UPDATE accounts SET balance = balance + 100 WHERE id=2;
COMMIT;
```

- **TCL (Transaction Control Language)** - Atomic, Consistent, Isolated, Durable

Isolation levels: - READ COMMITTED (default) - REPEATABLE READ - SERIALIZABLE

---

14. Views & Materialized Views

```
CREATE VIEW active_users AS SELECT * FROM users WHERE active;
```

Materialized:

```
CREATE MATERIALIZED VIEW stats AS SELECT COUNT(*) FROM users;
REFRESH MATERIALIZED VIEW stats;
```

---

15. JSONB Power

```
SELECT data->>'name' FROM products WHERE data @> '{"active":true}';
```

Operators: - `->`, `->>` - `@>` contains

GIN index recommended

---

16. Functions & Procedures

```
CREATE FUNCTION add(a int, b int)
RETURNS int LANGUAGE sql AS $$ SELECT a+b $$;
```

Procedures (can commit):

```
CREATE PROCEDURE log_cleanup() LANGUAGE plpgsql AS $$ BEGIN DELETE FROM logs;
END $$;
```

---

17. PL/pgSQL Essentials

```
DECLARE total int;
BEGIN
  SELECT COUNT(*) INTO total FROM users;
END;
```

---

18. Triggers

```
CREATE TRIGGER trg BEFORE INSERT ON users
FOR EACH ROW EXECUTE FUNCTION check_email();
```

Use cases: - audit - validation - denormalization

---

19. Security & Roles (DCL)

```
CREATE ROLE app_user LOGIN PASSWORD 'x';
GRANT SELECT, INSERT ON users TO app_user;
```

- **DCL (Data Control Language)** → permissions

Row Level Security:

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
```

---

20. Performance & EXPLAIN

```
EXPLAIN ANALYZE SELECT * FROM users WHERE email='x';
```

Look for: - Seq Scan vs Index Scan - Cost - Rows

---

21. Maintenance

- `VACUUM`
- `ANALYZE`
- `REINDEX`

Autovacuum is critical

---

22. Backup & Restore

```
pg_dump app_db > backup.sql
psql app_db < backup.sql
```

---

23. Replication & HA (Advanced)

- Streaming replication
- Logical replication
- WAL shipping
- Hot standby

---

24. Extensions

```
CREATE EXTENSION pg_trgm;
CREATE EXTENSION postgis;
```

---

25. Mental Model

- Tables = files
- Rows = records
- Indexes = lookup maps
- WAL = black box flight recorder

---

26. Learning Path

1. SQL basics
2. Indexing
3. Transactions
4. Query planning
5. JSON & extensions
6. Replication & tuning

Source: PostgreSQL 18.x Official Documentation