📘 PostgreSQL Notes

(From Basics to Advanced with Definitions & Examples)

🔹 What is PostgreSQL?

- **PostgreSQL** is an advanced **open-source relational database management system (RDBMS)**.
- It supports **SQL** and **NoSQL** features like JSON.
- Known for **performance, reliability, and extensibility**.
- Often called **Post SQL**.

🔑 Key Features

- ACID compliant (Atomicity, Consistency, Isolation, Durability)
- Supports complex SQL queries, joins, and transactions
- Supports **JSON, XML, Arrays**
- User-defined types, functions, and stored procedures
- Full-text search
- MVCC (Multi-Version Concurrency Control)

📦 Basic Syntax

sql
CopyEdit

```sql
-- Create Database
CREATE DATABASE college;

-- Connect to Database
\c college

-- Create Table
CREATE TABLE students (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  age INT,
  course VARCHAR(50)
);

-- Insert Data
INSERT INTO students (name, age, course) VALUES ('Dhanush', 21,
'BCA');

-- Query Data
SELECT * FROM students;
```

🔠 Data Types in PostgreSQL

Category	Data Types	Example
Numeric	INT, BIGINT, SERIAL, NUMERIC	100, 100000, 3.14
String	VARCHAR(n), TEXT, CHAR(n)	'Hello'
Date/Time	DATE, TIME, TIMESTAMP	'2025-07-26'
Boolean	BOOLEAN	TRUE, FALSE
Special	JSON, ARRAY, UUID	['A', 'B'], JSON

🧰 Constraints

sql

CopyEdit
```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) NOT NULL,
  age INT CHECK (age >= 18)
);
```

Constraint	Purpose
PRIMARY KEY	Uniquely identifies a row
UNIQUE	No duplicate values
NOT NULL	Must have a value
CHECK	Validates condition
FOREIGN KEY	Links to another table

🔍 Query Examples

sql
CopyEdit
```
SELECT * FROM students WHERE age > 20;
SELECT name FROM students ORDER BY age DESC LIMIT 3;
SELECT course, COUNT(*) FROM students GROUP BY course;
```

🔁 Joins

sql
CopyEdit
```
SELECT s.name, c.course_name
FROM students s
JOIN courses c ON s.course_id = c.id;
```

Join Type	Description
INNER JOIN	Matches in both tables

LEFT JOIN	All left + matched right
RIGHT JOIN	All right + matched left
FULL JOIN	All from both sides (need `FULL OUTER JOIN`)

🧠 Advanced PostgreSQL Features

✅ 1. Views

sql
CopyEdit
```
CREATE VIEW student_view AS
SELECT name, course FROM students;
```

✅ 2. Stored Functions (Procedures)

sql
CopyEdit
```
CREATE OR REPLACE FUNCTION get_students()
RETURNS TABLE(name VARCHAR, age INT) AS $$
BEGIN
  RETURN QUERY SELECT name, age FROM students;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_students();
```

✅ 3. Indexes

sql
CopyEdit
```
CREATE INDEX idx_name ON students(name);
```

✅ 4. JSON Support

sql
CopyEdit
```
CREATE TABLE profiles (
  id SERIAL PRIMARY KEY,
  data JSON
);

INSERT INTO profiles (data)
VALUES ('{"name": "Dhanush", "skills": ["Python", "SQL"]}');
```

✅ 5. Array Support

sql
CopyEdit
```
CREATE TABLE courses (
  id SERIAL PRIMARY KEY,
  subjects TEXT[]
);

INSERT INTO courses (subjects) VALUES (ARRAY['HTML', 'CSS', 'JS']);
```

🔄 Transactions

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

🔐 User and Role Management

sql
CopyEdit
```
CREATE ROLE dhanush LOGIN PASSWORD 'password';
GRANT CONNECT ON DATABASE college TO dhanush;
GRANT SELECT, INSERT ON students TO dhanush;
```

📦 Backup and Restore

- **Backup:**

bash
CopyEdit
```
pg_dump college > college_backup.sql
```

- **Restore:**

bash
CopyEdit
```
psql college < college_backup.sql
```

✅ PostgreSQL vs MySQL Summary

Feature	PostgreSQL	MySQL
Open-source	Yes	Yes
JSON Support	Advanced	Basic
SQL Features	Rich/Complex	Simpler
Indexing	Powerful	Good
Stored Procs	PL/pgSQL	SQL/Procedural

📘 PostgreSQL Notes – Part 2 (Advanced + Real World Concepts)

🔷 1. Subqueries

A query inside another query.

sql
CopyEdit
```
-- Find students older than average age
SELECT name FROM students
WHERE age > (SELECT AVG(age) FROM students);
```

🔷 2. Common Table Expressions (CTE)

Temporary result set used within a query. Improves readability.

sql
CopyEdit
```
WITH student_cte AS (
  SELECT name, age FROM students WHERE age > 20
)
SELECT * FROM student_cte;
```

🔷 3. Window Functions

Allows performing operations across rows related to the current row.

sql
CopyEdit
```
SELECT name, age,
       RANK() OVER (ORDER BY age DESC) AS age_rank
FROM students;
```

🔷 4. FULL OUTER JOIN

Includes all rows when there is a match in either table.

sql
CopyEdit
```
SELECT a.name, b.course_name
FROM students a
FULL OUTER JOIN courses b ON a.course_id = b.id;
```

🔷 5. Upsert (`INSERT ... ON CONFLICT`)

Inserts a record or updates if it already exists.

sql
CopyEdit
```
INSERT INTO students (id, name, age)
VALUES (1, 'Dhanush', 21)
ON CONFLICT (id)
DO UPDATE SET age = EXCLUDED.age;
```

🔷 6. CASE Statement

Used for conditional logic inside queries.

sql
CopyEdit
```
SELECT name,
  CASE
    WHEN age >= 21 THEN 'Adult'
    ELSE 'Teen'
  END AS age_group
FROM students;
```

🔷 7. Triggers

Auto-execute SQL on insert/update/delete.

sql
CopyEdit
```
CREATE FUNCTION update_log() RETURNS trigger AS $$
BEGIN
  INSERT INTO logs (info) VALUES ('Student Updated');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update
AFTER UPDATE ON students
FOR EACH ROW EXECUTE FUNCTION update_log();
```

🔷 8. Table Partitioning

Split large tables into smaller parts (partitions).

sql
CopyEdit
```
CREATE TABLE sales (
  id SERIAL,
  region TEXT,
  amount INT
) PARTITION BY LIST (region);

CREATE TABLE sales_south PARTITION OF sales FOR VALUES IN ('South');
CREATE TABLE sales_north PARTITION OF sales FOR VALUES IN ('North');
```

🔷 9. Full Text Search

Used for searching text efficiently.

sql
CopyEdit
```
SELECT * FROM articles
WHERE to_tsvector(content) @@ to_tsquery('PostgreSQL');
```

🔷 10. Extensions in PostgreSQL

PostgreSQL supports extensions to add functionality.

sql
CopyEdit
```
-- Enable UUID generation
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Use it in table
CREATE TABLE users (
  id UUID DEFAULT uuid_generate_v4(),
  name TEXT
);
```

🛠️ Real-world Project Tables Example

👨‍🎓 Students Table

sql
CopyEdit
```
CREATE TABLE students (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  dob DATE,
  email VARCHAR(100) UNIQUE,
  address TEXT,
  enrolled BOOLEAN DEFAULT TRUE
);
```

📚 Courses Table

sql
CopyEdit
```
CREATE TABLE courses (
  id SERIAL PRIMARY KEY,
  title VARCHAR(100),
  credits INT
);
```

📝 Enrollments (Relation Table)

sql
CopyEdit
```
CREATE TABLE enrollments (
  student_id INT REFERENCES students(id),
  course_id INT REFERENCES courses(id),
  enrollment_date DATE DEFAULT CURRENT_DATE,
  PRIMARY KEY (student_id, course_id)
);
```

📋 PostgreSQL Best Practices

1. Use `EXPLAIN` to check query performance.
2. Use **indexes** on frequently searched columns.
3. Regularly perform **VACUUM** to clean dead tuples.
4. Use **connection pooling** for production (e.g., pgBouncer).
5. Use `NOT NULL` + `CHECK` for data integrity.

🧪 PostgreSQL Admin Commands

Command	Purpose
`\l`	List all databases

`\dt`	List all tables
`\du`	List roles
`\c dbname`	Connect to database
`\d table_name`	Table structure (schema)
`\q`	Quit from psql