**Best Practices for PostgreSQL Administration**

**1. Configuration Best Practices**

**1.1 Memory Settings**

* **shared\_buffers**: Allocate 15–25% of total system memory.
* **work\_mem**: Per-operation memory for sort/hash. Start with 4–16MB.
* **maintenance\_work\_mem**: Use 64MB–1GB for vacuum and create index operations.

**1.2 WAL Configuration**

* **wal\_level = replica**: Enables replication and point-in-time recovery.
* **archive\_mode = on** + **archive\_command**: For WAL archiving.
* **max\_wal\_size** and **min\_wal\_size**: Tune based on disk and workload.

**1.3 Connection Tuning**

* **max\_connections**: Set based on connection pooling strategy.
* Use **pgbouncer** for connection pooling to avoid excessive backend usage.

**2. Performance Tuning**

**2.1 Indexing**

* Use **B-Tree** for equality/range.
* Use **GIN** for full-text search and JSONB.
* Use **GiST** for geometric/range queries.

**2.2 Analyze Query Plans**

* Use EXPLAIN (ANALYZE, BUFFERS) to troubleshoot queries.
* Look for sequential scans where index scans are expected.

**2.3 Autovacuum**

* Ensure **autovacuum is enabled**.
* Monitor and tune:

autovacuum\_vacuum\_threshold = 50

autovacuum\_vacuum\_scale\_factor = 0.02

autovacuum\_naptime = 10s

**3. Security Best Practices**

**3.1 Authentication & Authorization**

* Use **scram-sha-256** instead of md5 for password encryption.
* Secure pg\_hba.conf:

host all all 127.0.0.1/32 scram-sha-256

* Limit superuser roles.

**3.2 SSL Encryption**

* Enable SSL for client-server communication.
* Use ssl\_cert\_file and ssl\_key\_file directives.

**3.3 Row- & Column-Level Security**

* Use **Row Level Security (RLS)** to limit access:

ALTER TABLE sensitive\_data ENABLE ROW LEVEL SECURITY;

**4. Maintenance Best Practices**

**4.1 Regular Vacuum & Analyze**

* **Vacuum** reclaims space.
* **Analyze** updates planner statistics.
* Schedule:

vacuumdb --all --analyze

**4.2 Monitor Dead Tuples**

* Run:

SELECT \* FROM pg\_stat\_user\_tables WHERE n\_dead\_tup > 1000;

**4.3 Reindex Periodically**

* Use REINDEX to fix bloat or corruption.

**5. Monitoring & Logging**

**5.1 Enable Logging**

* Configure:

logging\_collector = on

log\_directory = 'pg\_log'

log\_min\_duration\_statement = 500 # ms

**5.2 Use Monitoring Tools**

* **pg\_stat\_statements** for tracking query performance.
* External tools: **pgAdmin**, **Prometheus + Grafana**, **pganalyze**.

**6. Backup & Recovery**

**6.1 Logical Backups**

* Use pg\_dump and pg\_restore:

pg\_dump -U postgres -F c -f db.backup dbname

pg\_restore -U postgres -d newdb db.backup

**6.2 Physical Backups**

* Use pg\_basebackup for streaming base backups.
* Enable **WAL archiving** for Point-in-Time Recovery (PITR).

**6.3 PITR Setup**

* Set:

restore\_command = 'cp /archive/%f %p'

* Create recovery.signal to trigger PITR.

**7. Schema & Application Design**

**7.1 Use Appropriate Data Types**

* Use jsonb for semi-structured data.
* Use range types for booking/reservation systems.

**7.2 Naming Conventions**

* Use clear and consistent names for tables, indexes, and constraints.

**7.3 Avoid Over-Normalization**

* Normalize wisely; overdoing it can hurt performance.

**Summary Checklist**

| **Task** | **Recommended Frequency** |
| --- | --- |
| Vacuum and Analyze | Daily / Autovacuum |
| Full backup | Daily |
| WAL archiving | Continuous |
| Monitor slow queries | Ongoing |
| Tune memory settings | During setup / traffic changes |
| Review access control | Monthly or after role changes |