PostgreSQL Deep Dive

SQL Basics → Indexing → Transactions → Query Planning → JSON & Extensions → Replication & Tuning

Designed as **PDF-ready learning material** with explanations + examples.

---

1. SQL BASICS (FOUNDATION)

PostgreSQL vs SQL

- **PostgreSQL**: Database engine (stores data, manages concurrency)
- **SQL**: Language used to talk to PostgreSQL

SQL Categories

- **DDL** (Data Definition Language): structure
- **DML** (Data Manipulation Language): data
- **DCL** (Data Control Language): permissions
- **TCL** (Transaction Control Language): transactions

1.1 Core DDL

```sql
CREATE TABLE accounts (
  id BIGSERIAL PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  balance NUMERIC(12,2) DEFAULT 0,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

Why: - `BIGSERIAL` = sequence-backed auto increment - `NUMERIC` = money-safe (no float errors) - `TIMESTAMPTZ` = timezone-safe timestamps

---

1.2 Core DML

```sql
INSERT INTO accounts(email, balance)
VALUES ('a@x.com', 1000);
```

```sql
UPDATE accounts SET balance = balance - 200 WHERE id = 1;
DELETE FROM accounts WHERE id = 99;
```

PostgreSQL guarantees **ACID** for each statement.

1.3 SELECT Order of Execution

```sql
SELECT email, balance
FROM accounts
WHERE balance > 100
ORDER BY balance DESC
LIMIT 5;
```

Logical order: 1. FROM 2. WHERE 3. SELECT 4. ORDER BY 5. LIMIT

---

2. INDEXING (PERFORMANCE CORE)

2.1 What Indexes Really Are

Indexes = **sorted lookup structures**, not magic. Tradeoff: - Faster reads - Slower writes - More disk

---

2.2 B-Tree Index (Default)

```sql
CREATE INDEX idx_accounts_email ON accounts(email);
```

Used for: - `=` - `< >` - `ORDER BY`

2.3 Composite Index

```sql
CREATE INDEX idx_accounts_email_balance
ON accounts(email, balance);
```

Works for: - email - email + balance ❌NOT balance alone

2.4 Partial Index

```
CREATE INDEX idx_positive_balance
ON accounts(balance)
WHERE balance > 0;
```

Smaller, faster, smarter.

2.5 GIN Index (JSON / Arrays)

```
CREATE INDEX idx_data_gin
ON products USING gin(data);
```

Required for JSONB performance.

---

3. TRANSACTIONS (DATA SAFETY)

3.1 ACID Explained

- **Atomic**: all or nothing
- **Consistent**: rules enforced
- **Isolated**: no dirty reads
- **Durable**: WAL-backed

3.2 Transaction Example

```
BEGIN;
UPDATE accounts SET balance = balance - 500 WHERE id = 1;
UPDATE accounts SET balance = balance + 500 WHERE id = 2;
COMMIT;
```

If any statement fails → ROLLBACK.

3.3 Isolation Levels

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Levels: - READ COMMITTED (default) - REPEATABLE READ - SERIALIZABLE

Higher = safer, slower.

---

4. QUERY PLANNING (HOW POSTGRES THINKS)

4.1 EXPLAIN ANALYZE

```
EXPLAIN ANALYZE
SELECT * FROM accounts WHERE email = 'a@x.com';
```

Look for: - Seq Scan ❌- Index Scan ✅- Rows vs actual rows

4.2 Common Planner Mistakes

- Missing index
- Wrong column order
- Outdated statistics

Fix with:

```
ANALYZE accounts;
```

4.3 Index-Only Scans

```
CREATE INDEX idx_covering
ON accounts(email) INCLUDE (balance);
```

Avoids table reads entirely.

---

5. JSON & EXTENSIONS (POSTGRES SUPERPOWER)

5.1 JSON vs JSONB

- `json` : stored as text
- `jsonb` : binary, indexable

Always prefer **jsonb**.

5.2 JSON Queries

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

Operators: - `->` json - `->>` text - `@>` contains

5.3 Powerful Extensions

```sql
CREATE EXTENSION pg_trgm;   -- fuzzy search
CREATE EXTENSION postgis;  -- geospatial
CREATE EXTENSION citext;   -- case-insensitive text
```

Postgres = extensible database, not just SQL.

---

6. REPLICATION & TUNING (ADVANCED)

6.1 WAL (Write Ahead Log)

- Every change written to WAL first
- Guarantees durability
- Powers replication & recovery

6.2 Streaming Replication

Primary → Standby - Physical replication - Hot standby supported

Use cases: - High availability - Read scaling

6.3 Logical Replication

```
CREATE PUBLICATION pub_all FOR ALL TABLES;
CREATE SUBSCRIPTION sub
CONNECTION '...'
PUBLICATION pub_all;
```

Used for: - Version upgrades - Selective replication

6.4 Performance Tuning Essentials

Key settings: - shared_buffers - work_mem - maintenance_work_mem - effective_cache_size

Golden rule:

> Measure first, tune second.

FINAL MENTAL MODEL

- SQL = language
- PostgreSQL = operating system for data
- Indexes = maps
- WAL = black box recorder
- Planner = cost-based optimizer

You are now at Senior PostgreSQL Level

Next upgrades: - Query rewriting - Lock analysis - Vacuum internals - Sharding strategies

---

Source: PostgreSQL 18.x Official Documentation