

# DBMS Advanced Concepts – Indexing, Performance, Concurrency & Architecture

One single, senior-level documentation covering INDEXING, query optimization, execution plans, locking, transactions, isolation levels, sharding, replication, partitioning, caching, and failure handling across:

- SQL (Generic)
- PostgreSQL (Production focus)
- MongoDB
- Node.js usage patterns

This is written at a 7–10 years backend / DB architect level.

---

## 1 INDEXING – THE MOST IMPORTANT DBMS CONCEPT

! 90% performance problems are indexing problems

### 1.1 What an Index REALLY is

- A **data structure** (usually **B-Tree**) that allows fast lookup
  - Trades **write speed + storage** for **read speed**
  - Indexes work only if the query uses them correctly
- 

## 2 INDEX TYPES (SQL vs PostgreSQL vs MongoDB)

### 2.1 Single Column Index

SQL / PostgreSQL

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

MongoDB

```
db.users.createIndex({ email: 1 });
```

✓ Used for `WHERE email = ?`

---

### 2.2 Compound Index (VERY IMPORTANT)

**Rule:** Order matters

## PostgreSQL

```
CREATE INDEX idx_users_status_salary ON users(status, salary);
```

## MongoDB

```
db.users.createIndex({ status: 1, salary: -1 });
```

Works for: -  -

 Does NOT work for only

---

## 2.3 Unique Index

### PostgreSQL

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

### MongoDB

```
db.users.createIndex({ email: 1 }, { unique: true });
```

Used for: - Emails - Usernames - Idempotency keys

---


## 2.4 Partial / Filtered Index (ADVANCED)

### PostgreSQL

```
CREATE INDEX idx_active_users ON users(email)  
WHERE status = 'ACTIVE';
```

### MongoDB

```
db.users.createIndex(  
  { email: 1 },  
  { partialFilterExpression: { status: 'ACTIVE' } }  
);
```

 Huge performance gain for selective data

---

## 2.5 Text / Search Index

### PostgreSQL

```
CREATE INDEX idx_users_search ON users USING GIN(to_tsvector('english', name));
```

### MongoDB

```
db.users.createIndex({ name: "text" });
```

---

## 3 QUERY EXECUTION PLAN (HOW DB DECIDES)

### 3.1 PostgreSQL – EXPLAIN ANALYZE

```
EXPLAIN ANALYZE  
SELECT * FROM users WHERE email = 'a@b.com';
```

Key things to read: - Seq Scan  - Index Scan  - Cost vs Actual Time

### 3.2 MongoDB – executionStats

```
db.users.find({ email }).explain('executionStats');
```

Look for: - COLLSCAN  - IXSCAN  - documentsExamined

---

## 4 TRANSACTIONS & ISOLATION LEVELS

### 4.1 ACID Breakdown

Property	Meaning
Atomicity	All or nothing
Consistency	Valid state
Isolation	No interference
Durability	Survives crash

## 4.2 Isolation Levels (VERY IMPORTANT)

Level	Problems Allowed
READ UNCOMMITTED	Dirty reads
READ COMMITTED	Non-repeatable
REPEATABLE READ	Phantom reads
SERIALIZABLE	None

**PostgreSQL default:** READ COMMITTED

**MongoDB:** Snapshot isolation (inside transaction)

---

## 5 LOCKING & CONCURRENCY

### 5.1 SQL Locks

- Row-level locks
- Table locks
- Deadlocks possible

```
SELECT * FROM users WHERE id = 1 FOR UPDATE;
```

---

### 5.2 MongoDB Concurrency

- Document-level locking
- No row/table lock
- Safer horizontal scaling

---

## 6 PARTITIONING & SHARDING (SCALE)

### 6.1 PostgreSQL Partitioning

```
CREATE TABLE orders_2024 PARTITION OF orders  
FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');
```

Used for: - Time-series - Logs

---

## 6.2 MongoDB Sharding

```
sh.shardCollection('db.orders', { userId: 1 });
```

Shard key rules: - High cardinality - Even distribution

---

## 7 REPLICATION & HIGH AVAILABILITY

### PostgreSQL

- Primary → Replica
- Read replicas

### MongoDB

- Replica Sets
  - Automatic failover
- 

## 8 CACHING STRATEGY (DB + REDIS)

### Golden Rules

- Cache **reads**, not writes
  - Invalidate on update
  - TTL always
- 

## 9 COMMON PRODUCTION ANTI-PATTERNS

✗ No indexes ✗ OFFSET pagination at scale ✗ Over-joining ✗ Large transactions ✗ Using MongoDB like SQL

---

## 10 FINAL DB ARCHITECT MINDSET

- Queries shape data
  - Indexes shape performance
  - Schema shapes queries
  - Scaling is planned, not added
- 

 This document completes DBMS mastery beyond CRUD and queries.