🧠 Core Idea

"There's no perfect answer — only trade-offs."

in @ankitrathi

Designing Data Systems

It's All About Trade-offs

🔄 Operational vs Analytical Systems

OLTP (Operational)

- Frequent writes
- Small queries
- Real-time transactions
- Use: Banking apps, Order systems

OLAP (Analytical)

- Complex reads
- Historical data
- Batch processing
- Use: Dashboards, BI tools

☁️ Cloud vs Self-hosted

Cloud-native

- Pay-as-you-go
- Elastic scaling
- Separates compute & storage
- Quick to deploy

Self-hosted

- More control
- Requires maintenance
- Can be cheaper long-term

🌐 Distributed Systems

When needed: scale, fault tolerance

Challenges: latency, consistency, complexity

Tip: Don't go distributed unless you must!

🔐 Privacy & Compliance

Driven by regulations (e.g. GDPR)

Not just legal — it's ethical

Hard to translate rules to code

Key Takeaway:

Architecture is a series of informed trade-offs. Make them wisely.

What Makes a Good System?

Think beyond features!

We need systems that are:

- Fast (Performance)
- Scalable
- Reliable
- Maintainable
- These are called Non-Functional Requirements (NFRs)

Performance

- Measure with response time percentiles (not just averages!)
- Track throughput (requests/second, etc.)
- Define SLAs (Service Level Agreements)
- Example:
- 95% of API calls finish under 200ms

Key Takeaway:

"Great systems are not just functional — they're flexible, reliable, and built to grow."

Scalability

- Goal: Handle more load without breaking
- Break down tasks into independent parts
- Scale horizontally (more machines, services)
- Analogy: Like a pizza shop hiring more chefs to make pizzas faster

Reliability

- Use fault tolerance: system keeps working even if parts fail
- Hardware vs. Software faults
- Human errors? → Build tools to recover gracefully
- Use blameless postmortems to learn from failures

Mistakes happen. Learn, don't blame!

Maintainability

- Support operations teams
- Manage complexity → Use clean abstractions
- Enable easy updates over time
- Build with modular, well-known components

@ankitrathi

Good Data Systems

Meeting Non-functional Requirements

A Tour of Data Models

With Query Languages

@ankitrathi

📦 Relational Model

📊 Used in analytics & data warehousing
⭐ Star & ❄️ Snowflake schemas
Strong SQL support
Structured, reliable, and still going strong after 50+ years.

🔗 Graph Model

🌐 Highly connected data
Supports recursive queries
Used with Cypher, SPARQL, Datalog
When everything is related to everything

📄 Document Model

📑 JSON-based, self-contained docs
Best when relationships between docs are rare
Great for flexible, nested data

🧮 Dataframes

🧠 Wide tables with many columns
Perfect for ML & statistical computing
Bridge between databases and machine learning

🔁 Emulating Models

Models can mimic each other
But: may get awkward (e.g., recursion in SQL)
Possible but not always elegant

🧩 Polyglot Databases

Databases expanding across models
Relational → JSON
Document → Joins
SQL → Graph
The lines are blurring

📚 Event Sourcing & CQRS

📜 Append-only log of events
🔄 Materialized views for querying
Write-fast, read-smart pattern

🔗 Schema: Explicit vs Implicit

- Relational: schema on write
- Document: schema on read

Flexibility vs control

🧬 Specialized Models

🧬 GenBank: for DNA string search
💰 Ledgers: double-entry accounting
🔗 Blockchains: distributed ledgers
🔍 Full-text search & vector search
Niche needs, custom tools

Key Takeaway:
Different models for different minds
The key is choosing what fits best

📍 OLTP vs OLAP

OLTP = Fast small reads/writes

- *Indexed access (Primary / Secondary)*
- *Handles user-facing apps*
- *Uses B-Trees*

💡 *e.g. Shopping cart updates*

OLAP = Complex analytical queries

- *Scans large data volumes*
- *Columnar storage + compression*
- *Used in reporting, BI tools*

💡 *e.g. Sales dashboard analytics*

Storage & Retrieval

How Databases work?

in @ankitrathi

🗺️ Index Types

Multidimensional Indexes (R-Trees)
For spatial queries (e.g., lat-long)
🗺️ *"Find all stores near me"*

Full-Text Search Index

🔍 *Finds docs by keywords*
📚 *Used in blogs, articles*

Vector Search (Semantic)

🤖 *For similarity search (AI/ML)*
🧠 *Compares high-dimensional vectors*
"Find docs like this one"

📁 Storage Engines

Log-Structured Storage

- ✅ *Append-only*
- ✅ *High write throughput*
- 🧱 *Examples: LSM Trees, SSTables, RocksDB*
- 🔁 *Deletes handled by compaction*

Update-in-Place Storage

- ✅ *Overwrites fixed-size pages*
- ✅ *Better read latency*
- 📘 *B-Trees dominate relational DBs*
- 👉 *Use case decides the engine choice!*

💡 Developer Wisdom

- 🔧 *Understand storage = Better tuning*
- 🛠️ *Choose DB based on read/write pattern*
- 📚 *Now you can decode DB docs with confidence!*

1 Why Encoding Matters

Converts data structures → bytes (for network or disk)

Influences performance, compatibility, and application design

Key for evolvability (changing systems safely over time)

2 Rolling Upgrades Need Compatibility

Deploy new versions gradually (rolling upgrades).

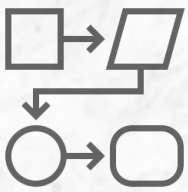
Old and new systems must understand each other's data

Goal:

- ✅ Backward compatibility (new reads old)
- ✅ Forward compatibility (old reads new)

3 Encoding Types & Trade-offs

Type	Pros	Cons
🔤 Language-Specific	Easy to implement	Tied to language, poor compatibility
📄 Text (JSON, XML, CSV)	Human-readable, flexible schema	Loose types, verbose
📦 Binary (Avro, ProtoBuf)	Compact, strict schema, good for APIs	Not human-readable



4 Where Encoding Matters (Dataflow)

- 🗂️ Database: Encode → store → decode
- 🌐 APIs: Encode request → decode/encode response
- 📩 Events: Messages sent between services

✅ Final Nugget

Design for compatibility from Day 1 and make evolution painless

Encoding & Evolution

How data flows?

in @ankitrathi

Replication

Making Data Available Everywhere

@ankitrathi

Why Replicate?

- High Availability - Survive machine/region failures
- Disconnected Operation - Keep apps running offline
- Low Latency - Keep data close to users
- Scalability - Handle more reads

Challenges

Not just copying — must handle:

- Node failures
- Network issues
- Bugs & data corruption
- Replication lag

Replication Types

Single-Leader

Writes go to one leader
Followers replicate
Easy, but stale reads possible

Multi-Leader

Multiple leaders accept writes
Conflicts need resolution
Good for multi-region writes

Leaderless

Write to multiple nodes
Read from many nodes
Needs conflict detection

4. Consistency Models

- Read-After-Write - See your own write
- Monotonic Reads - No time-travel
- Consistent Prefix - Replies don't come before questions

5. Conflict Resolution

- Version vectors - Track concurrent writes
- Last Write Wins - Easy, but risky
- Manual resolution - Needs dev input
- CRDTs - Auto-merge friendly data types

Closing Note

Replication isn't just copying. It's designing for resilience, performance, and peace of mind

Sharding

Scaling Data Across Machines

@ankitrathi

Why Sharding?

Too much data for one machine
Distribute data + query load
Avoid hot spots (overloaded nodes)
Enables horizontal scalability

Two Main Sharding Strategies

🔑 Key Range Sharding

Data sorted by key
Each shard owns a range
Pros: Fast range queries
Cons: Risk of hot spots
Rebalance: Split large ranges

Hash Sharding

Apply hash(key)
Shards own hash ranges
Pros: Even load
Cons: No efficient range queries
Rebalance: Move entire shards

Composite Keys

Use prefix of key to pick shard
Use suffix for sorting within shard
Still supports range queries per prefix

Secondary Indexes

🔗 Local Secondary Index

Stored in same shard as data
✅ Efficient writes
❌ Reads hit all shards

🌎 Global Secondary Index

Indexed values are sharded separately
✅ Reads from one shard
❌ Writes touch many shards

Query Routing

Coordinator keeps shard-to-node map
Routes queries to correct shard

Challenges

Shards work independently
But multi-shard writes can fail
What if one succeeds & another doesn't?
❓ Solution: Covered in next chapter

🧱 Key Takeaway

Sharding lets us scale data systems...
...but choosing the right strategy is key to performance, flexibility, and fault tolerance.

Follow: @ankitrathi

Ankit Rathi (He/Him)

I simplify Data & AI through Visual Storytelling – One Concept at a Time | All views are my own

Noida, Uttar Pradesh, India · Contact info

Check My Visual Notes

Bookmark: Access all Live Notes here

https://github.com/ankit-rathi/The-Data-AI-Visual-Notebook

40+ Visual Notes for FREE