# 🏆 Everything You Need to Know About Schema & Indexes for Interviews

😵 **Interviewer : Mr. Pessimo, the Skeptic**

*Can you design TwoFaTwoSh in 30 minutes?*

😋 **Interviewee : Mr. Optimo, the Persister**

*Oh Yes! Let's go for it.*

## 🌱 Humble Beginnings: Relational Tables

- **The Birth of TwoFaTwosh (FastShopFastShip):** A simple e-commerce system.
- **3NF Design:** Orders, Products, Users tables.
- **Normalization Benefits:** Avoiding redundancy, transitive relationships.
- **1**:N* Relationship:** Introduced `order_items` to link Orders & Products.
- **Indexing for Efficiency:**
  - **Primary Keys (PK)** for unique identification.
  - **Composite Indexes** for optimized queries.
  - **Covering Indexes** to minimize lookups.

## 📌 Indexing Types and Their Uses

### Relational Database Indexes

- **Primary Index:** Auto-created on PK, used for fast lookups.
- **Clustered Index:** Physically sorts table rows, efficient for range queries.
- **Non-Clustered Index:** Stores pointers to actual rows, supports multiple indexes.
- **Composite Index:** Indexes multiple columns for specific query patterns. Supports sorting.
- **Covering Index:** Includes all columns needed for query execution, reducing lookups.
- **Full-Text Index:** Optimized for searching text fields (e.g., MySQL, PostgreSQL).
- **Pessimo:** "Normalization is great, but what about JOIN performance at scale?"
- **Pessimo:** "Indexes are good, but don’t they slow down writes?"
- **Pessimo:** "Okay, but what happens when your database starts struggling at scale?"

## 📈 Scaling Up: The Latency Bottleneck

- **Challenges:** Increasing query response time, read/write contention.
- **Solutions:**
  - **Single Table Design in DynamoDB**
    - PK & **Static Sort Keys** for entity identification queries.
    - **Hierarchical Relationships** modeled using sort keys.
  - **LSM Trees & SortedSet Tables** for high-throughput writes.
  - **Global Secondary Indexes (GSI), Local Secondary Indexes (LSI)** for flexible queries.

## 🔍 Search Optimization with Elasticsearch

- **Why Elasticsearch?**
  - Full-text search, ranking, and filtering.
  - Complementing DynamoDB's key-value pattern.
- **Indexing Strategies:**
  - **LSM Indexing** for rapid ingestion.
  - **Optimized Queries** leveraging precomputed indices.

### NoSQL Indexes (DynamoDB, Cassandra)

- **Partition Key (PK):** Used for fast lookups, distributes data.
- **Sort Key (SK):** Enables range queries within a partition.
- **Global Secondary Index (GSI):** Enables queries on non-PK attributes.
- **Local Secondary Index (LSI):** Allows additional sort key on the same partition.

### Search Engine Indexing (Elasticsearch, Solr)

- **Inverted Index:** Efficient for text searches, maps terms to documents.
- **SortedSet Index:** Maintains sorted values for range queries.
- **LSM Tree Index:** Log-structured merges optimize write-heavy workloads.

## 🎭 Pessimo Strikes Again

- **Pessimo:** "Why did you introduce a GSI? Isn't that costly?"
- **Pessimo:** "Can your system handle millions of writes per second?"
- **Pessimo:** "How would you optimize Elasticsearch indexing for real-time queries?"

- **Pessimo:** "Why not just use a relational database for everything?"
- **Pessimo:** "A covering index sounds cool, but does it work for every query?"

## ⚖️ Trade-offs in Indexing

✔ **Pros:** Faster lookups, scalable, reduced read contention. ❌ **Cons:** Increased storage, more complex writes, eventual consistency in search.

## 🛠️ Key Takeaways

- **Schemas**: How to design a schema in various technologies?
- **Indexing Patterns:** Different types of indices and when to use what? The pitfalls.
- **Search & Scalability:** Using Elasticsearch alongside structured data.
- **Real-World Performance Tuning:** Choosing the right index for the right workload.