**Databases**

Types:

* Relational
* Non-relational – Key/Value stores, column based DBs, Document based DBs, Search DBs
* File
* Network, etc.

**Relational DB**

Schema: Structure of data.

Constraints: Must follow rules by data

If data can be stored in form of table and rows, use relational DB.

Ex: Employee management system, banking, etc.

Pros:

* Database for interrelated complex data can be designed easily.
* By putting constraints, we ensure that null or garbage value is not stored
* Ensures all other schema constraints are followed
* Follow ACID

**ACID Property**

Atomicity:

* A transaction should happen completely or don’t happen at all.
* Relational DBs make sure their transactions are atomic

Consistency

* If two people trying to read same data, both request must have same response.
* Relational DBs should be consistent

Isolation

* Two transactions should not know about each other.
* Read and Write should be independent, and we get respective values on reading and writing as per timestamp of transaction. Read first or Write first

Durability

* Guarantees that transactions are logged properly and saved permanently in disk storage.

**Cons**

* Cannot be used if we are unaware of columns in the table that can come up as the product evolves.
* There are ways to update table but, it becomes difficult when product scales to large extent and new columns keep adding or updating
* When we try to fetch data from multiple tables in huge tables, Join Operations can become expensive
* Easy to scale vertically, Horizontal scaling is difficult in relational DBs

**Non-Relational DBs**

**Key Value Store:**

* Stores values as key-value like request-response, etc.
* Very useful in caching solutions.
* Ex: Redis, DynamoDB, MemCache, etc.

**Document Based:**

* No fixed schema, can be used when not sure how it will evolve over time
* They can support heavy reads and writes.
* They have collections (like tables) and documents (like rows)
* Might have null values, need to handle in code.
* They do not provide ACID transactions, can be handled in application code.
* Highly Scalable, Sharding, Query optimization.

**Column Based**

* Mid way between relational and document based.
* There is fixed schema in form of columns but do not support ACID transactions
* Used when we require heavy reads like event or streaming data (music player)
* All actions performed on a song (like, change, pause) anything is stored as event data so that analytics can be derived from it later.
* Health tracking data, IOT Sensor data.
* Heavy writes, special reads (table structure defined by kind of queries you have to make, Ex: in music player, tables can be – Users, Songs, SongsLikedByUsers, UsersBySongsLiked, etc)
* Support distributed Database
* Ex: Cassandra, HBase, Sylla.

**Search DBs**

* Booking flight, movie, searching on amazon. All those full text search queries are stored in SearchDBs.
* Data against queries are stored in advanced indexes In Search DB.
* Ex: Elastic DB, etc.
* Results of search queries or data on which frequent queries are executed is stored in Search DBs. The main relational DB of the product will still be there.
* Data in search DB is refreshed as per frequency of queries.