***Sharding Technique:***

we cannot build a scalable model without understanding the **Sharding Technique.**

The Sharding is technique of dividing the large DB into small DB. Each Small DB is called Shards.

There is multiple technique of dividing the DB in small shards.

1. **Horizontal Sharding**:

It is range based partitioning. Example: Table Location. If we have divided DB into 3 Shards. If Pin Code is less then 5K, we will store Location in 1st Shards, for Pin Code between 5k-10k in second Shards and grater than 10k in Third Shards.

This kind of Sharding is called Horizontal Sharing.

Problem: If the field/column based on which horizontal Sharding is done, in exam pin code, And If that field is not chosen properly.

For Example : If Pin Code is always grater than 10k, then all the load/request will be coming to 3rd partition only

1. **Vertical Sharding:**

In this Sharding, DB is divided based on the feature.

For exam: User Module or feature: 1 DB, For AAIS 1 DB, For IP\_FootPrint another DB etc.

Problem : If any one feature is widely used as compared to other features ,in that case DB with widely used feature can have more request coming in as compare to other DB.

So If application experiences the additional growth, than it may be necessary to further divide or partition feature specific DB.

1. **Directory based Sharding: Lookup Server**

**It** is widely used Sharding technique. It is very efficient in rebalancing.

Here one directory server is kept in front of all partitioned DB. This Directory server is server which known current partitioning scheme.

So Directory server know where particular data resides. So First we connect to directory server, then directory server will lookup data/table in any of the existing partitioned DB.

1. **Hashing based Sharding:**

This is very rarely used Sharding, where partitioning is done based on hash function.

Problem: If we need to add few more DB partition, then we need to change Hash function also.

**Sharding cons :**

1. **Joins and Denormalization :** One DB is partitioned and spread across multiple server, It is not feasible to perform Join operation.
2. **Referential Integrity:** If one table is present in one partition in any server, and foreign key table resides in another machine or db in different server. Then referencing becomes very complex in case of Sharding.

**SQL/ NOSQL**

SQL :

It has schema, fixed table structure, Normalization concept like every piece of data stored in DB only once, and other places are connected/uses any other data via relationship etc.

Limitation:

1. Number of columns in a table is fixed. As it has fixed schema, then it is not possible that for one table, one row has different number of columns and other row has different number column.
2. We need to have a clear schema to relation DB table. And some time because of these relations, SQL becomes too complex and slow.

**No-SQL : No fixed Schema**

1. Used in Scalable product as it can store a huge amount of data.

Type:

1. **Key-value DB** : Redis uses this kind of the DB

How does a key-value database work?

A key-value database, aka *key-value store*, associates a value (which can be anything from a number or simple string, to a complex object) with a key, which is used to keep track of the object. In its simplest form, a key-value store is like a dictionary/array/map object as it exists in most programming paradigms, but which is stored in a persistent way and managed by a Database Management System (DBMS).

Key-value databases use compact, efficient index structures to be able to quickly and reliably locate a value by its key, making them ideal for systems that need to be able to find and retrieve data in constant time. Redis, for instance, is a key-value database that is optimized for tracking relatively simple data structures (primitive types, lists, heaps, and maps) in a persistent database. By only supporting a limited number of value types, Redis is able to expose an extremely simple interface to querying and manipulating them, and when configured optimally is capable of extremely high throughput.

1. **Document DB:**

A document database is a type of nonrelational database that is designed to store and query data as JSON-like documents. Document databases make it easier for developers to store and query data in a database by using the same document-model format they use in their application code. The flexible, semistructured, and hierarchical nature of documents and document databases allows them to evolve with applications’ needs. The document model works well with use cases such as catalogs, user profiles, and content management systems where each document is unique and evolves over time. Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents.

Reference: <https://www.documentdb.com/sql/tutorial>

1. **Graph DB:**

**CAP Theorem :**

**C-Consistency:** A system will be consistence if Just after the write operation, for next read operation it provides correct results.

**A-Availability:** Every Request should get response, either success or failure weather you have high load or network failure, System should alwaays be available.

**and P-Partition tolerance:**  A System continue to work despite to message loss or partial failure. A partition tolerance System can sustain any number of network failure and does not result in entire network failure.

Means we will always have a backup server to keep the server up in case of any discrepancy.

As per this theorem, Any distributed system, we can achieve any of two property, we can’t achieve all 3 property in one System.

Exam : SQL Server , MYSQL , Oracle, PostgreSQL => Consistency, Availability

Cassandra , CouchDB, DynamoDB => Availability, Partition tolerance

MongoDB, Redis => Consistency, Partition tolerance