SKILL<S/ASH>

SYSTEM DESIGN
CHEAT SHEET

Primary Database

Primary database is your single source of truth when you save data. You always rely on this data.

Which data base to choose?

We always get confused with the database to use as our primary database. Here are some points to keep in mind to make a better decision.

SQL Database

- Relational, Table-based
- When you require ACID (Atomicity, Consistency, Isolation, and Durability) properties.
- You choose SQL Database for saving financial transactions which gives you ACID guarantee that multiple records belongs to the same transaction are saved either all or nothing.
- You choose SQL Database when rigid schema is required. For e.g. you want to maintain strict relationships between the tables. you want to maintain unique, not_null constraints.
- You choose SQL Database when records are updated very frequently.
- You choose SQL Database when Joins and complex queries are required.
- SQL Databases are: Oracle, MySQL, MariaDB, Microsoft SQL Server, and PostgreSQL

NoSQL Database

- Document-based, key-value pairs, Graph database
- You choose NoSQL Database when flexible schema is required. For e.g. e-commerce like Amazon can have Products in varied schema. Electronic products can have capacity (lt), power-rating (3- star) whereas Clothing products can have size (S,M,L,XL), material (cotton).
- You choose NoSQL when horizontal scaling is required. They scale very well.
- NoSQL Databases are: MongoDB, BigTable, Redis, RavenDb, Neo4j, and CouchDb

Scalability

As your company grow, data grow and scalability comes into picture:-

Vertical Scaling

1. Database can be scaled vertically by adding more power (CPU, RAM or SSD) to same machine, so that database can handle more load.
2. Vertical scaling has limitations as can not add power to same machine beyond a point.
3. SQL Databases are often scaled vertically

Horizontal Scaling

1. Databases can be scaled horizontally by adding more machines into the pool of databases. Data is saved in pool of databases using **Database Sharding** or also known as **Horizontal Partitioning**
2. Database sharding is easier in NoSQL databases as compare to SQL databases.

Database sharding means partitioning of data in multiple databases. Incoming data can be saved in databases based on some key. for e.g. userId in case of saving user profile.

total database in horizontal partitioning = 5 [DB_1, DB_2, DB_3, DB_4, DB_5]
consistent hashing = 5 databases makes a consistent hash ring

DB_1

↗ ↘

DB_5 DB_2

↖ ↙

DB_4 ⟵ DB_3

write user_1

hash(userId_1) = 1002
database to write this record = 1002%5 = 2 (DB_2)

write user_2

hash(userId_2) = 5119
database to write this record = 5119%5 = 4 (DB_4)

read user_1

hash(userId_1) = 1002
database to read this record = 1002%5 = 2 (DB_2)

DB_4 goes down....

write user_3

hash(userId_3) = 1234
database to write this record = 1234%5 = 4 (DB_4)
DB_4 is not available, write to the next available db in consistent hash ring i.e. DB_5

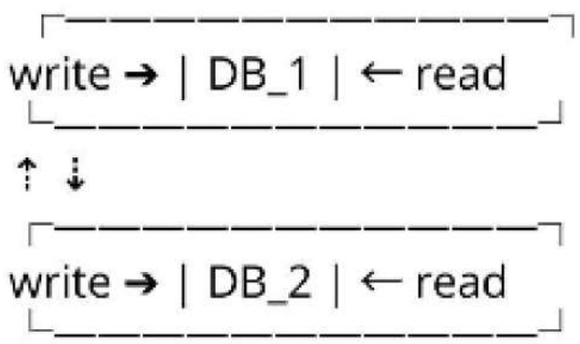
Availability

Availability is measured based on replication factor. Replication factor = 3 means a database is having 2 additional copies which keep themselves in-sync
asynchronously.

Active-Active

Data read and write happen in any database replica.

(Eventual read and write Consistency, High Availability, High Performance)



Eventual Read Consistency:-

1. record_A write request to DB_1
2. record_A read request from DB_2 immediately
record_A is not be available in DB_2 if sync is yet to happen

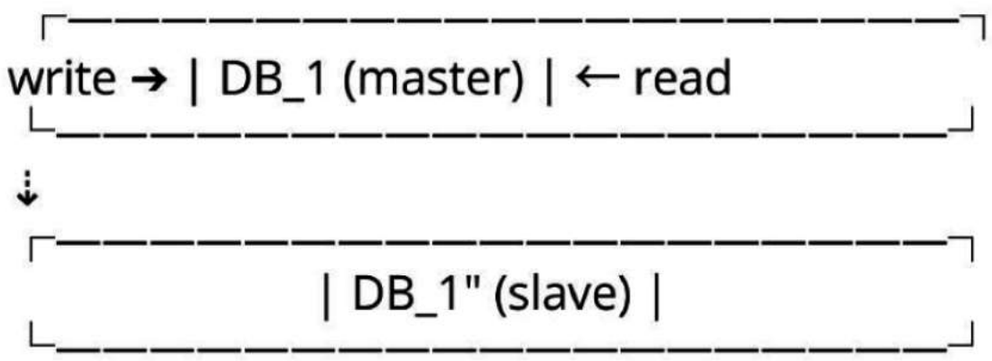
Eventual Write Consistency:-

1. record_A write request to DB_1
2. record_A update request to DB_2 immediately
record_A is not be available in DB_2 if sync is yet to happen

Active-Passive (Master-Slave)

Data read and write happen only in master database. Data is synced with slave database asynchronously. When Master goes down, Slave started serving as master.

(High read and write Consistency, High Availability, Low Performance)

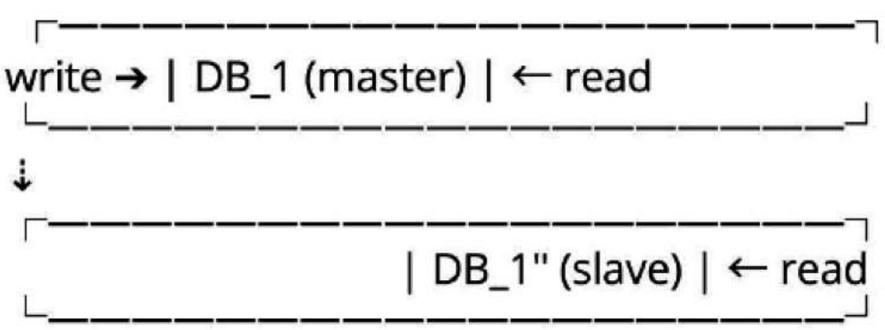


Active-Passive is generally used in financial sector where read and write consistency of financial transactions is very important.

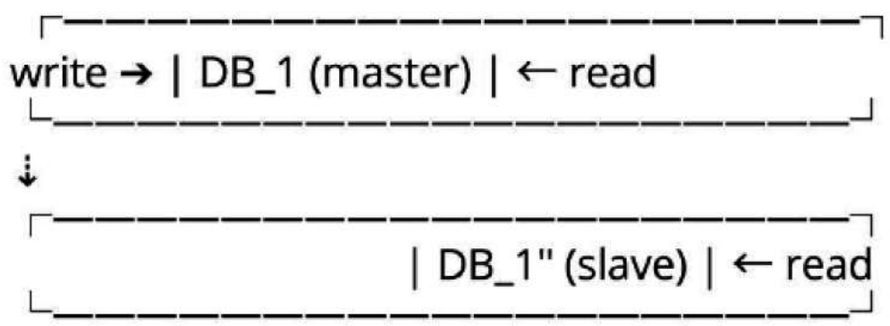
Write-Read (Master Slave)

Data write happen in master database. Data read happen from any database.

(High write and eventual read Consistency, High Availability, Low write and High read Performance)



Write-Read is generally used in social media such as Facebook, LinkedIn, Twitter, Instagram where read/write ratio is very high and eventual read consistency is acceptable.



Write-Read is generally used in social media such as Facebook, LinkedIn, Twitter, Instagram where read/write ratio is very high and eventual read consistency is acceptable.

Cluster

When data is partitioned across multiple database

total database in cluster 5 [DB_1, DB_2, DB_3, DB_4, DB_5]
replication factor = 3
consistent hashing = 5 databases makes a consistent hash ring

DB_1
↗ ↘
DB_5 DB_2
↖ ↙
DB_4 ⟵ DB_3

Data write for DB_1 is copied to DB_2 and DB_3 (next 2 databases in the ring)
Data write for DB_2 is copied to DB_3 and DB_4 (next 2 databases in the ring)
...
...
Data write for DB_5 is copied to DB_1 and DB_2 (next 2 databases in the ring)

In consistent hash ring, next two databases either can contain 100% data copy of previous database or share the 50%-50% data copy depending upon the requirement.

Geo Sharding

Geo sharding is nothing but database sharding based on geo location.

Use-case

1. It is used by ride hailing apps such as Uber, Grab to store live locations of drivers and passengers.
2. It is also used by tinder and other dating sites to provide recommendations based on location.

How to do?

1. Divides the geo locations into cells
   (for e.g. 1 cell = 100x100 miles square).
2. You get the live location (coordinates lat, long) of user and pass to S2 library, which returns the cell_id in which that location falls into.
3. Use this cell_id as a key to partition the data.

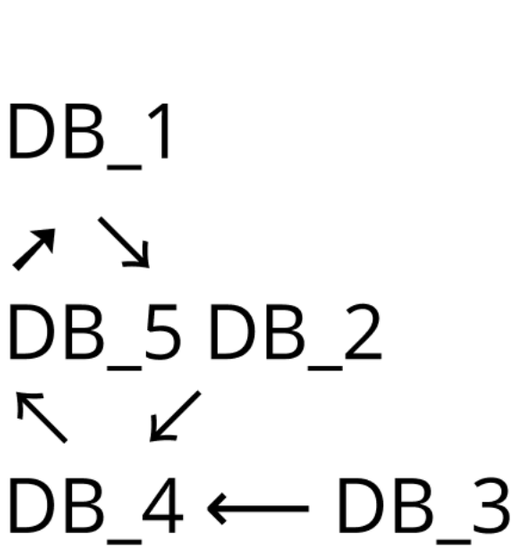
Consistent Hashing

Consistent hashing is the algorithm used by -

1. Most of the load balancers to distribute the load across multiple services.
2. Most of the database horizontal partitioning to distribute the data across multiple databases.

The idea of consistent hashing is to place all the services (or databases) across the consistent hash ring and distribute the load based on some hashing algorithm. This gives us the flexibility to remove or add the service (or database) to the ring without disturbing the whole cluster with minimum or no data loss.

total database in cluster = 5 [DB_1, DB_2, DB_3, DB_4, DB_5]
consistent hashing = 5 databases makes a consistent hash ring



- If DB_2 goes down, Next database in the ring i.e. DB_3 is going to take additional load (future writes). If replication is not implemented then existing data of DB_2 will be lost but other database will still work.
- If DB_6 is added newly between DB_5 and DB_1 then it share the 50% load of the next database in the ring i.e. DB_1.

File Storage Options (Image/Video)

Scalability

E-Commerce like Amazon, Social Media like Instagram, Facebook, Twitter, Streaming Provider like Netflix requires to store a huge amount of Images and Videos.
Images and Videos are stored a BLOB (Binary Large Object) and Database Storage is not a good option for this kind of storage. Why?

1. You are not going to query on BLOB Storage
2. You do not need any ACID guarantees for BLOB storage as such provided by RDBMS
3. Costly to store in Database

Cheaper and scalable options to store such files are Distributed File Systems (DFS) such as Sharepoint, Amazon S3.
Details about the image/video such as its DFS URL, metadata can be save in Database, referred by Image ID.

ID	URL	METADATA	IMAGE_THUMBNAIL/VIDEO_COVER_PHOTO

It is also a smart idea to save thumbnail of image, or cover photo of the video in database, size of which should not exceed few KBs. Initially the thumbnail or cover photo can be sent to the client along with the URL of image/video. It is a good experience for the user to see something while image/video is downloading/streaming from DFS.

Availability

You may want to use Content Delivery Network (CDN) which distribute the same images/videos to different locations geographically near to the client locations.

Caching

You need to use Caching at some point of time in your system design for faster read.

Few famous Caching solutions are -

1. Redis(recommended)
2. Memcached
3. Hazelcast

Text Search Capabilities

Almost all the websites requires searching capabilities in one way or another. For e.g. Amazon for product search, Netflix for movies search, Youtube for video search, Social media websites for user profile search, and so and so forth. Some of the popular text search engine -

1. Elastic Search(recommended)
2. Apache Solr
3. Apache Lucene

- Apache Lucene is the core framework written based on Map-Reduce and Inverted-Index algorithm.
- Elastic Search and Apache Solr has been written on top of Apache Lucene provide additional search capabilities such as fuzzy search, type ahead, search suggestions.
- You don’t rely on these for storage. You generally save data in your primary database and write to Elastic Search or Apache Solr asynchronously.

Reporting and Analytics

Data warehouse where you generally dump a lot of data and they provide very complex querying capabilities.

1. Apache Hadoop
2. Amazon Redshift
3. Elastic Search

Top Hashing Algorithms

Consistent Hashing

Consistent Hashing is used to route the request to distributed resources. For e.g.

- Load Balancer use Consistent Hashing algorithm to route the request to specific Application Server.
- Used in database sharding (also known as horizontal partitioning) to save the date in specific shard.

Bloom Filter

Bloom filter takes the constant time and space to answer Yes/No to this kind of questions:-

1. Does this username exist or already taken?
2. Is this item exist in the set?

Bloom filter gives:-

1. 100% accurate result when it says “No”
2. Probabilistic result when it says “Yes”. Sometimes it might be false positive.

Bloom filter data-structure is represented as an array of bits like this:-

Index	0	1	2	3	4	5	6	7
8 Bit Array	0	0	0	0	0	0	0	0

Case 1

Check username “John” exist?

hash1(John) = 100 => 100%8 = 4
hash2(John) = 54 => 54%8 = 6

Since 4th and 6th bit of our array is not set, “John” doesn’t exist for sure. Set the bits 4th and 6th bit now:-

Index	0	1	2	3	4	5	6	7
8 Bit Array	0	0	0	0	1	0	1	0

Case 2

Check username “Bill” exist?

hash1(Bill) = 81 => 81%8 = 1
hash2(Bill) = 31 => 31%8 = 7

Since 1st and 7th bit of our array is not set, “Bill” doesn’t exist for sure. Set the 1st and 7th bit now:-

Index	0	1	2	3	4	5	6	7
8 Bit Array	0	1	0	0	1	0	1	1

Case 3

Check username “Garry” exist?

hash1(Garry) = 89 => 89%8 = 1
hash2(Garry) = 90 => 90%8 = 2

Since 1st bit is set but 2nd bit is not set, “Garry” doesn’t exist for sure. Set the 2nd bit now:-

Index	0	1	2	3	4	5	6	7
8 Bit Array	0	1	1	0	1	0	1	1

Case 4

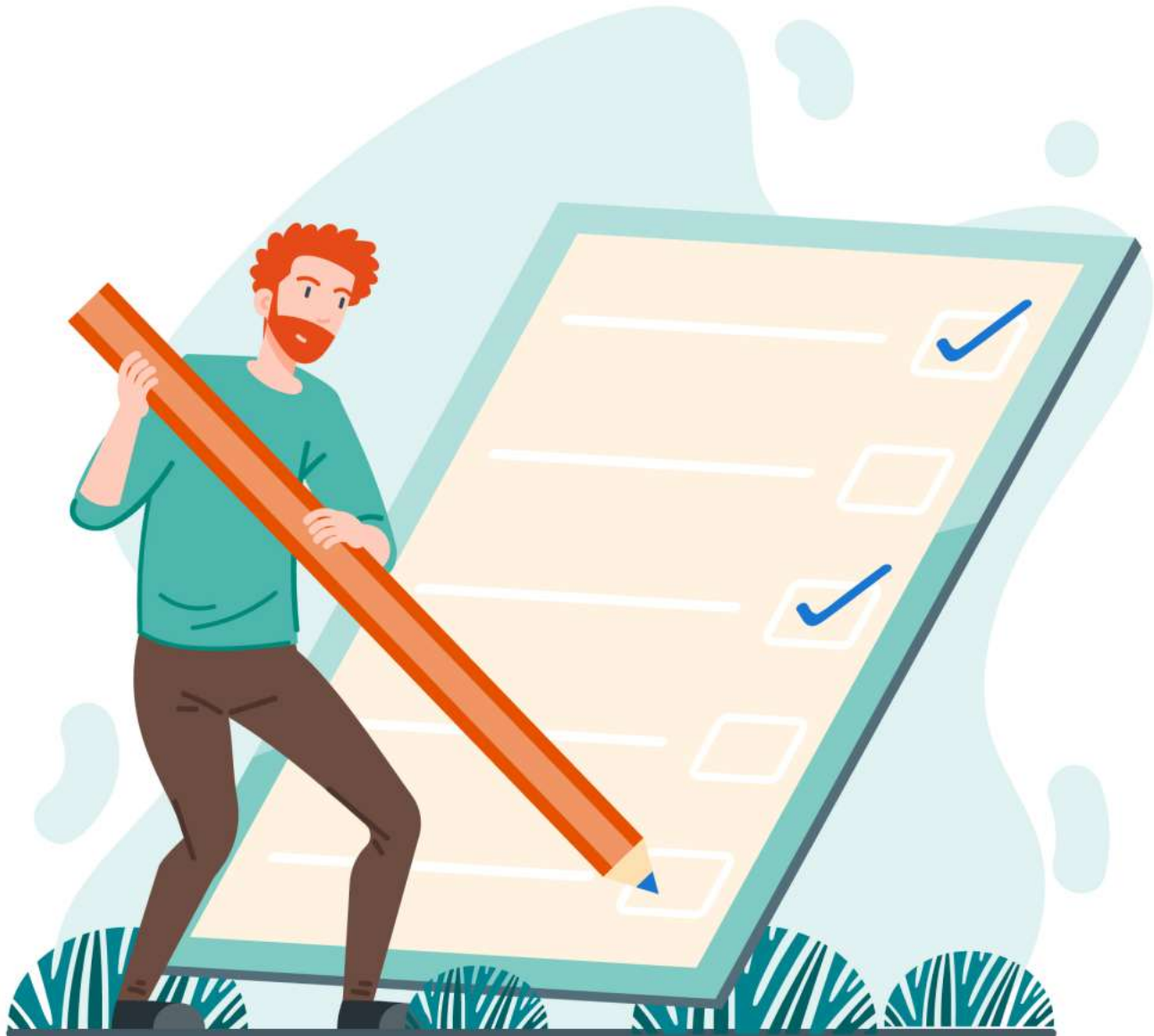
Check username “Jack” exist?

hash1(Jack) = 89 => 89%8 = 1
hash2(Jack) = 90 => 90%8 = 2

Since both 1st and 2nd bit are set, “Jack” may or may not exist. Check the database.

Conclusion

1. Number of bits takes very less space and in practice numbers of bits in array can be huge providing more precise results. We have used only 8 bits in the example
2. Bloom filter provide 100% accurate result if user doesn’t exist
3. Bloom filter provide estimated result if user exist. Estimated result is more precise if hash function is evenly distributed and more hash functions are used. We have used only 2 hash functions in the example



SKILL<S/ASH>

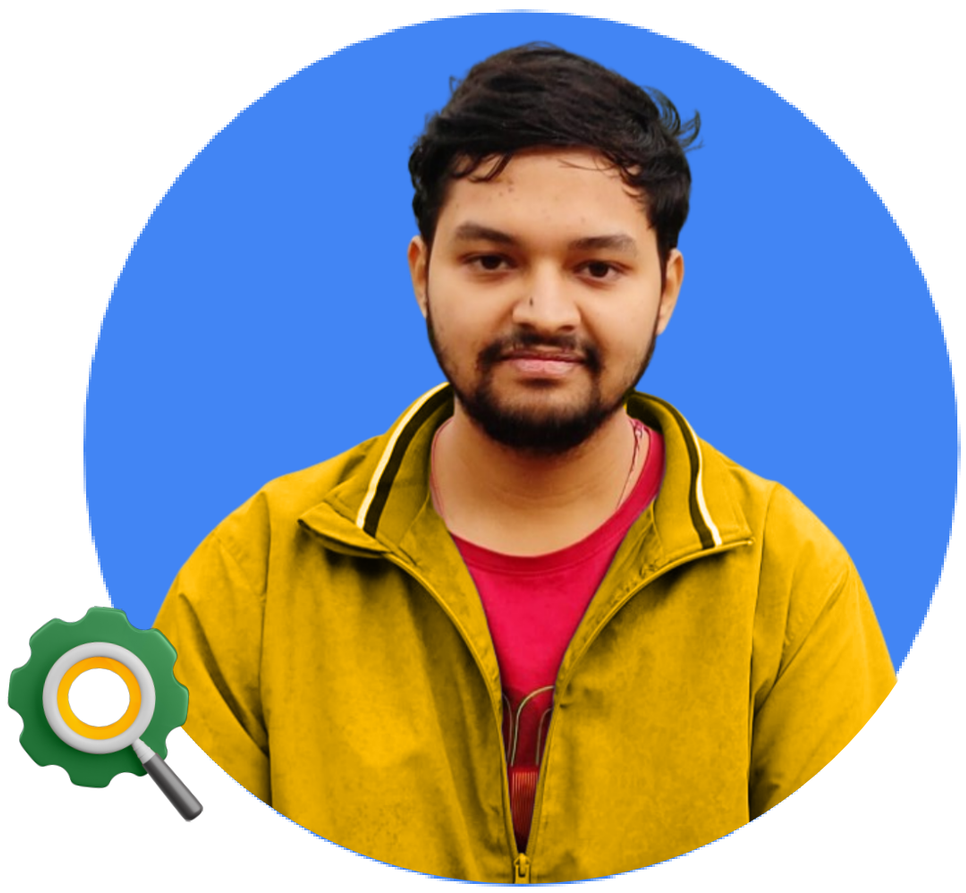
Architecture of a Search Engine

Better search through architecture

FREE LIVE MASTERCLASS

7 th Feb, 2023

8pm - 9:30pm



Register Now

Akshat Bhargava

Sr.SDE Google Cloud