**Vertical partitioning vs horizontal partitioning**

[[A diagram of a database

Description automatically generated](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fbucketeer-e05bbc84-baa3-437e-9518-adb32be77984.s3.amazonaws.com%2Fpublic%2Fimages%2Fa0018b6a-0e64-4dc6-a389-0cd77a5fa7b8_1999x1837.png)](https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fbucketeer-e05bbc84-baa3-437e-9518-adb32be77984.s3.amazonaws.com%2Fpublic%2Fimages%2Fa0018b6a-0e64-4dc6-a389-0cd77a5fa7b8_1999x1837.png" \t "_blank)

Horizontal partitioning is widely used so let’s take a closer look.

**Routing algorithm**  
The routing algorithm decides which partition (shard) stores the data.  
  
🔹 Range-based sharding. This algorithm uses ordered columns, such as integers, longs, timestamps, to separate the rows. For example, the diagram below uses the User ID column for range partition: User IDs 1 and 2 are in shard 1, User IDs 3 and 4 are in shard 2.  
  
🔹 Hash-based sharding. This algorithm applies a hash function to one column or several columns to decide which row goes to which table. For example, the diagram below uses **User ID mod 2** as a hash function. User IDs 1 and 3 are in shard 1, User IDs 2 and 4 are in shard 2.  
  
**Benefits**  
🔹 Facilitate horizontal scaling. Sharding facilitates the possibility of adding more machines to spread out the load.  
  
🔹 Shorten response time. By sharding one table into multiple tables, queries go over fewer rows, and results are returned much more quickly.  
  
**Drawbacks**  
🔹 The order by operation is more complicated. Usually, we need to fetch data from different shards and sort the data in the application's code.  
  
🔹 Uneven distribution. Some shards may contain more data than others (this is also called the hotspot).

What is the shard key here?

When using mysql here,

1. It’s important if we use this

What happens in the event of a network failure? Well you can choose between the 2. You can choose

* 1. High consistency comes at the cost of lower availability.
* 2. High availability comes at the cost of lower consistency.

And then here we have the code:

Consistency means having the most update to date informatino here

1. This is important

2.

The following image represents what databases guarantee what aspects of the CAP Theorem simultaneously. We see that RDBMS databases guarantee consistency and Availability simultaneously. Redis, MongoDB, Hbase databases guarantee Consistency and Partition Tolerance. Cassandra, CouchDB guarantees Availability and Partition Tolerance. [**Complete Video Tutorial**](https://www.youtube.com/watch?v=8UryASGBiR4).

What’s acid in sql?

Sql vs nosql

ACID is an acronym that refers to the set of 4 key properties that define a transaction: Atomicity, Consistency, Isolation, and Durability. If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems. ACID transactions guarantee that each read, write, or modification of a table has the following properties:

Atomicity - each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed. This property prevents data loss and corruption from occurring if, for example, if your streaming data source fails mid-stream.

Consistency - ensures that transactions only make changes to tables in predefined, predictable ways. Transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table.

Isolation - when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously.

Durability - ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

How to scale nosql db?

How to choose between

**Why Using 2 phase commit protocol?**

We have now decided to scale our database, to cater to increasing customers. Data is distributed across multiple database servers. So, user A and user B’s database records may fall in different shards.

A diagram of a diagram

Description automatically generated

If 1 sql query runs and then fails that’s not good for us. So do sth about this here.

We have to ensure that either the transaction completes successfully or fails. We don’t want to leave the transaction midway in an inconsistent state. 2-

A diagram of a financial diagram

Description automatically generated

Who are the actors here?

Client, transaction coordinator, shard A and shard B here.

Later, the client sends a commit message to the Transaction Coordinator. The transaction commit is now divided into two phases by the Transaction Coordinator.

Phase 1: here

In the first phase, a RequestCommit the message is sent to all the participant servers. Every server has to respond to this message either with an OKor FAIL message. The server replies with an OKif it’s able to execute the transaction successfully. A FAIL message will be returned if there are any errors during the execution. For eg:- If the account balance went negative during the debit transaction.

**What’s diff between eventual and strong consistency?**

Eventual consistency here:

1. There is the problem of delayed consistency here, Typically offers higher performance and availability than strong consistency.
2. : A social media platform's distributed database that uses eventual consistency might show different users different versions of a post's like count for a short period after it's updated. Over time, all users will see the correct count.

2nd phase

The Transaction Coordinator waits for a response from all the servers. Once it receives a response, it will decide to either Commit or Abort the transaction. This becomes the second phase of the commit. The transaction will be committed only if every server replies with a OK message. If at least one server responds with a FAILmessage, the transaction will be aborted.

Drawback

1. Latency could be slow here
2. **Transaction Coordinator:**The Transaction Coordinator becomes a single point of failure at times. The Transaction Coordinator may go down before sending a commit message to all the participants. In such cases, all the transactions running on the participants will go in a blocked state. They would commit only once the coordinator comes up & sends a commit signal.

This is so important here:

**How to handle eventual consistency?**

**Part 1:**

C**onflict Resolution Mechanisms**

Conflict resolution mechanisms are used when there is a high chance of conflicting updates to the same data item in different data stores. For example, this can widely happen in e-commerce applications where multiple users update a single product simultaneously. Last-writer-wins (LWW) and vector clocks are 2 of the widely used methods in such situations to decide which version of the data should be displayed.

**1. Last-writer-wins (LWW)**

In the LWW approach, the most recent update is considered valid. For example, let’s consider a scenario where two users add the same item to the cart in an e-commerce application. If the first user’s write operation is timestamped later than the second user’s write operation, the version of the item added by the first user will be stored in the cart.

**2. Vector clocks**

The vector clock approach assigns a list of integers known as a vector clock for each data record replica. When an update is made to a data replica, the [vector clock](https://en.wikipedia.org/wiki/Vector_clock#:~:text=A%20vector%20clock%20is%20a,the%20sending%20process's%20logical%20clock.) for that replica is incremented. Then, when a replica receives an update from another replica, it can compare the vector clocks to determine which version of the data is more recent.

For more of this please refer to the design a key value store in the database here

**What does high availability mean?**

High availability refers to a database’s ability to remain operational and accessible to users without significant downtime or interruptions. To achieve high availability, a database system must be designed with redundancy and failover mechanisms and have no single points of failure. High availability is particularly important in the e-commerce, financial, and healthcare industries.

**How to achieve high availability?**

Multiple copies of the same database are maintained on different servers so that if one server fails, the database can still be accessed from another server. Different techniques are used to achieve database high availability, including primary-replica replication or multi-primary replication.Jun 22, 2023

Which is basically the same idea as the replication here.

**Part 2: How to handle replication technique?**

**Replication Techniques**

Replication techniques are used when there is a high write throughput and a low likelihood of conflicts. Here are some of the most common replication techniques to ensure eventual consistency.

**1. Master-slave replication**

In this method, a single master node is responsible for writing data, and multiple slave nodes are there to read data. Hence, updates are first written to the master node and gradually propagated to the slave nodes.

A diagram of a product

Description automatically generated with medium confidence

In this example, a Customer Care Agent is updating an Order and as part of the operation, you are expected to capture a new amount, reserve the stock, and update the shipping information at the warehouse.

**7. How is sharding different from partitioning?**

Vertical sharding splits a table into 2 or more tables containing diff columns as said

A screenshot of a computer

Description automatically generated

Sharding is just a nother type of partioning here

Basically sharding is horizontal partioninig here

The above tells you what you need to know

What's DNS?

DNS translates domain names to [IP addresses](https://www.cloudflare.com/learning/dns/glossary/what-is-my-ip-address/) so browsers can load Internet resources.

2. So no need for human to memorize anything here

3.

What's round robin DNS?

1.

1. And then here we have the code here then

2. Here we have the load balancing algorithm here

- and then we have the code

Part 1: Design the API's what does the url look like here

the params and the ressponse of the API here

**3. How to support high concurrrecy and high availiabitliy?**

- High avaliailibyt means sharding the database

- High concurrency means trasnaction needs to be done right as mentioned before some sort of locking, like no race condition here

4. What is hte best way to store files?

General answer is Amazon S3 supports same region and cross-region application. This way there won't be any actual losses here

5. What do you usually write down in the system design doc of each file?

- vocabulary

- questions or topics that's covered that's new to you, interesting stuff here s

6. How do you handle faliure in the db?

* Load balancer failure: If a load balancer fails, the secondary would become active and pick up the traffic. Load balancers usually monitor each other using a heartbeat, a periodic signal sent between load balancers. A load balancer is considered as failed if it has not sent a heartbeat for some time.
* Block server failure: If a block server fails, other servers pick up unfinished or pending jobs.
* Cloud storage failure: S3 buckets are replicated multiple times in different regions. If files are not available in one region, they can be fetched from different regions.
* API server failure: It is a stateless service. If an API server fails, the traffic is redirected to other API servers by a load balancer.
* Metadata cache failure: Metadata cache servers are replicated multiple times. If one node goes down, you can still access other nodes to fetch data. We will bring up a new cache server to replace the failed one.
* Metadata DB failure.
* Master down: If the master is down, promote one of the slaves to act as a new master and bring up a new slave node.
* Slave down: If a slave is down, you can use another slave for read operations and bring another database server to replace the failed one.
* Notification service failure: Every online user keeps a long poll connection with the notification server. Thus, each notification server is connected with many users. According to the Dropbox talk in 2012 [6], over 1 million connections are open per machine. If a server goes down, all the long poll connections are lost so clients must reconnect to a different server. Even though one server can keep many open connections, it cannot reconnect all the lost connections at once. Reconnecting with all the lost clients is a relatively slow process.
* Offline backup queue failure: Queues are replicated multiple times. If one queue fails, consumers of the queue may need to re-subscribe to the backup queue.

**Master vs slave architecture in code:**

**Using the master slave architecture here:**

How do we prevent the single point of fialure here

So single point of faliure is there for a long time here,

Using the master slave architecture

A person standing in front of a whiteboard

Description automatically generated

If the database fails

A diagram of a diagram of a different type of cylinder

Description automatically generated with medium confidence

Slave for read and then master then uses write

2. slaves but an additional task iscontrollers in this case

the masters are chosen arbitrarily. between one of these replicas and if the master fails then the replicas choose the new master

**how to deal with data sync issue in master slave database?**

You usually

There is a reason why u ! change db on the slave db as said. This is very important, only change the master node slave database

**What’s the master slave architecture in the database?**

Because the master database has the genuine data, writing takes place only there. The slave, on the other hand, is the only one who carry out read requests.

**Handling faliure in the database**

**What happens in the event of a faliure**

**Failures of the databases**

Databases are the core components of most systems. Although the probability of the the databases failure is low but it’s not zero.

Part 1 here:

**A.** **Redundancy and Replication :**Having a backup DB with all the data replicated from the main DB in it will reduce the probability of single point of failure and in case of the failure this redundant DB can be used to serve the data needs till the main DB is back.

**B. Fallbacks and retries :** The application can use fallback mechanisms for the upcoming requests till the DB is ready to take up the load again. Reads can be served either from cache or from redundant DB. Writing data to files on disk or pushing the write payload to streams will give the flexibility to the retry writing in the DB when DB is up and running. Even the redundant DB can be used for writing the data and once the main DB is up data can be synced between the two.

And then here we have the code

What’s the diff between the sharding and partioningi?

 Sharding distributes data across multiple servers, while partitioning splits tables within one server.

**How to handle failure in database?**