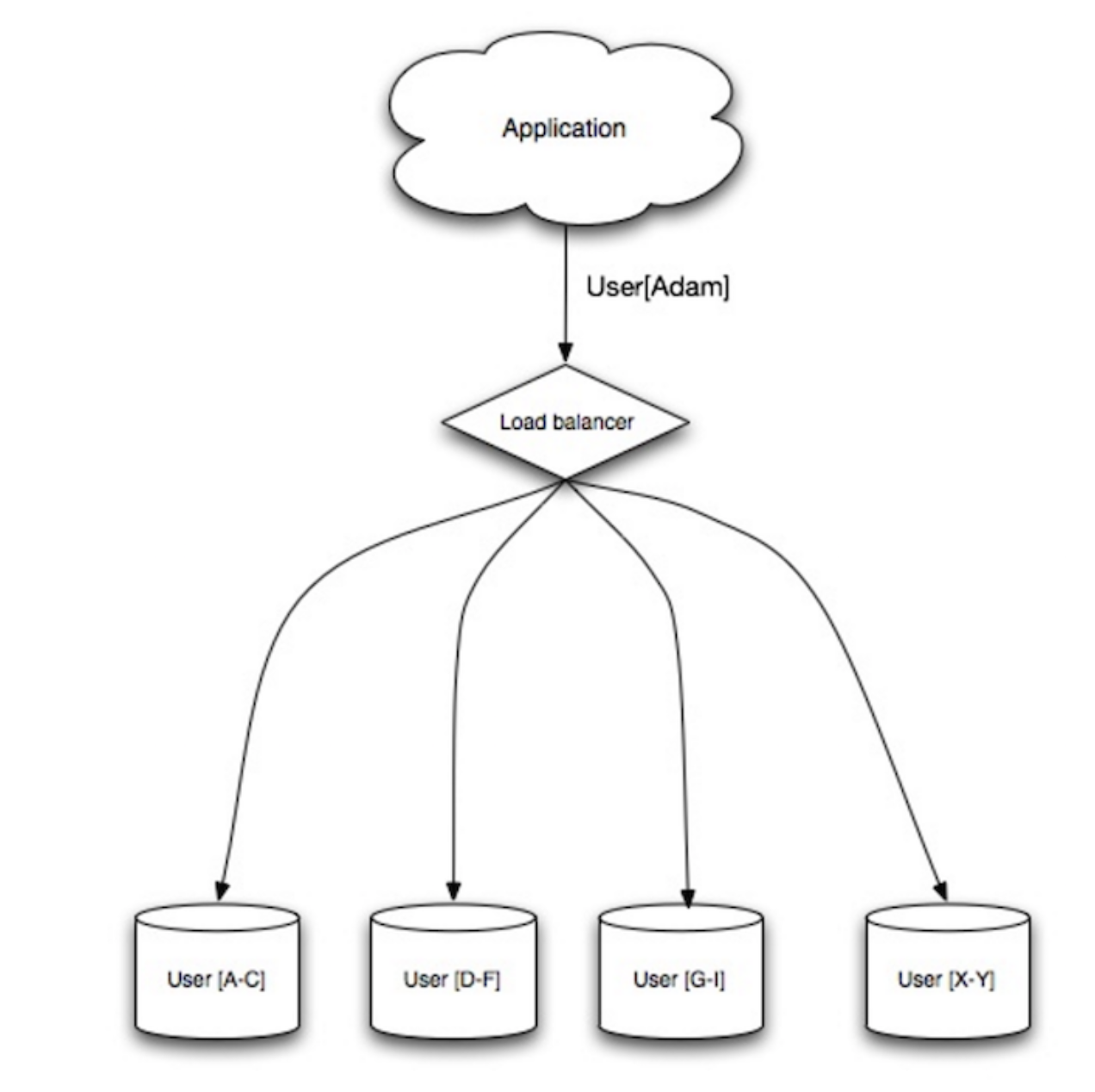
**Database Concepts**

**Sharding**

*A type of Horizontal Partitioning*



[Source](https://www.slideshare.net/jboner/scalability-availability-stability-patterns/)

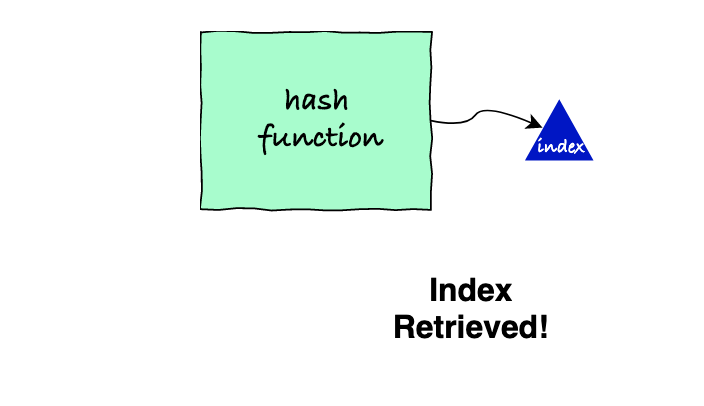
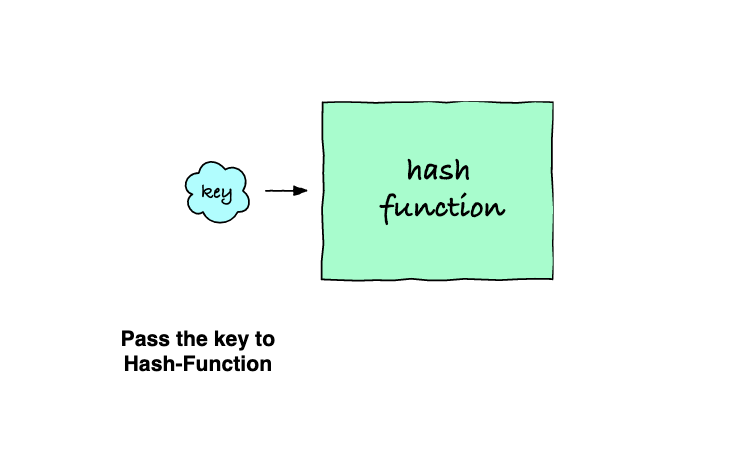
Good Things:

* High availability - if one box goes down, others are still up??
  + Doubt this, if one shard goes corrupt - the entire table will get corrupted? Isn’t HA config supposed to get rid of Single point failures like these? But sharding is also said to be shared nothing architecture so technically has no single point failures.
* Less Read and write traffic
* More cache hits, smaller indexes leading to faster queries
* Write bandwidth - Parallel writes

Not so good things:

* Application logic change
* Data Skew in the shards can be an issue (*Need to* [*rebalance data*](https://blog.yugabyte.com/how-data-sharding-works-in-a-distributed-sql-database/) *if this happens - Google (Cloud Spanner Sharding)/Flikcr do this rebalancing - should be built in from the start*)
* Joins can be complex and to be handled at application level/ SQL queries complex
* Hardware/Ops complexity (each shard resides on a db server per se) - failover/backup can be complex

**Consistent Hashing**



[Source: Educative.io](https://www.educative.io/courses/data-structures-in-java-an-interview-refresher)

* Index is also known as the hash code in some references. Point is, the same key will always return same index/hash code for a particular hash function.
* These indices work as the index of the Hash Table data structure.
* [Akamai invented](https://www.akamai.com/us/en/multimedia/documents/technical-publication/consistent-hashing-and-random-trees-distributed-caching-protocols-for-relieving-hot-spots-on-the-world-wide-web-technical-publication.pdf) Consistent Hashing.

But wait? Why are we talking about Hashing all of a sudden?

Turns out Consistent Hashing can determine where data is

It affects sharding -***read more and update***

Implementations:

* Avoids Hotspots
* Scale out storage nodes in NoSQL and Hadoop like systems

**Denormalization**

Good things:

* Improving read performance at the cost of write performance! This avoids Joins by making redundant copies of data in different tables.
* In most systems, reads have a higher frequency than writes (100:1 sometimes) so joins can be very expensive and slow to use everytime. Denormalization helps in such cases.

**Materialized view**

Storing results of a query in this DB object will help you use it again.

**Materialized view vs view:**

* both are technically the same except that view is not stored into the disk meaning that view will be run each that it’s accessed.
* Also materialized view will be updated if needed using the query definition

Not so good things:

* Data is redundant
* Data consistency issues, what will be the correct value of data. Can be solved by constraints?
* Mayb not do well on a write heavy load.

**SQL Tuning:**

How do we know when to tune?

Benchmark - load testing ([ab](http://httpd.apache.org/docs/2.2/programs/ab.html), Hammer DB, Db Monster etc.)

Profile: finding slow queries - MySQL [slow query log](https://dev.mysql.com/doc/refman/5.7/en/slow-query-log.html), postGres - [log\_min\_duration\_statement](https://wiki.postgresql.org/wiki/Logging_Difficult_Queries),

[auto\_explain](https://www.postgresql.org/docs/8.4/auto-explain.html)

Overhead when using these things?

* Because we’re asking DB to log and or explain each step

Some general things to keep in mind:

* CHAR is better than VARCHAR
* because it is a fixed block, easier for DB to access (*think of an array of int - 4 bytes - when accessing it has to look for the next element after 4 bytes so it can keep going on and on after 4 bytes but when using varchar - it’ll need to access the end point of the first element to track where it stops and the next one starts!*)
* Use NOT NULL where possible
  + Not in issue ([sql server example](https://stackoverflow.com/questions/129077/null-values-inside-not-in-clause)), NULL is treated as UNKNOWN
  + NULL uses less space than “” (empty)
  + NULL can be any type : int, string, decimal
* [Avoid BLOBS, store locations/pointers instead and fetch](https://dba.stackexchange.com/questions/174678/why-is-it-recommended-to-store-blobs-in-separate-sql-server-tables)
* Use TEXT for large texts/ blogs etc.
* DECIMAL for currency
* INT for large numbers upto 4 billion (2^32)
* Good index (indices)
  + Columns you use in group by, order, join etc.
  + Writes can be slower with indices as it needs to be updated
  + Index is stored in memory so will require space

**No SQL:**