Game Service

m.solimanz@hotmail.com

My Thoughts:

In this document I will try to give different scenarios/behaviors and how the database would designed in each case:

Big data solutions came to spread and strengthen the ideas of partitioning and replication. Partitioning data helps in splitting our data across fleet or cluster of machines while replications empowers the read capacity we are expecting or that might even come unexpectedly (the ideas behind elasticity where the system should be smart enough to be resilient to the high traffic or workload that it might expect and also to shrink down or shut down the resources that might be idle due to low workload, those ideas are spreader in cloud computing solutions especially)

Database should be designed to support both ideas partitioning and replication to enhance scalability of our system.

Partitioning or Sharding data can be done using a consistent hashing function, its main job is to distribute the data based on a key across cluster of machines while replication should copy the same data into different machines for increasing durability and high availability of our data.

However it seems it is very simple and promising however it is coming with big burdens to maintain and keep data consistent moreover limitations in accessing the data (not to be as easy as relational databases).

Data denormalization helps a lot in tackling problems of joining that we might face during partitioning our data and also super fastly accessing or reading the data however it is affecting in performance especially when it comes to high write-traffic.

NoSQL databases came to simplify partitioning and replicating data however still we should be smart enough to design the database by choosing the right indexes and partition keys as it is affecting in how our data physically distributed and stored. (e.g. Hbase, Column Family, HFiles)

Game Service

m.solimanz@hotmail.com

Design:

USERS	
_id	**Index**
rank	**Index**
userName	**Compound Index with password**
email	**Compound Index with password**
password	**Hashed**
myMatches	
currentMatch	
awards (issueDate, type)	**Sorted by issueDate desc** **Types are enumerated values**
wins / deaths / kills	

- Passwords should be hashed (service should be deployed in HTTPS)
- I supposed that types of awards are limited (few types) that is why I made in **enum**
- Awards are kept sorted every time it is updated (to list the most recent awards for user)
- MyMatches, I supposed we are storing only hundreds and removing old ones (recent history only) otherwise if it is thousands it better to keep them in separate collection.
- Compound Indexes used for quick retrievals of userName/password and email/password
- Ranks are calculated based on wins/losses/deaths/..etc. and it is kept indexed to get top N.
- wins/deaths/kills..etc and rank can be separated out to different collection based on required access patterns or scenarios.

Game Service

m.solimanz@hotmail.com

MATCHES	
_id	**Index**
name	
startedAt	**Index (desc)**
players	
playersCount	

- **Matches** are separated out as I assumed we might have access patterns like
  - Get all active matches o along with the players who are playing
  - Get all matches

- **playersCount** are atomic counters that keep track of how many players in the match which simplify validations or restrictions on how many users should play in the match.

- startedAt might be used to fetch the matches within specific time frame, in this case they should be indexed. Also sorting in desc to support get the last matches.

Game Service

m.solimanz@hotmail.com

Design In Relational Database:

For more info please email me m.solimanz@hotmail.com

I applied data normalizations and also referential integrity. Although reference keys keep data consistent (players should NOT like game or have a game as one of his/her favorites if it is not existing at all) but also it affects in performance when it checks every time it is writing to likes table whether that game exists or not.

Matches and Games can be separated into shared database however all other table can be partitioned using “userid” which is the main key in our database. Each user should reside in the same server long with his/her like/stats/matches/awards.