**Design Twitter – A System Design Interview Question**

* Difficulty Level : [Expert](https://www.geeksforgeeks.org/expert/)
* Last Updated : 15 Jan, 2023

 Read

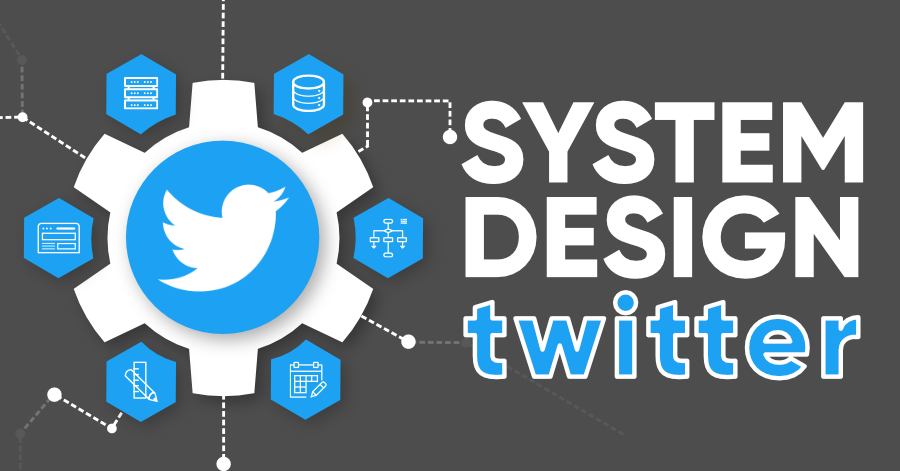
 Discuss

 Courses

 Practice

 Video

**Twitter**…You might be good at checking this social media all the time but what if somebody asks you to design this gigantic system within just 45 minutes? (*Is it a joke….lolz?*).   
Yes, this is what you are expected to do in your system design round of interviews. We are not joking and you need to tell your approach about designing a system like Twitter (within 45 minutes or less) that has hundreds of software engineers working on it for a decade? Designing Twitter (or Facebook feed or Facebook search..) is a quite common question that interviewers ask the candidates. A lot of candidates get afraid of this round more than the coding round because they don’t get the idea that what topics and tradeoffs they should cover within this limited timeframe. Firstly, remember that the system design round is extremely **open-ended** and there’s no such thing as a standard answer. Even for the same question, you’ll have a different discussion with different interviewers.



In this blog, we will discuss how to design a website like Twitter but before we go further we want you to read the article [“How to crack system design round in interviews?”](https://www.geeksforgeeks.org/how-to-crack-system-design-round-in-interviews/). It will give you an idea that what this round looks like, what you are expected to do in this round and what mistakes you should avoid in front of the interviewer.

Now let’s jump to the question straightforward “***How would you design Twitter?***”

**How Would You Design Twitter?**

Don’t jump into the technical details immediately when you are asked this question in your interviews. Do not run in one direction, it will just create confusion between you and the interviewer. Most of the candidates make mistakes here and immediately they start listing out some bunch of tools or frameworks like MongoDB, Bootstrap, MapReduce, etc. Remember that your interviewer wants **high-level ideas** about how you will solve the problem. It doesn’t matter what tools you will use, but how you define the problem, how you design the solution, and how you analyze the issue step by step.

You can put yourself in a situation where you’re working on real-life projects. Firstly, define the problem and clarify the problem statement. In this question, we will compress Twitter to its MVP (minimum viable product). Nobody expects you to design the whole service. So, we will only design the **core features** of Twitter instead of everything.

**1. Discuss The Core Features**

So firstly divide the whole system into several core components and talk about some core features. If some other features your interviewer wants to include he/she will mention there. For now, we are going to consider the following features on Twitter…

* The user should be able to tweet in just a few seconds.
* The user should be able to see Tweet Timeline(s)
* **Timeline:** This can be divided into three parts…
  1. *User timeline:* User sees his/her own tweets and tweets user retweet. Tweets that users see when they visit their profile.
  2. *Home Timeline:* This will display the tweets from people users follow. (Tweets when you land on twitter.com)
  3. *Search timeline:* When users search some keywords or #tags and they see the tweets related to that particular keywords.
* The user should be able to follow another user.
* Users should be able to tweet millions of followers within a few seconds (5 seconds)

**2. Naive Solution (Synchronous DB queries)**

To design a big system like Twitter we will firstly talk about the Naive solution. That will help us in moving towards high-level architecture. You can design a solution for the two things:

1. **Data modeling:** You can use a relational database like MySQL and you can consider two tables **user table (id, username)** and a **tweet table[id, content, user(primary key of user table)]**. User information will be stored in the user table and whenever a user will tweet a message it will be stored in the tweet table. Two relations are also necessary here. One is the user can follow each other, the other is each feed has a user owner. So there will be a one-to-many relationship between the user and the tweet table.
2. **Serve feeds:** You need to fetch all the feeds from all the people a user follows and render them in chronological order.

**3. Limitation of Architecture (Point Out Bottleneck)**

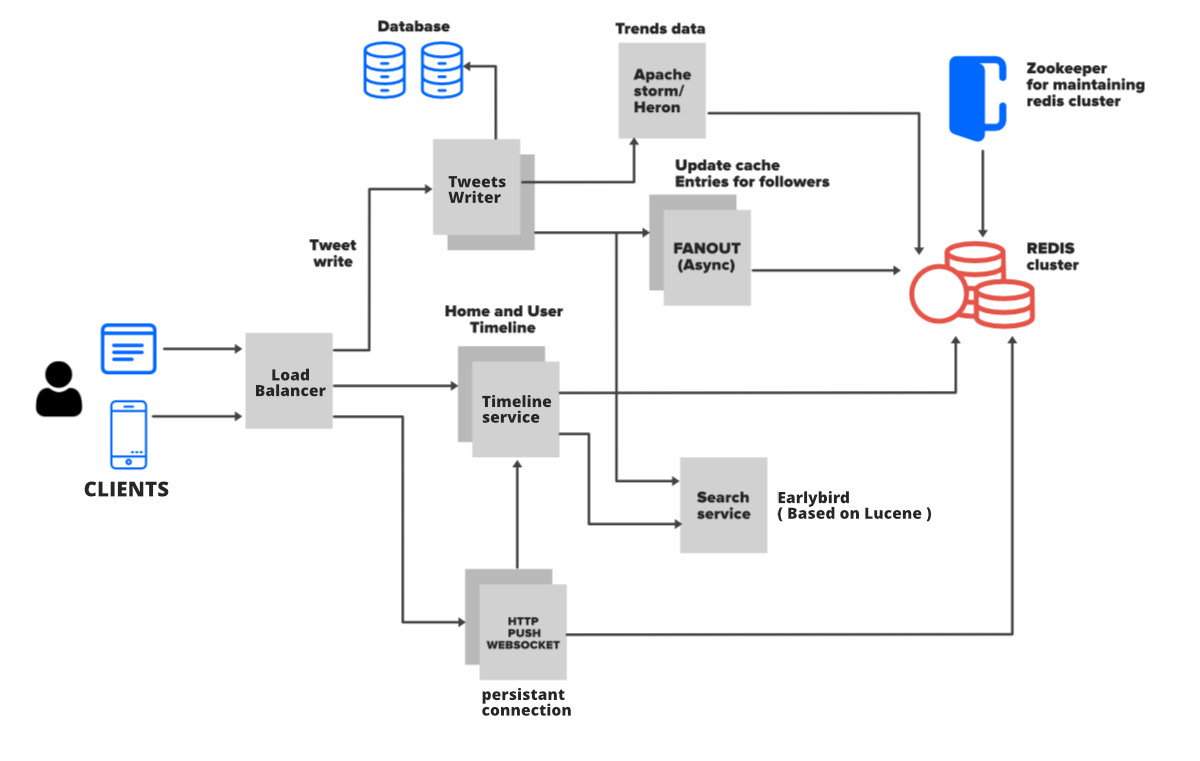
You will have to do a big **select**statement in the tweet table to get all the tweets for a specific user whomsoever he/she is following, and that’s also in chronological order. Doing this every time will create a problem because the tweet table will have huge content with lots of tweets. We need to optimize this solution to solve this problem and for that, we will move to the high-level solution for this problem. Before that let’s understand the characteristics of Twitter first.

**4. Characteristics of Twitter (Traffic)**

Twitter has **300M** daily active users. On average, every second **6, 000 tweets** are tweeted on Twitter. Every second **6, 00, 000 Queries** are made to get the timelines. Each user has on average 200 followers and some users like some celebrities have millions of followers. This characteristic of Twitter clears the following points…

1. Twitter has a **heavy read** in comparison to writing so we should care much more about the **availability**and scale of the application for the heavy read on Twitter.
2. We can consider **eventual consistency** for this kind of system. It’s completely ok if a user sees the tweet of his follower a bit delayed
3. Space is not a problem as tweets are limited to 140 characters.

**High-Level Solution**



As we have discussed that Twitter is read-heavy so we need a system that allows us to read the information faster and also it can scale horizontally. **Redis**is perfectly suitable for this requirement but we can not solely depend on Redis because we also need to store a copy of tweets and other users’ related info in the Database. So here we will have the basic architecture of Twitter which consists of three tables…**User Table, Tweet Table, and Followers Table**.

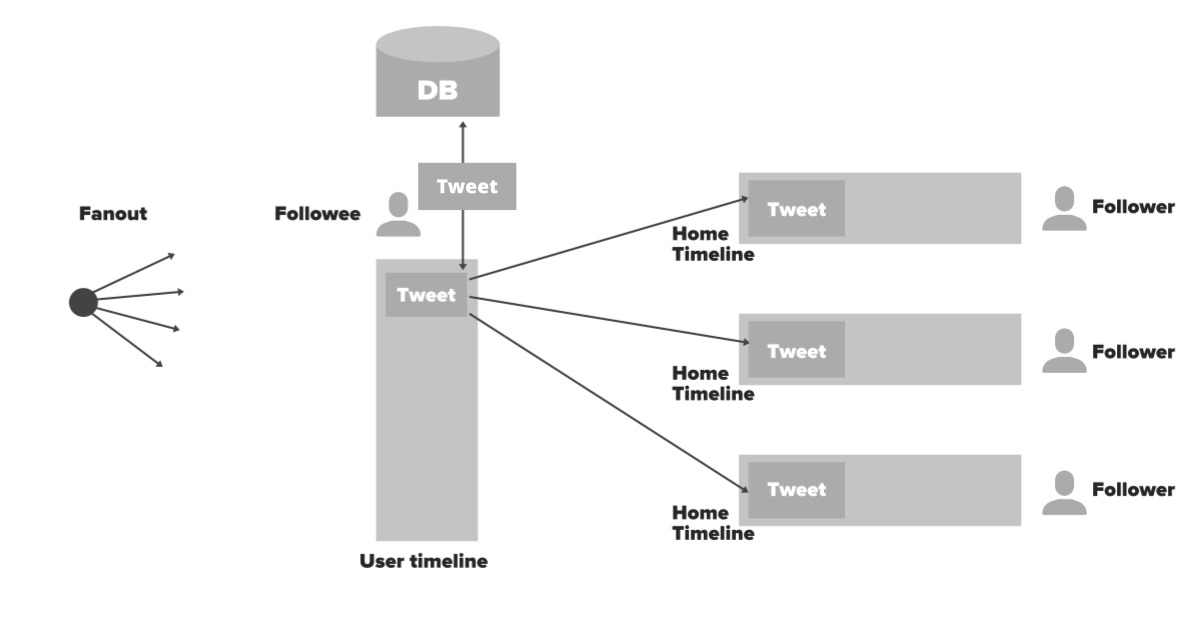
* Whenever a user will create a profile on Twitter the entry will be stored in the User table.
* Tweets tweeted by a user will be stored in the Tweet table along with the User\_id. Also, the User table will have **1 to many relationships** with the Tweet table.
* When a user follows another user, it gets stored in Followers Table, and also caches it Redis. The User table will have **1 to many relationships** with the Follower table.

**1. User Timeline Architecture**

* To get the User Timeline simply go to the user table get the user\_id, match this user\_id in the tweet table and then get all the tweets. This will also include retweets, save retweets as tweets with original tweet references. Once this is done sort the tweet by date and time and then display the information on the user timeline.
* As we have discussed that Twitter is read-heavy so the above approach won’t work always. Here we need to use another layer i.e **caching layer** and we will save the data for user timeline queries in Redis. Also, keep saving the tweets in Redis, so when someone visits a user timeline he/she can get all the tweets made by that user. Getting the data from Redis is much faster so it’s not much use to get it from DB always.

**2. Home Timeline Architecture**

* A user’s Home Timeline contains all the latest tweets of the person and the pages that the user follows. Well, here you can simply fetch the users whom a user is following, for each follower fetch all the latest tweets, then merge all the tweets, sort all these tweets by date and time and display them on the home timeline. This solution has some **drawbacks**. The Twitter home page loads much faster and these queries are heavier on the database so this huge search operation will take much more time once the tweet table grows to millions. Let’s talk about the solution now for this drawback…   
  ***Fanout Approach:*** Fanout simply means spreading the data from a single point. Let’s see how to use it. Whenever a tweet is made by a user (Followee) do a lot of **preprocessing**and distribute the data into different users (followers) home timelines. In this process, you won’t have to make any database queries. You just need to go to the cache by user\_id and access the home timeline data in Redis. So this process will be much faster and easier because it is an in-memory we get the list of tweets. Here is the complete flow of this approach…
  1. User X is followed by three people and this user has a cache called user timeline. X Tweeted something.
  2. Through Load Balancer tweets will flow into back-end servers.
  3. Server node will save tweet in DB/cache
  4. The server node will fetch all the users that follow User X from the cache.
  5. The server node will inject this tweet into the in-memory timelines of his followers (fanout)
  6. All followers of User X will see the tweet of User X in their timeline. It will be refreshed and updated every time a user will visit his/her timeline.



***What will happen if a celebrity will have millions of followers? Is the above method efficient in this scenario?***   
**Weakness (Edge Case):** The interviewer may ask the above question. If there is a celebrity who has millions of followers then Twitter can take up to 3-44 minutes for a tweet to flow from Eminem(a celebrity) to his million followers. You will have to update the millions of home timelines of followers which is not scalable. Here is the solution…   
**Solution [Mixed Approach (In-memory+Synchronous calls)]:** 

1. Precompute the home timeline of User A (follower of Eminem) with everyone except Eminem’s tweet(s)
2. Every user maintains the list of celebrities in the cache as well as whom that user is following. When the request will arrive (tweet from the celebrity) you can get the celebrity from the list, fetch the tweet from the user timeline of the celebrity and then mix the celebrity’s tweet at runtime with other tweets of User A.
3. So when User A access his home timeline his tweet feed is merged with Eminem’s tweet at load time. So the celebrity tweet will be inserted at runtime.

**Other Optimization:** For inactive users don’t compute the timeline. People who don’t log in to the system for a quite long time (say more than 20 days.).

**3. Searching**

Twitter handles searching for its tweets and #tags using **Earlybird**which is a real-time, reverse index based on Lucene. Early Bird does an **inverted full-text indexing** operation. It means whenever a tweet is posted it is treated as a document. The tweet will be split into tags, words, and #tags, and then these words are indexed. This indexing is done at a big table or distributed table. In this table, each word has a reference to all the tweets which contain that particular word. Since the index is an exact string-match, unordered, it can be extremely fast. Suppose if a user searches for ‘*election*‘ then you will go through the table, you will find the word ‘*election*‘, then you’ll figure out all the references to all the tweets in the system, then it gives all the results that contain the word ‘*election*‘.

Twitter handles thousands of tweets per second so you can’t have just one big system or table to handle all the data so it should be handled through a distributed approach. Twitter uses the strategy **scatter and gather** where it set up the **multiple servers or data center** which allow indexing. When Twitter gets a query (let’s say #geeksforgeeks) it sends the query to all the servers or data centers and it queries every **Early Bird shard**. All the early bird that matches with the query return the result. The results are returned, sorted, merged, and reranked. The ranking is done based on the number of retweets, replies, and the popularity of the tweets.

So far we have talked about all the core features and components of Twitter. There are some other in-depth components you can talk about. For example, you can talk about **Trends / Trending topics** (using Apache Storm and Heron framework), you can talk about **notifications** and how to incorporate **advertisement**.

*Want to get a Software Developer/Engineer job at a leading tech company? or Want to make a smooth transition from SDE I to SDE II or Senior Developer profiles?*If yes, then you’re required to **dive deep into the System Design world!** A decent command over System Design concepts is very much essential, especially for the working professionals, to get a much-needed advantage over others during tech interviews.

*And that’s why, GeeksforGeeks is providing you with an in-depth interview-centric*[***System Design – Live Course***](https://practice.geeksforgeeks.org/courses/system-design-live?utm_source=gfg-article&utm_medium=Q1-2023&utm_campaign=system-design-live)*that will help you prepare for the questions related to System Designs for Google, Amazon, Adobe, Uber, and other product-based companies.*