How to pass the data modeling round in big tech data engineering interviews

ZACH WILSON
23 SEPT 2023

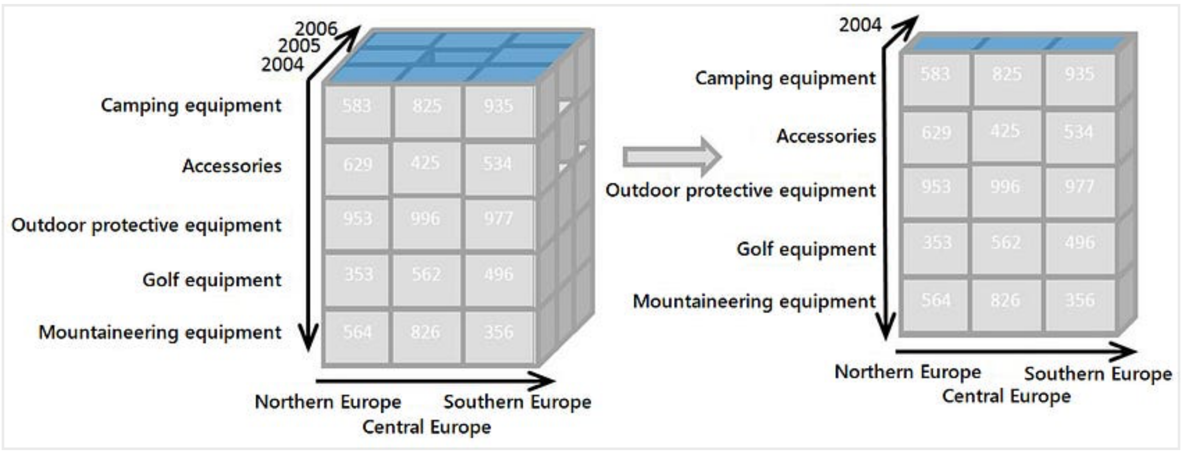
91

Share

The data modeling interview separates data engineers who can solve business problems efficiently from those who can't. Being able to deconstruct business requirements into efficient data sets that solve problems is the key skill that you want to demonstrate throughout this entire interview.

Knowledge of the following concepts will serve you very well:

- Asking the right questions!
    - Remember the interviewer will often give you more requirements than is given in the problem statement. It's your job to discover these requirements! The data modeling interview is 50% about technicals and 50% about communication!
- Dimensional data modeling
    - Cumulative dimensions versus daily dimensions (Maxime Beauchemin covers this in-depth on his Medium)
    - Slowly-changing dimension types
- Fact data modeling
    - Kimball data modeling method and Normalized versus denormalized facts
    - One big table methodology
- Aggregate data modeling
    - Daily metrics vs. cumulative metrics
    - Enable rapid slice-and-dice of metrics with cube aggregates
        - OLAP cubes are fundamental to rapid analytics.
            - Your data gets crunched down to the number of combinations of dimensions.
            - You can model the dimensions by time, product, location, etc. This enabled you to “slice” the aggregates you care about.



Example - modeling user growth

Often times in interviews you'll be given a vague problem like, **"Give me an efficient data model that models user connection growth on a social media site. Here are the upstream schemas"**

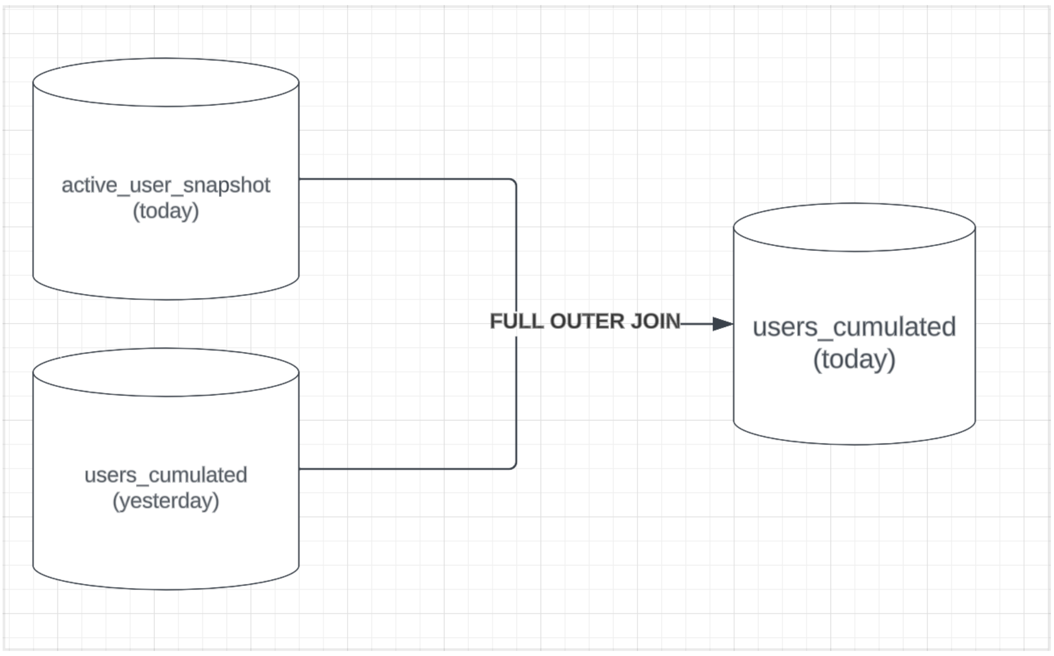
- **connection_events**
  - event_time TIMESTAMP
  - sending_user_id BIGINT
  - receiving_user_id BIGINT
  - event_type STRING (values ["sent", "reject", "accept"]
  - event_date DATE PARTITION
- **active_user_snapshot** (contains one row for every active user on **snapshot_date**)
  - user_id BIGINT
  - country STRING
  - age INTEGER
  - username STRING
  - snapshot_date DATE PARTITION

So how do you take these two schemas and create a data model that efficiently tracks how many connections are sent, accepted, and rejected?
One of the first indicators here that you should lean into cumulative table design is that **active_user_snapshot** does not have all the users for each **snapshot_date.**
So to start off, you'd want to cumulate **active_user_snapshot** in a table called: **users_cumulated**
The schema of this table might look something like:

- **users_cumulated**
  - user_id BIGINT
  - dim_is_active_today BOOLEAN
  - l7 INTEGER (how many days were they active in the last 7 days)
  - active_datelist_int INTEGER (a binary integer that tracks the monthly activity history, see this article on how to leverage

    powerful data structures like this)
    - dim_country STRING
    - dim_age INTEGER
    - partition_date DATE PARTITION

This table is populated by taking **active_user_snapshot WHERE snapshot_date = 'today'** and FULL OUTER JOIN it with **users_cumulated WHERE partition_date = 'yesterday'**
This table will have one row for every user each day regardless of it they are active or not.
In the interview, you'll probably need to come up with the schema above and a diagram that looks something like this.

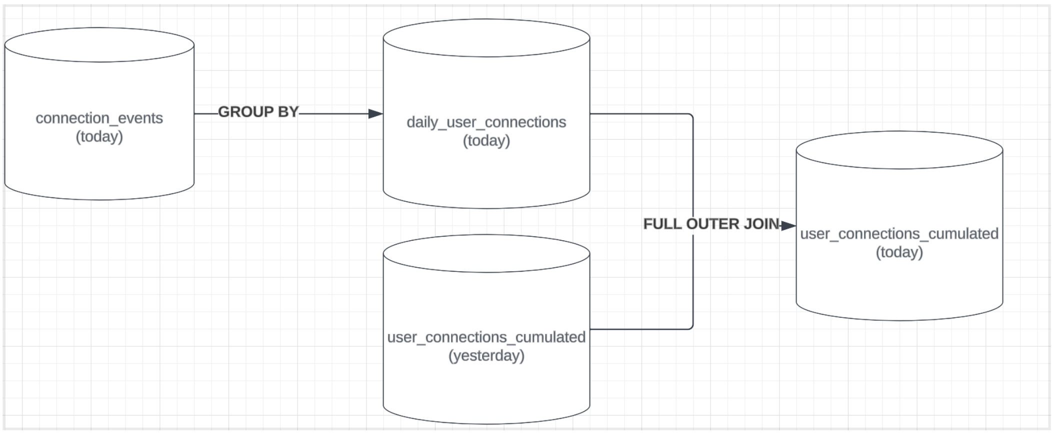


Great, now we have a good user dimension table to use further downstream.
The next thing we need to build is two tables. One daily dimension table called **daily_user_connections** and one cumulative table called **user_connections_cumulated.**
We create **daily_user_connections** for the purpose of simpler backfills in the future and fast daily-level analytics.
You can imagine a schema like

- **daily_user_connections**
    - sender_user_id BIGINT
    - receiver_user_id BIGINT
    - sent_event_time TIMESTAMP
    - response_event_time TIMESTAMP (this is NULL if they have not accepted or rejected)
    - connection_status STRING ["accepted", "rejected", "unanswered"]
    - partition_date DATE PARTITION
- **user_connections_cumulated**

    - (the same schema except contains all historical connections)

The schemas above and a diagram that looks something like this would be expected in the interview.



Now that we have **daily_user_connections**, **user_connections_cumulated**, and **users_cumulated** we are ready to JOIN these two dimensions together to create aggregate tables and analytical cubes.
What type of aggregates do we care about? You can use the upstream schemas as hints. (They probably care about age and country). If you're unsure what aggregates matter, make sure to ask questions in the interview.

If we join together **user_connections_cumulated** and **users_cumulated,** both on sender_user_id and receiver_user_id, we can get a new schema. Let's call it **user_connections_aggregated**

- **user_connections_aggregated**
    - dim_sender_country STRING
    - dim_receiver_country STRING
    - dim_sender_age_bucket STRING
    - dim_receiver_age_bucket STRING
    - m_num_users BIGINT
    - m_num_requests BIGINT
    - m_num_accepts BIGINT
    - m_num_rejects BIGINT
    - m_num_unanswered BIGINT
    - aggregation_level STRING PARTITION KEY
    - partition_date DATE PARTITION KEY
-

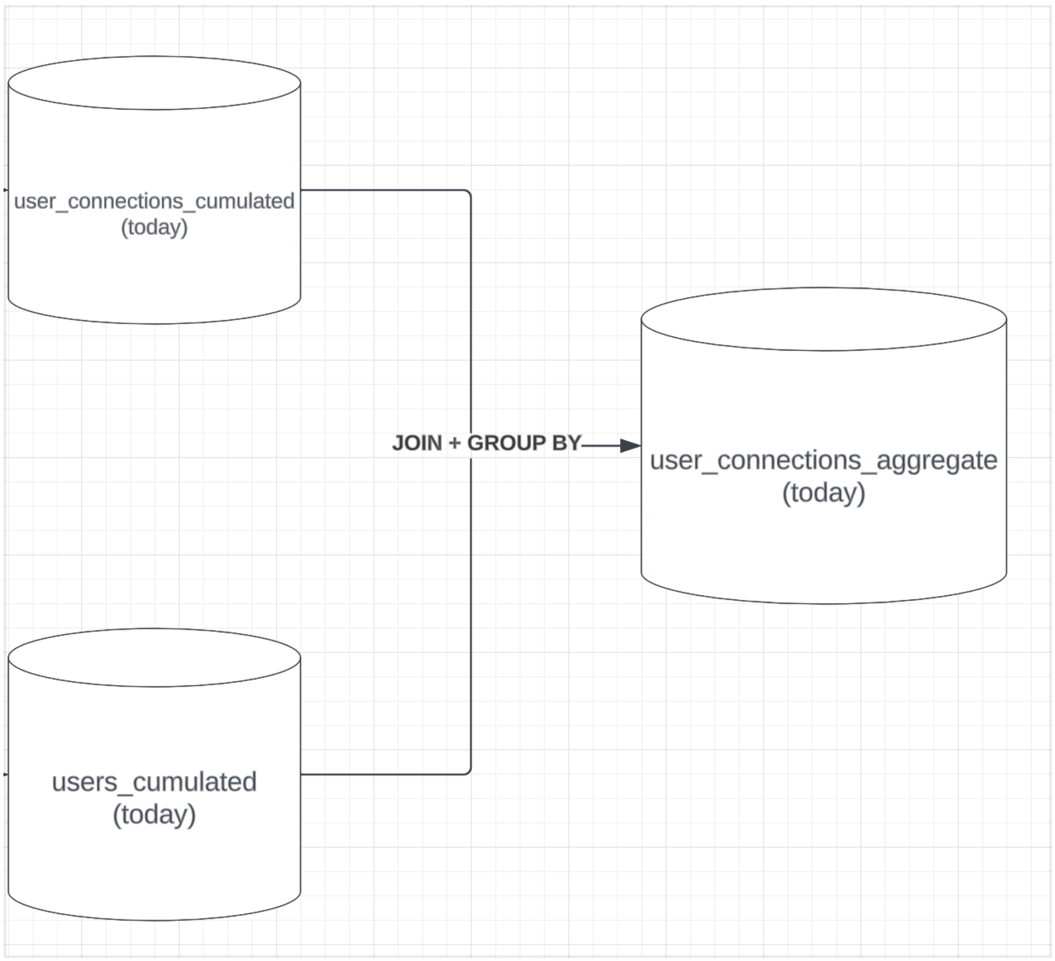
This this case we can generate this type of table by doing a **GROUPING SETS** query on top of a JOIN between **user_connections_cumulated** and **users_cumulated.** We also want to bucketize age into categorys like <18, 18-30, 30-50, etc. This is to lower the cardinality and make the dashboards more performant.
The GROUPING SETS statement would probably look something like this:
GROUP BY GROUPING SETS (

```
(),
(dim_sender_country),
(dim_sender_country, dim_receiver_country),
(dim_sender_country, dim_sender_age),
(dim_receiver_country, dim_receiver_age),
(dim_sender_country, dim_receiver_country, dim_sender_age,
dim_receiver_age)
)
```

The aggregation_level partition is determined by which columns we are grouping on. If it's just **dim_sender_country**, then the aggregation_level would be the string literal '**dim_sender_country',** at Facebook they concatted the aggregation_levels with '**__'** so if you grouped on **dim_sender_country** and **dim_receiver_country** the aggregation_level would be '**dim_sender_country__dim_receiver_country'**

A lot of the details here around GROUPING SETS aren't needed in the interview though. You just need to talk about the different grains and aggregates you need to produce not the nitty-gritty details I'm going over here.

Aside from the schema for user_connections_aggregate, you'll need to produce a diagram that looks something like this:

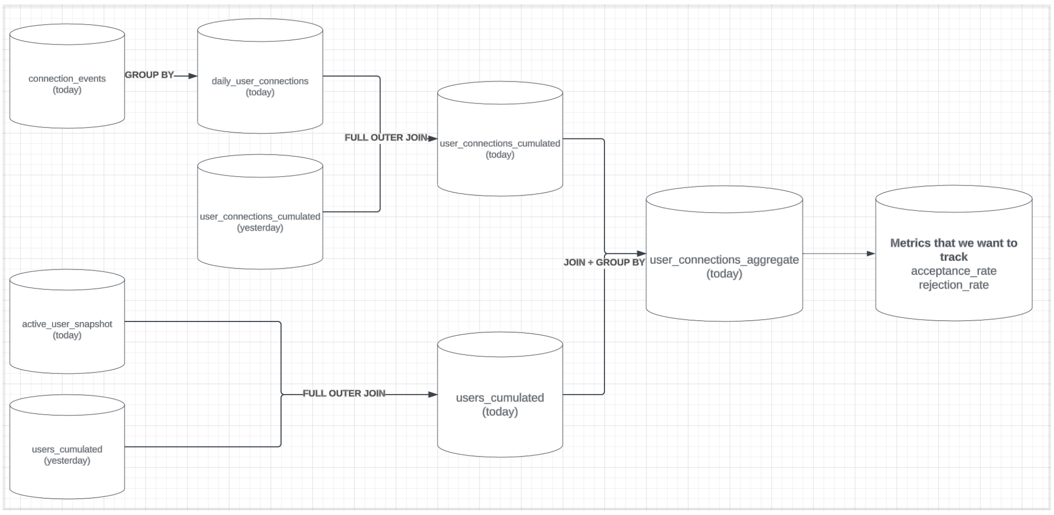

user_connections_cumulated (today)
users_cumulated (today)
JOIN + GROUP BY
user_connections_aggregate (today)

You'll see that **user_connections_aggregate** is an OLAP cube that. is ready to be sliced and diced by analysts!

The last piece of the puzzle is coming up with metrics based on this aggregate table. There are easy ones like:

- **rejection_rate** which is defined as **m_num_rejects / m_num_requests**
- **acceptance_rate** which is defined as **m_num_accepts / m_num_requests**
- **connections_per_user** which is defined as **m_num_accepts / m_num_users**

You don't have to come up with a ton of metrics during the interview but a few is good to wrap up the modeling exercise!
After defining a few important business metrics, you'll end up with a diagram that looks something like this:



connection_events (today)
GROUP BY
daily_user_connections (today)
user_connections_cumulated (yesterday)
FULL OUTER JOIN
user_connections_cumulated (today)
active_user_snapshot (today)
users_cumulated (yesterday)
FULL OUTER JOIN
users_cumulated (today)
JOIN + GROUP BY
user_connections_aggregate (today)
Metrics that we want to track
acceptance_rate
rejection_rate

Conclusion

If you can produce diagrams and schemas like the ones above and talk intelligently about the tradeoffs, you'll pass this round of interview with ease. I've found this round of interview to be fun and engaging since the correct answer is often ambiguous and requires a lot of back and forth with the interviewer!

If you want to learn more about data modeling and other critical data engineering concepts, join my six week intensive course that covers everything from data modeling to Kafka to Flink to Spark to Airflow and more!

——

How to pass data engineering SQL interviews in big tech

ZACH WILSON
11 SEPT 2023

100

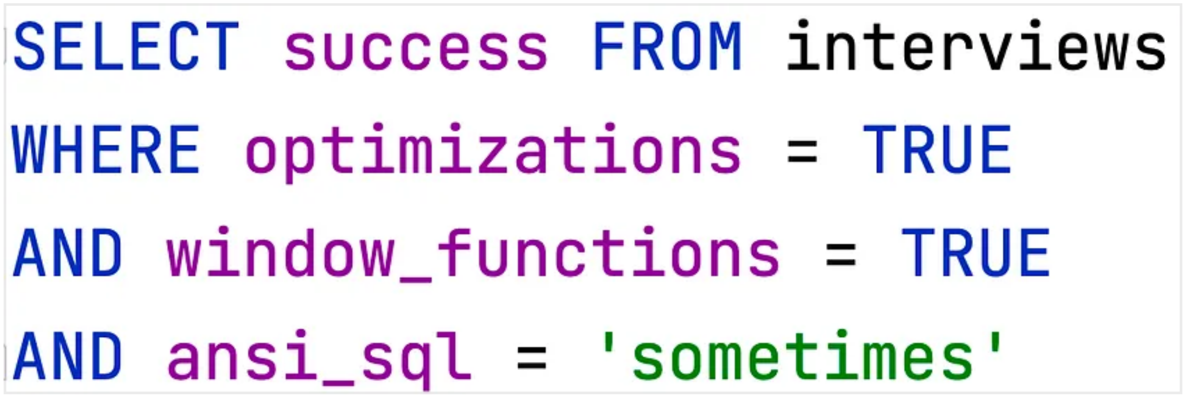
11
Share

SQL and data engineering are like peanut butter and jelly. In every single data engineering role I've ever had, I've been asked SQL questions for at least an hour. This is a round that I personally feel very confident in since I have never failed it. In this newsletter, I'll be covering the key details that you'll need to know to pass these common interviews.

What are the types of SQL interviews in big tech companies

Big tech companies like to ask a lot of questions from their engineers. There are usually two types of interviews for SQL.

- The screener interview
    - This interview is 45-60 minutes. The questions here are usually pretty easy compared to the onsite round. The goal here is to show that yes, you can code in SQL. Answering the questions is usually enough to pass the round.
- The onsite SQL interview
    - This interview is 60 minutes. The questions here go more in-depth. You'll be asked a lot about table scans and optimizations. It's unlikely your first solution will be the most optimal one.



```
SELECT success FROM interviews
WHERE optimizations = TRUE
AND window_functions = TRUE
AND ansi_sql = 'sometimes'
```

The screener interview in detail

In the screener interview, they usually ask four to five questions:

- A simple question that requires a WHERE condition with a GROUP BY.
    - This question is mostly to build your confidence and to give everyone

who interviews a small win even if they don't pass.

- A question that requires you to do a JOIN + aggregation
    - Make sure to be able to talk about the differences between LEFT, RIGHT, INNER, and FULL OUTER JOIN. If you don't know the differences, you won't pass
- A question that requires you to do a window function
    - The most common question here is the "second highest salary" question. The use of RANK, DENSE_RANK, and ROW_NUMBER functions is very common here. Know the differences between these functions and also how to use PARTITION BY in window functions.
- A question that requires you to use a common table expression or subquery plus a technique from one of the questions above
    - Using the WITH keyword and creating a common table expression is usually better than using a subquery for this problem. It'll help you maintain readability while also making it easier to keep your thought process clear
    - Use good alias names here so that in the subsequent queries you can filter or aggregate easily
- A question that requires a self-join or one that starts to dig deep into optimizations
    - A self-join is one way to solve a problem. Usually, another way to solve the same problem is usually LAG/LEAD window functions. Some places like, Facebook/Meta want you to use only ANSI SQL so knowing that you can replicate LAG/LEAD window functions with a self-join. This is kind of annoying since LAG/LEAD are pretty standard and more performant!

If you can solve four or five of these SQL questions in 45 minutes, you'll pass and be able to go to the onsite round of the interview.

Most common mistakes in screener interview

I've given over a hundred of these screener interviews in my time in big tech and here are the most common mistakes I see.

- Jumping right into coding before clearly understanding the problem
    - This mistake is critical because it wastes the valuable 45 minutes that you have. Spending 1-2 minutes asking questions to save yourself 5-10 minutes debugging is critical for success in these interviews. It also illustrates to the interviewer that you have good communication skills!
- Relying too heavily on UDFs or engine-specific code
    - Most of the interviews are going to be using basic MySQL or Postgres. If you need a fancy SparkSQL UDF to solve your problem, you're going to fail
- Relying too heavily on window functions
    - Not knowing how to replicate window functions with self-joins and pure ANSI SQL is a common mistake I see engineers make.
- Not communicating with the interviewer
    - The interviewer will give you hints and guidance. Trying to solve the

problems through sheer coding will is not the right approach and I've seen many people fail this way

The onsite SQL interview in detail

This interview is one of the last interviews that stands in the way of you getting that shiny big tech offer! You can think of it as the last SQL boss. This interview is considerably different from the screener interview. You'll usually be asked fewer questions but in much more detail.

- You'll be given a set of data tables and how they relate. The questions that you'll be asked are various so I'll give you some keyword mappings to help solve them
- If you hear ordinal words in the question (i.e. first, second, third, etc), this is a signal you should be using a window function.
    - This usually means you'll solve the problem with DENSE_RANK or ROW_NUMBER. Having a firm understanding of how ties work between these window functions is important.
- If you need to count multiple things with a condition, it's much better to do:
    - SELECT COUNT(CASE WHEN condition = 'value' THEN 1 END), COUNT(CASE WHEN condition2 = 'value2' THEN 1 END) FROM table
    - vs
    - SELECT COUNT(1) FROM table WHERE condition = 'value' UNION ALL SELECT COUNT(1) FROM table WHERE condition2 = 'value2'
    - The reason for this is table scans and performance. You need to be able to solve the problems with the fewest number of table scans possible.
- If you hear the word "rolling" or "cumulative":
    - You should think window function again. For example, here's a 30 days rolling sum
    - SUM(value) OVER (ORDER BY date ROWS BETWEEN 30 PRECEDING AND CURRENT ROW)
- If you might hear something like, "likes by country" or "metric by dimension"
    - This should be a signal to either use GROUP BY or PARTITION BY + window function depending on if it's coupled with an ordinal term or a "rolling/cumulative" term
- Find all the things that don't have X attributes. For example, "find all the Facebook users who have no friends"
    - This should signal to use **LEFT JOIN + WHERE IS NULL**
    - SELECT * FROM users u LEFT JOIN friends f ON u.user_id = f.user_id WHERE f.user_id IS NULL
- Remember that speed in answering questions here is important but so is the detail and expertise that you express. Be prepared to also answer follow-up questions about
    - Query plans and table scans
        - I used the **EXPLAIN** keyword during my interview with Airbnb and actually stepped through the plan with them. I did get the job.
    - Optimizations, index usage, and partitioning

- Can you talk about the read-write trade-offs of index creations?
- What would happen if we partitioned the table by date or some other low-cardinality column?

Interview tactics that I used to also help

Remember that technical interviews are interviews and not tests!

- I make it a goal of mine to try and make the interviewer laugh. Being personable and treating the interviewer as a person and not a test-giver can do wonders. Especially if you're on the edge between passing and failing the interview. Being likable can help bias the interviewer's rating of you into landing the job.
- Give your brain a little bit of time to reset and refresh. Sometimes big tech wants you to do four to six hours of interviews in one block. I ask for at least 30 minutes between each interview so I can reset my brain and research about the next interviewer so I can look more prepared and ask better questions. These types of accommodation are rarely asked for but almost always given! You'd be surprised how asking something vaguely personal like, "How have you liked working at Netflix for the last 7 years?" gets the interviewer to think you're cooler than you might actually be.
- For my readers with ADHD and anxiety, I highly recommend getting some exercise in before the interview to reduce the jitters. I always do a short 10-15 minute run before these interviews and that helps a lot!

What other things do you think are needed to pass these interviews? They're tough but if you focus on the right words and translating them into SQL, you can definitely pass these rounds!
If you liked this content, please share it with your friends and comment on what other type of data engineering interview content you'd like to see!
I cover all types of interview content in my data engineering course on DataEngineer.io!, including data modeling, SQL, data architecture, behavioral and data structures, and algorithms interviews!

——

How to pass the data architecture interviews in big tech

Trading off complexity, latency and correctness

ZACH WILSON
3 OCT 2023

The architecture interview is often what stands in the way of you and getting a fancy senior+ data engineering role in big tech. The part of this interview that people fear the most is its open-ended nature and the depth you need to have in order to not sound like an idiot. In this newsletter, I'll be covering data architecture in depth and how to not sound like an idiot and ace these interviews with a little bit more ease!

What does the data architecture interview look like?

If you've made it to the data architecture interview, congratulations! This interview round is usually one of the very last rounds before you get an offer.

This interview most often consists of a 60-90 minute discussion (possibly including some whiteboarding) about the tradeoffs of various technical decisions and how you could overcome or compensate for these tradeoffs.

What is tested in the data architecture interview

The key thing to remember about these interviews is tradeoffs!

- A part of this will be considering the differences between Lambda and Kappa data architectures.
  - Some key points I want to talk about here:
    - Lambda (used by companies like Meta and Airbnb)
      - More complex since you have a "speed layer" and a "batch layer"
      - More likely to be correct because it can "true up" the speed layer. The batch layer is more trustworthy
      - Picks up low latency and correctness at the expense of complexity
    - Kappa (used by companies like Uber)
      - Simpler since it's a "streaming-only" paradigm
      - Has a harder time with data quality alerts
      - Picks up low latency and simplicity at the expense of correctness
    - The serving layer
      - This is often a low-latency store like Redis, Memcached, or Druid. Or it could be a NoSQL store like Cassandra or MongoDB.
      - It can be a higher-latency store like Iceberg too though
      - A key thing to remember here for the serving layer is picking the right sized database, Redis for example can't store huge data!
- Knowing where you can insert data quality checks
  - Data contract patterns like signal table vs. write-audit-publish
  - How to test streaming pipelines for errors
    - Options (in order from most taxing to least):
      - Fail the job on egregious errors to troubleshoot
      - Output error rows to a separate Kafka topic
      - Ignore the error rows with filter conditions
- Knowing the tradeoffs of different database choices
  - Alex's book and newsletter on the System Design interview will serve you well here and with a lot of these other topics

    - CAP Theorem is very important when determining which database
        - You have consistent and available databases like Postgres, which have a hard time scaling horizontally
        - You have available and partition-tolerant databases like Cassandra which are “eventually consistent” which means they will return a consistent result given enough time
        - You have consistent and partition-tolerant databases like HBase and MongoDB which will give you consistent results but sacrifice the availability of data in the event of a failure!

Example: Airbnb guest viewing counter problem

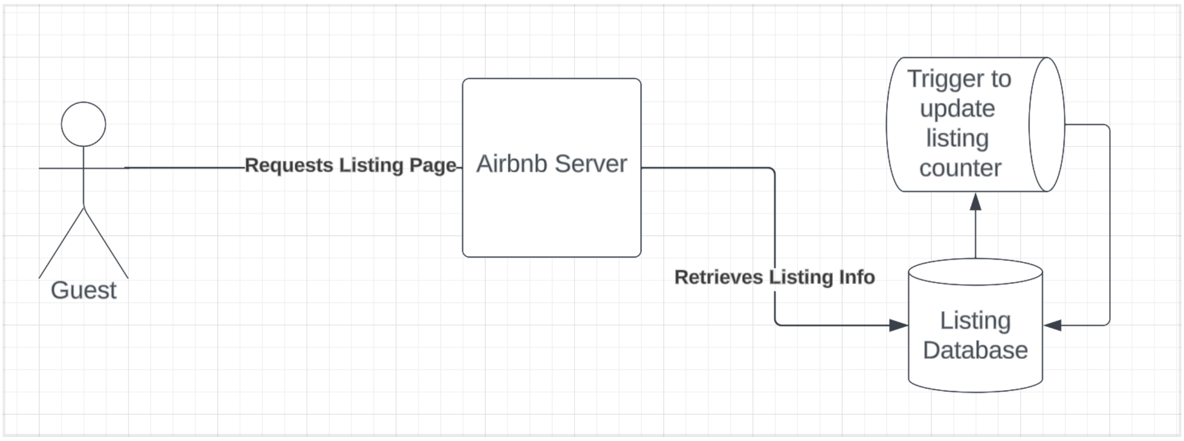
When I was interviewing for Airbnb, I was asked the following question in my data architecture interview, “How would you design a counter on a web page that would count the number of guests that have viewed a listing in a given day?”
Your initial instinct as a data engineer is to “log the event data and update with some sort of batch process.”
But that’s not the point of these interviews! The point is to engage with the interviewer and delve deeper into what the actual business requirements are to find an architecture that would work best.

Asking good follow-up questions is really important in these interviews. My immediate set of questions were:

- How quickly does this data need to be updated?
    - Asking latency-based questions will be critical. Remembering in this case there are actually a few latencies to consider
        - The latency between client and server
            - This should always be kept low since slow server response times destroy revenue for companies
            - In other words, you should probably not keep this data in a data lake but in a low-latency store like Redis or Memcached
        - The latency between the correct data generation and what the client sees
            - Intuitively, this latency is caused by the delay in the schedule of the logged data. If the count isn’t materialized and aggregated instead, it will increase the latency above. Materialized counts will have a latency in displaying the correct number and what they currently display.
        -
- How accurate does this data need to be?
    - Accuracy and latency often trade-off in data architectures. Asking about accuracy expectations will show that you know there’s no such thing as zero-latency, one-hundred-percent accurate data.
        - The interviewer responded with, “It needs to be at least 90% accurate all the time”

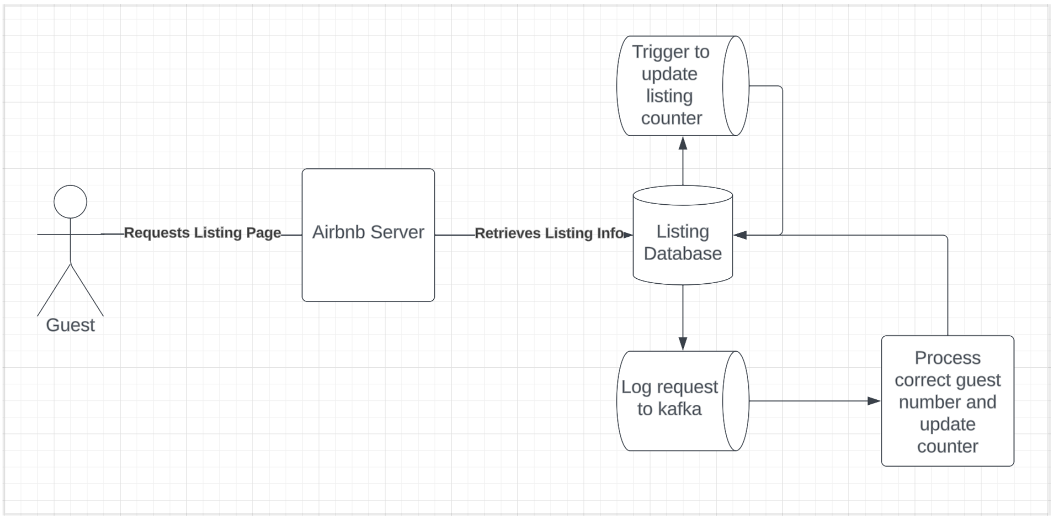
After understanding the latency and correctness requirements of this problem. My initial thought was a diagram that looked like this:

Every time a guest asks for a listing page, it triggers the database to update a counter for that listing which can be served to the user as well!



The interviewer wasn't happy with this design because he said that the correctness of the number of guests would slowly drift as guests refreshed the page.
I fumbled a bit with this pushback. Initially, I was thinking of a trigger that updated a unique array of guests. But this wouldn't be performant on the app side.

Then I remembered we don't have to do everything in the "real-time" path and we could solve this problem with a batch job.
I added a small piece where the requests get logged to Kafka and then there's an hourly process that updates the counters with their correct, deduped value.



This was something that really pleased the interviewer. I came up with this design in about 20-25 minutes and we spent the next 30 minutes talking about Airbnb and what the role looked like!

Data architecture interviews can be much more complicated than this example.

I’ve failed a few architecture interviews with Clubhouse and Robinhood because they were asking for something very specific that I couldn’t figure out! But that’s for another newsletter!

Conclusion

If you enjoyed this newsletter, please share it with your friends! Did I miss anything else that has helped you pass these interviews?

We’ll be launching the next iteration of the EcZachly boot camp on November 6th and sign-ups will start this week so get ready for another email for those!

People who buy the self-paced course here will get a large discount and first priority for the November 6th boot camp seats!