Wipro

**1. How did you handle the initial historical data load and the subsequent incremental loads in your project?**

**3. Let's consider a scenario: A data pipeline fails for two hours, causing a data gap. The files are still in the landing zone. How would you recover this missing data, and what steps would you take to prevent this from happening again?**

**Interviewer's Detailed Explanation (as a correction/clarification):** The interviewer provides a comprehensive solution for this common production issue. The steps include:

1. **Stop the production pipeline:** This is the first action to prevent further issues.
2. **Assess the backlog:** Determine how many hours of data are missing.
3. **Increase cluster size:** Scale up the compute resources to process the backlog quickly.
4. **Run a separate historical load pipeline:** Manually or automatically trigger a dedicated process to load the missing data.
5. **Restart the main pipeline:** Once the backlog is cleared and the data is caught up, resume normal operations.
6. **Preventive measures:** The interviewer notes that having two distinct pipelines—one for regular batches and one for historical or recovery loads—is a robust strategy to handle such failures without disrupting the main data flow.

**For a data migration of 10 TB from Oracle to Snowflake, what's your preferred strategy, and why, from the following options: Azure Data Factory, Snowflake Stored Procedures, Databricks, and Snowpipe?**

**1. Batch Data Migration: Oracle to Snowflake**

**Interviewer's Question:** The interviewer presents a scenario of migrating **10 petabytes of data from Oracle to Snowflake**. The goal is a one-time historical load, followed by a daily incremental batch load. The interviewer asks which of the following options is best and why:

1. **Azure Data Factory (ADF)**
2. **Snowflake Stored Procedures with Tasks**
3. **Databricks**
4. **Snowpipe**

**Prabhakar's Initial Answer:**

* He correctly suggests that **Snowpipe** is the preferred option, but he incorrectly states that Snowpipe would be a poor choice for a petabyte-scale historical load.
* He recommends **Databricks** for its ability to handle large data volumes, but then struggles to explain the specific integration steps between Databricks and Snowflake. He mentions that PySpark integration with Snowflake is not possible from a Python worksheet, which is a key misunderstanding.
* He says **Snowpark** is not suitable for this scenario because it's meant for data already inside Snowflake, not for large-scale ingestion.

**Interviewer's Detailed Explanation (as a correction and best practice):** The interviewer corrects Prabhakar's misconceptions and provides a detailed strategy:

* **Snowflake Stored Procedures with COPY INTO is the best approach.**
* **Why COPY INTO?** This command is highly optimized for bulk loading data. It can be used for both the large historical load and the subsequent daily incremental batches.
* **Batch vs. Streaming:** The interviewer clarifies the roles of other tools:
  + **Databricks:** Best for **complex transformations** or **massive-scale streaming data**. It should be considered when the data needs significant processing before loading.
  + **Snowpipe:** Best for **streaming data** or **small batches**, but not typically recommended for huge historical loads.
* **The Overall Strategy:**
  1. Have Oracle export the data into files in an object storage location like **AWS S3** or **Azure Data Lake Storage (ADLS)**.
  2. Use a **Snowflake stored procedure** with a **COPY INTO** command to pull the data from the cloud storage into a Snowflake staging table.
  3. For performance, you can use a **multi-cluster warehouse** to process the data in parallel.
  4. The interviewer also mentions that you can establish a direct connection to Oracle to pull data, but pushing to object storage first is a common and efficient pattern.

**2. In-Database Complex Transformations**

**Interviewer's Question:** The interviewer presents another scenario:

* Data is loaded into a Snowflake staging table.
* The data is semi-structured (**JSON**) and requires **complex transformations**, including parsing and flattening.
* The processing is done once a day as an overnight batch.
* The interviewer asks which of these two options is better for the transformations and why:
  1. **Snowflake Stored Procedures (SQL)**
  2. **Snowpark (Python/PySpark)**

**Prabhakar's Answer:**

* He suggests **Snowpark** is the better option.
* His justification is that Snowpark has built-in libraries like Pandas and NumPy, which make complex transformations easier and faster than writing the same logic in SQL stored procedures.
* He also says that because the transformations happen *inside* Snowflake, the processing is faster. However, he struggles to provide a definitive, technical reason why Snowpark is superior to SQL for this specific task.

**Interviewer's Detailed Explanation (as a correction and best practice):**

* The interviewer agrees that **Snowpark is the better option** and provides the technical reasons:
* **Complex Transformations:** Writing JSON parsing and flattening logic in SQL can be tedious and verbose. For example, extracting 15 columns from a JSON object would require writing code for each one.
* **Simplified Code:** **Python/PySpark** in Snowpark provides powerful libraries and functions that can handle these complex transformations more efficiently and with less code.
* **Resource Optimization:** With Snowpark, you can process smaller chunks of data more efficiently.
* **Data Validation:** Snowpark allows you to build a more robust data validation framework directly into your code, which is more flexible than what you can achieve with standard SQL.

**3. General Feedback and Suggestions**

**Interviewer's Question:** "Do you have any questions about that?"

**Prabhakar's Question:** "Yeah, your questions are very interesting... if you want to give some suggestions, just give suggestions, that would be for me honestly."

**Interviewer's Final Feedback:**

* The interviewer commends Prabhakar for his experience with PySpark and his general understanding of data volumes.
* He points out a key area for improvement: **deepening his knowledge of specific Snowflake features and architecture**.
* **Specific examples of missing knowledge:**
  + **Query Pruning:** A core Snowflake feature that optimizes query performance by skipping micro-partitions that don't contain relevant data.
  + **Micro-partitions:** The underlying data storage structure in Snowflake, which is critical to understanding how the platform works and why performance tuning techniques like clustering are effective.
* The interviewer concludes by saying that a deeper focus on these core Snowflake-specific features would be beneficial for the candidate's professional growth.

**Candidate's Corrected Responses**

This is a breakdown of the candidate's interview answers, with corrections and improvements to provide more comprehensive and professional responses.

**Question 1: How do you handle historical and incremental loads in your current project?**

**Correction:** The candidate's response was weak because they admitted to not being involved in the initial implementation. A better answer would demonstrate knowledge of common industry practices and explain how they would approach this problem even if they haven't personally built the specific pipelines.

**Improved Answer:** "While I wasn't part of the initial implementation, my daily responsibilities require a deep understanding of the existing pipeline logic. Our project uses a hybrid approach for data ingestion. The initial **historical load** was a one-time event, likely performed using a bulk data transfer tool like **Snowflake's Data Loading Wizard** or an orchestration tool like **Azure Data Factory** to efficiently move a large volume of data from our source (mainframe files) to the Snowflake staging area.

For **incremental loads**, we have two primary methods:

1. **Stored Procedures and Tasks:** For our batch-oriented data, we use a combination of Snowflake **Tasks** and **Stored Procedures**. The tasks are scheduled to run at regular intervals, and the stored procedures contain the MERGE logic. This MERGE statement is a key component, allowing us to efficiently insert new records and update existing ones in our target tables based on a unique key, preventing duplicates and ensuring data consistency.
2. **Streams (for future-proofing):** Although we haven't implemented it yet, I've done extensive research and POCs on using **Snowflake Streams**. This is a superior method for handling real-time or near-real-time data as it tracks changes (inserts, updates, and deletes) in a source table and makes them available for consumption by a downstream table. This approach is highly scalable and would be my recommendation for any new incremental pipeline build to reduce manual MERGE logic and increase efficiency."

**Question 2: A data pipeline failed for two hours. How would you recover the missing data and prevent this from happening again?**

**Correction:** The candidate's answer was overly simplistic and lacked a structured, production-ready plan. The interviewer had to step in and provide a better answer. The correct response should show a clear, decisive, and proactive approach.

**Improved Answer:** "This is a critical production incident that requires a clear, step-by-step recovery plan. My approach would be as follows:

1. **Immediate Action & Communication:** The first step is to immediately **stop the downstream processes** that consume this data to prevent further errors or inconsistent reporting. I would then notify the relevant stakeholders (e.g., business owners, data analysts) about the issue and the estimated recovery time.
2. **Assess and Isolate the Issue:** I would then analyze the root cause of the failure. In this case, an expired security key. I would fix the key and verify that the pipeline can now execute successfully.
3. **Backfill the Missing Data:** Instead of simply re-running the main pipeline, which could cause data inconsistencies, I would execute a **backfill process**. This involves running a separate, purpose-built pipeline specifically designed to ingest the missing files from the landing zone and load them into our staging and target layers. This backfill job might run on a larger, temporary Snowflake warehouse to speed up processing.
4. **Resume Normal Operations:** Once the backfill is complete and the data is caught up to the current time, I would resume the regular pipeline schedule.

To **prevent this from happening again**, I would implement two key measures:

* **Proactive Monitoring and Alerting:** Set up monitoring on key metrics like the pipeline's execution time, the number of processed records, and, most importantly, the expiration date of all security keys or credentials.
* **Robust Pipeline Design:** We would build a more resilient pipeline that can automatically retry failed jobs and has a separate, parameterized **backfill script** ready for quick execution. This separates the logic for a one-time historical load from the daily incremental loads, making recovery more robust."

**Question 3: What's your preferred strategy for migrating 10 TB of data from Oracle to Snowflake?**

**Correction:** The candidate's answer was confused. They initially chose the wrong option (stored procedures) and didn't provide a strong justification for their final choice (Databricks), showing a lack of understanding of the full capabilities of each tool.

**Improved Answer:** "Given the scale and the requirement for both a historical load and daily incremental updates, the best approach is a hybrid one, leveraging the strengths of multiple tools.

For the **historical 10 TB load**, I would choose **Databricks** or **Azure Data Factory**. These are powerful orchestration tools ideal for a one-time bulk migration. I would use them to:

* **Extract and Chunk:** Extract the data from Oracle in parallel and write it to a cloud storage layer (like Azure Blob Storage) in manageable chunks (e.g., Parquet or Avro files).
* **Transform (Optional):** Perform any necessary schema changes or initial transformations using PySpark within Databricks.
* **Bulk Load:** Use Databricks or ADF's native connectors to efficiently load these large files into Snowflake using a single COPY INTO command, which is optimized for bulk data ingestion.

For the **daily incremental loads**, **Snowpipe** is the superior solution.

* **Continuous Ingestion:** We would set up a process to automatically export new or updated data from Oracle and land it in the cloud storage.
* **Automated and Serverless:** Snowpipe would then automatically and continuously load these new files into our Snowflake staging tables as they arrive, without the need for manual scheduling or a running virtual warehouse.

The combination of **Databricks** or **Azure Data Factory** for the initial heavy lift and **Snowpipe** for the continuous, real-time ingestion provides the most efficient, scalable, and cost-effective solution for this scenario."