

‘1. Why use 3rd-party tools like NiFi or Juniper instead of GCP native solutions for file transfer to GCS?’

Third-party tools like NiFi or Juniper are sometimes preferred over native GCP solutions (e.g., Cloud Functions, Cloud Scheduler) for the following reasons:

- **Complex Workflows:** Third-party tools allow for complex, real-time data flow orchestration, data transformations, and error handling, which may be challenging to implement using GCP’s simpler tools.
 - **Integration with Diverse Sources:** NiFi, for instance, supports connections with various data sources beyond GCP, offering high versatility in hybrid or multi-cloud environments.
 - **Enhanced Monitoring and Control:** They provide advanced monitoring, data lineage, and control over data flows, making them suitable for large-scale or enterprise environments.
 - **Data Transformation Flexibility:** Tools like NiFi provide built-in processors for data cleaning, filtering, and enriching as data moves, which is ideal for complex pipelines.
-

2. Avro vs. Parquet File

- **Avro:** Avro is a **row-based** file format, efficient for write-heavy workloads, serialization, and streaming data. It supports schema evolution, which makes it flexible for cases where data structures may change over time.
 - **Parquet:** Parquet is a **columnar** storage format optimized for analytical and read-heavy queries. It provides excellent compression and query performance for large datasets with complex, nested structures, commonly used with big data tools like BigQuery and Spark.
-

3. Data Lake vs. Data Warehouse

- **Data Lake:** A storage system designed for handling **unstructured, semi-structured, and structured data** in its native format. Data lakes are cost-effective for storing vast amounts of raw data but require additional processing for analytics.
 - **Data Warehouse:** Optimized for **structured data** and analytical queries, using schema-on-write (data is cleaned/organized before storage). Ideal for business intelligence and reporting, with faster query performance but higher storage costs compared to data lakes.
-

4. Data Modeling

Data modeling is the process of defining the **structure, relationships, and constraints** of data in a database. Effective modeling helps ensure data consistency, integrity, and efficiency, often involving **conceptual, logical, and physical schemas**. Key benefits include:

- Improved data accuracy and consistency
 - Reduced redundancy and optimized query performance
 - A clear blueprint for data management and growth
-

5. Data Lineage

Data lineage is a record of the **data's lifecycle**, tracking its origin, transformations, and movements across systems. This helps with:

- **Compliance and Auditing:** Knowing where data originated and how it's been processed is crucial for regulatory requirements.
 - **Debugging and Issue Resolution:** Data lineage clarifies where errors may have occurred in a pipeline. Tools like Apache Atlas and Collibra are commonly used for managing data lineage.
-

6. Airflow – Versions and Types

- **Versions:** Airflow's latest versions (2.x) offer enhanced features, including task groups, REST APIs, and better DAG scheduling.
 - **Types:** Two main types include the **Community Edition** (open-source) and **Cloud Composer** (GCP's managed version of Airflow). Community Edition offers flexibility, while Cloud Composer provides Google-managed infrastructure with added security, scalability, and integration benefits.
-

7. Time Travel in BigQuery

BigQuery Time Travel allows users to query historical data by specifying a previous timestamp. This can be done using the `FOR SYSTEM_TIME AS OF` clause, which can retrieve data up to **7 days in the past** for standard tables.

Example:

```
sql
Copy code
SELECT *
FROM your_table
FOR SYSTEM_TIME AS OF TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 3 DAY)
```

This feature is useful for data auditing, tracking changes, or debugging without needing to restore backups.

8. Data Governance

Data governance involves policies, standards, and roles for managing data securely and effectively. Key elements include:

- **Data Quality:** Ensuring data accuracy and consistency.
 - **Data Security:** Implementing access controls and protection for sensitive data.
 - **Compliance:** Adhering to regulations (e.g., GDPR).
 - **Data Lineage and Cataloging:** Tracking data's origins and transformations.
- Effective governance enhances decision-making, data protection, and compliance.
-

9. Data Validation

Data validation involves verifying data for accuracy, quality, and consistency before it's processed. Techniques include:

- **Type Checking:** Ensuring values meet expected data types.
 - **Range and Format Validation:** Confirming values fall within specified ranges and formats.
 - **Referential Integrity:** Checking for valid relationships between tables. Validation can occur at multiple stages, such as during ETL, ensuring data is accurate and reliable.
-

10. High Availability in BigQuery

BigQuery offers high availability by:

- Distributing data across **multiple zones** within a region for redundancy.
 - Automatic **failover mechanisms** ensure minimal disruption. This ensures BigQuery remains available even if one zone fails, maintaining data accessibility for critical workloads.
-

11. Materialized Views

A materialized view in BigQuery stores precomputed query results, ideal for **frequently accessed, aggregated data**. It improves performance by reducing repetitive computations. These views are **automatically refreshed** at set intervals, ensuring up-to-date data for fast retrieval.

12. What is a DAG?

A DAG (Directed Acyclic Graph) in Airflow is a workflow with **tasks arranged sequentially** without cycles. DAGs define both **task order and dependencies** and are used to schedule complex workflows in ETL or data processing. Each task runs independently, ensuring a structured flow from start to finish.

13. GCS Storage Classes

Google Cloud Storage offers these storage classes based on access frequency:

- **Standard:** For frequently accessed data.
 - **Nearline:** For data accessed about once a month.
 - **Coldline:** For data accessed less than once a year.
 - **Archive:** For rarely accessed data with long-term storage needs.
-

14. Best Storage Class for Access After 90 Days

Coldline storage is optimal for data accessed every 90 days due to its low storage costs and 90-day minimum storage duration. For data rarely accessed but retained long-term, **Archive** storage would be more cost-effective.

15. BigQuery AI

BigQuery AI, via BigQuery ML, allows users to build and deploy ML models using SQL directly within BigQuery. Supported models include **classification, regression, clustering, and time-series forecasting**. This enables ML on large datasets without external infrastructure, making it easy for analysts to leverage ML on big data.

16. Is Indexing a Best Practice in BigQuery?

Traditional indexing is not used in BigQuery. Instead, BigQuery uses:

- **Partitioning:** Dividing tables into segments (e.g., by date).
 - **Clustering:** Organizing data within partitions. These methods improve query performance, especially for large tables.
-

17. Object Versioning in GCS

Object versioning in Google Cloud Storage keeps multiple versions of an object, providing backup in case of accidental deletion or modification.

Enabling Object Versioning:

- **gcloud CLI:**

```
bash
Copy code
gcloud storage buckets update gs://YOUR_BUCKET_NAME --versioning
```

- **Console:**

- Go to Cloud Storage > Buckets.
- Select your bucket, enable **Object versioning** in settings.

List versions:

```
bash
Copy code
gsutil ls -a gs://YOUR_BUCKET_NAME/YOUR_OBJECT
```

18. Data Masking in BigQuery

Data masking in BigQuery can be done with the **DLP API** or SQL functions like `REGEXP_REPLACE` for basic masking.

SQL Example:

```
sql
Copy code
SELECT
  name,
  REGEXP_REPLACE(phone_number, r"(\d{3})\d{4}(\d{3})", r"\1****\2") AS
masked_phone_number
FROM your_table
```

Using DLP API:

Configure data masking transformations for sensitive data fields. The DLP API applies transformations, like redacting or tokenizing data for enhanced protection.

19. Python Code to Split JSON Array into Rows

Using Pandas, you can split array values in JSON into separate rows with `explode`:

```
python
Copy code
import pandas as pd
```

```
# Sample data
data = [{"id": 1, "city": ["New York", "Los Angeles"]}, {"id": 2, "city":
["Chicago", "Houston"]}]]
df = pd.DataFrame(data)

# Explode 'city' column
df = df.explode('city').reset_index(drop=True)
print(df)
```

This code will create a DataFrame with each array element as a separate row for easier analysis.

20 . what if we dont know the column name first.... so how to add the column... so cant we load.... without creating the column first

ChatGPT said:

ChatGPT

If you don't know the column name beforehand and want to load data from a raw file into a BigQuery table that may have additional columns, you can take the following approach:

Steps to Load Data with Unknown Columns into BigQuery

1. **Load the Data into a Staging Table:** Create a staging table with a flexible schema (using `STRUCT` or `JSON` types) to accommodate the unknown column. This allows you to load the data without specifying every column name.

Example Staging Table Schema:

```
sql
Copy code
CREATE TABLE `your_project_id.your_dataset.staging_table` (
  col1 STRING,
  col2 STRING,
  extra_column STRUCT<column_name STRING>
);
```

You can also create it without the `extra_column` if you want to keep the schema flexible, like:

```
sql
Copy code
CREATE TABLE `your_project_id.your_dataset.staging_table` (
  col1 STRING,
  col2 STRING,
  raw_data STRING
);
```

2. **Load Data into the Staging Table:** Use the `bq` command-line tool or the BigQuery console to load the raw file into the staging table.

Using bq Command Line:

```
bash
Copy code
bq load \
--source_format=CSV \
your_project_id:your_dataset.staging_table \
gs://your_bucket/your_raw_file.csv
```

3. **Extract Schema from Staging Table:** After loading, you can query the staging table to inspect the data and identify any new columns.

Example Query:

```
sql
Copy code
SELECT *
FROM `your_project_id.your_dataset.staging_table`
LIMIT 10;
```

4. **Update the Main Table Schema:** If you identify the extra column(s) you want to keep, you can add them to your existing table using the `ALTER TABLE` statement.

Using SQL:

```
sql
Copy code
ALTER TABLE `your_project_id.your_dataset.your_main_table`
ADD COLUMN extra_col_name STRING; -- Replace with the actual column
name and data type
```

5. **Insert Data into the Main Table:** Once the main table has been updated, you can insert the relevant data from the staging table into your main table. You can choose which columns to include based on what you identified.

Example Insert:

```
sql
Copy code
INSERT INTO `your_project_id.your_dataset.your_main_table` (col1,
col2, extra_col_name)
SELECT col1, col2, raw_data.extra_column.column_name -- Use STRUCT
if applicable
FROM `your_project_id.your_dataset.staging_table`;
```

6. **Cleanup:** After you have successfully transferred the data, you can drop the staging table if it is no longer needed.

Drop Staging Table:

```
sql
Copy code
DROP TABLE `your_project_id.your_dataset.staging_table`;
```

Key Points

- **Staging Table:** This allows you to flexibly load data without defining all columns upfront.
- **Data Inspection:** After loading, you can analyze the staging table to understand the structure and contents.
- **Dynamic Handling:** By using the staging approach, you can adapt to changes in incoming data without prior knowledge of the schema.

21. Basic Checks While Loading Data into GCS

When loading data into Google Cloud Storage (GCS), consider the following basic checks:

- **File Format Validation:** Ensure the file format (e.g., CSV, JSON) matches the expected format for downstream processing.
 - **Data Integrity:** Check for data corruption or incomplete files before upload.
 - **File Naming Conventions:** Use consistent naming patterns to simplify identification and retrieval.
 - **Size Limitations:** Verify that file sizes adhere to GCS limits; large files may require chunking.
 - **Access Permissions:** Ensure proper permissions are set for the bucket to avoid access issues post-upload.
 - **Metadata Consistency:** Include relevant metadata for the files to facilitate management and tracking.
-

22. How to See Metadata in BigQuery

You can view metadata in BigQuery using the `INFORMATION_SCHEMA` views or via the BigQuery console. Here's how to do it:

Using SQL:

```
sql
Copy code
SELECT *
FROM `your_project_id.your_dataset.INFORMATION_SCHEMA.TABLES`
WHERE table_name = 'your_table_name'
```

This will return metadata about the specified table, including schema, creation time, last modified time, etc.

In BigQuery Console:

- Go to the BigQuery console.
 - Navigate to your dataset and select the table.
 - Click on the "Details" tab to view metadata like schema, size, and row count.
-

23. Data Lake vs. Data Warehouse

(This question was previously answered; however, here's a brief recap)

- **Data Lake:** A storage repository that holds vast amounts of raw data in its native format (structured, semi-structured, unstructured). It's cost-effective and flexible but requires processing before analysis.
 - **Data Warehouse:** A centralized repository optimized for structured data storage and retrieval, supporting complex queries and analytics. Data is cleaned and transformed before storage, providing high performance for reporting and BI tasks.
-

24. Display Only Duplicate Records in Python

To find and display only duplicate records from a list in Python, you can use the following code:

```
python
Copy code
a = [3, 3, 4, 6, 7, 1, 9, 7, 1, 1, 1, 0]

# Using a set to track duplicates
seen = set()
duplicates = set()

for num in a:
    if num in seen:
        duplicates.add(num)
    else:
        seen.add(num)

# Displaying duplicates
print(list(duplicates))
```

Output:

```
csharp
Copy code
[1, 3, 7]
```

25. SCD Type 1 or Type 2 in BigQuery

- **SCD Type 1:** Updates existing records in place, overwriting old data with new data. No history is preserved.
- **SCD Type 2:** Adds new records for changes, preserving the historical data while maintaining the current data in a way that allows tracking changes over time. This often involves using an additional column to indicate the active version of the record.

If currently implemented, you would specify whether your process uses SCD Type 1 or Type 2 depending on the requirement for historical data retention.

26. What is SCD Type 2

Slowly Changing Dimension (SCD) Type 2 is a data warehousing technique used to manage and preserve historical data in dimensional tables. It allows you to track changes over time by:

- Adding a new record for each change instead of overwriting the existing record.
- Maintaining additional columns (like `start_date`, `end_date`, and `is_active`) to indicate the validity period of each record.
- This approach enables analytics on historical data, providing a complete view of changes to dimensions over time.

Example Table Structure:

CustomerID	Name	StartDate	EndDate	IsActive
1	Alice	2020-01-01	2021-01-01	0
1	Alice	2021-01-02	NULL	1

27. Different Airflow Operators

Apache Airflow provides various operators to define tasks within a DAG. Key operators include:

- **BashOperator:** Executes a bash command or script.
- **PythonOperator:** Calls a Python function.
- **DummyOperator:** Does nothing, often used as a placeholder.
- **BranchPythonOperator:** Allows conditional branching based on a Python function's output.
- **SqlOperator:** Executes SQL queries against a database.
- **HttpOperator:** Makes HTTP requests to external APIs.
- **DockerOperator:** Runs tasks inside a Docker container.
- **KubernetesPodOperator:** Launches tasks in Kubernetes pods.

Operators can be combined to build complex workflows that orchestrate a variety of tasks.

28. In order to minimize the cost of storage in GCS, what will you do?

To minimize storage costs in Google Cloud Storage (GCS), you can:

- **Choose Appropriate Storage Classes:** Utilize different storage classes (Standard, Nearline, Coldline, Archive) based on access frequency. For infrequently accessed data, prefer Nearline or Coldline, which are cheaper.
- **Implement Object Lifecycle Management:** Set up lifecycle rules to automatically transition or delete objects based on age or other criteria, minimizing costs by ensuring you're not paying for unused data.

- **Use Compression:** Store data in compressed formats (e.g., Gzip, Snappy) to reduce the amount of storage space used.
- **Consolidate Small Files:** Combine smaller files into larger ones to minimize overhead and storage costs associated with many small objects.

29. Have you done partitioning in your project?

Yes, I have implemented partitioning in my projects, particularly in BigQuery. Partitioning helps improve query performance and manage large datasets more efficiently by dividing tables into smaller, more manageable pieces based on specific criteria (e.g., date).

30. What are the different methods of partitioning?

In BigQuery, the main methods of partitioning are:

- **Time-based Partitioning:** Divides tables based on a **TIMESTAMP** or **DATE** column, typically used for time-series data. This allows you to query specific time ranges efficiently.
- **Integer Range Partitioning:** Segments tables based on an integer column, useful for numerical datasets.
- **Partitioned Tables:** When creating a table, you can specify it as partitioned using SQL, which helps in managing and optimizing query performance.

31. What is clustering, and what is the difference between partitioning and clustering? Can we do both at a time?

Clustering in BigQuery organizes data within each partition based on the values in specified columns, enabling faster query performance for those columns. The main differences are:

- **Partitioning** splits the data into separate segments (partitions), usually by time or range, while **clustering** sorts data within those partitions based on one or more columns.
- Both techniques can be used together: you can have a partitioned table that is also clustered by specific columns to enhance performance further.

32. What are views, and what is the level of access to downstream people?

Views in BigQuery are virtual tables that represent the result of a SQL query. They do not store data themselves but provide a way to simplify complex queries or restrict access to specific data. The level of access to views can be controlled using IAM roles, allowing downstream users to view the data without direct access to the underlying tables.

33. What is the benefit of using Cloud Composer when an open-source service (Apache Airflow) is available?

Cloud Composer, as a managed service for Apache Airflow, offers several benefits over self-managed Airflow installations:

- **Managed Infrastructure:** Google manages the underlying infrastructure, reducing operational overhead and maintenance efforts.
- **Automatic Scaling:** Cloud Composer can automatically scale resources based on the workload, ensuring efficiency.
- **Integration with GCP:** It provides seamless integration with other Google Cloud services, making it easier to create workflows that involve different GCP components.
- **Simplified Upgrades:** Google handles upgrades, allowing users to leverage the latest features without manual intervention.

34. How have you followed Agile methodology?

I have followed Agile methodology by participating in regular stand-up meetings, sprint planning, and retrospectives. Our teams used tools like JIRA to manage tasks and track progress, enabling flexibility and iterative improvements based on stakeholder feedback and changing requirements.

35. Explain how you used GitHub.

I used GitHub for version control and collaboration on code. We created repositories for our projects, allowing multiple team members to contribute simultaneously. GitHub's branching strategy helped manage feature development and code reviews, and we utilized pull requests to ensure quality and maintainability before merging code into the main branch.

36. Difference between Jupyter Notebook and VS Code.

- **Jupyter Notebook:** Primarily used for data analysis, visualization, and prototyping. It supports inline code execution, making it great for exploratory data science work. Notebooks can combine code, text, and visualizations.
- **VS Code:** A full-featured code editor that supports multiple programming languages. It is designed for software development, offering extensive extensions, debugging capabilities, and version control integration, making it suitable for larger projects.

37. What is the use of Dataflow?

Google Cloud Dataflow is a fully managed service for processing and analyzing streaming and batch data. It allows for real-time data processing and provides a unified programming model using Apache Beam, enabling developers to build complex data pipelines efficiently and at scale, with automatic resource management and scaling.

38. Explain briefly how you used Airflow and Pub/Sub.

I used Apache Airflow to orchestrate data workflows that interacted with Google Cloud Pub/Sub for event-driven data processing. Airflow DAGs would listen for messages on Pub/Sub topics, triggering data ingestion tasks upon receiving new data events. This setup allowed us to react to real-time data streams and ensure timely processing.

39. Have you ever done data masking?

Yes, I have implemented data masking techniques to protect sensitive information in datasets. This involved using methods such as redaction, tokenization, or SQL functions to obfuscate sensitive data fields (e.g., credit card numbers, personal identifiers) while maintaining usability for analysis and reporting.

40. What is the use of encryption? Explain with an example.

Encryption is used to protect sensitive data by transforming it into an unreadable format for unauthorized users. In BigQuery, data is automatically encrypted at rest and in transit using Google-managed encryption keys, which helps maintain data confidentiality and compliance with regulations.

Example:

When you load sensitive data into BigQuery, you don't need to do anything special to enable encryption, as it is managed by Google Cloud. However, if you want to use customer-managed encryption keys (CMEK) for additional control, you can specify a Cloud KMS key during the table creation or data load.

SQL Example of Creating a Table with CMEK:

```
sql
Copy code
CREATE TABLE `your_project.your_dataset.your_table`
OPTIONS (
  description="This table contains sensitive data.",
  encryption_configuration = (
    kms_key_name="projects/your-project/locations/global/keyRings/your-
keyring/cryptoKeys/your-key"
  )
) AS
SELECT *
FROM `your_project.your_dataset.source_table`;
```

In this example:

- **Table Creation:** The SQL statement creates a new table in BigQuery.
- **Encryption Configuration:** The `encryption_configuration` option specifies a Cloud KMS key to encrypt the table's data. This provides the organization with control over the encryption keys used to protect sensitive data.

When querying this table, the data will be automatically decrypted on-the-fly by BigQuery, allowing users with appropriate permissions to access the data seamlessly.

Summary

In summary, encryption in BigQuery is crucial for safeguarding sensitive data, and using CMEK allows organizations to manage their encryption keys for enhanced security and compliance. If you have further questions or need more specific details, let me know!

4o mini

You said:

kms_key_name="projects/your-project/locations/global/keyRings/your-keyring/cryptoKeys/your-key" here what is there in this link..... what and how we configured

ChatGPT said:

ChatGPT

In the `kms_key_name` option of the BigQuery table creation SQL command, you specify the resource path to a Cloud KMS (Key Management Service) encryption key that will be used to encrypt the data in the table. Here's a breakdown of the components in that path and how to configure it:

KMS Key Path Breakdown

plaintext

Copy code

```
projects/your-project/locations/global/keyRings/your-  
keyring/cryptoKeys/your-key
```

- **projects/your-project:** This part indicates the Google Cloud project where the KMS key is located. You need to replace `your-project` with your actual project ID.
- **locations/global:** This specifies the location of the KMS resources. It can be either `global` or a specific region (e.g., `us-central1`). Using `global` allows you to create keys that are accessible across multiple regions.
- **keyRings/your-keyring:** A key ring is a collection of keys. Replace `your-keyring` with the name of your key ring where you want to store the encryption key.
- **cryptoKeys/your-key:** This is the specific key within the key ring that will be used for encryption. Replace `your-key` with the name of your actual encryption key.

How to Configure a KMS Key

1. **Create a Key Ring:**
 - Go to the **Google Cloud Console**.
 - Navigate to **Security > Cryptographic Keys**.
 - Click on **Create key ring**.
 - Enter a name and select a location (either `global` or a specific region).
 - Click **Create**.
2. **Create an Encryption Key:**
 - After creating a key ring, click on it.
 - Click on **Create key**.
 - Specify a name for your encryption key.
 - Choose **Symmetric** or **Asymmetric** key type (for most use cases, use **Symmetric**).
 - Configure the key options, including rotation and permissions if needed.
 - Click **Create**.
3. **Set IAM Permissions:**
 - Ensure that the service account used by BigQuery has the appropriate permissions to use the KMS key. You can grant roles like `roles/cloudkms.cryptoKeyEncrypter` and `roles/cloudkms.cryptoKeyDecrypter` to the service account.

Example of Full KMS Key Path

If your project ID is `my-gcp-project`, your key ring is named `my-keyring`, and your encryption key is `my-encryption-key`, the KMS key path would look like this:

```
plaintext
Copy code
projects/my-gcp-project/locations/global/keyRings/my-keyring/cryptoKeys/my-encryption-key
```

Summary

By specifying the KMS key path in your BigQuery table options, you ensure that the data loaded into the table is encrypted using your custom key, giving you greater control over data security. If you have any further questions or need additional examples, feel free to ask!

41. How did you do data validation and cleaning?

Data validation and cleaning involved several steps:

- **Validation Checks:** Implemented checks to ensure data integrity, such as verifying data types, ranges, and uniqueness constraints.
- **Handling Missing Values:** Employed strategies like imputation, removal, or replacement of missing data based on business rules.
- **Data Transformation:** Cleaned and transformed data to conform to required formats (e.g., date formats, categorical values) using tools like SQL and Dataflow.
- **Automated Pipelines:** Developed automated data validation pipelines in Dataflow to continuously monitor and clean incoming data before it reached the final storage.

This comprehensive approach ensures that the data is reliable, accurate, and ready for analysis.

40. What is Data Catalog and Data Governance?

Data Catalog: Google Cloud Data Catalog is a fully managed metadata management service that helps organizations discover, manage, and understand their data assets. It provides a central repository for metadata, enabling easier data discovery, tagging, and annotation across Google Cloud services, improving collaboration and efficiency.

Data Governance: Data governance refers to the overall management of data availability, usability, integrity, and security within an organization. It involves defining policies, procedures, and standards to ensure that data is accurate, consistent, and used appropriately. Effective data governance helps organizations comply with regulations, maintain data quality, and protect sensitive information.

41. What are the optimization techniques used in BQ that you implemented in your current project?

1. **Partitioning:** Segmenting tables based on date or another column to reduce the amount of data scanned during queries.

2. **Clustering:** Organizing data within tables based on specific columns to improve query performance by minimizing scanned data.
3. **Materialized Views:** Creating pre-computed views to speed up access to frequently queried data, reducing query execution time.
4. **Query Optimization:** Using best practices to write efficient queries, such as avoiding `SELECT *`, using proper filters, and minimizing joins.
5. **Denormalization:** In specific scenarios, combining tables to reduce the need for joins, thus improving query performance.

42. What happens to the data when not used for a long time in BQ?

Data that is not accessed for 90 consecutive days in BigQuery is classified as "inactive." While this does not incur additional storage costs, it may be subject to deletion if the dataset is not preserved or if it is deleted. Users can set retention policies for datasets and tables to manage their lifecycle effectively.

43. What is Cloud Spanner and the difference between Cloud Spanner, BigQuery, and Cloud SQL?

Cloud Spanner: A fully managed, horizontally scalable, relational database service designed for mission-critical applications, offering strong consistency and high availability.

Differences:

- **Data Model:** Cloud Spanner uses a relational model, BigQuery is optimized for analytics (handling semi-structured data), and Cloud SQL supports MySQL, PostgreSQL, and SQL Server.
- **Use Cases:** Spanner is best for operational workloads, BigQuery for analytics and reporting, and Cloud SQL for traditional database applications.
- **Scalability:** Spanner scales horizontally, BigQuery is serverless and scales automatically for analytics, while Cloud SQL has vertical scaling limitations.

44. How do you connect the API to Spanner?

To connect an application to Cloud Spanner via its API:

1. **Set Up Authentication:** Use a service account with appropriate IAM roles (e.g., `roles/spanner.databaseAdmin`).
2. **Use Client Libraries:** Utilize official client libraries for the programming language being used (e.g., Python, Java).

Python Code Example:

```
python
Copy code
from google.cloud import spanner

# Create a Spanner client
spanner_client = spanner.Client()

# Specify your instance and database
```



```
instance = spanner_client.instance('your-instance-id')
database = instance.database('your-database-id')

# Perform operations
with database.batch() as batch:
    batch.insert(
        table='YourTable',
        columns=('Column1', 'Column2'),
        values=[('value1', 'value2')]
    )
```

45. What is Broadcast Join?

A **Broadcast Join** is a join strategy used when one of the tables in the join operation is small enough to be sent to all nodes in a distributed system. By broadcasting the smaller table, each node can perform the join locally, reducing the amount of data transferred over the network and significantly improving performance.

Example in PySpark:

```
python
Copy code
from pyspark.sql import SparkSession
from pyspark.sql.functions import broadcast

spark = SparkSession.builder.appName("BroadcastJoinExample").getOrCreate()

# Create two DataFrames
large_df = spark.read.csv("large_data.csv", header=True)
small_df = spark.read.csv("small_data.csv", header=True)

# Perform a broadcast join
joined_df = large_df.join(broadcast(small_df), "join_key")
```

46. ACID Properties example with your project where you have used it?

ACID Properties refer to the four key properties of database transactions:

- **Atomicity:** Ensures that all operations within a transaction are completed; if not, the transaction fails and the database state is unchanged.
- **Consistency:** Ensures that a transaction brings the database from one valid state to another.
- **Isolation:** Ensures that transactions are executed independently without interference.
- **Durability:** Guarantees that once a transaction is committed, it will remain so, even in the case of a system failure.

Example: In a financial application using Cloud Spanner, when a money transfer occurs between two accounts, the transaction updates the balance of both accounts. If either update fails, the entire transaction is rolled back, maintaining data consistency and integrity.

47. How many types of SCD and related to your current project and example it briefly?

Types of Slowly Changing Dimensions (SCD):

1. **Type 0:** Retain original data; no changes.
2. **Type 1:** Overwrite old data with new data.
3. **Type 2:** Maintain historical data by creating new records for changes.
4. **Type 3:** Store current and previous values in the same record.

Example: In my project, I implemented **Type 2 SCD** for customer records. When a customer's address changes, a new record is created with a new effective date, while the old record is preserved with an end date.

48. What are the optimization techniques you take if the pipeline is running too long?

1. **Data Filtering:** Apply filters early in the pipeline to reduce the amount of data processed.
2. **Parallel Processing:** Leverage parallelism in processing tasks to improve execution time.
3. **Optimize Transformations:** Rewrite inefficient transformations to improve performance (e.g., using built-in functions).
4. **Resource Allocation:** Ensure that the appropriate amount of resources (e.g., memory, CPU) is allocated for the job.
5. **Monitoring and Logging:** Use monitoring tools to identify bottlenecks and optimize based on logs.

49. What is Docker and Kafka? Explain it briefly with deployments.

Docker: A platform that enables developers to automate the deployment of applications in lightweight, portable containers. Containers encapsulate an application and its dependencies, ensuring consistent environments across development, testing, and production.

Example Deployment:

```
bash
Copy code
# Dockerfile example
FROM python:3.8
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]

# Build and run the container
docker build -t myapp .
docker run -p 5000:5000 myapp
```

Kafka: A distributed streaming platform used for building real-time data pipelines and streaming applications. It allows you to publish and subscribe to streams of records and store them in a fault-tolerant way.

Example Deployment:

```
bash
Copy code
# Run Kafka using Docker
docker run -d --name kafka -p 9092:9092 \
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092 \
  wurstmeister/kafka
```

50. Explain UDF Dataflow templates used for streaming pipelines?

User-Defined Functions (UDF) in Dataflow allow you to create custom processing logic that can be applied to data within streaming pipelines. UDFs can be written in Java or Python and can be used to perform complex transformations or enrich data streams.

Example:

```
python
Copy code
import apache_beam as beam

# Define a UDF
def add_prefix(element):
    return f"prefix_{element}"

# Use the UDF in a Dataflow pipeline
with beam.Pipeline() as pipeline:
    (pipeline
     | 'ReadFromPubSub' >> beam.io.ReadFromPubSub(subscription='your-
subscription')
     | 'ApplyUDF' >> beam.Map(add_prefix)
     | 'WriteToSink' >> beam.io.WriteToText('output.txt'))
```

If you have more questions or need further details, feel free to ask!

51. Why are you using Dataproc and why loading into BigQuery?

Dataproc is a managed Spark and Hadoop service that simplifies running big data frameworks. It's used for processing large datasets using batch or stream processing. Loading data into BigQuery enables fast and scalable analysis of the processed data using SQL-like queries. This combination allows for efficient data processing and analysis in the Google Cloud environment.

52. How would you verify your migration is correct or not?

Verification can be done by:

- **Data Sampling:** Compare sample records between the source and destination to ensure consistency.
- **Count Validation:** Ensure the row counts match between the source and destination tables.
- **Checksums:** Compute checksums for data sets to verify integrity.

- **Validation Queries:** Run queries to check for expected results or anomalies in the migrated data.
- **End-to-End Testing:** Validate data flow through the entire pipeline to ensure that transformations were applied correctly.

53. How are you optimizing your migration?

- **Batch Processing:** Break down large datasets into smaller chunks to avoid timeouts and improve manageability.
- **Data Filtering:** Only migrate the necessary data to reduce the volume and time taken.
- **Parallel Processing:** Utilize multiple workers for concurrent data loading to speed up the migration.
- **Data Transformation:** Clean and transform data during the migration to minimize post-migration processing.

54. Do you know about BigQuery SQL?

Yes, BigQuery SQL is a variant of SQL designed specifically for querying large datasets in BigQuery. It supports standard SQL features like SELECT, JOIN, GROUP BY, and supports functions for analytics, windowing, and more. BigQuery also offers unique features like nested and repeated fields to work with semi-structured data.

55. Can you explain one complex transformation you have done using Dataproc?

In a recent project, I used Dataproc to perform a complex ETL process involving multiple data sources. This included:

- **Data Ingestion:** Reading data from multiple sources (GCS, Cloud SQL).
- **Data Cleansing:** Removing duplicates, handling null values, and standardizing formats using Spark transformations.
- **Aggregation:** Grouping data by certain criteria and calculating metrics (e.g., total sales per region).
- **Output:** Writing the transformed dataset back to GCS for further analysis in BigQuery.

56. What is the most complicated query you have written?

A complex query involved multiple joins across several tables with window functions to calculate rolling averages and rank users based on their activity. For example:

```
sql
Copy code
SELECT user_id,
       order_date,
       SUM(order_amount) OVER (PARTITION BY user_id ORDER BY order_date
                               ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_total
FROM orders
JOIN users ON orders.user_id = users.id
WHERE users.status = 'active';
```

57. Have you used joins in BigQuery?

Yes, I have utilized various types of joins in BigQuery, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN, to combine data from multiple tables based on matching criteria. These joins facilitate complex queries and enhance data analysis capabilities.

58. Where are you triggering Dataproc jobs?

Dataproc jobs can be triggered through various means, such as:

- **Cloud Composer:** Orchestrating workflows with Apache Airflow.
- **Direct API Calls:** Using the Google Cloud SDK or client libraries to submit jobs programmatically.
- **Cloud Functions:** Triggering jobs based on events, like file uploads in GCS.
- **Scheduled Jobs:** Setting up scheduled triggers through Cloud Scheduler to automate regular processing.

59. Which language you used in DAG?

In DAG (Directed Acyclic Graph) definitions within Cloud Composer, I primarily use Python. This allows me to define workflows, tasks, dependencies, and parameters effectively, leveraging Python's capabilities for complex task execution.

60. What is object versioning in GCS?

Object versioning in Google Cloud Storage (GCS) allows you to keep multiple versions of an object in a bucket. When versioning is enabled, GCS retains older versions of an object whenever it is overwritten or deleted. This feature is useful for recovering from accidental deletions or modifications. To enable versioning, you can set the `versioning` configuration on a bucket using the GCS Console or via the `gsutil` command line tool:

```
bash
Copy code
gsutil versioning set on gs://your-bucket-name
```

61. What are the optimization techniques in BigQuery?

Optimization techniques in BigQuery include:

1. **Partitioning:** Organizes tables into segments, typically based on a date column, to reduce data scanned in queries.
2. **Clustering:** Sorts data within a partition by one or more columns, improving query performance by reducing data scanned.
3. **Materialized Views:** Stores pre-computed query results for faster access, reducing execution time.
4. **Query Optimization:** Writing efficient SQL queries by avoiding `SELECT *`, using filtering conditions, and optimizing joins.
5. **Denormalization:** Combining related tables into a single table to minimize joins and improve read performance.
6. **Table Sampling:** Using sampling methods for analysis on large datasets to reduce costs and improve performance.

62. What are the authorized views and materialized views and their differences?

Authorized Views: These are views that can be shared with users, allowing them to query the view without granting them direct access to the underlying tables. They provide a way to control data access and security.

Materialized Views: These are physical copies of the query result stored in BigQuery, which can be refreshed periodically. They speed up query performance by avoiding the need to re-execute the underlying query each time.

Differences:

- **Storage:** Materialized views consume storage space, while authorized views do not as they query the underlying tables directly.
- **Performance:** Materialized views provide faster query performance since they store pre-computed results, while authorized views do not inherently improve performance.
- **Use Cases:** Authorized views are primarily for access control, while materialized views are for performance optimization.

63. What are the types of joins and explain them?

The types of joins include:

1. **Inner Join:** Returns only the rows where there is a match in both tables.

```
sql
Copy code
SELECT * FROM TableA
INNER JOIN TableB ON TableA.ID = TableB.ID;
```

2. **Left Outer Join:** Returns all rows from the left table and matched rows from the right table; if no match, returns NULL for right table columns.

```
sql
Copy code
SELECT * FROM TableA
LEFT JOIN TableB ON TableA.ID = TableB.ID;
```

3. **Right Outer Join:** Returns all rows from the right table and matched rows from the left table; if no match, returns NULL for left table columns.

```
sql
Copy code
SELECT * FROM TableA
RIGHT JOIN TableB ON TableA.ID = TableB.ID;
```

4. **Full Outer Join:** Returns all rows when there is a match in either table; returns NULL for non-matching rows from both tables.

```
sql
Copy code
SELECT * FROM TableA
FULL OUTER JOIN TableB ON TableA.ID = TableB.ID;
```

64. Number of records retrieved in each join on the below data:

Given the data:

A	B
1	1
0	1
0	NULL
1	3

- **Left Join** with Table B:
 - Records: 4 (All records from Table A)
- **Inner Join:**
 - Records: 3 (Only matching records with B = 1)
- **Full Outer Join:**
 - Records: 4 (All records from both tables, matches included)
- **Right Join:**
 - Records: 3 (All records from Table B with matching records from A)

65. Write a query to remove duplicate records using `DENSE_RANK()`

To remove duplicates based on a specific column (e.g., `ColumnName`):

```
sql
Copy code
WITH RankedData AS (
    SELECT *,
           DENSE_RANK() OVER (PARTITION BY ColumnName ORDER BY SomeColumn)
    AS rank
    FROM YourTable
)
SELECT *
FROM RankedData
WHERE rank = 1;
```

66. What is Airflow Composer?

Cloud Composer is a fully managed workflow orchestration service on Google Cloud that is built on Apache Airflow. It allows you to create, schedule, and manage workflows for data processing, ETL, and other tasks. Composer handles the setup and maintenance of Airflow environments, making it easier for teams to build and deploy workflows in a scalable and reliable manner.

67. What is a DAG?

A Directed Acyclic Graph (DAG) is a collection of tasks with directed edges that show dependencies between them. In Airflow, a DAG defines a workflow of tasks that are executed in a specific order. Each task in a DAG is an operator, and the DAG ensures that tasks are executed in the correct sequence based on their dependencies.

68. What are the dependencies in a DAG?

Dependencies in a DAG are the relationships between tasks that dictate the order in which they should be executed. These can be defined using the `set_upstream()` and `set_downstream()` methods or by using bitwise operators (`>>` or `<<`) in Airflow. Dependencies ensure that a task only runs after its upstream tasks have completed successfully.

69. What is a list and tuple?

- **List:** A mutable collection that can hold multiple items, allowing for modification (addition, deletion, change of elements) after creation.

```
python
Copy code
my_list = [1, 2, 3, 4]
my_list.append(5) # List after addition: [1, 2, 3, 4, 5]
```

- **Tuple:** An immutable collection, meaning once created, its elements cannot be modified. Tuples are typically used to group related data together.

```
python
Copy code
my_tuple = (1, 2, 3, 4)
# my_tuple[1] = 5 # This will raise an error
```

70. What are some libraries you have used in Python?

Some common libraries used in Python include:

1. **Pandas:** For data manipulation and analysis.
2. **NumPy:** For numerical computing and working with arrays.
3. **Requests:** For making HTTP requests to APIs.
4. **SQLAlchemy:** For database interaction using SQL.
5. **Scikit-learn:** For machine learning tasks.
6. **Matplotlib/Seaborn:** For data visualization.

71. Difference between mutable and immutable

- **Mutable:** Objects that can be changed after they are created. Examples include lists, dictionaries, and sets. You can modify the content without creating a new object.

```
python
Copy code
my_list = [1, 2, 3]
my_list[0] = 4 # The list is now [4, 2, 3]
```

- **Immutable:** Objects that cannot be changed after they are created. Examples include tuples, strings, and frozensets. Any modification creates a new object instead.

```
python
Copy code
```



```
my_tuple = (1, 2, 3)
# my_tuple[0] = 4 # This will raise an error
new_tuple = (4,) + my_tuple[1:] # Creates a new tuple (4, 2, 3)
```

72. Difference between list, set, and tuple

- **List:**
 - **Mutable:** Can be modified after creation.
 - **Ordered:** Maintains the order of elements.
 - **Allows duplicates:** Multiple occurrences of the same value are permitted.

```
python
Copy code
my_list = [1, 2, 2, 3]
```

- **Set:**
 - **Mutable:** Can be modified after creation.
 - **Unordered:** Does not maintain order.
 - **No duplicates:** Only unique elements are allowed.

```
python
Copy code
my_set = {1, 2, 2, 3} # Result: {1, 2, 3}
```

- **Tuple:**
 - **Immutable:** Cannot be modified after creation.
 - **Ordered:** Maintains the order of elements.
 - **Allows duplicates:** Multiple occurrences of the same value are permitted.

```
python
Copy code
my_tuple = (1, 2, 2, 3)
```

73. What is the difference between RDD and DataFrame?

- **RDD (Resilient Distributed Dataset):**
 - A fundamental data structure in Spark that represents a distributed collection of objects.
 - Allows for low-level transformations and actions, giving more control but requiring more complex operations.
 - Does not optimize query execution or utilize Spark's Catalyst optimizer.
- **DataFrame:**
 - A higher-level abstraction built on top of RDDs that represents data in a tabular format with named columns.
 - Optimized for performance through Spark's Catalyst optimizer and Tungsten execution engine.
 - Allows for more expressive operations, similar to SQL, and is easier to work with for data manipulation.

74. What is a lambda function?

A lambda function is an anonymous, small function defined with the `lambda` keyword in Python. It can take any number of arguments but can only have one expression. Lambda functions are often used for short, throwaway functions in places where a function is required temporarily.

Example:

```
python
Copy code
# Lambda function to add two numbers
add = lambda x, y: x + y
result = add(3, 5)  # Result: 8
```

75. What is a driver manager?

In the context of database connectivity, a driver manager is a component that manages the connection to a database. It acts as an interface between the application and the database driver, allowing the application to establish connections, manage transactions, and execute SQL statements. In Java, for example, the `DriverManager` class is used to manage database drivers and connections.

Example in Java:

```
java
Copy code
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"password");
```

76. What is a cluster manager?

A cluster manager is a system that manages resources in a distributed computing environment. It allocates resources (such as CPU, memory, and storage) to various applications and services running on the cluster. In Apache Spark, the cluster manager can be YARN, Mesos, or Spark's standalone cluster manager. The cluster manager is responsible for scheduling tasks, monitoring resource usage, and handling failures.

- **YARN (Yet Another Resource Negotiator):** A resource management platform in Hadoop that allows multiple data processing engines to handle data stored in a single platform.
- **Mesos:** A cluster manager that abstracts resources across an entire datacenter, providing efficient resource isolation and sharing across distributed applications.
- **Spark Standalone:** A simple cluster manager included with Spark that makes it easy to set up a cluster without needing external services.

78. SQL questions to convert the first letter of a word to lower case

To convert the first letter of a word to lower case in SQL, you can use the `LOWER()` function in combination with `SUBSTRING()` and `CONCAT()`. Here's an example:

```
sql
Copy code
SELECT CONCAT(LOWER(SUBSTRING(Name, 1, 1)), SUBSTRING(Name, 2)) AS
UpdatedName
FROM YourTable;
```

This code takes the first letter of the `Name`, converts it to lower case, and concatenates it with the rest of the name.

79. SQL question: Give me the position of 's' in "Krishna"

To find the position of the letter 's' in the string "Krishna", you can use the `INSTR()` function:

```
sql
Copy code
SELECT INSTR('Krishna', 's') AS PositionOfS;
```

This will return the position of 's' in the string (it returns 3 in this case, as positions are 1-based).

80. One PySpark code to read a text file, count the number of words, and write to a file

Here's a PySpark example to read a text file, count the number of words, and write the result to a new text file:

```
python
Copy code
from pyspark.sql import SparkSession

# Create a Spark session
spark = SparkSession.builder \
    .appName("WordCount") \
    .getOrCreate()

# Read the text file
text_file = spark.read.text("path/to/input.txt")

# Count the number of words
word_count = text_file.rdd.flatMap(lambda line: line.value.split(" ")) \
    .count()

# Write the result to a text file
with open("path/to/output.txt", "w") as output_file:
    output_file.write(f"Number of words: {word_count}\n")

# Stop the Spark session
spark.stop()
```

81. PySpark commands you used in the project

Common PySpark commands include:

- **Reading data:**

```
python
Copy code
df = spark.read.csv("path/to/file.csv", header=True,
inferSchema=True)
```

- **Writing data:**

```
python
Copy code
df.write.mode('overwrite').parquet("path/to/output")
```

- **Transformation example:**

```
python
Copy code
df_filtered = df.filter(df['column_name'] > 100)
```

- **Aggregation example:**

```
python
Copy code
df_grouped = df.groupBy("column_name").agg({"other_column": "sum"})
```

82. Where do you send the Python code in relation to Airflow?

In Apache Airflow, Python code is typically placed in DAG (Directed Acyclic Graph) files, which are Python scripts that define the workflow and tasks. These DAG files are usually located in the `dags/` folder of your Airflow environment. You can also use the `PythonOperator` to execute specific Python functions or scripts as part of the workflow.

83. BigQuery: Internal vs External Tables

- **Internal Tables:** These tables are stored within BigQuery's managed storage. They are optimized for performance and cost, and you can perform queries on them without worrying about the underlying storage.
- **External Tables:** These tables reference data stored outside BigQuery, such as in Google Cloud Storage (GCS). They allow querying of data without loading it into BigQuery, which can save costs, but performance may be slower compared to internal tables.

84. Cloud Run

Cloud Run is a fully managed compute service that enables you to run containerized applications in a serverless environment. It automatically scales your applications based on incoming requests and allows you to deploy applications without worrying about the underlying infrastructure. You can use any language or framework that can run in a container.

85. Cloud Functions

Cloud Functions is a serverless compute service that allows you to run your code in response to events without managing servers. You can trigger Cloud Functions through various events from other GCP services (like Cloud Storage uploads, Pub/Sub messages, HTTP requests, etc.). It's great for building microservices and automating workflows.

86. What is your source?

This question likely pertains to how you obtain your data. Common sources could include:

- Google Cloud Storage (GCS)
- Google Cloud Pub/Sub
- External databases (e.g., Cloud SQL, BigQuery)
- APIs (e.g., third-party REST APIs)

87. Explain how the data is getting into the bucket

Data can be ingested into a Google Cloud Storage (GCS) bucket in various ways:

- **Manual Upload:** Users can upload files through the Google Cloud Console or `gsutil` command-line tool.
- **Automated Scripts:** Use scripts or applications to move data from other sources (like local file systems or databases) into GCS using the `gsutil` command or GCS API.
- **Streaming:** Stream data directly to GCS using services like Cloud Pub/Sub, where messages can be published and stored as files in a GCS bucket.
- **Scheduled Jobs:** Use tools like Cloud Scheduler or Dataflow to periodically transfer data from various sources into GCS.

88. BigQuery: Hierarchy

In BigQuery, the hierarchy consists of:

- **Project:** The top-level container for your datasets and resources. Each Google Cloud account can have multiple projects.
- **Dataset:** A collection of tables and views within a project. It serves as a way to organize your data.
- **Table:** A structured collection of data within a dataset. Tables can be internal or external.
- **View:** A virtual table based on the results of a query. It can simplify complex queries and provide a layer of security.

89. Name a few `gcloud` commands

Here are some commonly used `gcloud` commands:

- **Authenticate:** `gcloud auth login`
- **Set project:** `gcloud config set project PROJECT_ID`
- **List storage buckets:** `gcloud storage buckets list`

- **Deploy a Cloud Function:** `gcloud functions deploy FUNCTION_NAME --runtime RUNTIME --trigger-http`
- **Create a BigQuery dataset:** `bq mk --dataset DATASET_NAME`

90. What is DBT (Data Build Tool)?

DBT is an open-source tool that enables data analysts and engineers to transform data in their warehouse more effectively. It allows users to write modular SQL queries, run them in a data warehouse, and create dependencies between them. DBT compiles SQL files into raw SQL queries and executes them against the data warehouse, facilitating the creation of tables and views with clear lineage.

Example of a DBT model:

```
sql
Copy code
-- models/my_model.sql
SELECT
    user_id,
    COUNT(order_id) AS order_count
FROM
    {{ ref('orders') }}
GROUP BY
    user_id
```

91. What is Time Theory?

Time theory in data processing refers to how temporal data is stored, managed, and queried. It encompasses concepts like timestamps, time zones, and the granularity of time data. Understanding time theory is crucial for data analytics, as it impacts how time-based analyses are performed.

92. What is Time Series?

Time series data is a sequence of data points collected or recorded at specific time intervals. This type of data is often used in forecasting, trend analysis, and other statistical applications. In BigQuery, time series analysis can be performed using SQL queries to aggregate or analyze data over time.

Example SQL for Time Series Analysis:

```
sql
Copy code
SELECT
    DATE(order_date) AS order_date,
    SUM(order_amount) AS total_sales
FROM
    `project.dataset.orders`
GROUP BY
    order_date
ORDER BY
    order_date
```

93. What is Time Travel in BigQuery?

Time Travel in BigQuery allows users to query historical data as it existed at a specific point in time, up to 7 days in the past. This feature can be particularly useful for auditing and debugging, as it enables access to previous versions of tables without the need for backups.

Example of Time Travel Query:

```
sql
Copy code
SELECT *
FROM `project.dataset.table` AS OF TIMESTAMP '2024-10-20 12:00:00'
```

94. How does Dremel work?

Dremel is a query engine developed by Google that is designed for interactive analysis of large datasets. It allows users to run SQL-like queries on data stored in Google's distributed file system. Dremel uses a multi-level execution model and can efficiently process petabytes of data using a tree architecture for query execution, where leaf nodes handle the execution of queries and aggregate results up the tree.

95. What is Pipe Syntax in BigQuery?

Pipe syntax in BigQuery refers to the use of the pipe (|) operator to denote the flow of data through a series of transformations. This is particularly useful when chaining multiple transformations together.

Example of Pipe Syntax in BigQuery:

```
sql
Copy code
SELECT
    column1,
    column2
FROM
    `project.dataset.table`
WHERE
    column1 = 'value'
| SELECT
    column1, SUM(column2)
| GROUP BY
    column1
```

96. Explain BigQuery Architecture

BigQuery is a serverless, highly scalable, and cost-effective multi-cloud data warehouse. Its architecture is composed of the following components:

- **Storage Layer:** Uses a distributed file system to store data in columnar format for efficient querying.
- **Query Engine:** Utilizes a Dremel-based architecture to execute SQL queries in parallel across multiple nodes.

- **Data Loading:** Supports various methods to load data, including batch loading and streaming inserts.
- **Access Control:** Uses IAM roles and permissions for data security.

97. Explain Databricks Architecture

Databricks is a cloud-based platform that provides a unified analytics workspace. Its architecture includes:

- **Workspace:** A collaborative environment for data engineers and data scientists.
- **Clusters:** Managed Apache Spark clusters for processing big data.
- **Jobs:** Scheduled or on-demand execution of notebooks and workflows.
- **Delta Lake:** Provides ACID transactions and scalable metadata handling for reliable data lakes.

98. Explain Spark Architecture

Apache Spark is a distributed computing framework that enables big data processing. Its architecture consists of:

- **Driver:** The main program that creates the SparkContext and coordinates tasks.
- **Cluster Manager:** Allocates resources and manages cluster nodes (e.g., YARN, Mesos, or Kubernetes).
- **Executors:** Worker nodes that execute tasks and return results to the driver.
- **Tasks:** Small units of work that are distributed across the cluster for parallel processing.

99. Explain Airflow Architecture

Apache Airflow is an open-source platform for orchestrating complex workflows. Its architecture includes:

- **Scheduler:** Responsible for scheduling and monitoring jobs, triggering workflows based on dependencies.
- **Web Server:** Provides a user interface for monitoring and managing workflows.
- **Workers:** Execute tasks defined in Directed Acyclic Graphs (DAGs).
- **Metadata Database:** Stores information about DAGs, task instances, and logs.

100. 2 sql questions: 1. sql has columns Name, subject, total marks(100), marks_obtained Subjects are 5 Now give name, marksobtained, pass percent as new column 2. If all 5 subjects are in a list of single row..... Write it into multiple rows3. As subjects are in 5 different rows... Give a single row output Where Name, subjects(in List), Write syntax in sql and Bigquery

ChatGPT said:

ChatGPT

Here are the SQL queries for your requests:

100. SQL Query to Calculate Pass Percentage

Assuming a passing percentage is defined (for example, 40%), you can calculate the pass percentage as follows:

```
sql
Copy code
SELECT
    Name,
    Marks_Obtained,
    (Marks_Obtained / Total_Marks) * 100 AS Pass_Percent
FROM
    your_table_name
WHERE
    Total_Marks = 100; -- Ensure the total marks are 100
```

101. SQL Query to Convert Subjects from a Single Row to Multiple Rows

If the subjects are stored in a single row as a list and you want to split them into multiple rows, you can use `UNNEST` in BigQuery. If your subjects are stored in a column as an array, the query will look like this:

```
sql
Copy code
SELECT
    Name,
    Subject
FROM
    your_table_name,
    UNNEST(Subjects) AS Subject; -- Assuming Subjects is an ARRAY type
```

If the subjects are stored in a single string separated by commas, you can first convert it to an array and then `unnest`:

```
sql
Copy code
SELECT
    Name,
    SPLIT(Subjects, ',') AS Subject
FROM
    your_table_name;
```

102. SQL Query to Aggregate Subjects into a Single Row with a List

To get a single row output with subjects in a list, you can use `ARRAY_AGG` in BigQuery:

```
sql
Copy code
SELECT
    Name,
    ARRAY_AGG(Subject) AS Subjects -- This aggregates the subjects into an
array
FROM
    your_table_name
GROUP BY
    Name; -- Group by Name to ensure a single row per student
```

Summary

- **First Query** calculates the pass percentage based on marks obtained.
- **Second Query** converts subjects from a single row to multiple rows.
- **Third Query** aggregates subjects into a single row for each student.

You can replace `your_table_name` with the actual name of your table. Adjust the conditions as per your data schema if necessary!

103. What is Spark? What is Hadoop?

- **Spark:** Apache Spark is an open-source, distributed computing system designed for speed, ease of use, and sophisticated analytics. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark supports various programming languages like Scala, Java, Python, and R.
 - **Hadoop:** Apache Hadoop is an open-source framework that allows for the distributed processing of large data sets across clusters of computers. It uses a simple programming model and is designed to scale up from a single server to thousands of machines.
-

104. What is the difference between Spark and Hadoop?

- **Speed:** Spark processes data in memory, making it much faster than Hadoop MapReduce, which relies on disk storage for intermediate data.
 - **Ease of Use:** Spark has a more user-friendly API and supports multiple programming languages, while Hadoop primarily uses Java.
 - **Data Processing:** Hadoop is batch-oriented, whereas Spark supports batch, streaming, interactive, and machine learning workloads.
 - **Storage:** Hadoop uses HDFS for storage, while Spark can work with HDFS, S3, and other data sources.
-

105. Why use Spark?

- **Performance:** Spark is faster due to its in-memory data processing capabilities.
 - **Unified Engine:** It can handle diverse workloads such as batch processing, real-time stream processing, machine learning, and graph processing within a single framework.
 - **Ease of Development:** Its APIs are easy to use and understand, speeding up the development cycle.
 - **Rich Libraries:** Spark comes with built-in libraries for SQL, streaming, machine learning, and graph processing.
-

106. What is a Spark Job? A Spark job is the highest-level unit of work that Spark performs. It consists of a sequence of transformations and actions that Spark executes on a dataset. When an action (like `count()`, `collect()`, etc.) is called, Spark creates a job and breaks it down into stages based on the transformations applied.

107. What is the difference between `map` and `flatMap` in Spark? When to use what?

- **`map`:** The `map` transformation applies a function to each element in the RDD and returns a new RDD containing the results. Each input element maps to exactly one output element.
- **`flatMap`:** The `flatMap` transformation applies a function to each element, but it can return multiple elements for each input element. It flattens the output, resulting in an RDD with potentially fewer or more elements than the input.

When to use: Use `map` when you want a one-to-one transformation and `flatMap` when you need to return multiple values from a single input.

108. Does using `flatMap` affect partitioning? Yes, using `flatMap` can affect the number of partitions. The resulting RDD can have a different number of partitions than the original RDD, especially if the function used in `flatMap` results in varying numbers of output elements. This can lead to skewness if the distribution of the resulting data is uneven.

109. What is the difference between `repartition` and `coalesce`?

- **`repartition`:** This transformation reshuffles the data across partitions, which can increase or decrease the number of partitions. It can be used to improve parallelism and can lead to a full shuffle.
 - **`coalesce`:** This transformation reduces the number of partitions without a full shuffle, which is more efficient when reducing the number of partitions. It is often used after filtering operations to optimize performance.
-

110. If we are facing skewness in data, which should we use: `repartition` or `coalesce`?

When facing skewness in data, `repartition` is preferred as it performs a full shuffle, redistributing the data evenly across partitions. This can help mitigate issues caused by data skew. On the other hand, `coalesce` is better suited for reducing partitions without affecting the distribution significantly.

111. What happens internally after a Spark job is submitted? When a Spark job is submitted, the following occurs:

1. **DAG Creation:** Spark creates a Directed Acyclic Graph (DAG) representing the sequence of operations.
 2. **Stage Generation:** The DAG is divided into stages based on narrow and wide transformations.
 3. **Task Scheduling:** Each stage is further divided into tasks, which are distributed to different executors based on data locality.
 4. **Execution:** Executors run the tasks and store intermediate data in memory or disk.
 5. **Result Collection:** The final output is collected and returned to the driver.
-

112. How does Spark handle the job? Spark uses a master-slave architecture. The **driver** program runs the main function of the Spark application and is responsible for converting the user's code into a DAG of tasks. The **cluster manager** allocates resources and schedules tasks on worker nodes (executors), which perform the actual computations. Spark keeps track of the state of the job and monitors task execution, retrying failed tasks when necessary.

113. I have a CSV file in GCS, and in a column, I am getting alphabets instead of numeric. Write PySpark code to handle it.

```
python
Copy code
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when

# Create a Spark session
spark = SparkSession.builder.appName("Handle Non-Numeric").getOrCreate()

# Read the CSV file
df = spark.read.csv("gs://your-bucket/path/to/file.csv", header=True)

# Convert the column to numeric, handling non-numeric values
df = df.withColumn("numeric_column",
when(col("your_column").cast("float").isNotNull(),
col("your_column").cast("float")).otherwise(0))

# Write the cleaned DataFrame back to a new CSV file
df.write.csv("gs://your-bucket/path/to/cleaned_file.csv", header=True)
```

114. Joining three tables (emp_details, salary_details, dept_details) and finding the max salary using CTE and joins.

```
sql
Copy code
WITH EmployeeSalaries AS (
    SELECT e.EmployeeID, e.Name, s.Salary
    FROM emp_details e
    JOIN salary_details s ON e.EmployeeID = s.EmployeeID
```

```
)  
SELECT d.DepartmentName, MAX(es.Salary) AS MaxSalary  
FROM EmployeeSalaries es  
JOIN dept_details d ON es.EmployeeID = d.EmployeeID  
GROUP BY d.DepartmentName;
```

115. When do we get an executor out of memory error, and what can we do? An executor out of memory error occurs when the executor runs out of memory while processing a task, typically due to:

- Large datasets
- Inefficient code (e.g., excessive caching or large broadcast variables)

Solutions:

1. **Increase Executor Memory:** Configure more memory for the executor in the Spark settings.
 2. **Optimize Code:** Review and optimize your Spark transformations to reduce memory usage (e.g., using `mapPartitions`).
 3. **Repartition Data:** Use `repartition` to spread data evenly across partitions.
 4. **Tune Serialization:** Use Kryo serialization instead of Java serialization to reduce memory footprint.
 5. **Use Disk:** Configure Spark to spill data to disk when memory is insufficient.
-

116. groupByKey vs. reduceByKey.

- **groupByKey:** Groups all values for each key into a single iterable. It does not reduce the number of records before shuffling, which can lead to high memory consumption and performance issues.
- **reduceByKey:** Combines values for each key using a specified commutative and associative function before shuffling. This means fewer records are shuffled over the network, making it more efficient than `groupByKey`.

117. TRICKY question...if i have performed a filter operation and flatmap operation on a df or rdd.... what will

you choose..... repartition or coalesce... to handle the time lag or efficiency.....

(for this ANSWER IS.. as it is transformation only...so yet there is no action performed)

118. Unitemporal Data:

Unitemporal data captures changes based on a **single timeline**, typically either the valid time or transaction time.

- ***Valid Time:*** Refers to the time when the data is actually valid in the real world. For example, the period during which a price for a product was valid.
- ***Transaction Time:*** Refers to the time when the data was recorded in the system or database. This is the time when the record was created, updated, or deleted in the system.

In a unitemporal system, the data is tracked either by the ***valid time*** or the ***transaction time***, but not both. Unitemporal data is simpler but less flexible when it comes to reconstructing historical states or managing complex changes.

119 *Bitemporal Data:*

Bitemporal data captures changes based on ***two timelines***: both the valid time and transaction time.

- ***Valid Time:*** The real-world time period during which the data was correct or valid.
- ***Transaction Time:*** The time when the data was recorded in the system (i.e., when the record was created, updated, or deleted in the database).

In a bitemporal system, both of these time dimensions are tracked simultaneously. This allows the system to know not only what was true at a certain time in the real world but also when the system believed that information to be correct. It enables more accurate reconstruction of past events, handling retroactive corrections, and auditing changes more effectively.

Example:

Imagine an employee's salary data in a bitemporal system:

- ***Valid Time:*** The salary for the employee is \$50,000 from January 1st, 2020, to December 31st, 2020.
- ***Transaction Time:*** The record of that salary was entered into the system on January 10th, 2020. Later, on March 1st, 2020, a correction is made because there was an error, and the salary is updated to \$55,000 retroactively from January 1st.

A bitemporal system allows you to see both the corrected salary and the fact that it was initially entered incorrectly. A unitemporal system would not provide this level of detail.

119. Table t1 has 100 rows and t2 has 50 rows, what are maximum possible rows for left, right, and inner join?

- **Left Join:** Maximum possible rows = 100 (all rows from t1 and matched rows from t2).
- **Right Join:** Maximum possible rows = 50 (all rows from t2 and matched rows from t1).
- **Inner Join:** Maximum possible rows => $100 \times 50 = 5000$ (only the matching rows between t1 and t2).

120. In which join can we get more than 100 rows?

- In a **left join** or a **right join**, if there are multiple matching rows in the opposite table for a single row in the main table. For instance, if one row in t1 matches multiple rows in t2, the number of rows returned can exceed 100.

121. Lead and Lag in window functions and example cases we use them:

- **LEAD:** Accesses data from the following row in a result set without the need for a self-join. Example: Getting the next sale amount.
- **LAG:** Accesses data from the previous row. Example: Comparing this month's sales with last month's.
- **Example SQL:**

```
sql
Copy code
SELECT
    SaleDate,
    SaleAmount,
    LAG(SaleAmount) OVER (ORDER BY SaleDate) AS PreviousSale,
    LEAD(SaleAmount) OVER (ORDER BY SaleDate) AS NextSale
FROM SalesTable;
```

122. How to find duplicates in a table?

- To find duplicates based on a specific column (e.g., Name):

```
sql
Copy code
SELECT Name, COUNT(*) as Count
FROM YourTable
GROUP BY Name
HAVING COUNT(*) > 1;
```

123. While running queries in BigQuery, have you experienced any lag?

- It's common to experience lag due to several factors, such as complex queries, high data volume, or network latency. However, the performance can often be optimized by using partitioned tables, clustering, and optimizing SQL queries.

124. Have you used Git, did you face any difficulty while pushing code?

- Yes, using Git is standard practice. Common challenges can include merge conflicts, authentication issues, or ensuring the correct branch is checked out before pushing changes.

125. Have you experienced revert?

- Yes, reverting changes can be necessary in case of errors. It involves using Git commands like `git revert <commit>` to undo specific commits without losing the entire history.

126. In your project, are you following ETL or ELT?

- In my current project, we follow **ELT** (Extract, Load, Transform) as we load data into the data warehouse first and perform transformations afterward for efficiency and performance.

127. Do you have any idea about surrogate keys?

- Yes, surrogate keys are artificial keys used to uniquely identify a record in a table, typically a numeric value that has no business meaning, to maintain data integrity and enable better performance in joins.

128. Data modeling - star and snowflake schema:

- **Star Schema:** A simple design where a central fact table is connected to multiple dimension tables, optimizing query performance but leading to data redundancy.
- **Snowflake Schema:** A more complex design where dimension tables are normalized into multiple related tables, reducing redundancy but complicating queries.

129. What are unitemporal and bitemporal data?

- **Unitemporal Data:** Data where the state is only recorded with respect to a single time dimension (e.g., when the data was created or updated).
- **Bitemporal Data:** Data that includes two time dimensions: valid time (when the data is valid) and transaction time (when the data was recorded).

130. What are the optimization techniques you follow in BigQuery?

- Techniques include:
 - Partitioning tables by date.
 - Clustering tables based on frequently queried columns.
 - Using materialized views for common aggregations.
 - Optimizing SQL queries to reduce data scanned (e.g., avoiding `SELECT *`).
 - Implementing query caching where applicable.

131. **Why are you avoiding SELECT ?*

- Avoiding `SELECT *` improves performance and reduces costs by only fetching necessary columns. It minimizes the amount of data scanned and can help in maintaining better query performance and clarity.

132. I'm executing two queries in BigQuery - one has to fetch 2 records using LIMIT, and the other query has to fetch two columns without using LIMIT. Which query will cost less?

- The query that uses `LIMIT` will likely cost less since it limits the amount of data processed. The query without `LIMIT` will incur costs based on the total data scanned for all columns specified.

133. What is coalescing and repartitioning?

Coalescing reduces the number of partitions in a DataFrame or RDD without performing a full shuffle. It combines existing partitions and is more efficient because it minimizes data movement. It's typically used when you want to reduce the number of partitions to improve performance for certain operations.

Repartitioning, on the other hand, involves shuffling the data across the network to create a new partitioning scheme. This allows you to increase or decrease the number of partitions. Repartitioning is useful when you need a specific number of partitions or want to evenly distribute data across them.

134. What is a task in Spark?

In Spark, a **task** is the smallest unit of work that a Spark job can execute. Each task corresponds to a partition of the data being processed. Tasks are distributed across the executors in a Spark cluster, and they handle operations such as transformations and actions on the data. Each task processes a portion of the data and produces results that are later combined in the final output.

135. What is the process after the submission of a Spark job?

After submitting a Spark job:

1. **Driver Program:** The driver program initializes and requests resources from the cluster manager.
2. **Cluster Manager:** Allocates resources (CPU, memory) across various worker nodes.
3. **Task Scheduling:** The driver divides the job into tasks and schedules them to be run on executors.
4. **Execution:** Executors execute the tasks and perform operations on the data.
5. **Results Collection:** The results from tasks are sent back to the driver.
6. **Completion:** The driver collects all results, combines them if necessary, and returns the final output.

136. What are the file types you are using in your project?

In typical data engineering projects, common file types include:

- **CSV (Comma-Separated Values):** For structured data interchange.
- **Parquet:** A columnar storage file format optimized for big data processing.
- **Avro:** A row-based storage format designed for serialization of large amounts of data.
- **JSON (JavaScript Object Notation):** For semi-structured data.
- **ORC (Optimized Row Columnar):** A columnar storage format optimized for Hive.

137. What are the other file types you know generally?

Other file types used in data processing and storage include:

- **Text Files:** Plain text for unstructured data.
- **XML (Extensible Markup Language):** For structured data interchange, especially in web services.

- **Binary Files:** For efficient data storage and transfer (e.g., images, audio).
- **TFRecord:** A TensorFlow data format used for storing large datasets.
- **Image Formats:** Such as PNG, JPEG for image data storage.

138. What is the difference between task and executor?

- **Task:** A unit of work that Spark schedules on an executor. It operates on a partition of data and is the smallest unit of execution.
- **Executor:** A distributed agent responsible for executing tasks. Each executor runs on a worker node, and multiple tasks can run in parallel on an executor depending on its resources.

139. What are the different Google Cloud services you worked on?

Common Google Cloud services include:

- **Google Cloud Storage (GCS):** For object storage.
- **BigQuery:** For data warehousing and analytics.
- **Cloud Pub/Sub:** For messaging and event-driven architectures.
- **Cloud Functions:** For serverless computing.
- **Cloud Dataproc:** For managed Spark and Hadoop services.
- **Cloud Dataflow:** For stream and batch data processing.

140. Given the tables `emp_data` and `emp_latest`, write a SQL query to flag employee address changes.

```
sql
Copy code
WITH latest_addresses AS (
    SELECT emp_id, mob_no, Address,
        CASE
            WHEN ed.Address <> el.Address THEN 'N'
            ELSE 'Y'
        END AS Flag
    FROM emp_data ed
    LEFT JOIN emp_latest el ON ed.emp_id = el.emp_id
)
SELECT emp_id, mob_no, Address, Flag FROM latest_addresses
UNION ALL
SELECT emp_id, mob_no, Address, 'Y' AS Flag
FROM emp_latest
WHERE emp_id NOT IN (SELECT emp_id FROM emp_data);
```

In SQL, the `<>` operator means "not equal to." In the expression:

```
sql
Copy code
WHEN ed.Address <> el.Address THEN 'N'
```

this means that if the value of `ed.Address` (from the `emp_data` table) is not equal to the value of `el.Address` (from the `emp_latest` table), then the condition is true, and the query will assign the value 'N' to the `Flag` column.

In the context of the query, it is used to check if an employee's address has changed between the two tables. If the addresses are different, it flags the record as 'N', indicating that the address has changed. If they are the same, it would assign 'Y' to indicate that the address remains unchanged.

40 mini

141. What are the first two types in Slowly Changing Dimension (SCD) tables?

- **Type 1:** Overwrite the existing record with new data, losing historical information. Useful when you only care about the current value.
- **Type 2:** Add a new record with the new value and keep the historical record. This includes start and end dates or a version number to manage history.

142. Where do you store your targeted data?

Targeted data is typically stored in a data warehouse or database. Common storage solutions include:

- **BigQuery:** For analytical querying.
- **Cloud SQL:** For relational database storage.
- **Cloud Spanner:** For scalable, high-availability databases.

143. What is data skewness in Spark?

Data skewness occurs when the data is unevenly distributed across partitions, leading to some partitions having significantly more data than others. This imbalance can cause certain tasks to take much longer than others, resulting in inefficient resource utilization and increased job execution time.

144. Explain about HDFS.

HDFS (Hadoop Distributed File System) is a distributed file system designed to run on commodity hardware. It is highly fault-tolerant and is optimized for high-throughput access to large datasets. HDFS stores files across multiple machines, breaking them into blocks (default size 128 MB or 256 MB) and replicating them for reliability. It allows for the storage of large files and is used in big data applications.

145. Explain MapReduce.

MapReduce is a programming model used for processing large data sets with a distributed algorithm on a cluster. It consists of two main steps:

- **Map:** Processes input data (key-value pairs) and produces intermediate key-value pairs.
- **Reduce:** Takes intermediate key-value pairs and merges them to produce the final output. This model is fault-tolerant and scales horizontally across many machines.

146. Did you work on HDFS, Hive...?

Yes, I have worked with HDFS for storing large datasets and with Hive for querying those datasets using SQL-like syntax. Hive provides an abstraction layer over HDFS and allows for easier data analysis.

147. What is the difference between fact and dimension tables?

- **Fact Tables:** Store quantitative data for analysis and are typically denormalized. They contain measurable, numeric performance metrics (e.g., sales amount, quantity sold).
- **Dimension Tables:** Store descriptive attributes related to the facts. They provide context to the facts (e.g., product details, customer information). Dimension tables are often denormalized to optimize query performance.

148. What is a surrogate key?

A **surrogate key** is an artificial key (typically a numeric value) used to uniquely identify a record in a table, independent of the natural key. It is commonly used in data warehousing to simplify relationships and improve performance. Surrogate keys are especially useful in slowly changing dimensions to track historical changes.

149. What is the memory allocation for `show()` in Spark?

The `show()` method in Spark is used to display the contents of a `DataFrame` in the console. The memory allocation for `show()` is generally dependent on the following factors:

- **Number of Rows:** By default, `show()` displays the first 20 rows of a `DataFrame`. The memory used will depend on the size of the rows being displayed.
- **DataFrame Size:** The total size of the `DataFrame` in memory can impact how much data is retrieved when `show()` is called.
- **Cluster Configuration:** The amount of memory allocated to each executor in the Spark cluster configuration can also affect how much data can be processed and shown at once.

Example:

```
python
Copy code
# Sample DataFrame
df = spark.createDataFrame([(1, "Alice"), (2, "Bob"), (3, "Charlie")],
["id", "name"])

# Show the DataFrame
df.show()
```

150. What is Partition Pruning?

Partition pruning is an optimization technique used in data processing systems like Spark and BigQuery to improve query performance. It refers to the ability of the query engine to skip reading unnecessary partitions of data based on the filter criteria specified in the query.

How it Works:

- When a query includes a filter condition on a partitioned column, the engine can identify which partitions do not satisfy the condition and skip them, reducing the amount of data read and processed.
- This is particularly effective for large datasets that are partitioned by certain columns (like date or region).

Example: Suppose you have a DataFrame partitioned by date:

```
sql
Copy code
SELECT * FROM sales WHERE sale_date = '2024-10-28';
```

In this case, if the data is partitioned by `sale_date`, the engine will only read the partition for 2024-10-28 instead of scanning the entire dataset.

151. How will you schedule your jobs?

Scheduling jobs in a data pipeline can be done using various tools and methods depending on the complexity and requirements of your workflows. Here are common approaches:

- **Apache Airflow:** Use Airflow for orchestrating complex workflows. Define Directed Acyclic Graphs (DAGs) to manage dependencies and schedule tasks.
- **Cloud Scheduler:** If you're working in Google Cloud, you can use Cloud Scheduler to trigger jobs at specified intervals or based on events.
- **Databricks Jobs:** Schedule jobs directly in Databricks to run notebooks or scripts at defined times.
- **Cron Jobs:** For simpler use cases, traditional cron jobs can be utilized on Unix-like systems to execute scripts at specified times.

Example (Airflow DAG):

```
python
Copy code
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from datetime import datetime

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2024, 10, 28),
}

dag = DAG('example_dag', default_args=default_args,
          schedule_interval='@daily')

start = DummyOperator(task_id='start', dag=dag)
end = DummyOperator(task_id='end', dag=dag)

start >> end # Define task dependencies
```

152. What is the maximum cluster size in your company?

The maximum cluster size can vary widely depending on your organization's infrastructure and workload requirements. Here are a few factors to consider:

- **Cloud Provider Limits:** If you're using cloud services like Google Cloud or AWS, there are specific limits for the number of nodes and resources available per project.
- **Company Policy:** Internal policies regarding resource allocation and cost management can affect the maximum cluster size.
- **Job Requirements:** The nature of the workloads (batch processing, streaming, etc.) may dictate how large your clusters need to be.

Example: In Google Cloud Dataproc, you can define the maximum number of worker nodes when creating a cluster. For instance, you might have a maximum cluster size of 100 nodes:

```
bash
Copy code
gcloud dataproc clusters create my-cluster \
  --region=us-central1 \
  --num-workers=100 \
  --worker-machine-type=n1-standard-2
```

153 . form the logic on DF to find the manager name of each employee

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import col
```

```
# Initialize Spark session
```

```
spark = SparkSession.builder \
    .appName("Employee Manager Self Join") \
    .getOrCreate()
```

```
# Sample employee DataFrame
```

```
employees_data = [
    (1, "Alice", 2),
    (2, "Bob", 3),
    (3, "Charlie", None),
    (4, "David", 2)
```

```

]

employees_columns = ["emp_id", "emp_name", "manager_id"]

employees_df = spark.createDataFrame(employees_data, employees_columns)


# Perform self-join to find manager names

result_df = employees_df.alias("emp").join(

    employees_df.alias("mgr"),

    col("emp.manager_id") == col("mgr.emp_id"),

    "left"

).select(

    col("emp.emp_id"),

    col("emp.emp_name"),

    col("mgr.emp_name").alias("manager_name")

)


# Show result

result_df.show()

```

154. read a file..... and count the number of words..... in DF.. pyspark code

```

from pyspark.sql import SparkSession

from pyspark.sql.functions import explode, split, col

# Initialize Spark session

spark = SparkSession.builder \

    .appName("Word Count in Text File") \

    .getOrCreate()

# Read the text file into a DataFrame

```

```

# Replace 'path/to/your/file.txt' with your actual file path

file_path = 'path/to/your/file.txt'

text_df = spark.read.text(file_path)

# Split each line into words and explode into separate rows

words_df = text_df.select(explode(split(col("value"), " ")).alias("word"))

# Count the number of words

word_count = words_df.count()

# Display the result

print(f'Total number of words: {word_count}')

# Optionally, if you want to see the DataFrame of words:

# words_df.show()

```

155. what is merge in BQ... why we use merge...

To avoid data duplicate.... We can use merge

In BigQuery, the `MERGE` statement allows you to perform an **upsert** operation on a table, which means you can update existing rows and insert new rows into a table in a single statement. The `MERGE` statement combines the functionalities of `INSERT`, `UPDATE`, and `DELETE` based on a condition that determines how the target table should be modified.

Syntax of MERGE

The basic syntax of a `MERGE` statement in BigQuery is as follows:

```

sql
Copy code
MERGE target_table AS target
USING source_table AS source
ON target.key_column = source.key_column
WHEN MATCHED THEN
  UPDATE SET target.column1 = source.column1, target.column2 =
source.column2
WHEN NOT MATCHED THEN
  INSERT (key_column, column1, column2) VALUES (source.key_column,
source.column1, source.column2)

```

Components of the MERGE Statement

- **target_table:** The table where you want to perform the upsert operation.
- **source_table:** The table or subquery that contains the new data to be merged.

- **ON clause:** Specifies the condition to identify matching rows between the target and source tables.
- **WHEN MATCHED:** Defines the actions to take when a row in the target table matches a row in the source table (typically an `UPDATE`).
- **WHEN NOT MATCHED:** Defines the actions to take when a row in the source table does not match any row in the target table (typically an `INSERT`).

Why Use MERGE in BigQuery?

1. **Efficiency:** Instead of running separate `INSERT` and `UPDATE` statements, `MERGE` allows you to do both in a single operation, reducing the number of API calls and improving performance.
2. **Atomic Operations:** `MERGE` ensures that the entire operation is performed atomically. This means either all the updates and inserts are applied, or none are, preventing data inconsistencies.
3. **Simplified Logic:** It simplifies the SQL logic when handling scenarios where you need to conditionally update or insert data based on whether it already exists in the target table.
4. **Better Readability:** The `MERGE` statement can make your SQL queries easier to read and maintain compared to a combination of multiple `UPDATE` and `INSERT` statements.

Example of Using MERGE

Here's an example of using `MERGE` in BigQuery:

```
sql
Copy code
MERGE dataset.target_table AS target
USING dataset.source_table AS source
ON target.id = source.id
WHEN MATCHED THEN
  UPDATE SET target.value = source.value
WHEN NOT MATCHED THEN
  INSERT (id, value) VALUES (source.id, source.value)
```

In this example, rows in `source_table` are used to update or insert into `target_table`. If the `id` exists in both tables, the value is updated; if it doesn't, a new row is inserted.

Summary

The `MERGE` statement in BigQuery is a powerful tool for managing data efficiently by enabling conditional updates and inserts in a single statement, ensuring data integrity and improving performance.