**DATA ENGINNER INTERVIEW QUESTIONS**

**‘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

);

1. **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

1. **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;

1. **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

1. **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`;

1. **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.

**FIn 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. **De.alization**: 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!

1. **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.
2. **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.

1. **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.

1. **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.
2. **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.

1. **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';

1. **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.
2. **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.

1. **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;

1. **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;

1. **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;

1. **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. **Databricks Architecture** is built around a unified analytics platform that leverages Apache Spark, providing a collaborative environment for data engineers, data scientists, and business analysts. At its core, Databricks integrates the Apache Spark engine with a cloud-native architecture that allows for scalable, distributed data processing.

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.

**Apache Spark Architecture** is designed for fast, distributed data processing, providing a unified analytics engine for large-scale data processing and machine learning. At its core, Spark operates on a **master-slave architecture** consisting of several key components:

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.

**Apache Airflow Architecture** is a platform to programmatically author, schedule, and monitor workflows. It is designed to help users manage complex data pipelines, typically by orchestrating the execution of tasks in a reliable and scalable way. The architecture of Apache Airflow follows a distributed and modular approach, and it consists of several components that work together to execute 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

### 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.

1. **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 \* 50 = 5000 (only the matching rows between t1 and t2).

1. **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.

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

* **LEAD**: A **lead** is the value of a variable from a future time period (or observation). It refers to how much time or steps ahead in the sequence you are looking. Example: Getting the next sale amount.
* **LAG**:  A **lag** is the value of a variable from a previous time period (or observation). It refers to how much time or steps back in the sequence you are looking.
*  : 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;

1. **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;

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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).

1. **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.

1. \**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.

1. **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.

4o 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.

**154. WHAT IS BIG QUERRY ?**

Google BigQuery is a powerful, serverless, and highly scalable data warehouse designed for enterprise-level analytics. By eliminating the need for infrastructure management, BigQuery allows users to focus solely on extracting insights from massive datasets. Its architecture supports standard SQL, machine learning capabilities through BigQuery ML, and seamless integration with business intelligence tools, enabling comprehensive data analysis. BigQuery's ability to handle structured, semi-structured, and even streaming data makes it a versatile solution for organizations seeking to derive meaningful insights and drive data-driven decisions from their information assets."

**BigQuery:**

"BigQuery is Google Cloud's serverless, highly scalable data warehouse. It allows you to analyze massive datasets quickly using SQL, without managing any infrastructure."

**Cloud Storage:**

"Cloud Storage is Google Cloud's object storage service. It provides durable and scalable storage for unstructured data, like images, videos, and large files."

**Dataflow:**

"Dataflow is Google Cloud's serverless service for stream and batch data processing. It uses Apache Beam to execute data pipelines, enabling transformations at scale."

**Dataproc:**

"Dataproc is Google Cloud's managed Apache Hadoop and Spark service. It simplifies running big data processing jobs on a scalable cluster."

**Databricks:**

"Databricks is a unified analytics platform, built on Apache Spark, that simplifies big data processing and machine learning. It provides a collaborative environment for data science and engineering."

**Airflow:**

"Airflow is an open-source workflow orchestration tool. It allows you to programmatically author, schedule, and monitor data pipelines."

**Pub/Sub:**

"Pub/Sub is Google Cloud's real-time messaging service. It enables asynchronous communication between applications by publishing and subscribing to messages."

**Kafka:**

"Kafka is an open-source distributed streaming platform. It's used for building real-time data pipelines and streaming applications."

**Cloud Composer:**

"Cloud Composer is Google Cloud's managed workflow orchestration service based on Apache Airflow. It simplifies the creation and management of data pipelines."

**Apache Spark:**

"Apache Spark is an open-source distributed computing system. It's designed for fast and general-purpose big data processing."

**Apache Beam:**

"Apache Beam is an open-source unified programming model for defining and executing data processing pipelines. It can run on various execution engines, like Dataflow and Spark."

**1. Cloud Function**  
Cloud Functions is a serverless compute service that allows you to run small, single-purpose functions in the cloud without managing servers. It is event-driven, meaning it executes in response to events (such as HTTP requests, file uploads, or messages in a queue).

**Key Points**:

* **Serverless**: No infrastructure management.
* **Event-driven**: Triggers based on events.
* **Scalable**: Automatically scales based on demand.
* **Use Cases**: Real-time file processing, webhooks, microservices.

**2. Cloud Run**  
Cloud Run is a fully managed platform that enables you to run containerized applications in a serverless environment. It can scale automatically based on incoming traffic, and it supports any language or framework that can run in a container.

**Key Points**:

* **Serverless**: No need to manage servers or clusters.
* **Containerized**: Supports running Docker containers.
* **Scalable**: Automatically scales up or down based on traffic.
* **Use Cases**: Web applications, APIs, background procefssing.

**3. Cloud Fusion**

Google Cloud Fusion is not an official Google Cloud product. However, you might be referring to **Oracle Cloud Fusion** or a similar concept used in integration, data processing, or hybrid cloud solutions. If you are asking about this for an interview, it's essential to clarify the context in which "Cloud Fusion" is being referenced.

**Key Points**:

* **Integration**: It is often used for integrating cloud applications with on-premise systems or other cloud services.
* **Hybrid**: Combines data from multiple sources.
* **Customizable**: Users can build integration workflows.

**4. Cloud SQL**

Cloud SQL is a fully managed relational database service provided by Google Cloud. It supports popular database engines like MySQL, PostgreSQL, and SQL Server. Cloud SQL handles routine tasks like patch management, backups, and scalability.

**Key Points**:

* **Managed Service**: Google manages infrastructure and maintenance tasks.
* **Relational**: Supports SQL-based databases.
* **High Availability**: Provides automated backups and replication.
* **Use Cases**: Web applications, analytics, enterprise databases.

**5. Cloud Spanner**

Cloud Spanner is a fully managed, scalable, globally distributed, and strongly consistent relational database service offered by Google Cloud. It combines the benefits of relational databases with the scalability of NoSQL databases, designed to handle large-scale applications.

**Key Points**:

* **Globally Distributed**: Can scale horizontally across regions.
* **ACID Transactions**: Supports strong consistency and ACID properties.
* **Scalable**: Handles large amounts of data with automatic scaling.
* **Use Cases**: Global applications, financial systems, mission-critical applications.

**Summary of Key Concepts:**

* **Cloud Function**: Serverless, event-driven, small, single-purpose tasks.
* **Cloud Run**: Serverless, containerized application execution.
* **Cloud Fusion**: (May refer to a specific integration service or hybrid cloud platform, clarify for the interview).
* **Cloud SQL**: Fully managed relational databases (MySQL, PostgreSQL, SQL Server).
* **Cloud Spanner**: Fully managed, scalable, globally distributed relational database with strong consistency.

**154. Explain Big Query Routines** **?**

BigQuery routines (UDFs and stored procedures) help you manage, automate, and reuse logic within queries, making complex data tasks easier to handle. You can use them for transforming data, performing calculations, and orchestrating multi-step workflows within BigQuery.

**155. Key Differences Between UDFs and Stored Procedures:**

* **UDFs** are functions used in SQL queries to return specific values or tables. They focus on transforming or processing data on a per-row basis.
* **Stored Procedures** are more comprehensive and can handle multiple SQL operations, including loops and conditional logic. They perform actions rather than returning a result directly in a query.

**1. User-Defined Functions (UDFs)**

A **UDF** in BigQuery is a custom function written by the user that can be called within SQL queries. UDFs allow users to write reusable logic for operations that are not directly available via standard SQL functions. There are two types of UDFs in BigQuery:

* **Scalar UDFs**: These return a single value for each row of input data.
* **Table-Valued UDFs**: These return a table of results instead of a single value and can be used in a SELECT statement.

**2. Stored Procedures**

A **stored procedure** is a set of SQL statements that you can save and reuse, which can include logic, loops, conditionals, and calls to other queries. Stored procedures are useful when you need to perform more complex, multi-step operations within BigQuery.

* Unlike UDFs, which return values directly, stored procedures typically perform an action, such as modifying tables, performing complex calculations, or handling multiple queries in sequence.
* You can pass **parameters** to stored procedures to customize their behavior.

**156. Benefits of BigQuery Routines:**

* **Reusability**: Both UDFs and stored procedures allow you to encapsulate logic that can be reused in multiple places within your BigQuery environment.
* **Modularity**: By breaking down complex queries into routines, you can improve code readability and maintainability.
* **Efficiency**: You can reduce the amount of repeated code and logic in your queries by defining common transformations as routines.

**157. BigQuery's Key Architecture Concepts:**

BigQuery’s architecture is designed to scale automatically, handle large datasets efficiently, and allow users to focus on data analysis without worrying about infrastructure management. It’s optimized for running complex queries and analytics workloads on large volumes of structured data.

* **Colossus** (Storage): Massive, distributed storage layer for structured data in columnar format.
* **Dremel** (Compute): Distributed query execution engine that processes SQL queries in parallel.
* **Serverless**: No server management; Google Cloud automatically handles scaling and infrastructure.
* **BigQuery Slots**: Virtual CPUs that execute queries in parallel, with flexibility in on-demand or reserved pricing models.
* **Separation of Storage and Compute**: Storage (Google Cloud Storage) and compute (BigQuery engine) scale independently.
* **Security and IAM**: Granular access control and encrypted data management for secure operations.

**158. GCS Optimization:**

Google Cloud Storage (GCS) optimization typically refers to practices that improve the performance and cost-efficiency of storing and accessing data. Some ways to optimize GCS include:

* **Choosing the Right Storage Class**: Select the appropriate storage class (e.g., Standard, Nearline, Coldline, Archive) based on how often you access the data.
* **Use Object Lifecycle Management**: Automatically move objects between storage classes or delete them after a certain period to optimize costs.
* **Data Compression**: Store compressed data to reduce storage costs and improve read/write performance.
* **Parallelization**: For large datasets, use parallel uploads/downloads to speed up the process.
* **Object Versioning**: Enable object versioning to keep track of changes and backups efficiently.
* **Use gsutil**: The gsutil tool is designed for high-performance data transfer in GCS and supports parallel uploads, resumable transfers, etc.

**159. GCS Storage and Classes:**

**Google Cloud Storage (GCS)** is an object storage service that provides scalable and secure storage for unstructured data. It supports multiple types of storage classes that are optimized for different use cases.

**Storage Classes**:

* **Standard**: Designed for frequently accessed data. It’s the most performant and has the highest availability but can be more expensive.
* **Nearline**: Best suited for data that is accessed less frequently, typically once a month or less. It’s cost-effective for backup and archiving.
* **Coldline**: Intended for long-term archival storage with infrequent access. Ideal for storing data you might only need to access once a year.
* **Archive**: Lowest-cost storage class for data that is rarely accessed and intended for long-term retention.

**160. What is Nearline:**

**Nearline** is one of the storage classes in Google Cloud Storage (GCS) for data that is accessed less frequently but still needs to be available for retrieval within seconds. It’s designed for backup and data that may be accessed less than once a month.

* **Use Cases**: Backup data, disaster recovery, data that might not be accessed regularly but needs to be available when needed.
* **Cost**: Nearline has lower storage costs compared to Standard storage but higher access costs.
* **Access Time**: Nearline is optimized for fast access, but the cost of retrieval is higher than for the Standard class.

**161. Optimizations in BigQuery:**

Optimizing BigQuery can lead to faster query performance and lower costs. Here are some common optimization techniques:

* **Partitioning and Clustering**: Partition tables by date or other relevant fields to reduce the amount of data scanned during queries. Cluster data by specific columns to speed up query execution.
* **Query Optimization**: Use filters early in your query to minimize the amount of data being processed. Avoid SELECT \* and instead query only the necessary columns.
* **Materialized Views**: Use materialized views to precompute and store results, which speeds up query performance for frequently executed queries.
* **Data Compression**: Store data in compressed formats like Avro or Parquet, which reduce the storage size and speed up query performance.
* **Denormalization**: Flatten nested data structures when it’s appropriate to reduce JOIN complexity and improve performance.
* **Use Partitioned Tables**: When your dataset is time-based (e.g., daily logs), partition your tables to reduce the amount of data queried.

**162. Partition By and Cluster By in BigQuery:**

**Partitioning** and **Clustering** are techniques in BigQuery used to optimize performance and reduce costs by organizing data.

* **Partitioning**:
  + BigQuery allows partitioning tables based on a column (typically a date or timestamp). Each partition stores a subset of data, and queries can be restricted to specific partitions, reducing the amount of data scanned.
  + Example: If you partition by date, queries that filter by date will only scan the relevant partitions rather than the entire table, improving performance.
  + **Partitioned tables** can be particularly helpful for time-series data, such as logs or transactional data.

Example of a partitioned table:

**CREATE OR REPLACE TABLE project\_id.dataset.table\_name**

**PARTITION BY DATE(timestamp\_column)**

**AS**

**SELECT \* FROM `project\_id.dataset.raw\_data`;**

* **Clustering**:
  + Clustering organizes data in the table by one or more columns. This optimizes queries that filter on those columns, as BigQuery can organize data on disk based on these fields.
  + Unlike partitioning, clustering doesn’t create physical partitions, but it optimizes data storage for faster access.

Example of a clustered table:

CREATE OR REPLACE TABLE project\_id.dataset.table\_name

CLUSTER BY column1, column2

AS

SELECT \* FROM `project\_id.dataset.raw\_data`;

* **Difference**:
  + **Partitioning** reduces the amount of data scanned by dividing the data into smaller chunks.
  + **Clustering** optimizes data storage to speed up access when filtering or aggregating on clustered columns.

**163. What is Dataproc and Dataflow?**

**Dataproc** and **Dataflow** are both managed data processing services from Google Cloud, but they serve different use cases:

* **Dataproc**:
  + A fully managed Apache Spark and Hadoop service. It allows you to run big data processing tasks using Spark, Hadoop, and other tools in the Hadoop ecosystem.
  + Best for batch processing, machine learning, and custom ETL workflows.
  + **Use Cases**: Big data processing, batch jobs, data transformation, custom analytics.
* **Dataflow**:
  + A fully managed service for stream and batch data processing built on Apache Beam. It simplifies the development of data pipelines that process real-time data.
  + **Use Cases**: Real-time streaming, ETL, data pipelines for continuous ingestion, and processing.

**Differences**:

* **Dataproc** is better for batch processing and running existing big data tools like Spark/Hadoop.
* **Dataflow** is better for stream processing and real-time analytics with support for both stream and batch data.

**164. SQL Query: Fetch the Top Three Highest Salary Employees in Each Department:**

Here’s the SQL query for fetching the top 3 highest-paid employees from each department:

WITH RankedEmployees AS (

SELECT

department,

employee\_id,

salary,

RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS salary\_rank

FROM `your\_project.your\_dataset.employees`

)

SELECT

department,

employee\_id,

salary

FROM RankedEmployees

WHERE salary\_rank <= 3;

* **RANK()** is used to assign a rank to each employee within their department based on salary.
* We use **PARTITION BY department** to restart the rank for each department.
* The query filters for the top 3 employees in each department.

**165. Reversing a String in Three Ways:**

Here are three different ways to reverse a string in Python:

1. **Using Slicing**:
2. s = "Hello"
3. reversed\_s = s[::-1]
4. print(reversed\_s)
5. **Using a Loop**:
6. s = "Hello"
7. reversed\_s = ""
8. for char in s:
9. reversed\_s = char + reversed\_s
10. print(reversed\_s)
11. **Using reversed() Function**:
12. s = "Hello"
13. reversed\_s = ''.join(reversed(s))
14. print(reversed\_s)

**166. Sum of Even and Odd Indexed Elements:**

Tosegregate and sum values at even and odd indices:

values = [1, 2, 3, 4, 5, 6, 7, 8]

even\_sum = sum(values[i] for i in range(0, len(values), 2)) # Even-indexed sum

odd\_sum = sum(values[i] for i in range(1, len(values), 2)) # Odd-indexed sum

print("Sum of even indexed elements:", even\_sum)

print("Sum of odd indexed elements:", odd\_sum)

**Example Output**:

Sum of even indexed elements: 16

Sum of odd indexed elements: 14

**INFOSIS :**

**167. What is BigQuery and Architecture Explanation?**

**BigQuery** is a fully managed, serverless data warehouse that allows you to run SQL queries on large datasets in real-time. BigQuery abstracts infrastructure management and scales automatically, making it easy for users to focus on data analysis without worrying about resource provisioning.

**Architecture of BigQuery**:

* **Storage Layer**: Data in BigQuery is stored in tables within datasets, using columnar storage. This allows efficient querying of large datasets.
* **Query Engine**: BigQuery uses a distributed execution engine built on **Dremel**, which efficiently processes complex queries by parallelizing tasks across a vast infrastructure.
* **Compute Layer**: BigQuery automatically handles compute resource allocation based on demand, scaling resources up or down as needed.

**168. Explain Scalability**

Scalability refers to the ability of a system to handle increasing amounts of work or its potential to accommodate growth. In **BigQuery**, scalability is **automatically handled**. It can scale up to process large amounts of data in petabytes and scale down when handling smaller datasets. This serverless architecture eliminates the need for manual resource management.

**169. Query Cost in BigQuery**

In **BigQuery**, query cost is based on the **amount of data scanned** by your queries. The more data your query processes, the higher the cost. BigQuery charges for the following:

* **Data Processed**: Charges are based on the bytes processed by a query.
* **Storage**: Charges for the data stored in tables, based on the size of the data.
* **Data Transfer**: Costs for transferring data between regions or between different Google Cloud services.

**170. BigQuery OLAP or OLTP?**

BigQuery is primarily designed for **OLAP (Online Analytical Processing)**, which is used for large-scale data analysis and reporting. OLAP is more suitable for read-heavy operations such as aggregations and complex queries, while **OLTP (Online Transactional Processing)** focuses on handling high-volume transaction processing, which is not BigQuery's primary use case.

**171. BigQuery ETL or ELT?**

BigQuery is best suited for **ELT (Extract, Load, Transform)**. In ELT, you extract data, load it into BigQuery, and then perform transformations using SQL queries directly within BigQuery. This contrasts with traditional ETL, where transformation happens before loading data into a data warehouse.

**172. BigQuery and PostgreSQL**

* **BigQuery** is a fully managed, scalable data warehouse primarily used for analytics (OLAP).
* **PostgreSQL** is an open-source relational database management system (RDBMS) used for transactional workloads (OLTP).

BigQuery is designed for analytics and large-scale data warehousing, while PostgreSQL is used for smaller-scale relational databases and transactional applications.

**173. Dataset and Table Difference Between?**

* **Dataset**: A dataset is a container for your tables, views, and other resources in BigQuery. It organizes and groups related data together.
* **Table**: A table stores data in rows and columns, similar to a table in a relational database. It contains the actual data in BigQuery.

**174. Explain Information Schema in BigQuery?**

The **INFORMATION\_SCHEMA** in BigQuery is a system database that allows you to query metadata about your datasets, tables, and other resources. For example, you can query it to get details about columns in a table or to list all tables within a dataset.

**175. Optimize Techniques in BigQuery?**

* **Partitioning**: Use partitioned tables to reduce the amount of data scanned by queries.
* **Clustering**: Cluster tables based on columns you filter or aggregate by to speed up query performance.
* **Column Pruning**: Only select the columns you need to minimize data scanned.
* **Query Caching**: Reuse cached query results for faster performance.
* **Limit Data Scanning**: Apply filters early to limit the data processed.

**176. Materialized Views and Authorized Views Difference Between?**

* **Materialized Views**: These are precomputed results stored in BigQuery. They can improve performance by avoiding recalculating the same results multiple times.
* **Authorized Views**: These are views that provide controlled access to data. Users with access to an authorized view can see data in a way that does not expose the underlying data directly.

**177. Materialized View and View Difference Between?**

* **Materialized View**: A materialized view stores the results of a query physically. It can be refreshed periodically and provides faster access to the data.
* **View**: A view is a virtual table based on the results of a query. It does not store data but executes the query each time it is queried.

**178. COUNT(\*) and COUNT(col) Difference Between?**

* **COUNT(\*)**: Counts all rows in a table or result set, including rows with NULL values.
* **COUNT(col)**: Counts only non-NULL values in the specified column. Rows with NULL in the column are excluded.

**179. CASE and WHEN Statements Difference Between?**

* **CASE**: The CASE expression allows you to perform conditional logic. It can be used for IF-ELSE type branching in SQL.
* **WHEN**: The WHEN clause is part of the CASE expression, specifying the conditions to evaluate.

**180. Explain Partition and Clustering Difference Between?**

Repeated question**. Q 162**

**181. Explain Time Travel Concept in BigQuery?**

Repeated question**. Q 07**

**182. Difference Between External Table and Internal Table?**

* **External Table**: An external table in BigQuery references data stored outside of BigQuery, like in Google Cloud Storage. Data is not stored inside BigQuery but is queried from its external location.
* **Internal Table**: An internal table is fully managed by BigQuery and stores data directly in BigQuery's storage infrastructure.

**183. SELECT, WHERE, GROUP BY, HAVING Difference Between?**

* **SELECT**: Specifies which columns to retrieve.
* **WHERE**: Filters rows before any grouping or aggregation.
* **GROUP BY**: Groups rows that share the same values into summary rows, often used with aggregate functions.
* **HAVING**: Filters groups created by GROUP BY after the aggregation.

**184. Table Name, Employee, User ID, Salary Desc Order Salary Partition by Dept A, B, C**

To retrieve a table with employee information, ordered by salary in descending order, and partitioned by department:

SELECT employee\_id, user\_id, salary, dept

FROM `project-id.dataset-id.employee\_table`

PARTITION BY dept

ORDER BY salary DESC;

This SQL will order employees by salary in descending order within each department partition (A, B, C).

**Company: Renexter**

**185. Your experience current project**

"In my current project, I'm focused on migrating Oracle databases to the Google Cloud Platform. This involves a comprehensive approach, including assessment of existing Oracle environments, designing target GCP architectures, implementing data migration strategies, and ensuring data integrity and performance post-migration. I'm actively working with GCP services like Cloud SQL for PostgreSQL/MySQL, Cloud Spanner, BigQuery, and Data Migration Service, as well as tools for data validation and performance optimization. This project is giving me hands-on experience in the complexities of large-scale database migrations."

**186. Are you worked any on-prem applications**

"Yes, the source Oracle databases are typically on-premises. My work involves understanding the on-premise Oracle configurations, dependencies, and performance characteristics. I'm also familiar with the challenges of connecting on-premise systems to GCP through VPNs or interconnects, and I understand the importance of secure data transfer during the migration process."

**187. What is the framework used in current project**

"We utilize a combination of GCP services and best practices as our 'framework'. This includes:

GCP's Data Migration Service (DMS): For streamlined and automated migration of Oracle databases.

Infrastructure as Code (IaC) tools like Terraform: For consistent and repeatable infrastructure deployment.

CI/CD pipelines: For automated testing and deployment of migration scripts and configurations.

Agile methodologies: For iterative development and collaboration.

Database assessment tools: For analysis of oracle databases, and providing the correct target GCP database.

We follow Google's Cloud Architecture Framework for designing and implementing secure, scalable, and reliable GCP solutions."

**188. At what time the batch and stream data consumed in a day**

"In this migration project, batch data migration is typically scheduled during off-peak hours to minimize disruption to production systems. Stream data migration, if applicable, might run continuously or during specific windows, depending on the application's requirements. We work closely with the client to determine the optimal timing for data transfers."

**189. How frequently it will done for a day**

"The frequency of data migration depends on the volume of data and the chosen migration strategy. For initial bulk migrations, we might perform daily or weekly transfers. Incremental data synchronization can occur more frequently, potentially hourly or even continuously for near real-time replication. The frequency is determined by the client's recovery point objective (RPO) and recovery time objective (RTO)."

**190. Who is your downstream what data they used**

"The 'downstream' users are the applications and business users that rely on the migrated data in GCP. This includes:

Business intelligence and analytics teams using BigQuery for reporting.

Application developers using Cloud SQL or Cloud Spanner for their applications.

Data scientists using data in Cloud Storage for machine learning.

The data used is the migrated Oracle database, transformed and optimized for GCP services."

**191. What are the requirement of project**

"The key requirements of this Oracle to GCP migration project are:

Minimize downtime and disruption to business operations.

Ensure data integrity and consistency throughout the migration.

Optimize performance and scalability of the migrated databases in GCP.

Reduce overall infrastructure costs.

Improve security and compliance.

Modernize the data platform."

**192. What are the analysis is doing in project**

**"**The analysis performed in this project includes:

Database schema and data analysis to understand the structure and dependencies of the Oracle databases.

Performance analysis to identify bottlenecks and optimize query performance.

Cost analysis to estimate the cost of running the migrated databases in GCP.

Risk assessment to identify and mitigate potential migration risks.

Application dependency mapping."

**193. How to fetch top 5 customers from your batch data**

"After the migration to GCP, if the data is in BigQuery, you can use a SQL query similar to this:

SQL

SELECT customer\_id, SUM(purchase\_amount) AS total\_purchase

FROM `your\_project.your\_dataset.customer\_purchases`

GROUP BY customer\_id

ORDER BY total\_purchase DESC

LIMIT 5;

If the data is in Cloud SQL (PostgreSQL or MySQL), the query would be very similar, adjusting the table and column names as needed. If the data is in Cloud Spanner, the query would also be similiar, but would follow Spanner SQL syntax.

Python

import pandas as pd

# Assuming 'purchases' is a pandas DataFrame

top\_customers = purchases.groupby('customer\_id')['purchase\_amount'].sum().nlargest(5)

print(top\_customers)

Okay, focusing on a GCP migration project will refine the answers and make them more impactful. Here's a breakdown tailored to that scenario:

**COMPANY : INCEDO**

**194.Have you worked on GCP migration projects?\*\***

\* "Yes, I've been involved in projects focused on migrating data and workloads to Google Cloud Platform, leveraging services like Cloud Storage, BigQuery, and Dataflow."

**195.How do you extract data from a source in a GCP migration?\*\***

\* "In GCP migrations, extraction often involves:

**\* \*\*Database Exports:\*\***

Using tools like `mysqldump` (for MySQL) or `pg\_dump` (for PostgreSQL) to export data into files.

**196.Cloud Storage Transfers:\*\***

Utilizing `gsutil` or Storage Transfer Service to move data from on-premises or other cloud storage to GCS.

\* **\*\*API Integrations:\*\***

Extracting data from third-party APIs using Cloud Functions or Dataflow.

**\* \*\*Dataflow:\*\***

For complex transformations and streaming data, Dataflow jobs are used to extract, transform, and load data."

**\* \*\*Which types of sources have you used in your GCP migration projects?\*\***

\* "Typical sources in GCP migrations include:

\* On-premises relational databases (MySQL, PostgreSQL, SQL Server).

\* On-premises file systems.

\* Other cloud providers (AWS, Azure).

\* APIs."

**\* \*\*You have 10 tables in MySQL, how do you extract 10 tables of data in a CSV file and how do you load them into a GCS bucket?\*\***

\* "For a GCP migration, I'd emphasize GCP tools:

1. \*\*Extraction:\*\*

\* Use `mysqldump` to create SQL dump files.

\* Use a script (Python with `mysql.connector` and `pandas`) to query each table, convert to CSV, and save locally.

**2. \*\*GCS Upload:\*\***

\* Utilize `gsutil cp` to upload the CSV files to a designated GCS bucket. Or use the google cloud storage python library.

**\* \*\*Once you have a file in your bucket, how do you load it into BigQuery?\*\***

\* "I'd use:

**\* \*\*BigQuery Load Jobs:\*\***

Through the BigQuery UI or the `bq load` command, specifying the GCS URI, file format (CSV), and schema.

**\* \*\*BigQuery Client Libraries:\*\***

Using Python's `google-cloud-bigquery` library for programmatic loading, especially within Airflow DAGs."

**\* \*\*Since you are getting every day files, would manually go and run bq command in console?\*\***

\* "No, automation is essential. I'd use Apache Airflow on Cloud Composer (GCP's managed Airflow service) to orchestrate the daily loading process."

**\* \*\*How to load data from a bucket to BigQuery in a DAG?\*\***

\* "Use the `BigQueryLoadOperator` within the Airflow DAG. Configure it with:

\* `source\_uris`: The GCS file URI(s).

\* `destination\_project\_dataset\_table`: The BigQuery table identifier.

\* `source\_format`: 'CSV', 'JSON', etc.

\* `schema\_fields`: The BigQuery table schema (if needed)."

**\* \*\*What kind of DAG have you developed, can you explain?\*\***

\* "For a GCP migration, I've developed DAGs for:

**\* \*\*Data Ingestion:\*\***

Extracting data from on-premises databases, uploading to GCS, and loading into BigQuery.

**\* \*\*ETL Pipelines:\*\***

Using Dataflow within a DAG to transform data in GCS and load it into BigQuery.

**\* \*\*Data Validation:\*\***

Running BigQuery queries to validate data integrity after migration.

**\* \*\*Cloud SQL backups:\*\***

creating cloud sql backups and storing them in GCS."

**\* \*\*How do you run any SQL code in Airflow?\*\***

\* "For BigQuery, I'd use the `BigQueryExecuteQueryOperator`. For Cloud SQL (MySQL or PostgreSQL), I'd use the appropriate database operator (e.g., `MySqlOperator`, `PostgresOperator`)."

**\* \*\*What are the operators that you are aware of?\*\***

\* "In a GCP context, I'd emphasize:

\* `BigQueryLoadOperator`

\* `BigQueryExecuteQueryOperator`

\* `DataflowStartFlexTemplateOperator`

\* `CloudSqlBackupOperator`

\* `GCSBucketCreateAclEntryOperator`

\* `GCSToBigQueryOperator`

\* `PythonOperator`

\* `BashOperator`"

**\* \*\*Environment variables in Airflow\*\***

\* "When using cloud composer, it is best practice to store sensitive data in google secrets manager, and then retreive those secrets using environment variables. This is the best practice for GCP."

**\*\*Key GCP Considerations:\*\***

\* Emphasize your familiarity with GCP services and best practices.

\* Highlight your ability to use `gsutil`, BigQuery, Dataflow, and Cloud Composer.

\* Showcase your understanding of security and access control in GCP.

\* Always mention best practices for security, like storing secrets in Google secrets manager.

**\*\*Incedo FF & GCP Migration Project\*\***

\* "In my GCP migration project, I worked within the Incedo FF framework, which facilitated the development and deployment of robust data pipelines on Google Cloud Platform. This involved leveraging standardized components and best practices to ensure efficient and scalable data processing."

**1. Data Pipelines**

* **Definition: Data pipelines are a series of processes that collect, transform, and load data (ETL). The goal is to automate the flow of data from source to destination (e.g., databases, data lakes).**
* **Components:**
  + Source Systems: Databases, APIs, file storage, etc.
  + ETL/ELT Process: Extraction, Transformation, and Loading of data.
  + Data Warehouse/Data Lake: Storage of transformed data (on cloud or on-prem).
  + Orchestration: Tools like Airflow, Cloud Composer, or Dataflow manage and schedule the pipeline.
* **Best Practices:**
  + Ensure data consistency.
  + Include monitoring, logging, and alerting mechanisms.
  + Optimize for performance to avoid bottlenecks.

**2. Views (in SQL)**

* **Definition: A view is a virtual table in SQL, created by querying data from one or more tables.**
* **Types:**
  + Simple View: Queries data from a single table.
  + Complex View: Queries data from multiple tables, can include joins, group by, or aggregates.
* **Use Cases:**
  + Abstracting complex queries for easy access.
  + Providing a security layer (restrict access to specific data).
* Important Consideration: Views don’t store data but run queries every time they are called, which can impact performance in large datasets.

**3. DAG (Directed Acyclic Graph) in Airflow**

* Definition: A DAG is a collection of tasks with dependencies defined in a sequence. Airflow uses DAGs to schedule and monitor workflows.
* Complete DAG Example:

**python**

**Copy**

**from airflow import DAG**

**from airflow.operators.python\_operator import PythonOperator**

**from datetime import datetime**

def task\_function():

print("Executing task")

with DAG('example\_dag', start\_date=datetime(2023, 1, 1), schedule\_interval='@daily') as dag:

task1 = PythonOperator(

task\_id='task1',

python\_callable=task\_function

)

**task1**

* **Important Points:**
  + Tasks in DAGs represent the unit of work.
  + Use operators for different tasks (e.g., PythonOperator, BashOperator).
  + DAGs are defined once and then executed on a schedule.
  + DAGs ensure tasks are executed in the right order, respecting dependencies.

**4. Python Basics for Data Engineering**

* **Key Concepts:**
  + Data Structures: Lists, dictionaries, sets, and tuples.
  + Functions: Functions to modularize code.
  + File Handling: Reading/writing CSV, JSON, etc.
  + Error Handling: Using try, except, and finally.
* **Libraries:**
  + pandas for data manipulation.
  + numpy for numerical operations.
  + requests for API calls.
  + json for working with JSON data.

**5. Dataflow Basics (Apache Beam on GCP)**

* **Definition: Dataflow is a fully-managed service on GCP for stream and batch data processing. It leverages Apache Beam for defining complex data processing pipelines.**
* **Key Concepts:**
  + Pipeline: The core of Dataflow. Defines the sequence of data processing steps.
  + PCollections: Unordered sets of data that are passed between processing steps.
  + Transformations: Operations applied to PCollections (e.g., ParDo, GroupByKey).
  + Windowing: Organizing data based on time.
* When to Use: Ideal for large-scale, real-time data processing needs.

**6. IAM (Identity and Access Management) in GCP**

* **Definition: IAM in GCP defines who (identity) has access to what resources and actions (permissions).**
* **Components:**
  + Roles: Predefined roles (e.g., Viewer, Editor, Owner) or custom roles with a set of permissions.
  + Policies: Permissions attached to resources, defining who can access or modify them.
  + Service Accounts: Used by applications to interact with GCP services.
* **Best Practices:**
  + Follow the principle of least privilege (give only the permissions necessary for the task).
  + Regularly audit permissions and service account keys.
  + Use IAM roles to manage permissions effectively.

**7. Window Functions in SQL**

* Definition: Window functions perform calculations across a set of table rows related to the current row.
* Common Functions:
  + ROW\_NUMBER(): Assigns a unique row number to each row.
  + RANK(): Ranks rows, assigning the same rank to rows with equal values.
  + SUM() OVER(): Calculates a cumulative sum over a specific window of rows.

**Example:**

sql

Copy

SELECT

employee\_id,

salary,

SUM(salary) OVER (PARTITION BY department) AS department\_total

FROM employees;

* Use Cases:
  + Moving averages, cumulative totals, ranking, etc.

**8. SQL Execution Order**

* The order in which SQL statements are executed, not necessarily the order in which they are written:
  1. FROM – Specify tables and joins.
  2. WHERE – Filter rows based on conditions.
  3. GROUP BY – Group rows.
  4. HAVING – Filter groups.
  5. SELECT – Define columns and expressions.
  6. ORDER BY – Sort results.
  7. LIMIT – Restrict the number of rows.

**9. Airflow vs. Cloud Composer**

* Airflow is an open-source tool to manage workflows and schedule tasks.
* Cloud Composer is a fully managed service by Google Cloud that runs Apache Airflow, offering seamless integration with GCP services.
* **Differences:**
  + Airflow: You manage everything (infrastructure, scaling, updates, etc.).
  + Cloud Composer: Managed by Google (auto-scaling, maintenance, and updates).
* **Architecture:**
  + Airflow: Web Server, Scheduler, Workers (managed by you).
  + Cloud Composer: Cloud-managed environment using Airflow, with Google handling the infrastructure.

**10. GCP Migration Project**

* **High-level Steps in GCP Migration:**
  1. Assessment and Planning: Understand current infrastructure and choose GCP services that match your needs (e.g., Compute Engine, BigQuery, Dataflow, etc.).
  2. Data Migration: Use Cloud Storage, Transfer Appliance, or Database Migration Service to move data.
  3. App Migration: Lift and shift apps, re-factor them, or use containers on GKE.
  4. Testing: Perform extensive testing for data integrity and app performance.
  5. Optimization and Monitoring: Leverage Stackdriver, cost optimization, and auto-scaling.

**BIG QUERY Architecture:**

When you submit a query to BigQuery, the Plato server (also called the query planner) breaks down the query into smaller tasks and distributes them to Leaf Nodes.

The Leaf Nodes are responsible for executing the query on data stored in Colossus, Google’s distributed storage system.

The execution engine used is Dremel, which enables parallel query execution across multiple nodes.

Different parts of the query are processed by different Leaf Nodes, and the results are sent to Mixers, which combine and aggregate them.

The Jupiter Network ensures fast communication between compute and storage layers, allowing smooth data transfer with minimal latency.

Once all tasks are completed and results are aggregated, the final output of the query is returned to the user.

This architecture makes BigQuery fast, scalable, and efficient.

Borg:

Borg is Google’s cluster management system, used within BigQuery's architecture.

It allocates and manages resources like compute and storage.

Borg makes BigQuery serverless and scalable by dynamically handling resource provisioning in the background.

**Borg's Role:**

* While Borg is a cluster management system, it's essential to emphasize that in the context of BigQuery, it's used for resource allocation and management, enabling BigQuery's serverless nature.
* It is also important to note that Borg has been superseded by a newer system called "Omega", and then "Kubernetes", but for the purpose of this explanation, Borg is sufficient.

**Dremel's Function:**

* Clarify that Dremel is the *multi-tenant, massively parallel query execution engine*. It's not just responsible for parallel execution, but also for breaking down queries into execution trees.

 **Colossus's Significance:**

* Emphasize that Colossus is a globally distributed file system, providing the underlying storage for BigQuery. It's designed for massive scale and durability.

 **Leaf Nodes and Mixers:**

* Explain that Leaf Nodes are responsible for reading data from Colossus and performing initial filtering and aggregation. Mixers perform further aggregation and shuffling of data.

 **Jupiter Network:**

* Highlight that the Jupiter network is a high-bandwidth, low-latency network crucial for efficient data transfer between compute and storage.

 **"Root Server" and "Leat Nodes"**

* "Root Server" should be "Root Node" and "Leat Nodes" should be "Leaf Nodes".

 **"Colessus"**

* "Colessus" should be "Colossus".

**CI CD PIPLINE :**

**CI = Continuous integration**

* Developers frequently push code to a shared Repository.
* Each push automatically triggers builders and tests.

**CD = Continuous Delivery \ Deployment**

* Continuous Delivered to a staging area.
* Continuous Deployed straight to production

**Company ; Prodapt**

1.Partion and clustering

2.Traditional databases vs Big query

3.find 2nd largest number in python and Big query

4.challenges you faced in Big query while loading the data

5.Exact role and responsibility in your project

6.size of the data ?

**cognizant**

1.what is routines in BQ?

2.What is federated queries and in which scenarios we use?

3.Authorized views and materialized views?

4.Actual tables and Temporary tables in BQ? and in which scenarios we use temporary tables?

5.write a sql query which one card is associated with single currency code and not associated with multiple currency,

Give a single mapped currency code?

Card\_num currency\_code

1 USD

2 USD

3 EURO

4 USD

4 EURO

5 USD

ans)

with card\_counts as(select card\_num, count(distinct cur\_code from table1 group by card\_num),

single\_currency\_cards as (select card\_num from card\_counts where currency\_count =1)

select c.card\_num,c.currency\_code from table1 join single\_currency\_cards scc on c.card\_num=scc.card\_num;

6.For the same above code Write a SQL query to find the first currency\_code for each card\_num, ordered alphabetically by currency\_code.

ans)

with ranked\_cards as (

select card\_num, currency\_code, row\_number() over(partition by card\_num order by currency\_code) as row\_num from card\_currency)

select card\_num,currency\_code from ranked\_cards where row\_num =1;

7.For the same above code write in pyspark by reading CSV file

ans)

data\_path= path

df= spark.read.option("header",true).csv(data\_path)

window\_spec=window.partitionby("card\_num").orderBy("currecncy\_code")

df\_ranked=df\_ranked.filter(df\_ranked.row\_num==1).select("card\_num","currency\_code")

df\_filtered.show()

**\*Random trees round 1\***

1.You are building an application that stores relational data from users. Users across the globe will use this application. Your CTO is concerned about the scaling requirements because the size of the user base is unknown. You need to implement a database solution that can scale with your user growth with minimum configuration changes. Which storage solution should you use?

2.Your organization is a financial company that needs to store audit log files for 3 years. Your organization has hundreds of Google Cloud projects. You need to implement a cost-effective approach for log file retention. What should you do?

3.Find the 2nd highest sales amount in each region.

region sales\_person

sales\_amount

North Alice 9000

North Bob 8500

North Charlie 8500

North David 7000

South Eva 9500

South Frank 9200

South George 9200

South Harry 8900

4.Generate a sequence of numbers using recursive CTE

**Randomtrees 2nd round**

1.Experience

2.Which tool do u specializise

3.Tools used

4.Different project u have worked on

5.Is your project injection or etl

6.What are the data source you have injested

7.current role and responsibilities

8.Any etl exposure u have( answered no )

9.How do you handle etl orchestration

10.Level of big query exposure

11.Did you use batch or streaming

Scenarios::

12.Under what Scenario you would use a materialized view in Big Query - When you already have a Normal View. Note - Your Data Set is in the Volume of 1 TB. The Data Sets Recieves Frequent Changes from the Source System

13.What do you mean by refresh materialized view

Q-Which Technique you would Use If you have been asked to perform→ physically organizes data within existing partitions based on one or more columns out of 1 TB of data Volume in BIG QUERY

14.Partion va clustering

15.Have u ever done Data sharding

16.How to do Data sharding

17.Have u worked on pub sub

18.You have an application packaged on a container that you plan to deploy on Cloud Run. The application performs some data analysis on messages from a Cloud Pub/Sub topic called order-queue.What are the Implementation steps you will foloow as Google-recommended practices for above Case?

19.PYTHON/PYSPARK CODE TEST

20.Replace the

→First 2 Occurrence of word BLR with KOL

→LAST BLR Occurance to DEL

from Below Dataset

1 SS 12 1234 10/12/2022 BLR

2 SS 12 1234 10/12/2022 HYD

3 SS 12 1234 10/12/2022 DEL

4 SS 12 1234 10/12/2022 BLR

5 SS 12 1234 10/12/2022 BLR

6 SS 12 1234 10/12/2022 BLR

7 SS 12 1234 10/12/2022 HYD

8 SS 12 1234 10/12/2022 PUN

E-This is CSV File & Last Column = CITY

21.Q-Consider a Scenario - While trying to delete records from a table that was updated using the GCP BigQuery Node.js table insert function and created through the GCP Console, you encounter the following error.

22.UPDATE or DELETE DML statements are not supported over table stackdriver-360-150317:my\_dataset.users with streaming buffer

23.What are the Steps of Implementation to resolve this issue?

24. Where u get this error and when ?

Based on the above error What challenges can we face

25.Q- You have a dataset in BigQuery where some fields contain NULL values.

26.How would you handle these NULL values in your queries - SPECIALLY WHEN YOUR Query is using TOO MANY AGGREGRATION?

27.How comfortable in slq

28.Q-Write a query to find the department with the highest average salary for employees who have been with the company for more than 2 years.

EMPLOYEE, DEPARTMENT Table

DATA PIPELINE ORCHESTRATION:

There is a Pipeline which loads Data from On Prem Source to Target - Approx 1 L Records Daily Requirement: From tomorrow Onwards Clients wants You to LOAD ONLY THOSE Records which are NEW in the Source System.

Eg-Out of 1L Records, Only 100 Records have Changed as NEW Recordc

How would you Design the Orchestration of this Pipeline using AIRFLOW/ADF/PYSPARK -> Solution so that You ONLY LOAD THOSE 100 NEW Records from Source System.

YESTERDAY BATCH: 1,2,3,4,5,6,7,8,9,10

TODAY BATCH: 1,2,3,4,5.5,6.5,7,8,9,10

E-THIS IS A CSV FILE, THERE IS NO DATE COLUMN IN THE SOURCE RECORD

29How to identify the change and then upload it

30.CORE PYTHON OOPS CODE TEST

MODULE A

CLASS A

TRUNC()

MODULE B

CLASS B

TRIM()

Write a Pseudo code -> Class B Object should use

TRUNC Method instead of TRIM

31.How would you Fill FIRST 2 Values with word KAS

from Below Dataset

1 SS 12 1234 10/12/2022

2 SS 12 1234 10/12/2022 HYD

3 SS 12 1234 10/12/2022 DEL

4 SS 12 1234 10/12/2022

5 SS 12 1234 10/12/2022

6 SS 12 1234 10/12/2022 BLR

7 SS 12 1234 10/12/2022 HYD

8 SS 12 1234 10/12/2022 PUN

32.What is the built in function for filling null values in python

33.What is the SQL Query Execution Order from below Series of SQL events :-SELECT, LIMIT& OFFSET, GROUP BY, FROM & JOIN WHERE, HAVING, ORDER BY

**\*\*INCEDO\*\***

1.self intro,Recent project explanation

2.Airflow DAG implementation

3.How you will handle failed cases in dataflow

4.SQL Blobs?

5.where,group by,having

6.storage classes in GCS

7.SQL commands,ddl dml dql

8.remove duplicates in a table

9.what is coalesce in sql

10.bigquery optimization techniques

11.what is partition

12.what is set in python

13.what are the operators in airflow

14.Write a SQL query to rank employees within each department based on their salaries in descending order by using window functions?

ans)

select empid,departmentid,salary,Dense\_rank() over(partition by department\_id order by salary desc)as salary\_rank

from employees;

15.find the employees in IT department who is spending more than 60000(assume 2 tables)

ans)

select e.emp\_id,e.departmentId,e.salary from employees e

join department d ON e.department\_ID=d.department\_id

where d.department\_name='IT'

and e.salary>60000;

**company: indium,**

**\*\*Hive Tables\*\***

\* \*\*Hive External vs. Managed Tables:\*\*

\* \*\*Managed Tables:\*\* Hive manages both the data and schema. If you drop a managed table, both the schema and the data are deleted.

\* \*\*External Tables:\*\* Hive only manages the schema. The data resides in an external location (e.g., HDFS, S3). If you drop an external table, only the schema is removed; the data remains intact.

**\*\*OLTP vs. OLAP\*\***

\* \*\*OLTP (Online Transaction Processing):\*\*

\* Designed for real-time transactional operations, such as inserting, updating, and deleting records.

\* Focuses on speed and concurrency for handling numerous short transactions.

\* Examples: Online banking, e-commerce order processing.

\* \*\*OLAP (Online Analytical Processing):\*\*

\* Designed for complex analytical queries and data warehousing.

\* Focuses on retrieving and analyzing large volumes of historical data.

\* Examples: Business intelligence reporting, data mining.

**\*\*Columnar vs. Row Storage\*\***

\* \*\*Row Storage:\*\*

\* Data is stored in rows, where all attributes of a record are stored together.

\* Efficient for OLTP workloads that require accessing entire records.

\* Examples: Traditional relational databases.

\* \*\*Columnar Storage:\*\*

\* Data is stored in columns, where all values of a particular attribute are stored together.

\* Efficient for OLAP workloads that require accessing specific columns.

\* Examples: Apache Parquet, Apache ORC, BigQuery.

**\*\*HDFS Block Size\*\***

\* \*\*Default Block Size:\*\*

\* The default block size in HDFS is 128 MB.

\* \*\*Impact of Changing Block Size:\*\*

\* \*\*Increasing:\*\*

\* Reduces the number of metadata operations, improving performance for large files.

\* May waste storage space for small files.

\* \*\*Decreasing:\*\*

\* Increases the number of metadata operations, potentially slowing down large file processing.

\* Improves storage efficiency for small files.

**\*\*HDFS File Storage and Jobs\*\***

\* \*\*1GB File:\*\*

\* HDFS splits files into blocks. The number of blocks depends on the block size. It will not result in 8 jobs. The number of map tasks will be related to the number of blocks.

\* \*\*1MB File:\*\*

\* Even a 1MB file will occupy a full block in HDFS. It will not result in 8 jobs. It will result in 1 map task, as it will take up one block.

**\*\*Python Lists vs. Tuples\*\***

\* \*\*Lists:\*\*

\* Mutable (can be modified).

\* Defined using square brackets `[]`.

\* \*\*Tuples:\*\*

\* Immutable (cannot be modified).

\* Defined using parentheses `()`.

**\*\*Python `\*args` and `\*\*kwargs`\*\***

**\* `\*args`:**

\* Used to pass a variable number of non-keyword arguments to a function.

\* Arguments are passed as a tuple.

**\* `\*\*kwargs`:**

\* Used to pass a variable number of keyword arguments to a function.

\* Arguments are passed as a dictionary.

**\*\*Python `list.append()` vs. `list.extend()`\*\***

\* `list.append()`:

\* Adds a single element to the end of a list.

\* If the element is a list, it adds the entire list as a single element.

\* `list.extend()`:

\* Adds multiple elements to the end of a list.

\* The elements are added individually from an iterable (e.g., another list).

**\*\*Finding the First Non-Repeating Character\*\***

\* \*\*Example (S = 'abxabyz'):\*\*

\* The first non-repeating character is 'x'.

**\*\*Finding the First Non-Repeating Character Using a Dictionary\*\***

```python

def first\_non\_repeating(s):

char\_counts = {}

for char in s:

char\_counts[char] = char\_counts.get(char, 0) + 1

for char in s:

if char\_counts[char] == 1:

return char

return None

```

**\*\*Spark Data Partitioning\*\***

\* \*\*Partitioning:\*\*

\* Divides a Spark DataFrame or RDD into smaller, manageable chunks.

\* Distributes data across multiple nodes in a cluster for parallel processing.

**\*\*When to Use Partitioning in Spark\*\***

\* When you need to optimize query performance by reducing data shuffling.

\* When you have frequently used filter or group-by columns.

\* When you want to distribute data evenly across the cluster.

\*\*Lazy Evaluation in Spark\*\*

**\* \*\*Lazy Evaluation:\*\***

\* Spark delays the execution of transformations until an action is called.

\* This allows Spark to optimize the execution plan and avoid unnecessary computations.

\* It allows spark to build a DAG of transformations, and then optimize the entire DAG before execution.

**Hexaware drive**

**### \*\*2. How do you process 5 TB of data in GCP?\*\***

- \*\*Batch Processing:\*\* Use \*\*Dataproc (Spark)\*\* or \*\*Dataflow (Apache Beam)\*\* for distributed processing.

**- \*\*Storage:**

\*\* Store raw data in \*\*Cloud Storage\*\* before processing.

**- \*\*ETL & Transformation:**

\*\* Use \*\*Dataflow\*\* for efficient parallel processing.

- **\*\*Loading into BigQuery:**

\*\* Use \*\*BigQuery’s native load\*\* or \*\*BigQuery Storage API\*\* for performance optimization.

- \*\***Partitioning & Clustering:**

\*\* To optimize query performance.

---

**### \*\*3. First and Last Punch of Employees (Attendance System)\*\***

- If \*\*punch data is in BigQuery\*\*:

```sql

SELECT employee\_id,

MIN(punch\_time) AS first\_punch,

MAX(punch\_time) AS last\_punch

FROM attendance\_table

WHERE punch\_date = CURRENT\_DATE()

GROUP BY employee\_id;

```

**- If \*\*real-time processing\*\* is needed**:

Use \*\*Pub/Sub → Dataflow → BigQuery\*\* pipeline.

---

**### \*\*4. Handling Metadata Differences (On-Prem vs BigQuery) Before Loading into GCS\*\***

- \*\*Identify Differences:\*\* Compare column names, data types, formats (e.g., date formats).

- \*\*Standardization in Staging Layer:\*\*

- Use \*\*Dataflow\*\* to transform data before writing to GCS.

- Convert inconsistent data types (e.g., \*\*VARCHAR → STRING, INT → NUMERIC\*\*).

- \*\*Schema Mapping:\*\* Create a mapping table to ensure consistency before loading into BigQuery.

---

**### \*\*5. What is Dataflow?\*\***

Google \*\*Dataflow\*\* is a \*\*serverless ETL tool\*\* based on \*\*Apache Beam\*\*, used for \*\*batch and streaming data processing\*\*. It handles:

- \*\*ETL Pipelines\*\* (Extract, Transform, Load)

- \*\*Data Enrichment\*\* (Joining, Filtering, Aggregation)

- \*\*Real-time Streaming\*\* (Using Pub/Sub & BigQuery)

---

**### \*\*6. Explain Real-time End-to-End Pipeline\*\***

For real-time \*\*customer order processing\*\*:

1. \*\*Source:\*\* Orders captured in \*\*Pub/Sub\*\* (Kafka alternative).

2. \*\*Processing:\*\* \*\*Dataflow\*\* applies transformations (cleaning, validation).

3. \*\*Storage:\*\* Processed data stored in \*\*BigQuery\*\* for analytics.

4. \*\*Visualization:\*\* \*\*Looker Studio\*\* or dashboards for real-time monitoring.

---

**### \*\*7. How to Remove Bad Data in a CSV File?\*\***

- \*\*Data Validation Before Processing:\*\*

- Use \*\*Cloud Functions or Dataflow\*\* to detect missing/null values.

- Check \*\*schema mismatches, incorrect date formats\*\*.

- \*\*During Processing (Dataflow/Dataproc)\*\*:

```python

import pandas as pd

df = pd.read\_csv("file.csv")

df = df.dropna() # Remove rows with null values

df = df[df["column\_name"].str.isnumeric()] # Remove invalid data

```

- \*\*BigQuery Handling:\*\* Use \*\*EXCEPT()\*\* or \*\*WHERE conditions\*\* to filter bad records.

---

**### \*\*8. Schema Change with No Downtime in Production\*\***

- \*\*BigQuery:\*\* Use \*\*schema evolution\*\* (add new columns without downtime).

- \*\*Rolling Updates:\*\*

- Use \*\*Cloud Composer (Airflow)\*\* for schema migration.

- Load new schema data into a separate table and \*\*use views\*\* to merge it.

- \*\*Dataflow Handling:\*\* If datatype change, \*\*cast it dynamically\*\* in the transformation layer.

**### \*\*9. Avoiding Duplicates While Moving Data from Staging to History Table\*\***

- \*\*Use MERGE in BigQuery:\*\*

```sql

MERGE INTO history\_table AS h

USING staging\_table AS s

ON h.id = s.id

WHEN MATCHED THEN UPDATE SET h.col1 = s.col1, h.col2 = s.col2

WHEN NOT MATCHED THEN INSERT (id, col1, col2) VALUES (s.id, s.col1, s.col2);

```

- \*\*Use Primary Key + Deduplication in Dataflow:\*\*

- Implement a \*\*deduplication mechanism\*\* before writing.

---

**### \*\*10. Data Loaded Partially in BigQuery – How to Handle?\*\***

- \*\*Check Job Status:\*\*

```sql

SELECT \* FROM `region-us.INFORMATION\_SCHEMA.JOBS\_BY\_PROJECT`

WHERE job\_id = '<your\_job\_id>' AND state != 'DONE';

```

- \*\*Common Issues & Fixes:\*\*

- \*\*Insufficient Quota:\*\* Increase resources.

- \*\*Failure in Source Pipeline:\*\* Retry logic in Dataflow.

- \*\*Transaction Failure:\*\* Use BigQuery’s \*\*retry and commit mechanism\*\*.

- \*\*Load Data in Chunks:\*\* Break large files into smaller batches.

---

**### \*\*11. What is SCD (Slowly Changing Dimension)?\*\***

SCD is used in \*\*Data Warehousing\*\* to handle \*\*historical changes in dimensions\*\*.

- \*\*SCD Type 1:\*\* Overwrite old data (no history).

- \*\*SCD Type 2:\*\* Maintain history with \*\*effective date, expiry date, or versioning\*\*.

- \*\*SCD Type 3:\*\* Store a limited history (e.g., current & previous values).

**For \*\*BigQuery\*\*:**

- \*\*SCD Type 2 Implementation:\*\*

```sql

MERGE INTO dim\_customer AS target

USING staging\_customer AS source

ON target.customer\_id = source.customer\_id

WHEN MATCHED AND target.end\_date IS NULL THEN

UPDATE SET target.end\_date = CURRENT\_DATE()

WHEN NOT MATCHED THEN

INSERT (customer\_id, name, start\_date, end\_date)

VALUES (source.customer\_id, source.name, CURRENT\_DATE(), NULL);

```

**Cognizant F2F**

1.Self intro

2.How to retreve the one hour back data in bigquery -time travel

3. Real project scenarios

1. What if job pipeline failed what are your furthur actions

2. What are the libraries you worked in pyspark

3. What is your actual expeience on data flow ,data proc

4. Write the one flow on data proc

5. How did you schduled the pipeline on cloud composer

6. What kind of data you loaded in bigquery -i mean in what way we usually load

7. What is your downstream what is their needs of requirements you deliver in sprint tasks

8. What is xcel operator which platform you use to write.

9. Write one dag

10. If one if the task in your dag needs to be hold whats your approach in that

Happened only for 20 mins

completely airflow ,cloud composer ,pyspark theory questions they asked

**TCS F2F**

1. difference between sort by and order by

2. If you are having a job I'd in airflow how do you find the error

3. Difference between app engine and cloud engine

4. Mostly questions are on project based

5. Any idea on data fusion

6. Purpose of data proc

7. In how many ways do you create a table with 2 columns in big query

8. Self intro

**Company: Motivity Labs**

1.intro

2.what are the tools used in your project

3. pipeline

4. How to load gcs to bq

5. frequency of source

6.size of source

7. frequency of running dag

8. Rate on python

9. Are you using cloud function and where

10. Are you worked on dataflow and dataproc

11. Do you know the ci cd pipeline

12. Do you aware of bigquery AI ML

**Company: Renexter**

1)intro

2) pipeline

3)how to load GCS to big query

4) big query architecture

5) explain dag

**HCA Healthcare**

**19/03/2025**

1.Introduce yourself, about project name and what tools you use?

2.How do you setup CI/CD pipeline using SQL

3.Master Classes in python

4.scenario based question in SQL?

if you buy products at a store like 4/5 products,

so how will you find the products together? How will you approach in SQL?

5.Secrets in GCP?

6.Diff between Merge, Join, Concat()?

7.Context Managers in python?

8.Decorators in python?

**HEXAWARE**

roles responsibilities

getting huge data how u handle the data in bigquery?

gcs contains data and u r loading the data but its taking too long time what u would do here?

from source history data is coming and adding in a table row by row what u will use?

what is dataflow, dataproc?

what u do in dataflow ?

diff between airflow and composer? what u do in composer?

what are all the libraries u have used in airflow dag code?

sql:

in an organisation there is in time and out time and with one name 3 people are there and it should not be duplicated?

what are scds?

python:

for data processing what u use in python ?

how many libraries u have used?

in pandas to do processing what u use?

how do u remove duplicates ?

pyspark:

batch data is coming in pyspark what u use?

**Hexaware drive Walk in Round L1**

**date: 15/03/25**

1. Self intro

2. How do you process 5 TB of data in gcp

3. In a office, we have few employees. How to get the first and last punch of employees.

4. Let us say, you have a file and that file has different metadata in onprem and Bigquery, how do you handle it before loading into GCS

5. What is dataflow?

6. Explain real time end to end pipeline

7. How to remove bad data in a csv file

8. The pipeline is already running in a production, now there is a change in schema datatype ane you have no downtime. How would you do that.

9. While transforming data from staging to history, how do you not insert duplicate records and how do you do that.

10. In end to end pipeline you see that data has only loaded half in bigquery table. How do you handle this situation?

11. What is SCD ?

last cha a 5 characters

select c1,c2 from tabloe where c1 like '\_a' and len(c1)=5;

WORKER

WORKER\_ID FIRST\_NAME LASTNAME SALARY JOINING \_DATE DEPART\_MENT

1 Monika Aora 10000 12-02-2012 HR

2 Niharila Verma 8000 05-03-2013 Admin

3 Vishal Singhal 30000 11-05-2010 HR

4 Amitabh Singh 50000 08-07-2012 Admin

5 vivek Bathi 50000 17-06-2011 Admin

6 Vipul Diwan 20000 21-06-2011 Account

7 Satish Kumar 7500 11-07-2010 Account

8 Geetha Gauthm 9000 11-07-2010 Admin

Bonus

WORKER\_ID Bonus\_date Bonus\_Amount

1 20-02-2016 5000

2 11-06-2016 3000

3 20-02-2016 4000

1 20-02-2016 4500

2 11-06-2016 3500

Get workers who has bonus amount >4500?

select w.worker\_id, w.first\_name,w.last\_name,b.Bonus\_Amount from WORKER w join Bonus b on w.worker\_id =b.WORKER\_ID where b.Bonus\_Amount >4500

workers who did not get any bonus ?

HR Admin Account

Total Sal 40000 157000 27500

select w.WORKER\_ID from WORKER w LEFT JOIN Bonus b on w.worker\_id =b.WORKER\_ID where b.Bonus\_Amount is null

select w.DEPART\_MENT,sum (w.SALARY) as total\_sal from WORKER w join Bonus b on w.worker\_id =b.WORKER\_ID group by w.DEPART\_MENT

select DEPART\_MENT ,sum(sala) as total\_sal from worker group

with even\_row as ( select \*,row\_number() over (order by (select null )) as row\_no from table )

select \* from even\_row where row\_no %2 =0

email =========== Myname@gmail.com testname@gmail.com ragini@yahoo.com kumaran@yahoo.com mugesh@Live.com magesh@Live.com raman@rediff.com

Expected output

===============

Domain count

gmail 2

yahoo 2

Rediff 1

**INCEDO**

1.introduction

2.real time challeges you have faced

3.real time experiance

4.stories in jira board

5.accumelators pyspark

6.Migration project

7.from where your data is coming

8.using groupby write a data and columns in pyspark

9.given 2colums and 3 rows of data, write rows as columns and colums as rows

**Lti Mindtree**

* Triangle program
* Project
* Self
* Joins
* Airflow which operator u have used
* Connection between sql to GCS
* Optimization u have used in your project
* Triangle program

**CAPGEMINI**

Self

1.olap and oltp

2.fact table and dimensions table

3.exampl of star scheme and snowfalke scheme

4.data modaling

5.what is cte

6.what is your source

7. What is lead and lag

8. How to find columns in a table

9. How to find null in a column

10. How Replace the null values in a column

11. How get duplicates in a table

12. Store procedure

13. String = I am giving interview find the longest word in the sentence

14. Word = [ 'm','w',] how many 'm' and 'w' count in given word

15. To retrieve the items bought by customers from the UK, we need to join the Customers and Orders tables using customer\_id. The SQL query

16. To find the total amount spent by each customer, we need to sum up the amount from the Orders table and group it by the customer\_id

**LTI MINDTREE**

**1.self**

**2. Gcp services Explain**

**3. What is Air flow**

**4. input=['Keyboard', 'laptop', 'mouse', 'bag', 'bag']**

**we need to convert above list to dictionary with no of occurences for each objects**

**like**

**{'Keyboard':1,**

**'laptop':1,**

**'mouse':1,**

**'bag':2**

VIRTUSA

Self

2. Pipeline

3. Data proc and without data proc

4. Data proc and data flow

5. PySpark

6.Partion and clustering

7. Time travel big queary

**LtiMindtree Interview questions**

Panel name: Yashwanth Singh

1)Have you seen your JD

2)Explain me what you understood from your JD

3) Self introduction

4) sql query to replace usa with america uk with United Kingdom with in customer table country column

5) sql query to count country wise customer count

6)What are different datatypes in python

7) Asked to write all data types with 1 element in it.

8) if there is a very slow running query within sql how to optimize it

9) Have u done data validations, data ingestion like that

**Interview Questions:**

**Company: Forefront(Deloitte)**

1) Unnest

2) Lag and Lead

3) Drop, Delete, Truncate

4) Snapshot , Timetravel

5) partitioning on which data types,

6) Clustering limit upto how many columns

7) Rank and DenseRank

**Prodapt**

1.what is cloud composer

2.what is dataflow, dataproc

3.materlized views

4.partitioning clustering

5.operators in Cloud composer

6.dataflow,pubsub usecases

7.have u used dataproc and pyspark explain

8.issues in project

9.fact and dimension table

10.joins and aggregation in bq

11.conflict in company (personal questions)

Capgemini L2 interview questions

Panel Name: Imran Khan

1) self introduction

2) explain about your recent project

3) olap and oltp difference

4) dataware house and data lake difference

5) Fact tables and dimension tables

6) Any examples for fact table what would be the data

7) Star schema and snowflake schema

8) Have u involved in data modelling

9)How would u rate yourself in sql

10)What is lead/lag function

11)What are the ways to find duplicates in table

12)how do you find nulls within a table

13) How can we replace null with any values

14) How can we get all columns within a table

15) 3 sql queries in programiz editor

16) python script s= "I am giving interview", give the word with max length

17) s=['w','m'] write python script to count no of times w is there

18) what is the reason for job change

**Pyspark Interview questions**

* What is Dataproc?
* Spark optimization techniques.
* Dataproc vs Dataflow.
* Spark and PySpark in Dataproc.
* RDD vs DataFrame.
* Spark Architecture.
* Spark Session.
* Transformations and Actions in Spark.
* Repartition vs Coalesce.
* Narrow vs Wide Transformations.
* Persist v/s cache
* Broadcast Join.
* Map vs FlatMap.
* GroupByKey vs ReduceByKey.
* Reading/Writing CSV, JSON, Parquet files.
* All Interview questions Topic wise
* Handling 2TB data distribution.
* Fault tolerance in Spark (DAG, RDD).
* What have you done for performance tunning??
* What is the block size in your project?
* How u will read a file if a file had multiple delimiters ?
* Write PySpark code to read a CSV file from an S3 bucket and convert it into Parquet format.
* How would you broadcast small datasets in PySpark?
* How do you handle data skewness in PySpark?
* Write PySpark code to remove duplicates based on multiple columns.
* Write PySpark code to implement Slowly Changing Dimension (SCD) logic.
* Dataproc cluster resizing.
* Different machine types in Dataproc.
* Real-time scenario-based questions.
* UDF in PySpark.
* PySpark Actions & Transformations.
* Groupby and reduced by key
* Logical plan and physical plan
* Hive Data Skewness
* Handling the airflow
* Executors
* Accumilators
* Who is responsible for reading a file ? driver or executor?
* If a 100gb file and workernode of 5 gb memory and 5gb driver node so, will the load of a cluster able to read the file?
* Who is responsible for running a job? Spark architecture?
* If 100gb file is there how will u calculate how many executors size and partitions?
* How an execution flow works for reading a file in the backend?
* What is difference between job stage and task in spark
* What are accumulators in spark

You need to calculate the cumulative sum of sales over time for each product. How would you approach this?

data = [("product1", "2023-12-01", 100), ("product2", "2023-12-02", 200), ("product1", "2023-12-03", 150), ("product2", "2023-12-04", 250)]

columns = ["product\_id", "date", "sales"]

Code writing

write a code in pyspark count word in file

write a code in pyspark count character in file

In pyspark write the code to change the dataframe

input

col1 col2

1 a,b

2 c,d

Output

col1 col2

1 a

1 b

2 c

2 d

Scenario

✅ You have a dataset of 100 million records, but only 2000 records are duplicates. How would you remove the duplicates?

✅ Your job is failing due to an Out of Memory (OOM) error during the final shuffle step — how would you debug and optimize this?

✅ How would you repartition data to enhance performance in PySpark?

✅ What is the best approach for handling null values during data transformations?

Python

* List vs Tuple.
* Lambda Function.
* Map Function.
* Reverse a string.
* List Comprehension.
* Count vowels in a string.
* Count frequency of elements in a list.
* Flatten a nested list using recursion.
* ODD/EVEN check using Lambda.
* Python program to read CSV.
* Merge DataFrames using Pandas.
* Extract substring from string.
* Add column in DataFrame.
* dict={A:1, B:2, C:2}
* Call all keys, values and total once.
* o/p={1:A,2:B,3:C}
* Converting Integer to String.
* a={'B':5,'A':9,'C':7} what will be the output of sorted(a)
* Deep copy vs shallow copy
* Empty string to NULL.
* Generator and Decorators.
* Handling large datasets in Pandas.
* Word Count in python
* Decorators in python with example
* L = [8, 9, 1, 2, 3, 4, 4, 5, 6]
* Find the list comprehension for odd numbers
* Arrange the elements in a descending order
* Need output as [8, 9, 1, 2, 3, 6, 5, 4, 4] changing of last 4 numbers
* Power the list of numbers
* reverse a list in python without using predefined functions
* Print the second largest value in the output
* Need output ‘8.9.1.2.3.4.4.5.6’
* G = [1, 2, 3] F = [ 4, 5, 6]
* Map the values into (1,4), (2,5), (3,6)
* reverse a list in python without using predefined functions
* pd data = {'Name': ['John', 'Cataline', 'Matt'],
* 'Age': [50, 45, 30], 'City': ['Austin', 'San Francisco', 'Boston'],
* 'Marks' : [70, 80, 95] }
* filter rows age>40 with all columns(dataframes)
* Ist = [0,0,1,1,1,2,2,3,3,4,2,2,2] o/p = [0,1,2,3,4,2]
* str = 'mississippi str\_op = 'misp’ try to write code in a single line
* What are scalar functions present in python

In python -Print unique values from the column value in row by row

Col1 Col2

101 'xyz,1,2,2,3,3,3,ab,4,4,4,5,5,5,5,5'

102 '3,3,xyz,4,5,ab'

output: 1 2 3 4 5 ab xyz

* Find the Missing Number in a List. write a python code snippet to find the missing number in a sequence of 1 to N?
* Write a python code snippet to reverse a string without using built-in functions reverse or [-::1]
* Write a code snippet to input two numbers assume a, b as two numbers.Create a exception when ever either of the numbers are 0 or -ve.if there is no exception then print the sum of the values
* Write a function that takes a string s as input and returns the index of the first non-repeating character in the string. If no such character exists, the function should return -1.

**kiran kumar**

**decision minds**

1.introduction

2.roles and responsibilities

3.about the project

4.challenges you faced in the project

| empId | name | year | Salary |prev\_salary

+-------+-------+------+--------+

| 1 | Clark | 2018 | 10000 |-

| 1 | Clark | 2019 | 10000 |10000

| 1 | Clark | 2020 | 15000 |10000

| 1 | Clark | 2021 | 15000 |15000

| 1 | Clark | 2022 | 17000 |15000

| 1 | Clark | 2023 | 18000 |17000

| 2 | Dave | 2019 | 20000 |18000

| 2 | Dave | 2020 | 30000 |20000

| 2 | Dave | 2021 | 30000 |

| 2 | Dave | 2022 | 40000 |

| 2 | Dave | 2023 | 40000 |

| 3 | John | 2020 | 30000 |

| 3 | John | 2021 | 30000 |

| 3 | John | 2022 | 40000 |

+-------+-------+------+-------

table: employee table

atleast twice two salary increments

with salary\_change as

(select empid, name, salary,

Lag(salary) over(partition by empid order by year) as prev\_salary

from employee

),

increments as

(select empid, name, count(\*) as increment\_count

from salary\_change

where salary > prev\_salary

group by empid, name)

select empid, name, increment\_count from increments

where increment\_count >= 2;

Customer

customer\_id product\_id amount

1 1 1000

2 2 2000

3 1 1100

4 1 1000

7 1 1000

7 3 4000

6 4 2000

1 5 1500

2 5 2000

4 5 2200

7 6 5000

1 2 2000

Product

product\_id category name

1 A A1

2 A A2

3 B B1

4 B B2

5 C C1

6 C C2

Write SQL to get the ids of the customers who has purchased at least one product from every product category

select c.customer\_id

from customer as c

join product as p

on c.product\_id = p.product\_id

group by c.customer\_id

having count(distinct p.category) = (select count(distinct category) from product);

customer 1

custid productid amount category

1 1 1000 A

1 5 1500 C

1 2 2000 A

FINAL OUTPUT

c.customer\_id

idname|designation|email|salary|dept|modified\_date

1|Amit|Manager|amit@abc.com|10000|IT|1-jan-2023

2|Ravi|Lead|ravi@tyt.com|20000|HR|1-jan-2023

3|Sumit|Manager|sumit@pqr.com|IT|15000|1-jan-2023

schema = StructType([StructField('id', integertype(), true),

([StructField('id', integertype(), true),

([StructField('id', integertype(), true),

([StructField('id', integertype(), true),

([StructField('id', integertype(), true),

([StructField('id', integertype(), true)

data = [1|Amit|Manager|amit@abc.com|10000|IT|1-jan-2023]

df = spark.createDataframe(data, schema)

df.show()

df=spark.read.option('header', True).option('inferSchema', True).csv('filepath.csv').

df.show()

Input : a string and a substring.

Print the number of times the substring occurs in the given input string

Example -

String : ABCDCDE

Substring : CD

Output : 2

string = input('enter any string ')

substring = input('enter any substring ')

count = 0

len\_sub = len(substring)

for i in range(len(string):

if string[i] == substring:

count += 1

print('Output', count)

**PYTHON AND PYSPARK**

**1. Accumulators in PySpark ?**

**An accumulator in PySpark is a shared variable used to accumulate values across multiple stages in parallel. They are write-only variables and are typically used for counting or summing values. PySpark supports accumulators of type integer and float.**

**from pyspark import SparkContext**

***CODE :***

**sc = SparkContext.getOrCreate()**

**accum = sc.accumulator(0)**

**def add\_to\_accum(x):**

**global accum**

**accum += x**

**rdd = sc.parallelize([1, 2, 3, 4, 5])**

**rdd.foreach(add\_to\_accum)**

**print(f"Accumulator Value: {accum.value}")**

**2. Using GroupBy to Write Data and Columns in PySpark ?**

**You can use groupBy in PySpark to group data by certain columns and then perform actions like count, sum, etc. Here’s an example of using groupBy to count the number of records for each group:**

***CODE :***

**from pyspark.sql import SparkSession**

**spark = SparkSession.builder.appName("GroupByExample").getOrCreate()**

**data = [(1, 'A'), (2, 'B'), (3, 'A'), (4, 'B'), (5, 'A')]**

**df = spark.createDataFrame(data, ['id', 'category'])**

**df\_grouped = df.groupBy('category').count()**

**df\_grouped.show()**

**3. Master Classes in Python**

**Master Classes are typically advanced or specialized courses on a particular Python topic. These courses provide in-depth knowledge and advanced coding techniques.**

**4. Context Managers in Python**

**Context managers in Python are used for resource management, particularly for file handling and network connections. They are implemented using the with statement. Example:**

**with open('file.txt', 'r') as file:**

**content = file.read()**

**print(content)**

**This ensures the file is properly closed after the block of code is executed.**

**5. Decorators in Python**

**A decorator is a function that allows you to modify the behavior of another function or method. It's often used for logging, access control, memoization, and more. Example:**

**def decorator(func):**

**def wrapper():**

**print("Before function call")**

**func()**

**print("After function call")**

**return wrapper**

**@decorator**

**def say\_hello():**

**print("Hello!")**

**say\_hello()**

**6. For Data Processing, What Do You Use in Python?**

**For data processing, I use libraries such as:**

* **Pandas for in-memory data manipulation.**
* **NumPy for numerical computations.**
* **PySpark for distributed data processing.**
* **Dask for handling larger-than-memory datasets.**
* **SQLAlchemy for SQL queries in Python.**

**7. How Many Libraries Have You Used in Python?**

**In Python, I have worked with many libraries across different domains:**

* **Data processing: Pandas, NumPy, PySpark, Dask**
* **Web development: Flask, Django**
* **Machine Learning: Scikit-learn, TensorFlow, Keras**
* **Visualization: Matplotlib, Seaborn, Plotly**
* **Networking: Requests, Flask**

**8. Pandas to Do Processing, What Do You Use?**

**In Pandas, I use functions like groupby(), merge(), pivot(), apply(), and agg() for data processing tasks. Pandas makes data manipulation straightforward for datasets that fit into memory.**

**9. How Do You Remove Duplicates?**

**You can use the drop\_duplicates() function in Pandas or PySpark to remove duplicates:**

**df = df.drop\_duplicates() # In Pandas**

**df = df.distinct() # In PySpark**

**10. Write Any Code in Python**

**Example: Reverse a string.**

**string = "Hello"**

**reversed\_string = string[::-1]**

**print(reversed\_string)**

**11. Where to Use Python in Our Project?**

**Python can be used in various parts of the project:**

* **Data Processing: ETL tasks using libraries like Pandas or PySpark.**
* **Web Development: Back-end development using frameworks like Django and Flask.**
* **Machine Learning: Model building and deployment using libraries like Scikit-learn and TensorFlow.**

**12. How to Remove Duplicates (List)**

**You can remove duplicates while preserving the order by converting the list into a set and back to a list:**

**lst = [1, 3, 2, 2, 2, 4, 4, 5]**

**result = list(dict.fromkeys(lst))**

**print(result)**

**13. Read and Write 10K Rows in Python**

**For large datasets, we use pandas or csv module to read and write:**

**import pandas as pd**

**# Read data**

**df = pd.read\_csv("file.csv")**

**# Write data**

**df.to\_csv("output.csv", index=False)**

**14. Libraries Used in PySpark**

**I have used libraries such as:**

* **pyspark.sql for DataFrames**
* **pyspark.streaming for real-time processing**
* **pyspark.ml for machine learning**

**15. Libraries You Have Used in Python**

**I have worked with libraries like:**

* **Data Processing: Pandas, NumPy, PySpark, Dask**
* **Web Frameworks: Flask, Django**
* **Machine Learning: Scikit-learn, TensorFlow**
* **Visualization: Matplotlib, Seaborn**

**16. Do You Know About Decorators in Python?**

**Yes, decorators in Python allow modifying the behavior of a function or method. They are implemented using the @decorator syntax.**

**17. What Happens in the Backend After Submitting a PySpark File**

**When a PySpark job is submitted, it is compiled into a Directed Acyclic Graph (DAG) of stages and tasks. It is then executed across a cluster of workers, where tasks are distributed and run in parallel. The final output is returned to the driver.**

**18. Python Script: Max Length Word in String**

**s = "I am giving interview"**

**words = s.split()**

**max\_word = max(words, key=len)**

**print(max\_word)**

**19. Count Occurrences of 'w' in a List**

**s = ['w', 'm', 'w', 'w']**

**count\_w = s.count('w')**

**print(count\_w)**

**20. Python Script to Reorder List**

**lst = [1, 2, 4, 5]**

**lst = lst[2:] + lst[:2]**

**print(lst)**

**21. Transformations Used in PySpark**

**Some common transformations in PySpark include:**

* **map()**
* **filter()**
* **flatMap()**
* **groupBy()**
* **distinct()**

**22. Correlated Queries in PySpark**

**Correlated subqueries in PySpark are generally used in SQL-like operations with joins. In PySpark, you can achieve similar functionality with join() and window() functions.**

**23. Data Governance in PySpark**

**Data governance involves ensuring data quality, security, compliance, and privacy. In PySpark, data governance can be implemented through access controls, logging, and data validation.**

**24. Broadcast Join in Spark**

**A broadcast join in Spark is used when one of the datasets is small enough to fit into memory. The smaller dataset is broadcasted to all worker nodes, and then the join happens locally.**

**25. Expand JSON List into Separate Rows in Python**

**import pandas as pd**

**data = [**

**{"category\_id": 210, "product\_id": [3001, 3002, 3005]},**

**{"category\_id": 211, "product\_id": [4001, 3032, 3305, 2317]}**

**]**

**df = pd.json\_normalize(data, 'product\_id', ['category\_id'], record\_prefix='product\_')**

**df.columns = ['category\_id', 'product\_id']**

**print(df)**

**REAL TIME**

**1. Real-Time Challenges You Have Faced?**

In real-time, I’ve faced challenges such as handling large data volumes, ensuring data quality across distributed systems, and managing system performance under heavy loads. One specific challenge was optimizing the performance of a data pipeline that was running slower than expected due to inefficient joins. We resolved it by partitioning the data and leveraging broadcast joins to speed up processing.

**2. Real-Time Experience in Project?**

In my current project, I work with big data pipelines in PySpark and Google Cloud Platform (GCP). The project involves processing millions of records daily, transforming raw data into actionable insights, and loading it into BigQuery for analysis. One key challenge was ensuring data consistency when working with different data sources, which I solved by building automated validation checks.

**3. Stories in Jira Board**

In our project, we use Jira for managing tasks and sprints. Stories are written to define specific pieces of functionality, such as "As a data engineer, I need to clean and transform raw data so that it can be loaded into BigQuery." Each story is broken down into sub-tasks for implementation, testing, and deployment.

**4. Migration Project**

In a migration project, I was responsible for migrating our data from on-premises infrastructure to GCP. We planned the migration in phases, starting with moving data to Google Cloud Storage (GCS) and then setting up data pipelines in Google Cloud Dataflow for processing. We ensured that the data was validated at each stage to prevent any loss during migration.

**5. What is the Source of Data?**

The source of data in my project typically comes from various systems, including transactional databases, APIs, and CSV files. We also collect data from third-party services and integrate it into our data pipeline for processing and analysis.

**6. Introduce Yourself, About Project Name, and Tools You Use?**

I am a Data Engineer with experience in building scalable data pipelines. In my current project, we focus on transforming and analyzing large datasets from multiple sources. We use tools like PySpark, BigQuery, GCS, and Apache Airflow to manage ETL pipelines and perform data transformations.

**7. How Do You Setup CI/CD Pipeline Using SQL?**

To set up a CI/CD pipeline with SQL, I would use a combination of GitLab CI, Jenkins, and cloud-based services like Google Cloud. The process typically involves:

* Storing SQL scripts in a Git repository.
* Setting up automated tests for the SQL queries.
* Using Jenkins or GitLab CI to run tests and deploy changes to production databases after validation.

**8. Roles & Responsibilities**

As a Data Engineer, my responsibilities include:

* Designing, building, and maintaining ETL pipelines.
* Ensuring the integrity and quality of data.
* Working with cross-functional teams to understand data needs.
* Optimizing data storage and query performance.
* Writing clean, maintainable code and automating processes.

**9. What is the Major Role in Your Project?**

My major role is to manage and optimize the data pipeline. This includes extracting data from different sources, transforming it to meet business requirements, and loading it into data warehouses like BigQuery. I also ensure that the pipeline runs efficiently, with minimal downtime, and that data is processed correctly.

**10. Project Name, How Are You Receiving the Data to GCS?**

In my project, we use APIs, FTP, and direct database connections to extract data from various sources and store it in Google Cloud Storage (GCS). We have set up scheduled jobs using Apache Airflow to automate data transfers to GCS.

**11. What is the Purpose of Jira Tool, How is Agile Methodology Used in Your Project?**

Jira is used to manage tasks and track the progress of sprints. In my project, we follow Agile methodology, where we break down the work into smaller tasks (user stories) and prioritize them in sprints. Each sprint focuses on delivering specific functionalities and improving the system iteratively.

**12. How to Remove Bad Data in a CSV File?**

To remove bad data in a CSV file, I would:

* Load the file into a DataFrame (using Pandas in Python).
* Check for missing values or invalid entries (like nulls or out-of-range values).
* Use dropna() or fillna() to handle missing values and filter out rows with invalid data.

import pandas as pd

df = pd.read\_csv("data.csv")

df = df.dropna() # Removes rows with missing values

df = df[df['column\_name'] > 0] # Filters rows with valid values

**13. The Pipeline is Already Running in Production, Now There is a Change in Schema Datatype and You Have No Downtime. How Would You Do That?**

I would handle this by implementing schema evolution. This involves:

* Creating a backward-compatible schema that supports both old and new data types.
* Using tools like Apache Avro or Parquet, which allow for schema versioning.
* Implementing conditional logic to handle the conversion of old data to the new schema during the data transformation stage.

**14. While Transforming Data from Staging to History, How Do You Not Insert Duplicate Records?**

To avoid inserting duplicates, I would:

* Use unique keys (like a combination of ID and timestamp) to identify records.
* Use a deduplication step in the ETL pipeline, where we can check the data before inserting it into the history table.
* In SQL, I would use INSERT IGNORE or ON DUPLICATE KEY UPDATE to handle duplicates.

**15. In End-to-End Pipeline, Data Has Only Loaded Half into BigQuery Table. How Do You Handle This Situation?**

I would:

* Check the pipeline logs to understand where it failed.
* Investigate if there was an issue with the data source or network connection.
* Restart the pipeline and implement checkpointing to ensure that the data loads completely in case of a failure.
* Use retries to handle intermittent failures, and make sure the process is idempotent.

**16. Real Project Scenarios**

In a real project, I’ve worked on building an end-to-end data pipeline for a retail client. We collected transaction data from their sales platform, transformed it for analysis, and loaded it into BigQuery. One key challenge was ensuring data consistency across different time zones and handling missing data effectively.

**17. What If Job Pipeline Failed, What Are Your Further Actions?**

If a job pipeline fails:

* I would immediately check the logs to identify the root cause.
* I would then fix the issue, whether it's a data issue, code issue, or resource allocation issue.
* Once fixed, I would re-run the pipeline and monitor it closely to ensure it completes successfully.
* I would also implement proper alerting mechanisms for future issues.

**18. Explain Your Current Role and What Are Your Daily Activities?**

I’m currently working as a Data Engineer where I design and build ETL pipelines. My daily activities involve:

* Writing and optimizing SQL queries.
* Managing data ingestion into cloud storage (GCS).
* Ensuring data quality and consistency.
* Collaborating with data scientists and analysts for data-related requirements.

**19. What is the Reason for Job Change?**

I am looking for new opportunities where I can grow professionally, work on more complex data engineering challenges, and contribute to projects that have a larger impact. I'm excited about the opportunity to work with new technologies and gain deeper experience in cloud-based data processing.

**20. What Challenges Faced in Your Project?**

Some challenges I faced include:

* Optimizing ETL pipelines for large datasets.
* Ensuring real-time data processing without impacting performance.
* Managing schema changes without downtime.
* Coordinating with cross-functional teams to understand and meet their data requirements.

**21. Have You Used System-Related Metadata Tables in Your Projects?**

Yes, I have worked with system metadata tables in relational databases and data warehouses to track table structure changes, query performance, and data lineage. These tables help in auditing, troubleshooting, and ensuring data integrity.

**22. If Task Fails, How Will You Handle It?**

If a task fails, I would:

* Review the error logs to diagnose the issue.
* Fix the issue based on the error message (e.g., missing data, syntax error, resource issue).
* Test the fix in a staging environment and then rerun the task.
* Implement better error handling or retries to minimize future failures.

**SQL**

**1. Given 2 Columns and 3 Rows of Data, Write Rows as Columns and Columns as Rows**

**To convert rows into columns and vice versa, we can use a pivot operation in SQL. Here's how you can do it:**

**SELECT MAX(CASE WHEN row\_num = 1 THEN column\_name END) AS column1,**

**MAX(CASE WHEN row\_num = 2 THEN column\_name END) AS column2,**

**MAX(CASE WHEN row\_num = 3 THEN column\_name END) AS column3**

**FROM (SELECT 'column\_name' AS column\_name, ROW\_NUMBER() OVER() AS row\_num FROM table\_name) AS temp**

**GROUP BY row\_num;**

**2. If You Buy Products at a Store Like 4/5 Products, How Will You Find the Products Together?**

**To find products bought together, you could use a self-join to identify transactions that contain the same customer ID, and filter based on the order IDs:**

**SELECT a.product\_id, b.product\_id**

**FROM transactions a**

**JOIN transactions b ON a.order\_id = b.order\_id**

**WHERE a.product\_id != b.product\_id;**

**3. Difference Between Merge, Join, Concat()**

* **Merge: Primarily used in Pandas or similar frameworks for combining DataFrames based on a key (like a database join).**
* **Join: SQL operation to combine rows from two or more tables based on related columns.**
* **Concat(): SQL or programming function used to concatenate strings or join tables vertically or horizontally (like UNION or UNION ALL in SQL).**

**4. What Are SCDs? Explain Type 2 and 3**

* **SCD (Slowly Changing Dimensions): These are dimensions that change over time but in a way that doesn't require frequent updates.**
* **Type 2: Keeps historical data and tracks changes by adding new rows with a version indicator or date range.**
* **Type 3: Tracks only a limited history (e.g., the previous value and current value) within the same row.**

**5. In an Organization There is IN-time and OUT-time, and with One Name, 3 People Are There and It Should Not Be Duplicated?**

**To avoid duplication based on name and ensure unique entries, you could use DISTINCT or handle this via grouping:**

**SELECT DISTINCT name, intime, outtime**

**FROM employees;**

**6. SQL Normalization and Denormalization**

* **Normalization: The process of organizing data to reduce redundancy, typically using multiple related tables.**
* **Denormalization: The process of combining tables to increase performance, which might increase redundancy.**

**7. In SQL, How to Get Only the Domain from Email?**

**You can extract the domain using SUBSTRING and CHARINDEX functions:**

**SELECT SUBSTRING(email, CHARINDEX('@', email) + 1, LEN(email)) AS domain**

**FROM users;**

**8. SQL All Types of Joins**

* **INNER JOIN: Returns only matching rows from both tables.**
* **LEFT JOIN: Returns all rows from the left table and matched rows from the right table.**
* **RIGHT JOIN: Returns all rows from the right table and matched rows from the left table.**
* **FULL OUTER JOIN: Returns all rows when there is a match in one of the tables.**
* **SELF JOIN: Joins a table with itself.**

**9. Write a SQL to Get the Output for the Given Data**

**To get the required output based on countries:**

**SELECT 'Team A' AS Team, Country**

**FROM countries**

**UNION ALL**

**SELECT 'Team B' AS Team, Country**

**FROM countries;**

**10. Write a SQL to Get the Given Output for the Customer Data**

**SELECT cust\_id,**

**MAX(CASE WHEN name = 'KYC' THEN 'Y' ELSE 'N' END) AS KYC,**

**MAX(CASE WHEN name = 'CIP' THEN 'Y' ELSE 'N' END) AS CIP**

**FROM customer\_table**

**GROUP BY cust\_id;**

**11. Extract Nested Fields Using JSON Functions**

**In SQL, to extract nested fields:**

**SELECT JSON\_VALUE(user\_info, '$.contact.email') AS email,**

**JSON\_VALUE(user\_info, '$.contact.address.city') AS city**

**FROM users;**

**12. SQL Query to Find Orders in Desc Order**

**Assuming a table orders:**

**SELECT order\_id, order\_date**

**FROM orders**

**ORDER BY order\_date DESC;**

**13. Write a Query to Remove Duplicates**

**To remove duplicates from a table:**

**SELECT DISTINCT \***

**FROM table\_name;**

**Or using a CTE with ROW\_NUMBER():**

**WITH cte AS (**

**SELECT \*, ROW\_NUMBER() OVER (PARTITION BY email ORDER BY id) AS row\_num**

**FROM table\_name**

**)**

**DELETE FROM cte WHERE row\_num > 1;**

**14. Get Customers Who Ordered in Both 2023 and 2024**

**SELECT custid**

**FROM orders**

**WHERE EXTRACT(YEAR FROM order\_date) = 2023**

**GROUP BY custid**

**HAVING COUNT(DISTINCT EXTRACT(YEAR FROM order\_date)) = 2;**

**15. Write a Query to Get Employees Whose Salary is Greater than the Avg Salary of the Department**

**SELECT emp\_id, emp\_name, salary**

**FROM employees e**

**WHERE salary > (SELECT AVG(salary) FROM employees WHERE dept\_id = e.dept\_id);**

**16. Delete Duplicates, Keep Only One Unique Email with Smallest ID**

**WITH cte AS (**

**SELECT \*, ROW\_NUMBER() OVER (PARTITION BY email ORDER BY id) AS row\_num**

**FROM customers**

**)**

**DELETE FROM cte WHERE row\_num > 1;**

**17. What is LEAD/LAG Function?**

* **LEAD: Returns the value of a row from the next row.**
* **LAG: Returns the value of a row from the previous row.**

**SELECT id, salary, LEAD(salary) OVER (ORDER BY salary) AS next\_salary**

**FROM employees;**

**18. Ways to Find Duplicates in Table**

**You can find duplicates by grouping the data:**

**SELECT column1, column2, COUNT(\*)**

**FROM table\_name**

**GROUP BY column1, column2**

**HAVING COUNT(\*) > 1;**

**19. How Do You Find Nulls Within a Table?**

**To find rows with NULL values:**

**SELECT \***

**FROM table\_name**

**WHERE column\_name IS NULL;**

**20. How Can We Replace NULL with Any Value?**

**To replace NULL with a default value, use COALESCE or IFNULL:**

**SELECT COALESCE(column\_name, 'default\_value') AS column\_name**

**FROM table\_name;**

**21. How Can We Get All Columns Within a Table?**

**You can use SELECT \* to get all columns:**

**SELECT \***

**FROM table\_name;**

**22. 3 SQL Queries in Programiz Editor Based on Tables Present in Editor**

**This is a more context-dependent question. You'll need to explore the tables available in the Programiz editor and build queries based on that.**

**23. How to Resolve Long-Running Query with Multiple Joins?**

* **Optimize the query by ensuring proper indexing.**
* **Use EXPLAIN to understand the query plan.**
* **Break the query into smaller subqueries or use CTEs.**
* **Consider partitioning large tables if needed.**

**24. SQL Query to Find Top 3 Employees from Each Department and Handle Less Than 3 Employees**

**WITH ranked\_employees AS (**

**SELECT \*, ROW\_NUMBER() OVER (PARTITION BY dept\_id ORDER BY salary DESC) AS rank**

**FROM employees**

**)**

**SELECT emp\_id, emp\_name, dept\_id, salary**

**FROM ranked\_employees**

**WHERE rank <= 3;**

**25. Group By and Order By – Do They Slow Down Queries?**

* **GROUP BY: Aggregates data, which can slow down queries when working with large datasets.**
* **ORDER BY: Sorts data, which also impacts performance due to sorting complexity.**

**26. Tables You Can Create for a Retail Store**

* **Customers: customer\_id, name, email, phone, address.**
* **Orders: order\_id, customer\_id, order\_date, total\_amount.**
* **Products: product\_id, name, category, price.**
* **Order\_Items: order\_id, product\_id, quantity, price.**

**27. How Do You Update a Customer Email in the Table?**

**UPDATE customers**

**SET email = 'new\_email@example.com'**

**WHERE customer\_id = 123;**

**28. Joins Types – Inner Join, Left Join & Self Join**

* **Inner Join: Returns only matching rows from both tables.**
* **Left Join: Returns all rows from the left table and matching rows from the right table.**
* **Self Join: Joins a table with itself, useful for hierarchical data.**

**29. Difference Between Drop, Delete, Truncate**

* **DROP: Removes the entire table or database.**
* **DELETE: Removes specific rows but keeps the structure.**
* **TRUNCATE: Removes all rows and resets the table.**

**30. Use Case for Window Functions**

**Window functions are used to perform calculations across a set of table rows related to the current row, like calculating running totals or rankings.**

**SELECT emp\_id, salary,**

**RANK() OVER (ORDER BY salary DESC) AS rank**

**FROM employees;**

**31. How to Migrate Data to the Cheapest Cost from On-Premises (Teradata/SQL Server)?**

**You can use cloud services like AWS S3, Google Cloud Storage (GCS), or Azure Blob Storage for inexpensive data transfer. Tools like AWS Database Migration Service (DMS), Google Cloud Data Transfer, or Azure Data Factory can assist in the migration.**

**32. Student Copy Status Based on Marks**

**SELECT student, marks,**

**CASE**

**WHEN marks = LEAD(marks) OVER (ORDER BY student) OR marks = LAG(marks) OVER (ORDER BY student)**

**THEN 'Copied'**

**ELSE 'Not Copied'**

**END AS message**

**FROM students;**

**33. Write SQL for Positive and Negative Values**

**SELECT**

**STRING\_AGG(CASE WHEN value > 0 THEN value END, ',') AS Positive,**

**STRING\_AGG(CASE WHEN value < 0 THEN value END, ',') AS Negative**

**FROM table\_name;**

**34. Rank and DenseRank**

* **RANK(): Assigns ranks with gaps in case of ties.**
* **DENSE\_RANK(): Assigns ranks without gaps.**

**SELECT emp\_id, salary,**

**RANK() OVER (ORDER BY salary DESC) AS rank,**

**DENSE\_RANK() OVER (ORDER BY salary DESC) AS dense\_rank**

**FROM employees;**

**BIG QUERRY**

**1. Difference between Merge, Join, Concat()**

* **Merge**: Combines two datasets based on a common column (like SQL JOIN but in data processing tools like Pandas).
* **Join**: Combines rows from two or more tables based on a common column or condition (INNER, LEFT, RIGHT, OUTER).
* **Concat()**: Combines multiple strings into one, or stacks tables/rows together vertically (like UNION in SQL).

**2. How to Handle Huge Data in BigQuery?**

When dealing with huge data in BigQuery, it's important to optimize queries by using partitioning, clustering, and avoiding SELECT \* queries. You can also use streaming inserts for real-time data and batch processing for large datasets. Proper use of caching and optimizations like materialized views can improve query performance.

**3. From Source History Data is Coming and Adding in a Table Row by Row, What Will You Use?**

For adding historical data row by row, you would likely use **streaming inserts** in BigQuery. You can use the bq load command or an API to stream data efficiently into the table.

**4. What is BigQuery?**

BigQuery is a fully managed, serverless data warehouse solution from Google Cloud that enables super-fast SQL queries using the processing power of Google's infrastructure. It supports large-scale data analytics with SQL-like syntax, and it integrates seamlessly with other Google Cloud services.

**5. What is ETL/ELT Process?**

* **ETL (Extract, Transform, Load)**: Data is extracted from various sources, transformed into a usable format, and then loaded into a data warehouse.
* **ELT (Extract, Load, Transform)**: Data is first extracted and loaded into the warehouse, then transformed as needed.

**6. Partition by Date, Month, Day, Year**

Partitioning allows data to be stored in separate segments for efficient querying. In BigQuery, you can partition tables by **date**, **month**, **day**, or **year**, which helps to improve performance by limiting the amount of data scanned during queries.

**7. Types of Index**

BigQuery doesn't have traditional indexes, but it supports **partitioning** (dividing tables into smaller, manageable chunks) and **clustering** (grouping rows with similar values to optimize query performance).

**8. One Database to Another Database**

To migrate data between databases, you can use tools like **Google Cloud Data Transfer Service**, **BigQuery Data Transfer Service**, or **ETL tools** to extract data from the source database, transform it as needed, and load it into BigQuery.

**9. From Local Desktop to GCS (How to Load Data Fast)**

You can upload data to Google Cloud Storage (GCS) using **gsutil** or the **Cloud Console**. For faster data transfer, use **parallel uploads**, optimize file size (e.g., by splitting large files), and use **Google Cloud Storage Transfer Service** for large-scale transfers.

**10. Local Desktop Moving Commands Speed**

Using **gsutil** for moving files from a local machine to GCS is a fast and reliable method. Example command:

gsutil -m cp -r /local/folder gs://your-bucket/

The -m flag allows parallel operations to speed up the process.

**11. Batch Data, Streaming Data, and GCS Data to Load into BigQuery**

* **Batch Data**: Use scheduled data loads from GCS using bq load for large files.
* **Streaming Data**: Use the BigQuery streaming API to send data in real time.
* **GCS Data**: Load data from files (CSV, JSON, Avro, Parquet) stored in GCS using bq load.

**12. Loading Data into BigQuery from GCS**

BigQuery uses **bq load** to load data into tables from GCS. You can specify file format (CSV, JSON, Parquet) and partitioning details while loading the data.

**13. Difference Between History Data vs Batch Data**

* **Historical Data**: Data accumulated over time that shows a trend or past information.
* **Batch Data**: Data collected and processed at scheduled intervals, typically in large volumes.

**14. Optimize Techniques in BigQuery**

To optimize BigQuery:

* Use **partitioning** and **clustering**.
* Use **materialized views** for frequently queried data.
* **Avoid SELECT \* queries**, only select needed columns.
* Use **sharded tables** for large datasets.
* Use **data compression formats** like Parquet for faster queries.

**15. How to Get the First and Last punch of Employees**

You can use **window functions** to get the first and last punch time for each employee:

SELECT employee\_id, MIN(punch\_time) AS first\_punch, MAX(punch\_time) AS last\_punch

FROM punch\_data

GROUP BY employee\_id;

**16. How to Handle Metadata Differences Between On-Prem and BigQuery Before Loading into GCS**

Before loading data, ensure the schema in the source (on-prem) matches the BigQuery table schema. Use **data transformation** tools like **Cloud Dataflow** or **Apache Beam** to reconcile schema differences, or preprocess metadata in **Cloud Dataprep** before loading into GCS.

**17. How to Retrieve One Hour Back Data in BigQuery (Time Travel)**

BigQuery supports **time travel**, allowing you to query data from up to 7 days ago using the FOR SYSTEM\_TIME AS OF clause.

SELECT \* FROM my\_table FOR SYSTEM\_TIME AS OF TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 1 HOUR);

**18. What Kind of Data You Loaded into BigQuery**

Typically, structured data like **CSV**, **JSON**, **Avro**, and **Parquet** are loaded into BigQuery. Data can be loaded in batch mode from GCS or streamed real-time for immediate analysis.

**19. Downstream Needs and Requirements in Sprint Tasks**

Downstream teams may need clean, structured data for reporting, analytics, or ML models. I would ensure that the data pipeline delivers accurate, timely data and that it aligns with their requirements by focusing on reliable data ingestion, transformation, and scheduling.

**20. What is Partitioning and Clustering?**

* **Partitioning**: Divides large tables into smaller, more manageable chunks (e.g., by date) for faster querying.
* **Clustering**: Groups rows based on column values to optimize queries that filter or aggregate on those columns.

**21. How Many Columns Can We Apply Partitioning On?**

BigQuery allows partitioning based on **a single column** (e.g., date, timestamp). Multiple columns can be used for **clustering** to improve query performance.

**22. Handling 2TB File for Daily Load in BigQuery**

To handle large files, use **parallel processing** with **gsutil** or **Cloud Storage Transfer Service**. You can also use **data sharding** (splitting data into smaller chunks) and load in parallel to improve performance.

**23. Deploying Changes to Prod from Dev Environment**

Changes are typically deployed using a **CI/CD pipeline**. In BigQuery, you can use **Terraform** or **Cloud Deployment Manager** for infrastructure as code (IaC). Ensure that all changes are tested in the dev environment before pushing them to production.

**24. Optimizing Queries in BigQuery**

To optimize queries, you can:

* Use **partitioned tables**.
* Use **clustering** for frequently queried columns.
* **Avoid cross joins** and large **cartesian products**.
* Utilize **materialized views** for recurring queries.

**25. Asked to Filter Above Fields to Calculate Sum Over Department**

You can use a **GROUP BY** query to calculate the sum for each department:

SELECT department\_id, SUM(salary)

FROM employees

GROUP BY department\_id;

**26. Asked About Merge**

The **MERGE** statement in BigQuery is used to perform **upserts** (insert or update) based on matching conditions:

MERGE INTO target\_table t

USING source\_table s

ON t.id = s.id

WHEN MATCHED THEN

UPDATE SET t.column = s.column

WHEN NOT MATCHED THEN

INSERT (column) VALUES (value);

**27. How to Replace Null Values with Any Value**

Use **COALESCE** or **IFNULL** to replace NULL values:

SELECT COALESCE(column\_name, 'default\_value') AS column\_name FROM table\_name;

**28. Data Warehouse vs Data Lake**

* **Data Warehouse**: A structured environment where data is stored in tables, optimized for reporting and analytics.
* **Data Lake**: A storage repository that holds raw, unstructured data in its native format for future processing.

**29. Fact Tables and Dimension Tables**

* **Fact Tables**: Store quantitative data (e.g., sales, transactions) and typically have foreign keys referencing dimension tables.
* **Dimension Tables**: Store descriptive data (e.g., customers, products) and are used for filtering and grouping in queries.

**30. Examples of Fact Tables**

An example of a **fact table** is a **sales** table with columns like order\_id, product\_id, quantity\_sold, sale\_amount.

**31. Star Schema and Snowflake Schema**

* **Star Schema**: A central fact table surrounded by dimension tables.
* **Snowflake Schema**: A normalized version of the star schema where dimension tables are split into additional tables.

**32. Involved in Data Modeling?**

Yes, I've been involved in designing **star** and **snowflake** schemas, creating **fact** and **dimension** tables, and ensuring the model aligns with business needs for reporting and analytics.

**33. Difference Between Star and Snowflake Schema**

* **Star Schema**: Denormalized, with a central fact table and surrounding dimension tables.
* **Snowflake Schema**: Normalized dimension tables, with multiple related sub-dimension tables.

**34. If You Are Keep on Retrying, What Would You Do?**

If retries are continuously failing, I would:

* Check logs to identify the root cause.
* Examine the data pipeline for errors.
* Look into retry configurations and error handling.
* If needed, isolate the failing process and rerun with data corrections.

**35. If Any Task Was Running Slow, What Would You Do?**

I would:

* Analyze the query execution plan with EXPLAIN.
* Check for any inefficient joins, large result sets, or missing indexes.
* Optimize the query by using partitioning, clustering, and materialized views.

**36. What Challenges Have You Faced?**

Common challenges include data inconsistencies, performance issues with large datasets, and maintaining the quality of ETL pipelines. These were resolved by implementing better data validation, partitioning strategies, and more efficient query optimization techniques.

**37. How Do You Manage Challenges?**

I prioritize identifying the root cause, discussing with the team if needed, and applying best practices such as optimizing data pipelines, ensuring data quality, and automating monitoring processes.

**38. Write All Data Types with One Element in It**

Example:

* **STRING**: 'Hello'
* **INTEGER**: 100
* **FLOAT**: 10.5
* **BOOLEAN**: TRUE
* **DATE**: '2022-01-01'
* **TIMESTAMP**: '2022-01-01 10:00:00'

**39. Slow Running Query Optimization**

To optimize a slow running query:

* Use **partitioning** and **clustering**.
* Avoid \*\*SELECT \*\*\*; select only the necessary columns.
* Optimize joins by ensuring appropriate indexes and keys.
* Use **materialized views** for frequently accessed queries.

**40. Data Validation and Data Ingestion**

Yes, I have worked on data validation (ensuring data integrity) and **data ingestion** using BigQuery’s data loading tools for batch and real-time streaming data.

**41. Snapshot Timetravel**

BigQuery allows you to query data from a point in time (up to 7 days ago) using the **time travel** feature.

**42. Partitioning on Which Data Types?**

Partitioning can be done on **DATE**, **TIMESTAMP**, and **DATETIME** columns in BigQuery.

**43. Clustering Limit**

Clustering can be applied to **up to 4 columns** in BigQuery.

**44. Handling Large CSV Files for Storage**

You can compress large CSV files into **gzip** or **bzip2** formats for faster loading into BigQuery via **Cloud Storage**.

**45. \*\*Using ISNULL and COALES**

CE\*\* You can replace NULL values using **ISNULL** or **COALESCE** functions in BigQuery:

SELECT COALESCE(column\_name, 'default\_value') FROM table\_name;

**46. Find Female and Male Data into Single Row Output**

You can use conditional aggregation:

SELECT

department\_id,

SUM(CASE WHEN gender = 'Female' THEN 1 ELSE 0 END) AS female\_count,

SUM(CASE WHEN gender = 'Male' THEN 1 ELSE 0 END) AS male\_count

FROM employees

GROUP BY department\_id;

**47. RDBMS vs BigQuery**

* **RDBMS**: Relational databases, typically used for transactional workloads, with rigid schema and ACID properties.
* **BigQuery**: A distributed data warehouse optimized for analytics and large-scale querying, but without ACID transactions.

**48. Create CDR Calls Table in BigQuery**

To create the **cdr\_calls** table in BigQuery, first define the schema based on the Parquet format data in the GCS bucket, then load it into BigQuery:

CREATE TABLE telco\_data.cdr\_calls

PARTITION BY event\_date

CLUSTER BY caller\_id, recipient\_id;

**49. Load 7 Days Data from GCS**

To load only the last 7 days of data from GCS:

bq load --source\_format=PARQUET --replace --autodetect telco\_data.cdr\_calls gs://your-hcs-bucket/cdr\_data/\*.parquet

**50. Partitioned by Event Date**

Partitioning ensures better performance for time-based queries:

CREATE TABLE telco\_data.cdr\_calls

PARTITION BY event\_date

AS SELECT \* FROM gs://your-hcs-bucket/cdr\_data/\*.parquet;

**51. Enable Schema Evolution**

BigQuery supports **schema evolution**, meaning it can handle changes in the schema like new columns. You can use the --schema\_update\_option flag in bq load to allow automatic schema updates.

**52. Enforce Partition Filtering**

To avoid full-table scans, always include a **WHERE clause** that filters on the partitioned column.

**53. Snapshot Timetravel to Get Last Hour Data**

Use **time travel** to retrieve data from the last hour:

SELECT \* FROM my\_table

FOR SYSTEM\_TIME AS OF TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 1 HOUR);

**54. Create a BigQuery Table Following SCD Types**

Use **MERGE** or **INSERT** to handle Slowly Changing Dimensions (SCD). For example, using the MERGE statement:

MERGE INTO destination\_table d

USING source\_table s

ON d.id = s.id

WHEN MATCHED THEN

UPDATE SET d.column = s.column

WHEN NOT MATCHED THEN

INSERT (columns) VALUES (values);

DATAPROC AND DATAFLOW :

**1. What is Dataflow?**

**Dataflow** is a fully managed service on Google Cloud for processing real-time and batch data using Apache Beam. It allows you to create data pipelines that can scale and handle massive amounts of data. Dataflow automatically handles resource management, providing efficient, scalable data processing.

**2. What is Dataproc?**

**Dataproc** is a fast, fully managed cloud service for running Apache Spark and Hadoop clusters in Google Cloud. It is designed for processing big data workloads like batch jobs, streaming, and machine learning, leveraging the power of Spark, Hadoop, and other big data technologies.

**3. What do you do in Dataflow?**

In **Dataflow**, you design, build, and execute data pipelines. This involves:

* Defining the source of data (like GCS, BigQuery, etc.)
* Transforming data using **Apache Beam** APIs
* Writing results back to destinations like **BigQuery**, **Cloud Storage**, or **Pub/Sub**
* Handling real-time streaming or batch processing tasks

**4. Diff Between Airflow and Composer?**

* **Airflow**: An open-source platform for orchestrating workflows. It allows you to create directed acyclic graphs (DAGs) to manage task execution workflows.
* **Cloud Composer**: A fully managed version of Airflow provided by Google Cloud. It helps you run and monitor Airflow DAGs at scale without managing the infrastructure.

**5. What do you do in Composer?**

In **Cloud Composer**, you:

* Create and manage **Airflow DAGs** (Directed Acyclic Graphs)
* Schedule and monitor workflows, from data extraction to transformation and loading (ETL)
* Use built-in operators for integration with various services like **BigQuery**, **Pub/Sub**, **Cloud Storage**, etc.
* Handle dependencies, retries, and monitoring for all tasks

**6. What Are All the Libraries You Have Used in Airflow DAG Code?**

Some common libraries used in **Airflow DAGs**:

* airflow.operators: For task definitions (e.g., BashOperator, PythonOperator)
* airflow.providers.google.cloud: For Google Cloud-specific operators (e.g., BigQueryOperator, DataflowTemplateOperator)
* pandas: For working with data frames
* datetime: For handling timestamps and scheduling
* requests: For HTTP requests in DAG tasks

**7. What is Airflow/Composer?**

* **Airflow**: An open-source workflow management platform used to schedule and monitor workflows in Python. It allows defining workflows as code using **DAGs**.
* **Cloud Composer**: A managed service for running Airflow workflows in the cloud. It automates infrastructure management, scaling, and scheduling, making it easier to use Airflow at scale.

**8. How Will You Connect to Airflow, Where Is It Used in Your Project?**

To connect to **Airflow**:

* Use the Airflow UI to monitor and trigger DAGs.
* You can also use the **Airflow REST API** for programmatically triggering and managing DAGs.
* Airflow is typically used for automating ETL tasks, data processing workflows, and integration between different cloud services.

**9. What Is Pub/Sub, How Are You Using It, and Can You Share Some Codes?**

**Pub/Sub** is a messaging service for building event-driven systems. It allows sending and receiving messages between independent applications.

* You can publish messages from one service (e.g., Dataflow) and consume them in another service (e.g., Cloud Functions, BigQuery).

Example Pub/Sub code to publish and subscribe:

# Publishing a message to Pub/Sub

from google.cloud import pubsub\_v1

publisher = pubsub\_v1.PublisherClient()

topic\_name = 'projects/your-project-id/topics/your-topic'

message\_data = b"Hello, Pub/Sub!"

publisher.publish(topic\_name, message\_data)

# Subscribing to a Pub/Sub topic

from google.cloud import pubsub\_v1

subscriber = pubsub\_v1.SubscriberClient()

subscription\_name = 'projects/your-project-id/subscriptions/your-subscription'

def callback(message):

print(f"Received message: {message.data}")

message.ack()

subscriber.subscribe(subscription\_name, callback=callback)

**10. What is Dataflow (Repeated)?**

Refer to the previous answer (Q1). Dataflow is a fully managed service for building scalable data pipelines using Apache Beam.

**11. What Are the Operators, and How Is the Sensor Operator Used When It Is Highly Costly?**

* **Operators**: Operators in Airflow define the tasks to be executed. Examples include PythonOperator, BashOperator, BigQueryOperator, DataflowOperator, etc.
* **Sensor Operators**: Used for waiting until a certain condition is met (e.g., FileSensor, ExternalTaskSensor).

**Handling Cost of Sensors**: Use **poke\_interval** and **timeout** to optimize the cost. Set appropriate intervals to check for the condition rather than continuously polling, and ensure the timeout doesn’t last too long.

**12. Can We Use Only Clustering in the Project?**

Yes, you can use **clustering** in BigQuery alone to improve query performance, especially for large tables. Clustering organizes data within partitions based on columns that you often query against. However, it may be less efficient without **partitioning** for very large datasets.

**13. Airflow catchup True or False?**

* **catchup=True**: Airflow will run missed DAG runs when it starts after the defined schedule.
* **catchup=False**: Airflow will only run the DAG for the current schedule, skipping any missed runs.

**14. Airflow Parameters**

Some common **Airflow parameters** include:

* schedule\_interval: How often to run the DAG.
* catchup: Whether to run missed DAG executions.
* retries: Number of retries before failing a task.
* retry\_delay: Delay between retries.

**15. Data Pipelines**

A **data pipeline** is a set of processes that move data from one or more sources to a destination (e.g., BigQuery, Dataflow). It typically involves extraction, transformation, and loading (ETL) tasks, and can be automated using tools like **Airflow**, **Dataflow**, and **Composer**.

**16. How to Handle Pipelines**

* Monitor pipeline performance using built-in **logging** and **monitoring**.
* Handle failures with retries or alternate workflows.
* Ensure that **data quality checks** are in place to prevent data corruption.

**17. How to Perform Pipelines?**

To perform data pipelines:

* Define tasks (e.g., using Airflow’s DAG).
* Set dependencies and triggers.
* Schedule the pipeline and monitor progress through the UI or programmatically.

**18. How Do You Build a Pipeline?**

You can build a pipeline by:

1. Defining the workflow using **Airflow DAGs**.
2. Configuring sources and destinations (e.g., **Cloud Storage**, **BigQuery**).
3. Adding operators (e.g., **DataflowOperator**, **PythonOperator**).
4. Setting dependencies between tasks and scheduling them to run.

**19. Scenario-Based Questions**

For example:

* **Scenario**: You need to process data from GCS, transform it, and load it into BigQuery.
* **Solution**: Create a pipeline using **Dataflow** to read data from GCS, perform transformations, and then load the results into BigQuery.

**20. Airflow January 24, 2024 - How Do You Trigger Automatic Pipelines?**

You can trigger pipelines automatically by setting the **schedule\_interval** in Airflow to your desired frequency or using **external triggers** (e.g., Pub/Sub messages) to start DAG runs when needed.

**21. How Will You Deploy the Pipeline?**

To deploy an Airflow pipeline, you:

* Write and test your DAG locally.
* Push your DAG files to **Cloud Composer**.
* Trigger the pipeline either manually or automatically based on the schedule.

**22. Querying How You Build a Pipeline**

To query and process data within a pipeline, you can use BigQuery operators in Airflow to trigger SQL queries, or use **PythonOperator** to execute SQL or Python code for querying.

**23. Which Operators Used in 2 DAGs?**

Common operators used in DAGs:

* **PythonOperator**: To run Python code.
* **BashOperator**: To run shell commands.
* **BigQueryOperator**: To run SQL queries in BigQuery.
* **DataflowTemplateOperator**: For running Dataflow jobs.
* **EmailOperator**: To send emails for notifications.

**24. Trigger Rule Operator**

The **Trigger Rule Operator** is used in Airflow to define how tasks are triggered based on the status of upstream tasks. Examples include:

* **all\_success**: Trigger if all upstream tasks succeed.
* **one\_failed**: Trigger if any upstream task fails.

**25. What Is Dataflow (Repeated)?**

Refer to the previous answer (Q1). Dataflow is a fully managed, serverless data processing service.

**26. Explain Real-Time End-to-End Pipeline**

A real-time pipeline involves processing data as it arrives. For example, using **Pub/Sub** to stream data into **Dataflow**, where transformations are applied, and results are written to **BigQuery** or other storage services in near real-time.

**27. What Is Your Experience with Dataflow and Dataproc?**

You could describe any experience where you:

* Built scalable pipelines using **Dataflow** for both batch and stream processing.
* Managed Hadoop/Spark clusters using **Dataproc** for distributed data processing.

**28. Write One Flow on Dataproc**

An example of a flow in **Dataproc** could be running a Spark job on a cluster to process data from **Cloud Storage** and store the result back in **BigQuery**.

**29. How Did You Schedule the Pipeline on Cloud Composer?**

In **Cloud Composer**, you can schedule a pipeline by defining a DAG and setting the **schedule\_interval** parameter to control when it runs (e.g., daily, weekly).

**30. What Is XCom Operator and Platform You Use to Write?**

**XCom** is a feature in Airflow that allows tasks to exchange messages or small data (such as the result of a task) with each other. You use it to pass data between tasks in a DAG.

**31. Write One DAG**

Here’s a simple DAG example:

from airflow import DAG

from airflow.operators.dummy\_operator import DummyOperator

from datetime import datetime

default\_args = {

'owner': 'airflow',

'start\_date': datetime(2023, 1, 1),

'retries': 1

}

dag = DAG('simple\_dag', default\_args=default\_args, schedule\_interval='@daily')

start = DummyOperator(task\_id='start', dag=dag)

end = DummyOperator(task\_id='end', dag=dag)

start >> end

**32. If One of the Tasks in Your DAG Needs to Be Held, What's Your Approach?**

You can hold a task in Airflow by setting **task dependencies** to make other tasks wait until certain conditions are met, or you can use **Sensors** with specific conditions.

**33. What Operators Have You Used in DAGS?**

I have used several operators, including:

* **PythonOperator** for running Python functions.
* **BigQueryOperator** for executing queries in BigQuery.
* **BashOperator** for running shell commands.
* **DataflowOperator** for managing Dataflow jobs.

**34. What Is a Dynamic DAG?**

A **dynamic DAG** is one where tasks or entire sections of the workflow are created dynamically at runtime, often using loops or conditional logic in the DAG definition.

**35. What Commands Did You Use in BashOperator?**

The **BashOperator** is used to execute shell commands. Example:

from airflow.operators.bash\_operator import BashOperator

bash\_task = BashOperator(

task\_id='run\_bash\_command',

bash\_command='echo "Hello World"',

dag=dag

)

**36. What Are the Different Operators You Have Used in Airflow?**

I have used

operators like:

* **PythonOperator**: To run Python code.
* **BigQueryOperator**: To run BigQuery queries.
* **BashOperator**: To run shell commands.
* **DataflowOperator**: To trigger Dataflow jobs.
* **EmailOperator**: To send emails.

**SQL**

**1.Write a SQL query to find the second highest salary from a table by dept wise**

**2.Find the number of users who logged in more than once in a day from user\_logs table**

**3.In BigQuery, how do you create a partitioned table and why use partitioning?**

**4.Get the top 3 products by sales in each category (Table: sales, columns: category, product, amount)**

**5..**

**🍽️ Zomato Orders Dataset (Example)**

**| order\\_id | customer\\_id | order\\_date | order\\_amount |**

**| --------- | ------------ | ----------- | ------------- |**

**| 101 | 1 | 2024-05-01 | 500 |**

**| 102 | 1 | 2024-05-10 | 700 |**

**| 103 | 1 | 2024-05-20 | 600 |**

**| 104 | 2 | 2024-05-02 | 400 |**

**| 105 | 2 | 2024-05-15 | 900 |**

**select**

**custumer\_id,**

**order\_date,**

**order\\_amount,**

**LAG(order\_amount) over(partion by customer\\_id order by order\\_date) as previous\_order**

**from zomato\_orders;**

**🔎 Output:**

**| customer\\_id | order\\_id | order\\_date | order\\_amount | previous\\_order |**

**| ------------ | --------- | ----------- | ------------- | --------------- |**

**| 1 | 101 | 2024-05-01 | 500 | NULL |**

**| 1 | 102 | 2024-05-10 | 700 | 500 |**

**| 1 | 103 | 2024-05-20 | 600 | 700 |**

**| 2 | 104 | 2024-05-02 | 400 | NULL |**

**| 2 | 105 | 2024-05-15 | 900 | 400 |**

**SELECT**

**customer\_id,**

**order\_id,**

**order\_date,**

**order\_amount,**

**LEAD(order\_amount) OVER (PARTITION BY customer\_id ORDER BY order\_date) AS next\_order**

**FROM zomato\_orders;**

**🔎 Output:**

**| customer\\_id | order\\_id | order\\_date | order\\_amount | next\\_order |**

**| ------------ | --------- | ----------- | ------------- | ----------- |**

**| 1 | 101 | 2024-05-01 | 500 | 700 |**

**| 1 | 102 | 2024-05-10 | 700 | 600 |**

**| 1 | 103 | 2024-05-20 | 600 | NULL |**

**| 2 | 104 | 2024-05-02 | 400 | 900 |**

**| 2 | 105 | 2024-05-15 | 900 | NULL |**

**6.| order\\_id | customer\\_id | order\\_date | item\\_name | quantity |**

**| --------- | ------------ | ----------- | ---------- | -------- |**

**| 1001 | 1 | 2025-05-29 | Biryani | 2 |**

**| 1002 | 2 | 2025-05-29 | Pizza | 1 |**

**| 1003 | 1 | 2025-05-30 | Biryani | 1 |**

**| 1004 | 3 | 2025-05-30 | Burger | 1 |**

**How many total orders were done between 29-May-2025 and 30-May-2025?**

**select count(\*) as total\_count**

**from zomato\_orders**

**where order\\_date between '2025-05-29' and '2025-05-30';**

**7.How many Biryani items were sold during the same period?**

**select sum(quantity) as Biryani\_sold**

**from zomato\_orders**

**where order\\_date between '2025-05-29' and '2025-05-30'**

**and item\\_name = 'biryani';**