1. How do you optimize queries in BigQuery?

Answer:
I optimize BigQuery queries by using partitioning and clustering on large tables, which helps reduce the amount of data scanned.
Avoid using Select *, select only the necessary columns.
Use materialized views for frequent access data (only for single table).
Break down complex queries using CTEs and use approximate.

2.What are the benefits of partitioning and clustering in BigQuery? Give an example where you used them.
Answer:
Partitioning – Divides a table into segments (partitions) based on a column (usually date or timestamp).
Clustering – Organizes the data within each partition (or whole table if no partitioning) based on one or more columns**.**
Benefits of Partitioning

1. **Improved Performance**: Queries only scan relevant partitions, not the entire table.
2. **Cost Efficiency**: You are billed only for the data scanned; fewer partitions scanned = lower cost.
3. **Faster Queries**: Especially useful for time-based data (e.g., logs, transactions).

Benefits of Clustering

1. **Efficient Filtering and Joins**: Queries with filters or joins on clustered columns are faster.
2. **Automatic Data Organization**: BigQuery physically stores related rows together.
3. **Improved Query Caching**: Clustered data leads to higher cache hits.

Example from My Project

In the Customer Data Analytics project for Mulberry, I implemented partitioning and clustering in BigQuery to optimize a large sales_transactions table (~1 billion rows)
What I did:

Partitioned the table by transaction_date
Clustered it by product_id and store_id

3. Can you explain your data pipeline architecture using GCS, BigQuery, and Airflow?

Answer:
We followed a layered architecture:

1. **Raw data** from source systems landed in GCS (via CSV/JSON).
2. Airflow DAGs triggered ingestion jobs that loaded this data into **raw tables** in BigQuery.
3. Transformations were performed in **staging and history layers** using SQL scripts.
4. **Exposition/authorized views** were created for business teams.
5. Airflow handled **scheduling**, dependency management, and failure alerts.

4. What is the difference between authorized views and regular views in BigQuery?

Answer:
An **authorized view** allows sharing of a dataset without giving users direct access to the underlying tables. It restricts access to only the data exposed through the view, making it ideal for **data security and role-based access control**.

5. How did you implement data quality checks in your pipeline?

Answer:
We wrote data validation queries to compare record counts, null ratios, and duplicates between staging and history layers. We generated discrepancy reports, which were reviewed daily.

6. How do you manage and automate data loading into BigQuery from GCS?

Answer:
To manage and automate data loading from Google Cloud Storage (GCS) into BigQuery, you typically use a combination of Airflow (or Cloud Composer), gsutil/gcloud commands, and BigQuery features like auto-detect, schema definition, and load jobs.

7. What transformation logic did you use between staging and history layers?

Answer:
We used the Data Ingestion Sheet to define mapping rules, data types, and business logic. Common transformations included:

1. Deduplication using ROW_NUMBER () in SQL
2. SCD Type 1/2 logic for tracking changes
3. Data type casting, date parsing, and lookups via joins with reference tables

8. How do you use Airflow (Composer) in your project?

Answer:
Airflow (on Composer) was used to:

1. Trigger data ingestion from GCS to BigQuery
2. Manage dependencies between raw → staging → history layers
3. Schedule daily/hourly pipelines
4. Send email alerts on failure
   We used BashOperator, PythonOperator, GCSOperator and BigQueryOperator extensively in our DAGs

9. What's the difference between external tables and native tables in BigQuery?

Answer:

Feature	External Table	Native Table (Managed Table)
Data Storage	Data is stored outside BigQuery (e.g., in GCS)	Data is stored inside BigQuery
Performance	Slower query performance (especially for large files)	Faster queries (optimized for BigQuery engine)
Cost	You're charged per query only, based on bytes read	Charged for both storage and query processing
Supports DML	❌ No	✅ Yes
Partitioning & Clustering	❌ Not supported directly (can simulate)	✅ Fully supported
Schema Updates	Limited	Fully supported
File Format Examples	CSV, JSON, Parquet, Avro in GCS	Table data stored natively in BigQuery

What is an External Table?
An external table lets you query data directly from GCS without loading it into BigQuery.
Used when:

1. You want to analyze data without importing
2. You're working with raw files
3. You're minimizing storage cost

What is a Native Table?
A native (or managed) table stores data inside BigQuery. This is the most common type.
Used when:

1. You need high-performance analytics
2. You're doing ETL or data modelling
3. You want partitioning, clustering, DML, etc.

10.Explain how you handle schema evolution in BigQuery when loading new data?

Answer:

Schema evolution refers to handling changes in the structure of incoming data — such as new columns, missing fields, or changes in data types — without breaking your data pipeline.

Common Schema Changes You May Face

1. New columns added to the data
2. Fields arriving in a different order
3. Optional fields missing
4. Data type mismatches

11.What is BigQuery and its features?

Answer:

BigQuery is Google Cloud's **serverless, highly scalable, and cost-effective** data warehouse designed for fast SQL analytics on large datasets. Its handle structure and semi-structure data (like JSON).

Features:

1.**Fullymanaged and Serverless**: No infrastructure to manage — Google handles provisioning, scaling, and maintenance. You just run queries.

2. **Standard SQL Support**: SQL with advanced features like window functions, arrays, structs, etc.

3.**Fast SQL Queries on Huge Dataset:** Can process petabytes of data quickly in sec to min using Dremel (Google internal Technology.

4.**Cost Effective:** Compute and storage are billed separately, Pay per Query and Flat pricing options.

5.**Built in Machine Learning:** Train and deploy ML model directly using SQL – no need to move data.

6.**Real Time Analytics:** Supports streaming data ingestion for real-time dashboards and monitoring.

7.**Partitioning & Clustering:** Optimize query performance and reduce costs by scanning only relevant data.

8**.Data Security & Governance:** Supports IAM, column-level security, row-level access policies, and full audit logging.

Uses:

1.Data Warehousing

2.BI Reporting

3.ETL/ELT Pipelines

4.Streaming Analytics

12. BigQuery Architecture?

Answer:

Architecture Workflow:

(a) Storage Layer (Colossus): Columnar Storage format and supports partition and clustering.

(b) Data Ingestion: Batch data (CSV, GCS), Streaming Data (Pub/Sub) and Federated Data (Sheets).

(c) Dremel Compute Engine: Serverless query execution.

Dremel uses multi-level tree architecture.

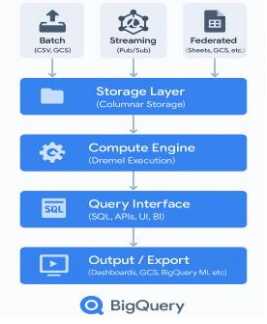
-**Root Server**: Co-ordinate the job.

-**Intermediate Server**: Break the task into smaller units.

-**Leaf Server**:(as called as Slots or Workers) Scan the actual data.

(d) Query Execution: Users submit SQL via Console / bq CLI / API / Airflow / Looker Studio.

(e) Result Delivery: Results returned in the interface or exported to GCS, Dashboard, BigQuery ML.



13.What are the advantages of BigQuery?

Answer:

1.Serverless Architecture
2. High Performance (Dremel Engine)
3. Separation of Storage & Compute
4. Standard SQL Support
5. Real-time Analytics
6. Cost Efficiency
7. Easy Integration
8.Built-in Machine Learning (BigQuery ML)

14.How to do Data Validation in BigQuery?

Answer:

We wrote data validation queries to compare record counts, null ratios, and duplicates between staging and history layers. We generated discrepancy reports, which were reviewed daily.

1.Row Counts Comparison: Compare row counts between the source and destination tables to check if all records are loaded**.**

```
-- Source row count
    SELECT COUNT (*) FROM `project.dataset.source_table`;
-- Destination row count
    SELECT COUNT (*) FROM `project.dataset.target_table`;
```

2. Null Checks for Key Columns: Ensure required fields are not null.

```
SELECT COUNT (*) AS null_count
FROM `project.dataset.table`
WHERE customer_id IS NULL;
```

3. Data Type & Format Validation: Check if values are in expected format (e.g., valid dates, numeric ranges)

```
-- Invalid dates
    SELECT *
    FROM `project.dataset.table`
    WHERE SAFE.PARSE_DATE ('%Y-%m-%d', date_column) IS NULL;
-- Check numeric ranges
    SELECT *
    FROM `project.dataset.table`
    WHERE sales_amount < 0;
```

4. Unique Constraint Validation: Check for duplicate values in a supposed unique column (e.g., order_id)

```
SELECT order_id, COUNT (*) AS count
FROM `project.dataset.table`
GROUP BY order_id
HAVING count > 1;
```

5. Referential Integrity: Ensure foreign key relationships hold (e.g., every customer_id in orders exists in customers).

6. Data Quality Discrepancy Reports: Automate discrepancy checks and store results in validation tables or dashboards.

15.What is Data Ware House and Data Lake?

Answer:

Two core components of modern data architecture.

A **Data Warehouse** is a centralized system used to store **structured data** that is cleaned, transformed, and optimized for **reporting and analysis**.

Key Characteristics:

1. **Structured data only** (from relational databases, business apps, etc.)
2. Uses **schema-on-write** (define schema before storing data)

3. Optimized for **SQL queries.**
4. Examples: **BigQuery**, Snowflake, Amazon Redshift, Azure Synapse

A **Data Lake** is a centralized repository that allows you to store **raw, unprocessed data** of all types — structured, semi-structured, and unstructured.

Key Characteristics:

1. Supports **all data formats** (JSON, Parquet, images, videos, logs).
2. Uses **schema-on-read** (define schema when reading data)
3. Built on cheap, scalable storage (e.g., **Google Cloud Storage**, AWS S3, Asure Data Lake)
4. Ideal for **big data, ML, and exploratory analysis**

A **Lakehouse** combines both systems — storing raw data like a lake, but offering analytics like a warehouse.
Example: **BigQuery with GCS**, **Databricks Lakehouse.**

16.What is Fact table and Dimension table?

Answer:

A Fact Table contains measurable, quantitative data — like sales, revenue, or counts — and foreign keys that link to dimension tables.

Key Characteristics:

- Stores **facts** or **metrics**
- Usually contains **numeric values** for aggregation (e.g., SUM, AVG)
- Includes **foreign keys** to dimension tables
- Grows rapidly as new data is added

A Dimension Table contains descriptive attributes (context) about the business dimensions like customer, product.

Key Characteristics:

- Stores **descriptive** or **categorical data**
- Usually contains **text or dates**
- Has a **primary key** referenced by fact tables
- Changes slowly (Slowly Changing Dimensions)

17. BQ Command to Fetch Records from 100th Row?

Answer:

To fetch records starting from the 100th row in BigQuery, you can use the OFFSET clause.

```
SELECT *
FROM `project.dataset.table`
ORDER BY column_name
LIMIT 100
OFFSET 99;
```

18. What are the types of partitions in Data Warehouse?

Answer:

In a Data Warehouse, partitioning is a strategy to divide large tables into smaller, manageable pieces — improving performance and scalability.

Types of Partitioning in a Data Warehouse:

1. **Range Partitioning:** Divides data based on a continuous range of values (often dates or numbers).
2. **List Partitioning:** Divides data based on predefined lists of values (e.g., region, country, category).

19. What are disposition issues in BigQuery?

Answer: Disposition issues happen when BigQuery tries to write or create a table but runs into a conflict because of how you've told it to behave.

Write Disposition - What to do if the table already exists

- WRITE_TRUNCATE: Replace the table (delete and overwrite)
- WRITE_APPEND: Add data to the existing table

- WRITE_EMPTY: Error if the table already exists

Create Disposition – What to do if the table does NOT exist

- CREATE_IF_NEEDED: Create it if it's missing
- CREATE_NEVER: Error if the table doesn't exist

20. What are the tools used to move data into BigQuery?

Answer:

1. Google Cloud Storage (GCS) + BigQuery Load Jobs
2. Cloud Dataflow
3. Cloud Data Fusion
4. Apache Airflow / Cloud Composer
5. BigQuery Data Transfer Service
6. Third-Party ETL Tools (Fivetran, Stitch, Talend, Informatica, Matillion)
7. Manual Methods

21. Array and Struct Data Types in BigQuery?

Answer:

In BigQuery, ARRAY and STRUCT are powerful data types used to handle nested and repeated data — which is common in JSON-like formats or hierarchical datasets.

Why Use ARRAY & STRUCT

- Support **semi-structured or nested data** (like JSON)
- Avoids table joins for embedded data
- Reduces data duplication
- Great for hierarchical schemas (e.g., orders → items)

22. What is a view in BigQuery? And What is the difference between a view and a table in BigQuery?

Answer:

View: A saved SQL query that behaves like a virtual table.

A table stores data; a view does not store data — it runs the query each time it is accessed.

Yes, you can create nested views (a view on top of another view).

When would you use a materialized view instead of a standard view?

- When you want to improve performance and avoid recalculating data repeatedly.

What happens if a referenced table in a view is deleted?

- The view will fail with an error when queried.

23. what are the limitation on Materialized Views?

Answer:

1.Only Single Table Queries Allowed

2.You cannot use joins, subqueries, or UNIONs.

3.Limited SQL Support

Certain SQL features are **not supported**, including:

- DISTINCT
- HAVING
- WINDOW FUNCTIONS (e.g., ROW_NUMBER ())
- LIMIT and OFFSET

4.No User-Defined Functions (UDFs).

5. No User-Defined Functions (UDFs)

6. Materialized views **cannot write into another materialized view**.

7.They also cannot be used to **partition or cluster** further.

BigQuery Views - Comparison Table

Type	Storage	Performance	Use Case
Standard View	No	Real-time, slower for complex queries	Encapsulating query logic
Materialized View	Yes	Faster (precomputed results)	Frequently accessed aggregated data
Authorized View	No	Depends on the query	Security and access control
Temporary View	No	Session-based, lightweight	Intermediate query results
Dynamic View	No	Depends on policies and query	Row-level security enforcement

24. What is a Row-Level Access Policy in BigQuery?

Answer:

Row-level access policies in BigQuery let you control access to individual rows in a table based on conditions, typically user identity or session context. Instead of restricting access to an entire table or column, you can filter rows based on who is querying the data.

```
CREATE ROW ACCESS POLICY west_coast_policy
ON dataset.sales_data
GRANT TO ("user:manager_west@example.com")
FILTER USING (region = "West");
DROP ROW ACCESS POLICY west_coast_policy ON dataset.sales_data
DROP ALL ROW ACCESS POLICIES ON dataset.sales_data;
```

Key Points:

- You can create **multiple row-level policies** on a single table.
- Only rows that match the **FILTER USING** condition for the user are visible.
- Works well with **authorized views**, **data governance**, and **multi-tenant datasets**.

25. Is BigQuery supports primary and foreign key?

Answer:

BigQuery does not natively enforce primary keys or foreign key constraints like traditional relational databases (e.g., MySQL, PostgreSQL

Why doesn't BigQuery enforce them?

- BigQuery is designed for analytical (OLAP) workloads, not transactional (OLTP) use.
- It Favors scalability and performance over strict schema enforcement.

26.What is Data Retention and their features?

Answer:

In BigQuery and modern data warehouses, data retention is handled using features like Time Travel, Snapshots, and Failsafe

1. **BigQuery Snapshot:**

   -- A **snapshot table** is a **read-only copy** of a base table **as it was at a specific point in time**.

   --Snapshots can be useful for backups, historical analysis, and data auditing.

   To create a snapshot of a table:

   ```
   CREATE SNAPSHOT TABLE CDE_B10.snapshot_table_name
   CLONE CDE_B10.original_table_name
   OPTIONS (expiration_timestamp = TIMESTAMP '2025-01-01 00:00:00 UTC');
   ```

2. **Time Travel:**

   --Allows you to access data from the past, typically within a limited time window.

   --Useful for recovering from accidental data deletions or changes.

   --Default: **7 days**

   -- Can query or restore table data as it was **up to 7 days ago** using FOR SYSTEM_TIME AS OF

   ```
   SELECT *
   FROM CDE_B10.table_name
   FOR SYSTEM TIME AS OF TIMESTAMP_SUB (CURRENT_TIMESTAMP (), INTERVAL 1 DAY);
   ```

- -Snapshots: Regularly create snapshots of critical tables to safeguard against accidental data loss or corruption. Ensure snapshots have a reasonable expiration time to manage storage costs.

- -Time Travel: Leverage time travel to undo unintended changes by querying the state of a table at a previous point in time. Regularly review and manage access to minimize accidental data modifications.

27. What is MERGE in BigQuery?

Answer:

The MERGE statement in BigQuery is used to perform insert, update, or delete operations on a target table based on conditions in a source table

```
MERGE dataset.customers AS target
USING dataset.customer_updates AS source
MERGE dataset.customers AS target
USING dataset.customer_updates AS source
ON target.customer_id = source.customer_id
WHEN MATCHED THEN
 UPDATE SET target.email = source.email,
            target.status = source.status
WHEN NOT MATCHED THEN
 INSERT (customer_id, email, status)
VALUES (source.customer_id, source.email, source.status);
```

28. How is MERGE better than running separate insert/update/delete queries?

--Reduces the number of queries.
--Maintains atomicity (all operations happen together).
--Optimizes performance for large datasets

29. Difference between BigQuery and other RDBMS?

Answer:

Feature	BigQuery	Traditional RDBMS
Type	Serverless, cloud-based data warehouse	On-prem/cloud transactional database
Use Case	OLAP (Analytics – large-scale data processing)	OLTP (Transactional – inserts/updates)
Scalability	Automatically scales to petabytes	Limited by server/storage capacity
Infrastructure Management	Fully managed (no server provisioning)	Requires manual setup and tuning
Performance Optimization	Partitioning, clustering, columnar storage	Indexes, normalization, tuning required
Storage Type	Columnar storage (optimized for analytics)	Row-based storage (optimized for transactions)
Pricing Model	Pay-per-query or flat-rate	License or fixed server cost
Data Loading	GCS, API, UI, Dataflow, etc.	ETL tools or bulk loaders
Joins & Relationships	No enforced primary/foreign keys	Enforced constraints for data integrity
Backup & restore	Time Travel, Snapshot Tables	Manual backups or built-in tools