BigQuery Interview Questions

A Comprehensive Guide for Data Engineers

Prepared for Technical Interviews

August 12, 2025

Contents

1 Basic BigQuery Interview Questions

1.1 What is BigQuery, and how does it differ from traditional databases?

BigQuery is a fully managed, serverless data warehouse on Google Cloud designed for large-scale data analytics. It allows high-speed SQL queries on massive datasets without management of the infrastructure, and it enables users to focus on insights rather than maintenance.

Unlike traditional on-premises relational databases, which are typically row-based and constrained by hardware limitations, BigQuery is a cloud-native, columnar storage system that offers near-infinite scalability. Its distributed architecture and pay-as-you-go pricing model make it more efficient for handling analytical workloads than conventional databases.

1.2 What is a dataset in BigQuery?

A dataset, in BigQuery, is defined as the topmost container that organizes tables, views, and other resources. This paves the way for access control and locating the data. By structuring data efficiently, datasets ensure better query performance and access management, making them a fundamental component of BigQuery's architecture.

1.3 How do you load data into BigQuery?

BigQuery offers several data ingestion methods intended for different purposes.

Batch load mode is typically used to upload a large historical dataset into BigQuery by using the BigQuery web UI, the bq command-line tool, or API calls. Streaming ingestion enables real-time data processing by pushing individual records or small batches using the BigQuery Streaming API. BigQuery's data transfer service can simplify scheduled data imports from Google Cloud storage, Google Ads, and SaaS sources. For advanced ETL workflows, Google Cloud Dataflow and other pipeline tools facilitate seamless data movement into BigQuery. Considering the best suitable ingestion method depends on data volume, latency, and processing needs.

1.4 What data types does BigQuery support?

BigQuery supports a variety of data types, categorized into: Basic types: BOOL, INT64, FLOAT64, STRING, BYTES, DATE, DATETIME, TIME, TIMESTAMP Complex types: ARRAY, STRUCT, JSON Specialized types: NUMERIC, BIGNUMERIC, INTERVAL, GEOGRAPHY, RANGE Each data type has a defined logical storage size, which impacts query performance and cost. For example, STRING storage depends on UTF-8 encoded length, whereas ARRAY<INT64> requires 8 bytes per element. Understanding these types helps in optimizing queries and managing costs efficiently.

1.5 What are the key advantages of using BigQuery?

Using BigQuery services brings five main benefits over traditional self-managed solutions:

Scalability: All investments in hardware are initially postponed. It allows us to easily scale resources up or down, depending on business requirements. Flexibility: Infrastructure can be modified according to the business needs. High security: Automatic backups and disaster recovery features are default features of cloud solutions like GCP. Cost effective: Pay-as-you-go options allow customers to pay only for the services they are using. Data sharing and collaboration: Using cloud-based services fosters data sharing and collaboration.

1.6 What is partitioning in BigQuery, and how does it improve performance?

Partitioning in BigQuery is a method of subdividing large tables into smaller manageable pieces based on a certain criterion like date, ingestion time, or values of integers. Thus, as it segments data into partitions, BigQuery can restrict the amount of data that is scanned during queries, ramping up performance and cutting costs considerably.

For example, a partitioned table storing daily transaction data allows queries to filter specific date ranges efficiently instead of scanning the entire dataset. This makes partitioning particularly beneficial for time-series analysis and large-scale analytical workloads.

1.7 What is clustering in BigQuery, and how does it improve query performance?

Clustering in BigQuery refers to the organization of data within partitions based on the values from one or more specified columns. By essentially grouping the related rows by clustering, BigQuery reduces the amount of data that is scanned, thus improving the performance of the queries and decreasing overall processing costs.

This way of optimizing queries has proved particularly efficient when filtering, sorting, or aggregating data based on clustered columns, where the query engine can skip all irrelevant data altogether rather than doing a full scan of the partition. Clustering works best when combined with partitioning and provides even more performance benefits when working with large datasets.

1.8 What is the difference between a table and a view in BigQuery?

In BigQuery, a table is a structured storage unit that physically holds data, while a view is a virtual table that dynamically retrieves data based on a predefined SQL query. The key distinction between them is that views cannot contain data by themselves but serve as a building block for writing large queries or enforcing security rules around data access by offering access to a subset of data without effectively duplicating it. Views also improve query execution times by permitting users to re-access data without having to

reload or reorganize tables, which makes them powerful for analytics, data abstraction, and security enforcement. So, to put it simply: • A table is a structured storage unit that physically holds data • A view is a virtual table that dynamically retrieves data based on a predefined SQL query without storing it

1.9 How does BigQuery handle data security?

BigQuery employs a layered approach to data security beginning with a secure model that is intended to safeguard data at all levels of the data lifecycle. Data is encrypted at rest using Google-managed encryption keys by default, or through customer-managed keys if one desires to hold managerial control over them. Data is encrypted while in transit via HTTPS/TLS and in transit between clients and BigQuery. Identity and Access Management (IAM) control allows for fine-grained access control by providing user and service account roles and assignments to ensure that only authorized parties can either access or change data. . Audit logging is aimed at tracking user and system activity by giving an audit trail that could be used for monitoring and compliance. These built-in security features help organizations maintain data confidentiality, integrity, and regulatory compliance.

1.10 What is the BigQuery Data Transfer Service, and how does it simplify data ingestion?

The BigQuery Data Transfer Service (BQ DTS) automates and schedules data imports from various external sources into BigQuery, eliminating the need for manual ETL processes. It natively integrates with Google services like Google Ads, YouTube, and Google Cloud Storage, as well as third-party SaaS applications. By enabling automated, scheduled data transfers, BQ DTS ensures that data remains up-to-date for analysis without requiring user intervention. This service is particularly useful for organizations managing recurring data ingestion workflows at scale, improving efficiency and reducing operational overhead.

1.11 What are nested and repeated fields in BigQuery, and why are they useful?

BigQuery allows for nested and repeated fields, enabling more efficient storage and querying of hierarchical or array-based data. Nested fields use the STRUCT data type, allowing a column to contain subfields, similar to a JSON object. Repeated fields function as ARRAYS, enabling a single column to store multiple values. These structures help eliminate the need for complex JOIN operations, improve query performance, and make BigQuery well-suited for processing semi-structured data such as logs, event streams, and NoSQL-like datasets.

1.12 How can you schedule and automate jobs in BigQuery?

BigQuery offers multiple methods to schedule and automate jobs, ensuring recurring tasks run without manual intervention. • Scheduled queries allow users to define recurring query executions directly through the BigQuery web UI or API. • Cloud Scheduler provides a fully managed cron-like service that triggers queries at predefined intervals. • For event-driven automation, Cloud Functions can execute BigQuery jobs in response to triggers from other Google Cloud services. These automation tools help streamline data pipelines, reduce manual workload, and ensure timely data processing for analytics and reporting.

1.13 How does Big Query handle data partitioning and clustering?

Partitioning splits large tables into smaller segments, improving query performance by scanning only relevant data. • Time-based: By DATE, TIMESTAMP, DATETIME (e.g., daily sales_date partitions). • Integer Range: By INTEGER values (e.g., user_id ranges). • Ingestion-time: By data load timestamp (_PARTITIONDATE). It is best for Time-series data and reducing query costs when filtering by date or numeric ranges. Clustering organizes data within a table or partition by sorting it based on selected columns, speeding up queries. It is best for filtering and aggregating by frequently queried fields like region or user_id.

2 BigQuery Architecture Interview Questions

2.1 How is Google BigQuery's architecture designed, and what makes it unique?

BigQuery features a serverless, fully managed architecture that decouples storage and compute, allowing them to scale independently based on demand. Unlike traditional cloud data warehouses or on-premises massively parallel processing (MPP) systems, this separation provides flexibility, cost efficiency, and high availability without requiring users to manage infrastructure. BigQuery's compute engine is powered by Dremel, a multi-tenant cluster that executes SQL queries efficiently, while data is stored in Colossus, Google's global distributed storage system. These components communicate through Jupiter, Google's petabit-scale network, ensuring ultra-fast data transfer. The entire system is orchestrated by Borg, Google's internal cluster management system and a precursor to Kubernetes. This architecture enables users to run high-performance, scalable analytics on massive datasets without worrying about infrastructure management.

2.2 How does BigQuery separate storage and compute, and why is this beneficial?

BigQuery follows a serverless, fully managed architecture where storage and compute are completely separated: • Storage: Data is stored in Colossus, Google's global distributed storage system. This ensures high availability, durability, and near-infinite scalability without requiring users to manage infrastructure. • Compute: Query execution is handled by Dremel, a distributed query engine that dynamically allocates computational resources (slots) based on workload demand. Key Benefits of Storage-Compute Separation: • Independent scaling: You can scale storage without affecting compute and vice versa. • Cost efficiency: You only pay for the data scanned during queries, avoiding unnecessary compute costs. • Optimized performance: Queries run faster since storage doesn't act as a bottleneck for computation. • Multi-region storage: Data can be stored across different locations without impacting query speed. This design allows BigQuery to process petabyte-scale queries efficiently while keeping costs low, making it an ideal choice for cloud-based analytics.

2.3 What is Dremel, and how does it enable BigQuery's fast query performance?

Dremel is a distributed query execution engine that powers BigQuery. Unlike traditional databases that use row-based processing, Dremel employs a columnar storage format and a tree-based query execution model to optimize speed and efficiency. How Dremel Enables Fast Queries: • Columnar Storage: Instead of reading entire rows, Dremel scans only the necessary columns, reducing the amount of data processed. • Tree-Based Execution: Queries are broken down into fragments and executed in parallel across thousands of nodes. Results are aggregated in a hierarchical tree structure, minimizing latency. • Serverless Resource Allocation: Dremel dynamically assigns compute slots to queries based on complexity, ensuring efficient resource utilization. Key Benefits of Dremel: • Blazing-fast queries on petabyte-scale datasets. • Cost-efficient processing by scanning only relevant columns. • Automatic parallelization for large workloads without manual tuning.

2.4 How does denormalization work in BigQuery, and when should it be used?

Denormalization in BigQuery intentionally introduces redundancy by merging tables and duplicating data to optimize query performance. Unlike normalization, which minimizes redundancy through smaller, related tables, denormalization reduces the need for complex joins, resulting in faster read times. This approach is especially useful in data warehousing and analytics, where read operations are more frequent than writes. However, denormalization increases storage requirements, which is why BigQuery recommends using nested and repeated fields to efficiently structure denormalized data while minimizing storage overhead.

3 SQL-Specific Questions

3.1 How do you perform a JOIN operation in BigQuery?

BigQuery supports various types of JOINs to combine data from multiple tables based on a shared column. For example, to retrieve product details along with their sales information, you can use an INNER JOIN between the SALES TABLE and the PRODUCT TABLE on Product_Id:

```
SELECT
    s.Id AS Sales_Id,
    p.Product,
    p.Price,
    s.Purchase_date
FROM sales_table s
JOIN product_table p
    ON s.Product_Id = p.Id;
```

For additional details, such as delivery dates, you can join the DELIVERY TABLE:

```
SELECT
    s.Id AS Sales_Id,
    p.Product,
    p.Price,
    s.Purchase_date,
    d.Deliver_date
FROM sales_table s
JOIN product_table p ON s.Product_Id = p.Id
JOIN delivery_table d ON s.Id = d.Sales_Id;
```

3.2 Write a SQL query to select the top 10 most expensive products.

To find the top 10 most expensive products, use ORDER BY to sort by price in descending order and LIMIT to restrict the result:

```
SELECT
    Product,
    Price
FROM product_table
ORDER BY Price DESC
LIMIT 10;
```

3.3 Write a SQL query to calculate the average number of sales per month.

To compute the average number of sales per month, use the EXTRACT function to group sales by month and COUNT to count the number of sales per month:

```sql
SELECT
    EXTRACT(MONTH FROM Purchase_date) AS Month,
    COUNT(Id) AS Total_Sales,
    AVG(COUNT(Id)) OVER () AS Average_Sales
FROM sales_table
GROUP BY Month;
```

3.4 Write a SQL query to find the total number of deliveries by product.

To count deliveries for each product:

```sql
SELECT
    p.Product,
    COUNT(d.Id) AS Total_Deliveries
FROM delivery_table d
JOIN product_table p
    ON d.Product_Id = p.Id
GROUP BY p.Product;
```

3.5 Write a SQL query to find the total number of deliveries for each product.

To get sales records from the last 30 days:

```sql
SELECT *
FROM sales_table
WHERE Purchase_date >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY);
```

3.6 Write a SQL query to count the number of unique products sold in the last week.

To count distinct products sold in the past 7 days:

```sql
SELECT COUNT(DISTINCT Product_Id) AS Unique_Products_Sold
FROM sales_table
WHERE Purchase_date >= DATE_SUB(CURRENT_DATE(), INTERVAL 7 DAY);
```

3.7 Write a SQL query to group sales data by product and calculate total revenue for each.

To calculate total revenue per product:

```
SELECT
    p.Product,
    SUM(p.Price) AS Total_Revenue
FROM sales_table s
JOIN product_table p
    ON s.Product_Id = p.Id
GROUP BY p.Product;
```

3.8 Write a SQL query to calculate the percentage of total sales contributed by each product.

To determine the percentage of total sales per product:

```
SELECT
    p.Product,
    SUM(p.Price) AS Product_Sales,
    (SUM(p.Price) / (SELECT SUM(Price) FROM product_table)) * 100
       AS Sales_Percentage
FROM sales_table s
JOIN product_table p
    ON s.Product_Id = p.Id
GROUP BY p.Product;
```

3.9 Write a SQL query to rank products based on total sales.

To rank products based on total sales:

```
SELECT
    p.Product,
    SUM(p.Price) AS Total_Sales,
    RANK() OVER (ORDER BY SUM(p.Price) DESC) AS Sales_Rank
FROM sales_table s
JOIN product_table p
    ON s.Product_Id = p.Id
GROUP BY p.Product;
```

3.10 Write a SQL query to calculate the average delivery time for each product.

To calculate the average delivery time per product:

```
SELECT
    p.Product,
    AVG(DATE_DIFF(d.Deliver_date, s.Purchase_date, DAY)) AS
       Avg_Delivery_Time
FROM sales_table s
```

```
JOIN delivery_table d
    ON s.Id = d.Sales_Id
JOIN product_table p
    ON s.Product_Id = p.Id
GROUP BY p.Product;
```

4 Advanced and Scenario-Based Questions

4.1 Can you describe your experience with implementing data pipelines in BigQuery?

The interviewer is looking for insights into your hands-on experience with designing and implementing data pipelines in BigQuery. You should discuss: • Data ingestion: Methods used (BigQuery Data Transfer Service, Cloud Storage, streaming ingestion). • Data transformation: Use of SQL-based transformations, scheduled queries, or tools like Dataflow and dbt. • Orchestration: How you manage pipeline workflows using Cloud Composer (Apache Airflow), Cloud Functions, or Cloud Scheduler. • Monitoring and optimization: How you track pipeline performance, debug issues, and optimize query costs. A strong response should include real-world examples of challenges you've faced and how you addressed them, such as handling large datasets, optimizing query performance, or ensuring data integrity in production pipelines.

4.2 How can you remove duplicate records from a column in a large BigQuery table while preserving the table name?

When dealing with a table containing millions of rows and duplicate values in a specific column, the best approach is to use the ROW_NUMBER() window function to identify and retain only the first occurrence of each duplicate while removing the rest. You can achieve this by overwriting the table with a deduplicated version using a CREATE OR REPLACE TABLE statement:

```
CREATE OR REPLACE TABLE tableX AS
SELECT * EXCEPT(row_number)
FROM (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY troubleColumn ORDER BY
              someOtherColumn) AS row_number
    FROM tableX
)
WHERE row_number = 1;
```

Here's how it works: • ROW_NUMBER() assigns a unique row number to each record within the same troubleColumn value, ensuring only one record per duplicate group is retained. • The PARTITION BY troubleColumn groups rows based on duplicates in that column. • The ORDER BY someOtherColumn (e.g., Purchase_date) ensures that a specific record is retained. • The outer query filters only the first occurrence

(row_number = 1), effectively removing duplicates. • The CREATE OR REPLACE TABLE statement ensures the table remains under the same name while replacing it with the cleaned data. This approach efficiently eliminates duplicates without creating a new table or affecting the overall table structure.

4.3 How do you use BigQuery for data visualization and reporting, and what are the most common tools for it?

BigQuery serves as a powerful backend for data visualization and reporting, enabling direct connections with tools like Google Looker Studio (formerly Data Studio), Tableau, and Power BI. These tools allow users to query data in real-time and create custom dashboards, charts, and reports without manual data exports. Key techniques for visualization include: • Aggregating and transforming data in BigQuery using SQL before visualization. • Building interactive dashboards with dynamic filters and drill-down capabilities. • Optimizing queries and materialized views to improve dashboard performance. A strong response should highlight real-world use cases, such as: • Tracking business KPIs • Monitoring trends • Generating automated reports

5 Questions about BigQuery Best Practices and Optimization

5.1 What are the best practices for managing BigQuery costs efficiently?

To manage BigQuery costs effectively, you should: • Optimize queries to scan only necessary columns and rows, avoiding SELECT *. • Use partitioned and clustered tables to minimize data scanning and improve query performance. • Leverage cost monitoring tools such as budgets, alerts, and cost control policies in Google Cloud. • Take advantage of slot reservations for more predictable pricing and cost savings. • Regularly clean up unused tables and partitions to optimize storage costs. A well-structured cost management strategy helps control expenses while maintaining efficient query execution.

5.2 What are the best practices for optimizing query performance in BigQuery?

To optimize query performance in BigQuery: • Design an efficient schema, selecting appropriate data types and avoiding excessive normalization. • Use partitioning and clustering to reduce the amount of data scanned. • Avoid SELECT *, retrieving only the necessary columns. • Optimize JOIN and GROUP BY operations, ensuring they follow best practices for large-scale queries. • Leverage caching and materialized views to store precomputed results and improve performance. • Use approximate aggregation functions, like APPROX_COUNT_DISTINCT, when exact precision is not required. By following these practices, you can speed up queries, reduce compute costs, and enhance overall efficiency.

5.3 What are BigQuery slots, and how do they impact query performance?

BigQuery slots are units of computational capacity used to process SQL queries. They are dynamically allocated based on workload demands, ensuring efficient resource utilization. Organizations can choose between: • On-demand slots, which automatically scale based on query needs. • Reserved slots, which provide more predictable performance and cost savings for consistent workloads. Efficient slot management can reduce query execution time, optimize resource allocation, and lower overall costs.

5.4 How can you monitor and optimize BigQuery costs effectively?

To monitor and optimize costs in BigQuery: • Analyze query execution using query history and EXPLAIN plans to identify expensive operations. • Enable audit logs to track usage patterns and detect inefficient queries. • Set up budgets and cost alerts to prevent unexpected expenses. • Optimize slot usage, leveraging reserved slots for steady workloads to reduce costs. • Regularly clean up unused tables and partitions to avoid unnecessary storage fees.

6 BigQuery Interview Cheat Sheet

6.1 Core Concepts

Serverless, fully-managed → No infrastructure setup; separation of storage (Colossus) & compute (Dremel).

Data Model: • Datasets → logical containers • Tables (native, external, materialized) • Views (standard, authorized)

Storage Types: • Native storage (Colossus) • External tables (Google Drive, GCS, Bigtable, Cloud SQL)

Partitioning: • Types: Time-based (DATE/TIMESTAMP), ingestion-time, integer range • Reduces scan size → lowers cost

Clustering: • Organizes data based on column values for faster filtering/sorting

Pricing: • Storage: $ per GB/month • Queries: $ per TB scanned (on-demand) or flat-rate slots

6.2 Architecture Overview

Colossus → Dremel Execution Tree → Jupiter Network

• Colossus: Distributed storage layer (columnar format) • Dremel: Query execution engine (tree architecture) • Jupiter: Low-latency network backbone in GCP

6.3 SQL Essentials

Joins: INNER, LEFT, CROSS JOIN (optimize with JOIN EACH for large tables pre-2016, now automatic)

Aggregations: GROUP BY, HAVING, ARRAY_AGG

Window Functions: OVER(PARTITION BY ...) for ranking, moving averages

Nested & Repeated Data: STRUCT and ARRAY types → Use UNNEST() to flatten

Date/Time: PARSE_DATE(), EXTRACT(), DATE_DIFF(), DATE_TRUNC()

JSON Functions: JSON_EXTRACT(), JSON_QUERY()

6.4 Optimization Strategies

• Partition + Cluster frequently queried columns • Use SELECT only required columns • Filter early using WHERE to reduce scan size • Avoid SELECT * in production • Use approximate functions (APPROX_COUNT_DISTINCT) for faster queries • Materialized views for pre-aggregated results • Monitor with Query Plan & Execution Details

6.5 Security & Access Control

• IAM roles: roles/bigquery.dataViewer, roles/bigquery.dataEditor, roles/bigquery.admin • Authorized views for controlled access • Column-level & row-level security (policy tags)

6.6 Common Scenario Q&A

Q: How to handle deduplication?
A: ROW_NUMBER() OVER (PARTITION BY id ORDER BY timestamp DESC) = 1

Q: Reduce cost on large joins?
A: Partition & cluster, filter before join, use WITH for pre-aggregations.

Q: How to join streaming & batch data?
A: Use MERGE with staging tables or Dataflow to transform before load.

7 Advanced Edge Topics (Top 1% Advantage)

7.1 Integrations

• Dataflow → BigQuery: Streaming ingestion pipelines for real-time analytics • Pub/Sub → BigQuery: Event-driven data updates • dbt → BigQuery: SQL-based transformations with version control • Looker Studio / Power BI → BigQuery: Direct BI connection for dashboards

7.2 Machine Learning with BigQuery ML

Train & deploy ML models without leaving BigQuery: • CREATE MODEL for regression, classification, time-series (ARIMA+), k-means clustering • ML.PREDICT for predictions directly in SQL • Use Explainable AI (ML.EXPLAIN_PREDICT)

7.3 Streaming & Real-Time Analytics

• Insert streaming data with tabledata.insertAll API • Partition streaming tables for optimized querying • Example use: IoT sensor monitoring, fraud detection

7.4 Cross-Cloud & External Data

• BigLake for unified lakehouse storage • Federated queries to external sources: • Cloud Storage (CSV, Parquet, Avro) • Cloud SQL • Bigtable • Sheets