☁ Cloud Databases: BigQuery Architecture

Generative AI Bootcamp – Week 2, Day 2, Session 1

November 25, 2025

🎯 Learning Objectives

- Understand **BigQuery architecture and principles**
- Explore how BigQuery supports **AI and LLM data pipelines**
- Learn about **datasets, tables, and query execution**
- Apply **performance and cost optimization** strategies

🌐 What Is BigQuery?

- Google Cloud’s **serverless, columnar data warehouse**
- Designed for **large-scale analytics** and **AI workflows**
- Fully managed – no infrastructure provisioning required
- Seamlessly integrates with **Python SDK**, **Vertex AI**, and **Looker Studio**

🧰 Python SDK Integration

```
from google.cloud import bigquery
client = bigquery.Client(project="my-project")
query = "SELECT name, SUM(sales) FROM dataset.table GROUP BY name"
df = client.query(query).to_dataframe()
```

➡️ Returns results as **Pandas DataFrame**, ready for further analysis or ML preprocessing.

💡 Use Case Example

Scenario: Retail company uses BigQuery to analyze real-time sales data.

```
from google.cloud import bigquery
client = bigquery.Client()

query = """
SELECT
  store_id,
  DATE_TRUNC(sale_date, ISOWEEK) AS week,
  SUM(revenue) AS weekly_revenue
FROM `retail.sales` GROUP BY store_id, week ORDER BY week;
"""
df = client.query(query).to_dataframe()
```

➡️ Result is ingested into a **Vertex AI time-series model** for sales forecasting.

🧩 Architectural Overview

- **Storage Layer** – columnar, compressed, distributed across Colossus
- **Compute Layer** – based on **Dremel**, organizes workers in a tree of “mixers” and “slots”
- **Query Engine** – supports ANSI SQL with **massively parallel execution**
- **Control Plane** – manages jobs, billing, metadata, and monitoring

🧱 Architecture Visualization

%%{init: {"themeVariables": { "fontSize": "20px" }}}%% flowchart TD T1["Control Pane (Metadata, Billing)"] T2["Compute Layer (Dremel Executors)"] T3["Storage Layer (Colossus Files)"] T1 --> T2 --> T3

💡 *Each layer can scale independently to meet workload demand.*

⚙️ Separation of Storage & Compute

- Enables **elastic scaling** and **isolation** for concurrent workloads
- Billing is **decoupled** — pay separately for storage and queries
- Improves **resilience** and **data sharing** between projects

💼 Example: Cross-Team Data Sharing

Teams can query shared datasets without duplication:

```
-- Data Science project querying Marketing dataset
SELECT campaign_id, AVG(conversion_rate)
FROM `marketing.analytics.campaign_metrics`
GROUP BY campaign_id;
```

✅ Zero-copy data access using project-level IAM permissions.

🧮 BigQuery Concepts

Concept	Description
Dataset	Logical container for tables
Table	Structured, columnar data storage
View	Saved SQL query (virtual table)
Job	Execution of a query or load task
Project	Resource boundary for billing and IAM

💡 Example: Creating a Dataset & Table (Python)

```python
dataset_ref = client.dataset("sales_data")

# Create dataset
dataset = bigquery.Dataset(dataset_ref)
dataset.location = "US"
client.create_dataset(dataset, exists_ok=True)

# Define schema and create table
schema = [
    bigquery.SchemaField("order_id", "STRING"),
    bigquery.SchemaField("amount", "FLOAT"),
]
table_ref = dataset_ref.table("transactions")
table = bigquery.Table(table_ref, schema=schema)
client.create_table(table, exists_ok=True)
```

💾 Data Ingestion Methods

- Upload CSV, JSON, or Parquet directly
- Stream data via `insertAll` **API**
- Load from **Cloud Storage**
- Use **federated queries** across Sheets, Drive, or Cloud SQL

🧠 Example: Streaming Data via API

```
rows_to_insert = [
    {"order_id": "A001", "amount": 53.2},
    {"order_id": "A002", "amount": 84.7},
]

table_id = "myproject.sales_data.transactions"
errors = client.insert_rows_json(table_id, rows_to_insert)
if errors:
    print("Errors:", errors)
else:
    print("Data streamed successfully!")
```

💡 Useful for **real-time analytics** pipelines (e.g., IoT, event streams).

📊 Query Processing Pipeline

1. **Parsing & Validation**
2. **Execution Plan Generation**
3. **Slot Allocation** in Dremel tree
4. **Parallel Execution** across workers
5. **Result Assembly & Caching**

🧩 Example: Query Execution Plan

```
EXPLAIN SELECT customer_id, COUNT(*) FROM `sales.transactions`
GROUP BY customer_id;
```

Returns JSON output describing the query stages and slot allocation — useful for debugging **performance bottlenecks**.

🧠 Why BigQuery for AI?

- Centralized store for **prompt datasets**, **embeddings**, and **RAG data sources**
- Integrates with **Vertex AI**, **LangChain**, **Pandas AI**
- Supports **vector embeddings**, **semantic joins**, and **feature engineering**

🤖 Example: Storing and Using Embeddings

```
CREATE TABLE `rag_store.embeddings` AS
SELECT
  id,
  ML.GENERATE_EMBEDDING(MODEL `vertexai.textembedding-gecko@latest`, content) AS vector
FROM `raw_documents`;
```

Then, semantic similarity query:

```
SELECT *
FROM `rag_store.embeddings`
ORDER BY ML.DOT_PRODUCT(vector, @query_embedding) DESC
LIMIT 5;
```

💰 Pricing Model

- **Storage:** pay per TB stored / month
- **Query:** pay per TB processed
- **Free Tier:** 10 GB storage + 1 TB queries/month

  💡 Use **query previews** and **EXPLAIN plans** for cost control.

💡 Example: Cost-Aware Querying

```
SELECT * FROM `sales.transactions`
WHERE DATE(order_date) > '2025-01-01'
OPTIONS (dry_run = true);
```

✅ Returns estimated cost **without executing** the query.

🛡️ Security & Access Control

- IAM roles: `roles/bigquery.user`, `roles/bigquery.admin`
- Table-, column-, and row-level security
- **Audit logs** and **data masking** for sensitive fields

🔐 Example: Row-Level Security

```
CREATE ROW ACCESS POLICY region_policy
ON `finance.revenue`
GRANT TO ("group:europe-team@company.com")
FILTER USING (region = "EU");
```

✅ Restricts access dynamically based on **user group**.

⚡ Performance Tips

- Use **partitioned** and **clustered tables**
- Avoid `SELECT *` in production
- Apply **materialized views** and **result caching**
- Monitor performance with **Query Insights**

🔍 Example: Partitioned Table

```
CREATE TABLE `sales.transactions_partitioned`
PARTITION BY DATE(order_date)
AS SELECT * FROM `sales.transactions`;
```

✅ Queries on specific date ranges will **scan fewer partitions**, reducing cost.

🧭 Real-World Scenario

Use Case: An AI-driven customer support system stores embeddings and conversation history in BigQuery.

- Analysts use SQL to monitor topic clusters
- Engineers connect BigQuery via LangChain for context retrieval
- Data scientists export datasets to Vertex AI for fine-tuning

💡 Summary

- BigQuery = scalable, cost-efficient analytics backbone for AI
- Combines **serverless analytics**, **SQL familiarity**, and **ML integration**
- Ideal for **data-centric AI systems** (RAG, feature stores, observability)

🚀 Next Session

Hands-on BigQuery Lab

> - Run queries via Python SDK
> - Ingest and analyze sample AI dataset
> - Explore cost optimization and performance tuning