Module 3: Data Warehouse and BigQuery

Table of Contents

Module Syllabus

This module covers the fundamentals of **Data Warehousing** with a focus on **Google BigQuery**. By the end of this module, you will understand:

- What a Data Warehouse is and how it differs from traditional databases
- OLAP vs OLTP systems
- BigQuery architecture and internals
- Creating and managing tables in BigQuery
- External tables vs Native tables
- Partitioning and Clustering for query optimization
- BigQuery best practices for cost and performance
- Introduction to BigQuery ML (Machine Learning)
- Deploying ML models from BigQuery

Theoretical Concepts

1. What is a Data Warehouse?

A **Data Warehouse (DW)** is a centralized repository designed for **analytical processing** and **reporting**. It consolidates data from multiple sources into a single, consistent store optimized for querying and analysis.

Key Characteristics: - Stores historical data (not just current state) - Optimized for read-heavy analytical queries - Uses denormalized schemas (Star/Snowflake) - Supports complex aggregations and joins - Typically updated in batches (ETL/ELT processes)

2. OLAP vs OLTP

Aspect	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Purpose	Day-to-day operations	Analysis and reporting
Queries	Simple, short transactions	Complex, aggregating queries

Aspect	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Data	Current, real-time	Historical, consolidated
Schema	Normalized (3NF)	Denormalized (Star/Snowflake)
Users	Clerks, customers	Analysts, data scientists
Examples	MySQL, PostgreSQL	BigQuery, Snowflake, Redshift

3. Google BigQuery Overview

BigQuery is Google's fully-managed, serverless data warehouse that enables super-fast SQL queries using the processing power of Google's infrastructure.

Key Features: - **Serverless**: No infrastructure to manage - **Scalable**: Handles petabytes of data - **Columnar Storage**: Optimized for analytical queries - **Separation of Compute and Storage**: Pay for what you use - **Built-in ML**: Train models using SQL (BigQuery ML) - **Geospatial Analysis**: Native support for geographic data - **Real-time Analytics**: Streaming inserts supported

4. BigQuery Architecture (Internals)

BigQuery uses a distributed architecture with several key components:

```
                     BigQuery Service



      Dremel           Colossus          Borg
     (Compute)         (Storage)      (Orchestr.)






                        Jupiter
                       (Network)
```

- **Dremel**: Execution engine that processes queries using a tree architecture
- **Colossus**: Google's distributed file system (stores data in columnar format)
- **Borg**: Cluster management system (allocates compute resources)
- **Jupiter**: Petabit network that connects compute and storage

Column-Oriented Storage:

```
Row-oriented:     Column-oriented:

  A   B   C       A1  A2  A3  ...  ← Column A

 A1  B1  C1
             B1  B2  B3  ...  ← Column B
 A2  B2  C2

                  C1  C2  C3  ...  ← Column C
```

Benefits of Columnar Storage: - Only reads columns needed for the query - Better compression (similar data types together) - Faster aggregations (SUM, AVG, COUNT)

5. External Tables vs Native Tables

Feature	External Table	Native (Managed) Table
Data Location	GCS, Drive, Bigtable	BigQuery Storage
Query Performance	Slower	Faster
Cost	Storage: External; Query: BQ	Both in BQ
Data Freshness	Always current	Requires refresh
Partitioning	Limited	Full support
Caching	No	Yes

When to use External Tables: - Data changes frequently in source - You want to avoid data duplication - Initial exploration before loading

6. Partitioning

Partitioning divides a table into smaller segments based on a column value, allowing BigQuery to scan only relevant partitions.

Partition Types: - **Time-unit column**: HOUR, DAY, MONTH, YEAR - **Ingestion time**: `_PARTITIONTIME` pseudo-column - **Integer range**: Partition by integer column ranges

Benefits: - Reduces data scanned (cost savings) - Improves query performance - Easier data management (delete old partitions)

Limits: - Maximum 4,000 partitions per table - One partition column per table

```sql
-- Example: Partition by date
CREATE TABLE dataset.my_table
PARTITION BY DATE(timestamp_column)
AS SELECT * FROM source_table;
```

7. Clustering

Clustering sorts data within each partition based on one or more columns, colocating related data.

Benefits: - Further reduces data scanned - Improves filter and aggregate performance - Works best with high-cardinality columns

Characteristics: - Up to 4 clustering columns - Order of columns matters - BigQuery auto-reclusters in background

```sql
-- Example: Partition + Cluster
CREATE TABLE dataset.my_table
PARTITION BY DATE(timestamp_column)
CLUSTER BY vendor_id, payment_type
AS SELECT * FROM source_table;
```

8. Partitioning vs Clustering: When to Use Which?

Use Partitioning When	Use Clustering When
Filter/aggregate on single column frequently	Need to filter on multiple columns
Large data volume (>1GB per partition)	Columns have high cardinality
Need to manage partition lifecycle	Partitioning alone isn't enough
Want predictable cost estimates	Need strict column ordering in queries

Combined Strategy: - Partition by DATE (time-based queries) - Cluster by frequently filtered columns

9. BigQuery Best Practices

Cost Optimization: - Avoid `SELECT *` — only query needed columns - Use partitioned and clustered tables - Use streaming inserts sparingly (more expensive) - Set up cost controls and quotas - Preview queries to estimate costs before running

Query Performance: - Filter early and filter often (WHERE clauses) - Denormalize data when possible - Use approximate aggregation functions (`APPROX_COUNT_DISTINCT`) - Avoid self-joins on large tables - Use `LIMIT` for exploration (but it doesn't reduce scan cost)

Data Management: - Use expiration settings for temporary tables - Set partition expiration for time-series data - Use `INFORMATION_SCHEMA` to monitor usage

---

Practical Tools Covered

1. Google BigQuery

Purpose: Serverless data warehouse for analytics **Use Cases:** - Ad-hoc SQL analysis on large datasets - Business intelligence and reporting - Data science and ML workflows - Real-time analytics with streaming

2. Google Cloud Storage (GCS)

Purpose: Object storage for data lake **Use Cases:** - Store raw data files (CSV, Parquet, JSON) - Data lake layer before warehouse - External table source for BigQuery - ML model storage and deployment

3. BigQuery ML

Purpose: Train ML models using SQL **Use Cases:** - Linear/logistic regression - K-means clustering - Time series forecasting - Recommendation systems - Classification models

4. bq Command Line Tool

Purpose: Interact with BigQuery from terminal **Use Cases:** - Run queries from scripts - Export/import data - Manage datasets and tables - Extract ML models

5. TensorFlow Serving

Purpose: Deploy ML models as REST APIs **Use Cases:** - Serve BigQuery ML models in production - Real-time predictions - Model versioning and A/B testing

---

Step-by-Step Tutorial

Prerequisites

Before starting, ensure you have: - Google Cloud Platform account with billing enabled - A GCP project created - BigQuery API enabled - Google Cloud SDK installed (for CLI operations) - Service account with BigQuery and GCS permissions

Part 1: Setting Up Your Environment

1.1 Create a Dataset in BigQuery

```sql
-- In BigQuery Console, create a new dataset
CREATE SCHEMA IF NOT EXISTS `your-project-id.nytaxi`
OPTIONS (
  location = 'US'
);
```

Or via the UI: 1. Go to BigQuery Console 2. Click on your project 3. Click "Create Dataset" 4. Name it `nytaxi` 5. Choose location (US recommended for this tutorial)

1.2 Upload Data to GCS You have two options:

Option A: Use the provided Python script

```python
# web_to_gcs.py - Upload taxi data to GCS
import os
import requests
import pandas as pd
from google.cloud import storage

# Set your bucket name
BUCKET = os.environ.get("GCP_GCS_BUCKET", "your-bucket-name")
init_url = 'https://github.com/DataTalksClub/nyc-tlc-data/releases/download/'

def upload_to_gcs(bucket, object_name, local_file):
    """Upload a file to GCS bucket."""
    client = storage.Client()
    bucket = client.bucket(bucket)
    blob = bucket.blob(object_name)
    blob.upload_from_filename(local_file)

def web_to_gcs(year, service):
    """Download taxi data and upload to GCS as parquet."""
    for i in range(12):
        month = str(i + 1).zfill(2)
        file_name = f"{service}_tripdata_{year}-{month}.csv.gz"

        # Download file
        request_url = f"{init_url}{service}/{file_name}"
        r = requests.get(request_url)
        open(file_name, 'wb').write(r.content)
        print(f"Downloaded: {file_name}")

        # Convert to parquet
        df = pd.read_csv(file_name, compression='gzip')
        parquet_file = file_name.replace('.csv.gz', '.parquet')
        df.to_parquet(parquet_file, engine='pyarrow')
        print(f"Converted to: {parquet_file}")

        # Upload to GCS
        upload_to_gcs(BUCKET, f"{service}/{parquet_file}", parquet_file)
        print(f"Uploaded to: gs://{BUCKET}/{service}/{parquet_file}")

# Run for yellow taxi data
web_to_gcs('2019', 'yellow')
web_to_gcs('2020', 'yellow')
```

Option B: Manual upload via gsutil

```
# Download parquet files
wget https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2019-01.parquet

# Upload to GCS
gsutil cp yellow_tripdata_*.parquet gs://your-bucket-name/yellow/
```

Part 2: Creating Tables in BigQuery

2.1 Create an External Table External tables reference data stored in GCS without copying it into BigQuery.

```
-- Create external table from parquet files in GCS
CREATE OR REPLACE EXTERNAL TABLE `your-project-id.nytaxi.external_yellow_tripdata`
OPTIONS (
  format = 'PARQUET',
  uris = ['gs://your-bucket-name/yellow/yellow_tripdata_2019-*.parquet',
          'gs://your-bucket-name/yellow/yellow_tripdata_2020-*.parquet']
);
```

For CSV files:

```
CREATE OR REPLACE EXTERNAL TABLE `your-project-id.nytaxi.external_yellow_tripdata`
OPTIONS (
  format = 'CSV',
  uris = ['gs://your-bucket-name/yellow/yellow_tripdata_2019-*.csv']
);
```

2.2 Verify the External Table

```
-- Preview data from external table
SELECT *
FROM `your-project-id.nytaxi.external_yellow_tripdata`
LIMIT 10;

-- Count total records
SELECT COUNT(*) as total_records
FROM `your-project-id.nytaxi.external_yellow_tripdata`;
```

2.3 Create a Native (Non-Partitioned) Table

```sql
-- Create a regular BigQuery table from external table
CREATE OR REPLACE TABLE `your-project-id.nytaxi.yellow_tripdata_non_partitioned` AS
SELECT * FROM `your-project-id.nytaxi.external_yellow_tripdata`;
```

Part 3: Partitioning Tables

3.1 Create a Partitioned Table

```sql
-- Create a table partitioned by pickup datetime
CREATE OR REPLACE TABLE `your-project-id.nytaxi.yellow_tripdata_partitioned`
PARTITION BY DATE(tpep_pickup_datetime) AS
SELECT * FROM `your-project-id.nytaxi.external_yellow_tripdata`;
```

3.2 Compare Query Performance

```sql
-- Query on NON-PARTITIONED table
-- This scans ~1.6GB of data
SELECT DISTINCT(VendorID)
FROM `your-project-id.nytaxi.yellow_tripdata_non_partitioned`
WHERE DATE(tpep_pickup_datetime) BETWEEN '2019-06-01' AND '2019-06-30';

-- Same query on PARTITIONED table
-- This scans only ~106MB of data (partition pruning!)
SELECT DISTINCT(VendorID)
FROM `your-project-id.nytaxi.yellow_tripdata_partitioned`
WHERE DATE(tpep_pickup_datetime) BETWEEN '2019-06-01' AND '2019-06-30';
```

3.3 Inspect Partitions

```sql
-- View partition information
SELECT
  table_name,
  partition_id,
  total_rows
FROM `nytaxi.INFORMATION_SCHEMA.PARTITIONS`
WHERE table_name = 'yellow_tripdata_partitioned'
ORDER BY total_rows DESC;
```

Part 4: Clustering Tables

4.1 Create a Partitioned and Clustered Table

```sql
-- Partition by date AND cluster by VendorID
CREATE OR REPLACE TABLE `your-project-id.nytaxi.yellow_tripdata_partitioned_clustered`
PARTITION BY DATE(tpep_pickup_datetime)
CLUSTER BY VendorID AS
SELECT * FROM `your-project-id.nytaxi.external_yellow_tripdata`;
```

4.2 Compare Clustered vs Non-Clustered

```
-- Query on partitioned table (no clustering)
-- Scans ~1.1 GB
SELECT COUNT(*) as trips
FROM `your-project-id.nytaxi.yellow_tripdata_partitioned`
WHERE DATE(tpep_pickup_datetime) BETWEEN '2019-06-01' AND '2020-12-31'
  AND VendorID = 1;

-- Same query on partitioned + clustered table
-- Scans ~864.5 MB (clustering reduces scan further)
SELECT COUNT(*) as trips
FROM `your-project-id.nytaxi.yellow_tripdata_partitioned_clustered`
WHERE DATE(tpep_pickup_datetime) BETWEEN '2019-06-01' AND '2020-12-31'
  AND VendorID = 1;
```

Part 5: BigQuery ML (Machine Learning)

5.1 Prepare Data for ML

```
-- Select relevant features for tip prediction
SELECT
  passenger_count,
  trip_distance,
  PULocationID,
  DOLocationID,
  payment_type,
  fare_amount,
  tolls_amount,
  tip_amount
FROM `your-project-id.nytaxi.yellow_tripdata_partitioned`
WHERE fare_amount != 0;
```

5.2 Create ML-Ready Table

```
-- Create table with proper data types for ML
CREATE OR REPLACE TABLE `your-project-id.nytaxi.yellow_tripdata_ml` (
  `passenger_count` INTEGER,
  `trip_distance` FLOAT64,
  `PULocationID` STRING,
  `DOLocationID` STRING,
  `payment_type` STRING,
  `fare_amount` FLOAT64,
  `tolls_amount` FLOAT64,
  `tip_amount` FLOAT64
) AS (
  SELECT
```

```
    passenger_count,
    trip_distance,
    CAST(PULocationID AS STRING),
    CAST(DOLocationID AS STRING),
    CAST(payment_type AS STRING),
    fare_amount,
    tolls_amount,
    tip_amount
  FROM `your-project-id.nytaxi.yellow_tripdata_partitioned`
  WHERE fare_amount != 0
);
```

5.3 Train a Linear Regression Model

```
-- Create a linear regression model to predict tip amount
CREATE OR REPLACE MODEL `your-project-id.nytaxi.tip_model`
OPTIONS (
  model_type = 'linear_reg',
  input_label_cols = ['tip_amount'],
  DATA_SPLIT_METHOD = 'AUTO_SPLIT'
) AS
SELECT *
FROM `your-project-id.nytaxi.yellow_tripdata_ml`
WHERE tip_amount IS NOT NULL;
```

5.4 Check Model Features

```
-- View feature information
SELECT *
FROM ML.FEATURE_INFO(MODEL `your-project-id.nytaxi.tip_model`);
```

5.5 Evaluate the Model

```
-- Get model evaluation metrics
SELECT *
FROM ML.EVALUATE(
  MODEL `your-project-id.nytaxi.tip_model`,
  (SELECT * FROM `your-project-id.nytaxi.yellow_tripdata_ml` WHERE tip_amount IS NOT NULL)
);
```

Key Metrics to Look For: - `mean_absolute_error`: Average absolute difference between predicted and actual - `mean_squared_error`: Average squared difference - `r2_score`: Proportion of variance explained (closer to 1 is better)

5.6 Make Predictions

```
-- Predict tip amounts
SELECT *
```

```
FROM ML.PREDICT(
  MODEL `your-project-id.nytaxi.tip_model`,
  (SELECT * FROM `your-project-id.nytaxi.yellow_tripdata_ml` WHERE tip_amount IS NOT NULL)
)
LIMIT 100;
```

5.7 Explain Predictions

```
-- Get feature importance for predictions
SELECT *
FROM ML.EXPLAIN_PREDICT(
  MODEL `your-project-id.nytaxi.tip_model`,
  (SELECT * FROM `your-project-id.nytaxi.yellow_tripdata_ml` WHERE tip_amount IS NOT NULL),
  STRUCT(3 AS top_k_features)
)
LIMIT 10;
```

5.8 Hyperparameter Tuning

```
-- Create model with hyperparameter tuning
CREATE OR REPLACE MODEL `your-project-id.nytaxi.tip_hyperparam_model`
OPTIONS (
  model_type = 'linear_reg',
  input_label_cols = ['tip_amount'],
  DATA_SPLIT_METHOD = 'AUTO_SPLIT',
  num_trials = 5,
  max_parallel_trials = 2,
  l1_reg = hparam_range(0, 20),
  l2_reg = hparam_candidates([0, 0.1, 1, 10])
) AS
SELECT *
FROM `your-project-id.nytaxi.yellow_tripdata_ml`
WHERE tip_amount IS NOT NULL;
```

Part 6: Deploying ML Models

6.1 Export Model to GCS

```
# Authenticate with gcloud
gcloud auth login

# Export the model to GCS
bq --project_id your-project-id extract -m nytaxi.tip_model gs://your-bucket/tip_model
```

6.2 Download Model Locally

```
# Create local directory
mkdir -p /tmp/model
```

```
# Copy model from GCS
gsutil cp -r gs://your-bucket/tip_model /tmp/model

# Create serving directory structure
mkdir -p serving_dir/tip_model/1
cp -r /tmp/model/tip_model/* serving_dir/tip_model/1
```

6.3 Deploy with TensorFlow Serving

```
# Pull TensorFlow Serving Docker image
docker pull tensorflow/serving

# Run the model server
docker run -p 8501:8501 \
  --mount type=bind,source=$(pwd)/serving_dir/tip_model,target=/models/tip_model \
  -e MODEL_NAME=tip_model \
  -t tensorflow/serving &
```

6.4 Test the Deployed Model

```
# Make a prediction request
curl -d '{
  "instances": [{
    "passenger_count": 1,
    "trip_distance": 12.2,
    "PULocationID": "193",
    "DOLocationID": "264",
    "payment_type": "2",
    "fare_amount": 20.4,
    "tolls_amount": 0.0
  }]
}' \
-X POST http://localhost:8501/v1/models/tip_model:predict
```

6.5 Check Model Status

```
# View model status
curl http://localhost:8501/v1/models/tip_model
```

Part 7: Useful BigQuery Queries

Query Public Datasets

```
-- Example: Query NYC Citibike public dataset
SELECT station_id, name
FROM `bigquery-public-data.new_york_citibike.citibike_stations`
LIMIT 100;
```

Check Table Size and Cost Estimates

```sql
-- Get table size information
SELECT
  table_name,
  ROUND(SUM(size_bytes) / POW(10,9), 2) AS size_gb,
  SUM(row_count) AS total_rows
FROM `your-project-id.nytaxi.__TABLES__`
GROUP BY table_name;
```

Monitor Query History

```sql
-- View recent queries and their costs
SELECT
  user_email,
  query,
  total_bytes_processed,
  total_slot_ms,
  creation_time
FROM `region-us`.INFORMATION_SCHEMA.JOBS_BY_PROJECT
WHERE creation_time > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)
ORDER BY creation_time DESC
LIMIT 20;
```

Documentation Links

BigQuery

Resource	Link
BigQuery Documentation	https://cloud.google.com/bigquery/docs
BigQuery SQL Reference	https://cloud.google.com/bigquery/docs/reference/standard sql/query-syntax
Partitioned Tables	https://cloud.google.com/bigquery/docs/partitioned-tables
Clustered Tables	https://cloud.google.com/bigquery/docs/clustered-tables
External Tables	https://cloud.google.com/bigquery/docs/external-tables
BigQuery Best Practices	https://cloud.google.com/bigquery/docs/best-practices-performance-overview
Pricing	https://cloud.google.com/bigquery/pricing

BigQuery ML

Resource	Link
BigQuery ML Documentation	https://cloud.google.com/bigquery-ml/docs
BigQuery ML Tutorials	https://cloud.google.com/bigquery-ml/docs/tutorials
ML Reference Patterns	https://cloud.google.com/bigquery-ml/docs/analytics-reference-patterns
Hyperparameter Tuning	https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-glm
Feature Preprocessing	https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-preprocess-overview
Export Models	https://cloud.google.com/bigquery-ml/docs/export-model-tutorial

Google Cloud Storage

Resource	Link
GCS Documentation	https://cloud.google.com/storage/docs
gsutil Tool	https://cloud.google.com/storage/docs/gsutil
Python Client	https://cloud.google.com/python/docs/reference/storage/la

Additional Resources

Resource	Link
bq Command Line Tool	https://cloud.google.com/bigquery/docs/bq-command-line-tool
TensorFlow Serving	https://www.tensorflow.org/tfx/guide/serving
Data Engineering Zoomcamp	https://github.com/DataTalksClub/data-engineering-zoomcamp
NYC TLC Trip Data	https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Video Tutorials (Course)

Topic	YouTube Link
Data Warehouse and BigQuery	https://youtu.be/jrHljAoD6nM
Partitioning vs Clustering	https://youtu.be/-CqXf7vhhDs
Best Practices	https://youtu.be/k81mLJVX08w
Internals of BigQuery	https://youtu.be/eduHi1inM4s
Machine Learning in BigQuery	https://youtu.be/B-WtpB0PuG4
Deploying ML Models	https://youtu.be/BjARzEWaznU

Quick Reference Card

Common SQL Patterns

```sql
-- Create partitioned + clustered table
CREATE OR REPLACE TABLE `project.dataset.table`
PARTITION BY DATE(date_column)
CLUSTER BY col1, col2
AS SELECT * FROM source;

-- Check bytes to be scanned (dry run in UI shows this)
-- Hover over query in BigQuery console before running

-- Delete partition
DELETE FROM `project.dataset.table`
WHERE DATE(date_column) = '2019-01-01';

-- Query partition metadata
SELECT * FROM `dataset.INFORMATION_SCHEMA.PARTITIONS`
WHERE table_name = 'your_table';
```

Cost Estimation Formula

```
Query Cost = (Bytes Scanned / 1 TB) × $5.00
(On-demand pricing as of 2024)
```

Key Limits to Remember

Limit	Value
Max partitions per table	4,000
Max clustering columns	4
Max columns per table	10,000
Max row size	100 MB
Streaming insert row size	10 MB

Notes compiled for Data Engineering Zoomcamp - Module 3