D607 – Cloud Databases (Task 2)

Morrell J. Parrish

Western Governors University

Table of Contents

[A. Google Cloud Platform Schema 4](#_Toc190281198)

[A.1. Schema Objects 4](#_Toc190281199)

[A.2. SQL Code to Create Schema 5](#_Toc190281200)

[A.2.1. Customers Table 5](#_Toc190281201)

[A.2.2. Transactions Table 5](#_Toc190281202)

[A.2.3. ShoppingCart Table 6](#_Toc190281203)

[B. Populating the Tables with Data 8](#_Toc190281204)

[B.1. Record Import 8](#_Toc190281205)

[B.2. Data Preprocessing 8](#_Toc190281206)

[B.2.1. Python Script – JSON to NDJSON 9](#_Toc190281207)

[B.2.2. Python Script to Split File 9](#_Toc190281208)

[C. Writing Queries and Running in GCP 10](#_Toc190281209)

[C.1. Query to List Unique Customers 10](#_Toc190281210)

[C.2. Query to List All Items in a Customer’s Shopping Cart 11](#_Toc190281211)

[C.3. Query to List Total Purchase Amounts for Customers (Desc) 12](#_Toc190281212)

[C.4. Panopto Video 13](#_Toc190281213)

[Conclusion 13](#_Toc190281214)

[References 14](#_Toc190281215)

D607 – Cloud Databases (Task 2)

****introduction****

After successfully convincing **Alliah Company** to transition from an **on-premises data storage solution** to a **cloud-based solution** in **Task 1**, your next responsibility is to set up a **sandbox environment** to validate that the **proposed cloud storage solution functions as expected**.

To achieve this, you will complete the following tasks using the provided **Alliah Company Scenario, Data Dictionary for Transaction Data, and Transaction Data** documents:

1. **Create the necessary database schema** in **Google Cloud Platform (GCP)** based on the optimal data store identified in Task 1.
2. **Populate the schema** with records from the **sample transaction dataset** to ensure data integrity.
3. **Execute and demonstrate queries** that validate the database functionality, proving that the data store performs as intended.

This process ensures that Alliah Company’s **cloud migration is successful**, supporting **scalability, efficiency, and reliability** in their new cloud environment (Western Governors University, n.d.).

# Google Cloud Platform Schema

A.1. Schema Objects

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A.2. SQL Code to Create Schema

### A.2.1. Customers Table

The **customers table** stores information about customers, including their unique ID, name, email, phone number, and address. This table helps manage customer-related data, such as personal details and contact information. It serves as the main reference table for identifying customers in other tables.

**-- Customers Table Schema**

CREATE TABLE `cal-2870-c81ef5710c2c.wgu\_d607\_25e398a3cfed3dbe.Customers` (

customer\_id STRING NOT NULL,

name STRING,

email STRING,

phone STRING,

address STRING

);

**Variables and Data Types:**

* customer\_id (STRING, **NOT NULL**): Unique identifier for each customer. This serves as the primary key.
* name (STRING): Customer's full name.
* email (STRING): Customer's email address for communication purposes.
* phone (STRING): Customer's phone number.
* address (STRING): Customer's physical address.

### A.2.2. Transactions Table

The **transactions table** records transaction details for each purchase made by a customer. This table includes the transaction ID, customer ID (foreign key), transaction date, shipping method, and total amount. It tracks the overall activity of a transaction and helps in analyzing sales trends and customer purchasing behavior.

**-- Transactions Table Schema**

CREATE TABLE `cal-2870-c81ef5710c2c.wgu\_d607\_25e398a3cfed3dbe.Transactions` (

transaction\_id STRING NOT NULL,

customer\_id STRING NOT NULL,

transaction\_date TIMESTAMP,

shipping\_method STRING,

total\_amount FLOAT64

);

**Variables and Data Types:**

* transaction\_id (STRING, **NOT NULL**): Unique identifier for each transaction. This serves as the primary key.
* customer\_id (STRING, **NOT NULL**): Foreign key referencing customer\_id in the Customers table, linking each transaction to a customer.
* transaction\_date (TIMESTAMP): Date and time when the transaction occurred.
* shipping\_method (STRING): Method of shipping chosen for the transaction (e.g., Standard, Express).
* total\_amount (FLOAT64): Total amount for the transaction.

### A.2.3. ShoppingCart Table

The **shoppingcart table** stores details about items in a customer's shopping cart during a transaction. This table includes the cart ID, transaction ID (foreign key), product ID (foreign key), vendor, product category, product name, quantity, and price. It provides item-level tracking for each transaction, enabling inventory analysis and reporting.

**-- ShoppingCart Table Schema**

CREATE TABLE `cal-2870-c81ef5710c2c.wgu\_d607\_25e398a3cfed3dbe.ShoppingCart` (

cart\_id STRING NOT NULL,

transaction\_id STRING NOT NULL,

product\_id STRING,

vendor STRING,

product\_category STRING,

product\_name STRING,

quantity INT64,

price FLOAT64

);

**Variables and Data Types:**

* cart\_id (STRING, **NOT NULL**): Unique identifier for each shopping cart. This serves as the primary key.
* transaction\_id (STRING, **NOT NULL**): Foreign key referencing transaction\_id in the Transactions table, linking each cart item to a transaction.
* product\_id (STRING): Foreign key that references a product in the products catalog, identifying the specific item purchased.
* vendor (STRING): Vendor selling the product.
* product\_category (STRING): Category of the product (e.g., Electronics, Clothing, Home Appliances).
* product\_name (STRING): Name of the product.
* quantity (INT64): Number of units purchased for this product.
* price (FLOAT64): Price per unit of the product.

# Populating the Tables with Data

## B.1. Record Import

A screenshot of a computer

Description automatically generated

## B.2. Data Preprocessing

Data can be added manually or via upload.The ***“TransactionData.json”*** file contains customer’s demographics, transactions, and shopping cart info*.*The current transaction file is not compatible with BigQuery**. BigQuery** does NOT support array-wrapped JSON files; we need to modify the JSON file slightly. The first python code was used to convert the original JSON file to Newline-Delimited JSON (NDJON), move the Customer’s ID to the root level, correct incorrect vendor key names, and then save the file as ***“TransactionData\_fixed.json***

The second code was used to separate the ***TransactionData\_fixed.json*** file into three different files – customers, shoppingcart, and transactions. The new json files are uploaded via BigQuery’s upload feature; this option will create the database schema and insert the data into the appropriate tables.

### B.2.1. Python Script – JSON to NDJSON

========================== **Convert JSON to Newline-Delimited JSON (NDJSON)** ======================

import json  
  
*# Load the JSON file*with open("TransactionData.json", "r") as f:  
 data = json.load(f)  
  
*# Fix JSON structure*for record in data:  
 *# Move customer\_id to root level* record["customer\_id"] = record["customer"]["customer\_id"]  
 record["name"] = record["customer"]["name"]  
 record["email"] = record["customer"]["email"]  
 record["phone"] = record["customer"]["phone"]  
 record["address"] = record["customer"]["address"]  
  
 *# Fix incorrect vendor key names* for item in record["shopping\_cart"]:  
 if "vendor: " in item:  
 item["vendor"] = item.pop("vendor: ") *# Rename the incorrect key  
  
# Save the fixed JSON file*with open("TransactionData\_fixed.json", "w") as f:  
 for entry in data:  
 f.write(json.dumps(entry) + "\n")  
  
print("✅ JSON structure fixed! Saved as TransactionData\_fixed.json")

### B.2.2. Python Script to Split File

**================= Python Script to Split and Save Data into Separate JSON Files =================**

import json  
  
*# Load JSON file*with open("TransactionData\_fixed.json", "r") as f:  
 data = [json.loads(line) for line in f]  
  
*# Prepare lists for separate tables*customers = []  
transactions = []  
shopping\_cart = []  
  
*# Extract data into separate tables*for record in data:  
 *# Extract customer details (avoid duplicates)* customer = {  
 "customer\_id": record["customer"]["customer\_id"],  
 "name": record["customer"]["name"],  
 "email": record["customer"]["email"],  
 "phone": record["customer"]["phone"],  
 "address": record["customer"]["address"]  
 }  
 if customer not in customers:  
 customers.append(customer)  
  
 *# Extract transaction details* transactions.append({  
 "transaction\_id": record["transaction\_id"],  
 "customer\_id": record["customer"]["customer\_id"],  
 "transaction\_date": record["transaction\_date"],  
 "shipping\_method": record["shipping\_method"],  
 "total\_amount": record["total\_amount"]  
 })  
  
 *# Extract shopping cart items* for item in record["shopping\_cart"]:  
 shopping\_cart.append({  
 "cart\_id": record["transaction\_id"], *# Each shopping cart is linked to a transaction* "transaction\_id": record["transaction\_id"],  
 "product\_id": item["product\_id"],  
 "vendor": item.get("vendor", "Unknown Vendor"), *# Fix vendor key issue* "product\_category": item["product\_category"],  
 "product\_name": item["product\_name"],  
 "quantity": item["quantity"],  
 "price": item["price"]  
 })  
  
*# Save extracted data as separate JSON files*customer\_file = "Customers.json"  
transactions\_file = "Transactions.json"  
shopping\_cart\_file = "ShoppingCart.json"  
  
with open(customer\_file, "w") as f:  
 for entry in customers:  
 f.write(json.dumps(entry) + "\n")  
  
with open(transactions\_file, "w") as f:  
 for entry in transactions:  
 f.write(json.dumps(entry) + "\n")  
  
with open(shopping\_cart\_file, "w") as f:   
 for entry in shopping\_cart:  
 f.write(json.dumps(entry) + "\n")  
  
print(f"✅ Extracted and saved Customers data to: {customer\_file}")  
print(f"✅ Extracted and saved Transactions data to: {transactions\_file}")  
print(f"✅ Extracted and saved ShoppingCart data to: {shopping\_cart\_file}")

# Writing Queries and Running in GCP

C.1. Query to List Unique Customers

SELECT DISTINCT customer\_id, name, email, phone, address

FROM `wgu\_d607\_d644e25097898fed.Customers`

LIMIT 10;

A screenshot of a computer

Description automatically generated

C.2. Query to List All Items in a Customer’s Shopping Cart

SELECT

c.customer\_id,

c.name,

sc.product\_name,

sc.quantity,

sc.price,

(sc.quantity \* sc.price) AS total\_price

FROM `wgu\_d607\_d644e25097898fed.ShoppingCart` AS sc

JOIN `wgu\_d607\_d644e25097898fed.Transactions` AS t

ON sc.transaction\_id = t.transaction\_id

JOIN `wgu\_d607\_d644e25097898fed.Customers` AS c

ON t.customer\_id = c.customer\_id

WHERE c.customer\_id = (

SELECT customer\_id

FROM `wgu\_d607\_d644e25097898fed.Customers`

LIMIT 1

); -- Dynamically selects one customer

Additionally, you can use the below script to select a specific customer

SELECT

c.customer\_id,

c.name,

sc.product\_name,

sc.quantity,

sc.price,

(sc.quantity \* sc.price) AS total\_price

FROM `wgu\_d607\_d644e25097898fed.ShoppingCart` AS sc

JOIN `wgu\_d607\_d644e25097898fed.Transactions` AS t

ON sc.transaction\_id = t.transaction\_id

JOIN `wgu\_d607\_d644e25097898fed.Customers` AS c

ON t.customer\_id = c.customer\_id

WHERE c.customer\_id = 'cb4ef4b2-9114-461c-97cf-2d8452bab9ae'; -- Replace with specific customer\_id

**A screenshot of a computer

Description automatically generated**

C.3. Query to List Total Purchase Amounts for Customers (Desc)

SELECT

c.customer\_id,

c.name,

SUM(t.total\_amount) AS total\_spent

FROM `wgu\_d607\_d644e25097898fed.Transactions` AS t

JOIN `wgu\_d607\_d644e25097898fed.Customers` AS c

ON t.customer\_id = c.customer\_id

GROUP BY c.customer\_id, c.name

ORDER BY total\_spent DESC;

**A screenshot of a computer

Description automatically generated**

C.4. Panopto Video

[Link](https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cdee07a8-0795-4059-8c9e-b27f011fdf61)

# **Conclusion**

In conclusion, transitioning to a cloud-based data storage solution provides Alliah Company with enhanced scalability, reliability, and efficiency. The successful creation of the database schema in Google Cloud Platform, coupled with the accurate population of tables and execution of queries, demonstrates the robustness of the proposed solution. This process ensures that Alliah Company is well-equipped to handle increasing data volumes while maintaining data integrity and accessibility. By validating the functionality of the database and the accuracy of data processing, the migration to the cloud is proven to be a strategic move, supporting the company's long-term success.

References

Western Governors University. (n.d.). DPN2 Task 2: Cloud Data Solution Implementation. Retrieved February 8, 2025, from https://apps.cgp-oex.wgu.edu/wgulearning/course/course-v1:WGUx+OEX0419+v01