D607 – Cloud Databases (Task 2)

Morrell J. Parrish

Western Governors University

Table of Contents

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**.

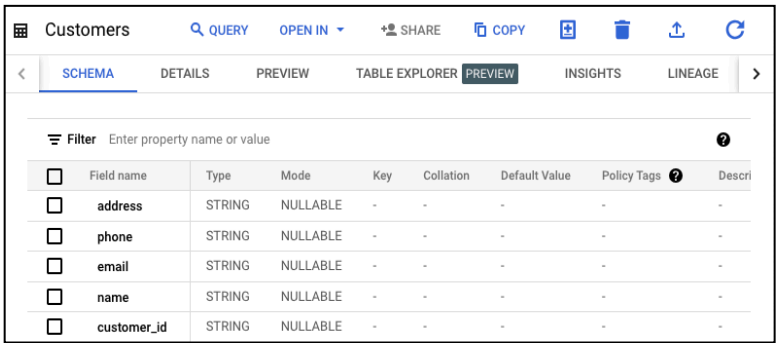
To achieve this, you will complete the following tasks using the provided **Alliah Company Scenario, Data Dictionary for Transaction Data, and Transaction Data** documents:

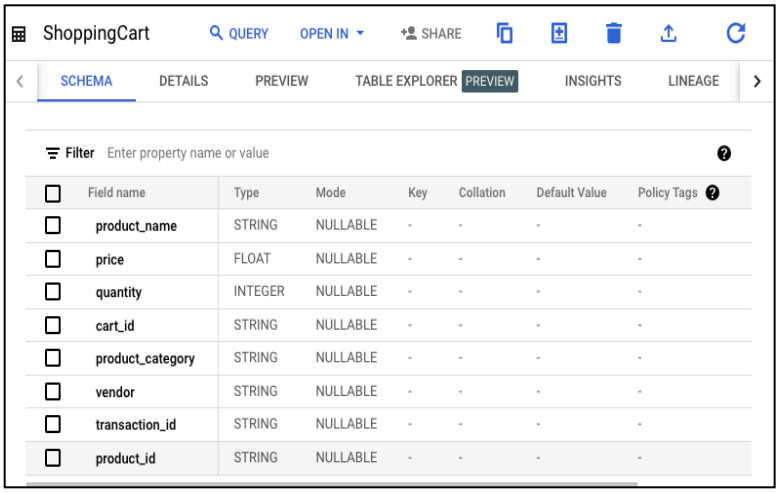
1. **Create the necessary database schema** in **Google Cloud Platform (GCP)** based on the optimal data store identified in Task 1.
2. **Populate the schema** with records from the **sample transaction dataset** to ensure data integrity.
3. **Execute and demonstrate queries** that validate the database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

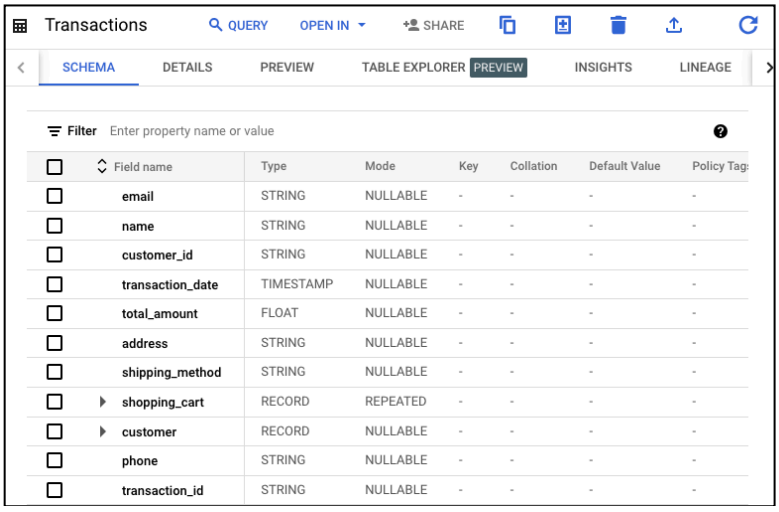
This process ensures that Alliah Company's **cloud migration is successful**, supporting **scalability, efficiency, and reliability** in their new cloud environment (Western Governors University, n.d.).

A. Google Cloud Platform Schema

A.1. Schema Objects







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,
```

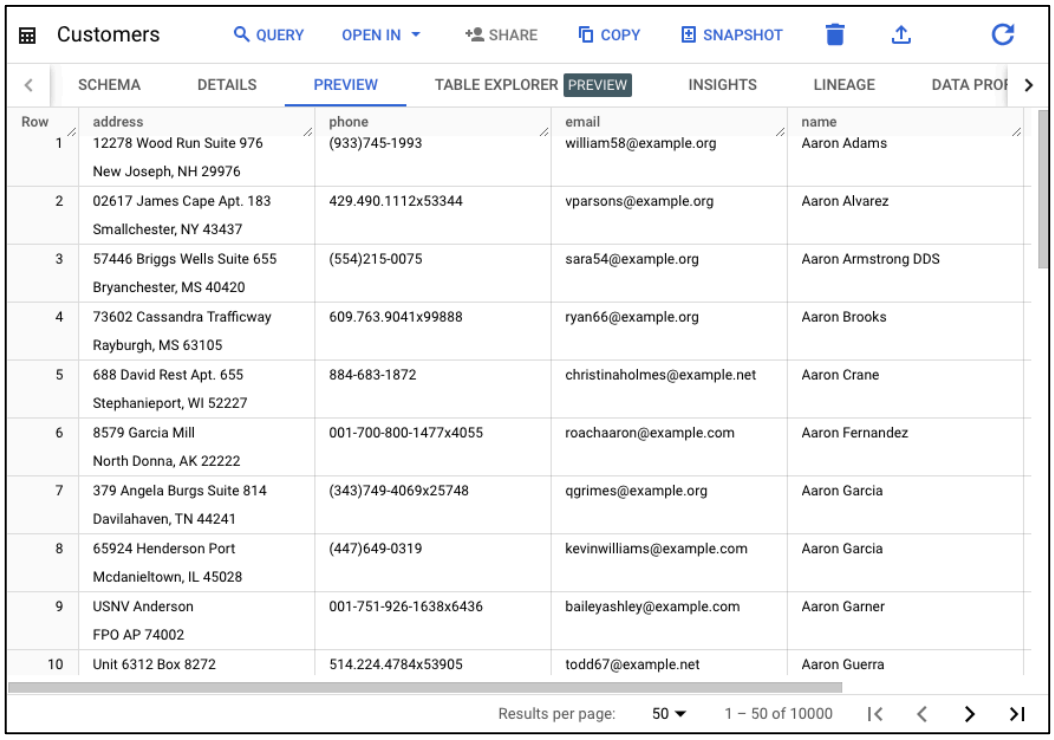
```
    quantity INT64,
    price FLOAT64
);
```

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.

B. Populating the Tables with Data

B.1. Record Import



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

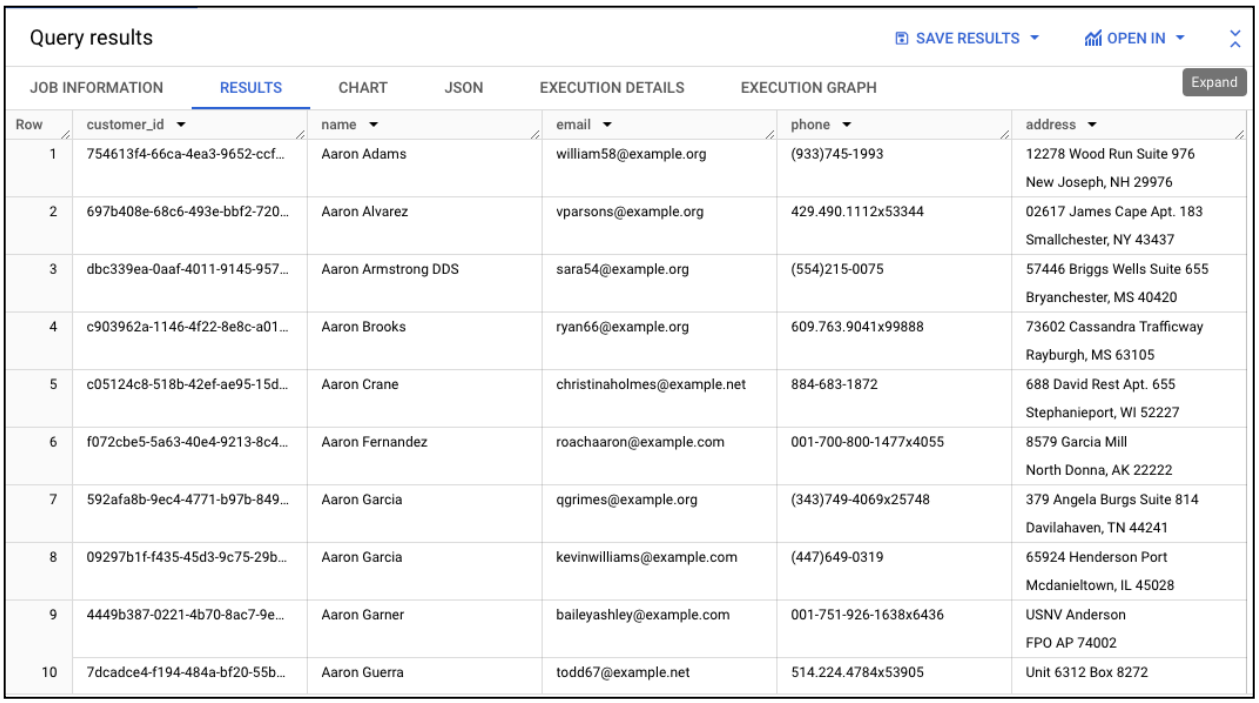
```
================= Python Script to Split and Save Data into Separate JSON Files =================

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}")
```

C. 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;
```



Query results

SAVE RESULTS OPEN IN

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Expand

Row	customer_id	name	email	phone	address
1	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	william58@example.org	(933)745-1993	12278 Wood Run Suite 976 New Joseph, NH 29976
2	697b408e-68c6-493e-bbf2-720...	Aaron Alvarez	vparsons@example.org	429.490.1112x53344	02617 James Cape Apt. 183 Smallchester, NY 43437
3	dbc339ea-0aaf-4011-9145-957...	Aaron Armstrong DDS	sara54@example.org	(554)215-0075	57446 Briggs Wells Suite 655 Bryanchester, MS 40420
4	c903962a-1146-4f22-8e8c-a01...	Aaron Brooks	ryan66@example.org	609.763.9041x99888	73602 Cassandra Trafficway Rayburgh, MS 63105
5	c05124c8-518b-42ef-ae95-15d...	Aaron Crane	christinaholmes@example.net	884-683-1872	688 David Rest Apt. 655 Stephanieport, WI 52227
6	f072cbe5-5a63-40e4-9213-8c4...	Aaron Fernandez	roachaaron@example.com	001-700-800-1477x4055	8579 Garcia Mill North Donna, AK 22222
7	592afa8b-9ec4-4771-b97b-849...	Aaron Garcia	qgrimes@example.org	(343)749-4069x25748	379 Angela Burgs Suite 814 Davilahaven, TN 44241
8	09297b1f-f435-45d3-9c75-29b...	Aaron Garcia	kevinwilliams@example.com	(447)649-0319	65924 Henderson Port Mcdanieltown, IL 45028
9	4449b387-0221-4b70-8ac7-9e...	Aaron Garner	baileyashley@example.com	001-751-926-1638x6436	USNV Anderson FPO AP 74002
10	7dcadce4-f194-484a-bf20-55b...	Aaron Guerra	todd67@example.net	514.224.4784x53905	Unit 6312 Box 8272

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
```

Query results

SAVE RESULTS OPEN IN

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	customer_id	name	product_name	quantity	price	total_price
1	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	Camera	1	944.99	944.99
2	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	Office Desk	1	249.99	249.99
3	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	Hair Dryer	2	56.99	113.98
4	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	Desk Chair	3	149.99	449.97
5	754613f4-66ca-4ea3-9652-ccf...	Aaron Adams	Inkjet Printer	3	209.99	629.97

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;
```

Query results

SAVE RESULTS OPEN IN

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EX

Row	customer_id	name	total_spent
1	21c24535-0a0d-4090-a298-aa7...	Kayla Jones	11959.89
2	fb4d59d0-609a-4102-89cc-d8b...	Daniel Meyer	11129.9
3	abece1e8-be9e-49ca-a6ec-c24...	Ashley Taylor	10319.89
4	2f370215-ee90-43e3-a54f-863...	Cynthia George	10293.88
5	4a611071-9d1d-4476-ace4-5c3...	Rebecca Ford	10264.91
6	c82883e3-9c26-4eee-9f90-2bf...	Emily Walker	10153.87
7	e825de81-74af-42b0-9d0f-c1c...	Juan Clark	10139.91
8	e434b95b-400a-43ff-b01a-d6c...	Daniel Glenn	10055.89
9	38f1a0e4-8c5c-4b9e-a838-d54...	Amber Anderson	9574.91
10	df37dd35-b5be-4e4f-9205-5b1...	Danielle Simon	9479.9

C.4. Panopto Video

Link

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