import time

from google.cloud import bigquery

from google.oauth2 import service\_account

from vertexai.generative\_models import FunctionDeclaration, GenerativeModel, Part, Tool

# Path to your service account key file

SERVICE\_ACCOUNT\_JSON = 'path/to/your/service-account-key.json'

# Create credentials object

credentials = service\_account.Credentials.from\_service\_account\_file(

SERVICE\_ACCOUNT\_JSON,

)

# Create BigQuery client

client = bigquery.Client(credentials=credentials, project=credentials.project\_id)

# Constants and Function Declarations

BIGQUERY\_PROJECT\_ID = "bigquery-public-data"

BIGQUERY\_DATASET\_ID = "thelook\_ecommerce"

list\_datasets\_func = FunctionDeclaration(

name="list\_datasets",

description="Get a list of datasets that will help answer the user's question",

parameters={"type": "object", "properties": {}},

)

list\_tables\_func = FunctionDeclaration(

name="list\_tables",

description="List tables in a dataset that will help answer the user's question",

parameters={

"type": "object",

"properties": {

"dataset\_id": {"type": "string", "description": "Dataset ID to fetch tables from."}

},

"required": ["dataset\_id"],

},

)

get\_table\_func = FunctionDeclaration(

name="get\_table",

description="Get information about a table, including the description, schema, and number of rows that will help answer the user's question. Always use the fully qualified dataset and table names.",

parameters={

"type": "object",

"properties": {

"table\_id": {"type": "string", "description": "Fully qualified ID of the table to get information about"}

},

"required": ["table\_id"],

},

)

sql\_query\_func = FunctionDeclaration(

name="sql\_query",

description="Get information from data in BigQuery using SQL queries",

parameters={

"type": "object",

"properties": {

"query": {"type": "string", "description": "SQL query on a single line that will help give quantitative answers to the user's question when run on a BigQuery dataset and table. In the SQL query, always use the fully qualified dataset and table names."}

},

"required": ["query"],

},

)

sql\_query\_tool = Tool(

function\_declarations=[list\_datasets\_func, list\_tables\_func, get\_table\_func, sql\_query\_func]

)

model = GenerativeModel(

"gemini-1.5-pro-001",

generation\_config={"temperature": 0},

tools=[sql\_query\_tool],

)

# Function to Handle User Prompt and Display Response

def handle\_user\_prompt(prompt):

chat = model.start\_chat()

# Append instructions to the user prompt

prompt += """

Please give a concise, high-level summary followed by detail in

plain language about where the information in your response is

coming from in the database. Only use information that you learn

from BigQuery, do not make up information.

"""

response = chat.send\_message(prompt)

response = response.candidates[0].content.parts[0]

api\_requests\_and\_responses = []

backend\_details = ""

function\_calling\_in\_process = True

while function\_calling\_in\_process:

try:

params = {}

for key, value in response.function\_call.args.items():

params[key] = value

if response.function\_call.name == "list\_datasets":

# Use the known public dataset

api\_response = [BIGQUERY\_DATASET\_ID]

api\_requests\_and\_responses.append([response.function\_call.name, params, api\_response])

if response.function\_call.name == "list\_tables":

dataset\_id = f"{BIGQUERY\_PROJECT\_ID}.{params['dataset\_id']}"

api\_response = client.list\_tables(dataset\_id)

api\_response = str([table.table\_id for table in api\_response])

api\_requests\_and\_responses.append([response.function\_call.name, params, api\_response])

if response.function\_call.name == "get\_table":

table\_id = f"{BIGQUERY\_PROJECT\_ID}.{params['table\_id']}"

api\_response = client.get\_table(table\_id)

api\_response = api\_response.to\_api\_repr()

api\_requests\_and\_responses.append([

response.function\_call.name,

params,

[str(api\_response.get("description", "")), str([column["name"] for column in api\_response["schema"]["fields"]])]

])

api\_response = str(api\_response)

if response.function\_call.name == "sql\_query":

job\_config = bigquery.QueryJobConfig(maximum\_bytes\_billed=100000000)

try:

cleaned\_query = params["query"].replace("\\n", " ").replace("\n", "").replace("\\", "")

query\_job = client.query(cleaned\_query, job\_config=job\_config)

api\_response = query\_job.result()

api\_response = str([dict(row) for row in api\_response])

api\_response = api\_response.replace("\\", "").replace("\n", "")

api\_requests\_and\_responses.append([response.function\_call.name, params, api\_response])

except Exception as e:

# If the execution fails, return the SQL logic for manual execution

print("BigQuery job execution failed. Returning SQL logic for manual execution.")

api\_response = cleaned\_query

api\_requests\_and\_responses.append([response.function\_call.name, params, api\_response])

response = chat.send\_message(

Part.from\_function\_response(

name=response.function\_call.name,

response={"content": api\_response},

),

)

response = response.candidates[0].content.parts[0]

backend\_details += f"- Function call:\n - Function name: `{api\_requests\_and\_responses[-1][0]}`\n"

backend\_details += f" - Function parameters: `{api\_requests\_and\_responses[-1][1]}`\n"

backend\_details += f" - API response: `{api\_requests\_and\_responses[-1][2]}`\n\n"

except AttributeError:

function\_calling\_in\_process = False

full\_response = response.text

return full\_response, backend\_details

# Example of using the function to handle a prompt

prompt = "What percentage of orders are returned?"

full\_response, backend\_details = handle\_user\_prompt(prompt)

print("Response:")

print(full\_response)

print("\nFunction calls, parameters, and responses:")

print(backend\_details)