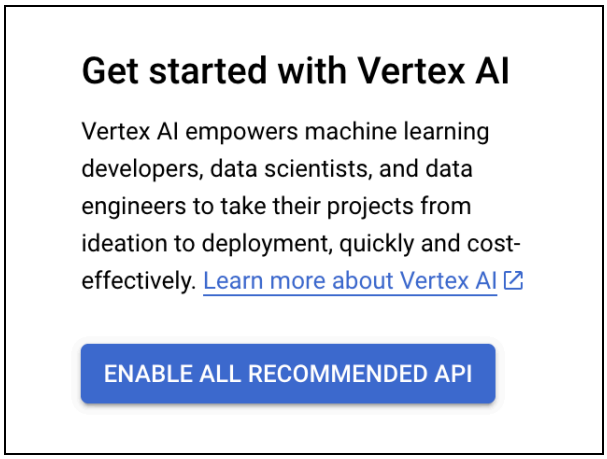
Agent Builder Advanced: BigQuery Structured Data with Cloud Function

Task 1. Set up your environment

Enable the Vertex AI API

1. In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI** or Utilize the search bar to find **Vertex AI.**
2. Within the "Get Started with Vertex AI" section, locate and click on the option that says "ENABLE ALL RECOMMENDED API."



Task 2. Create a table in BigQuery

Get the Data:

- Download the test data from this repository:

  https://github.com/themlguy-tf/generative-ai.git

  Then Navigate to the file agent-builder/sales.csv

  OR

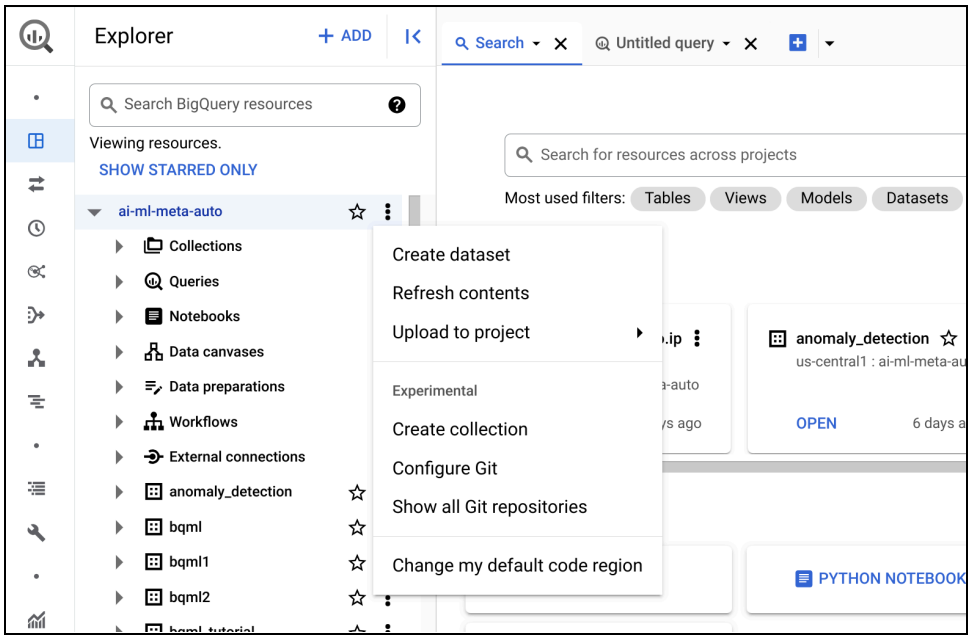
- Direct link to the file:

  https://github.com/themlguy-tf/generative-ai/blob/main/agent-builder/sales.csv

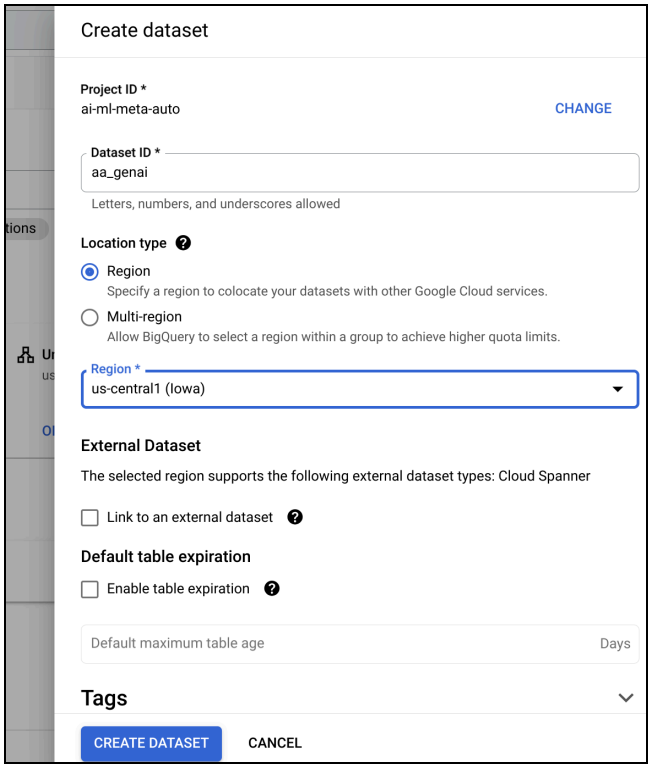
Create a dataset named (aa_genai) in BigQuery.

Follow the steps below:

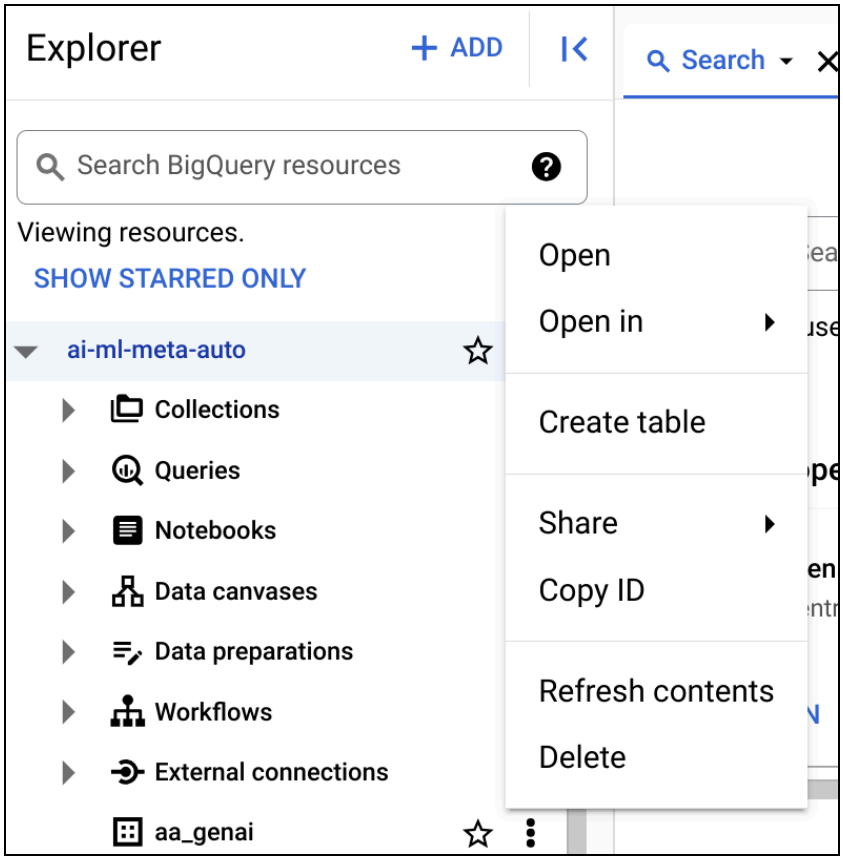
1. Navigate to the BigQuery Studio, and then create a dataset.



2. Create a dataset.
    a. **Name:** aa_genai
    b. **Region:** us-central1a
    c. Other default settings remain unchanged

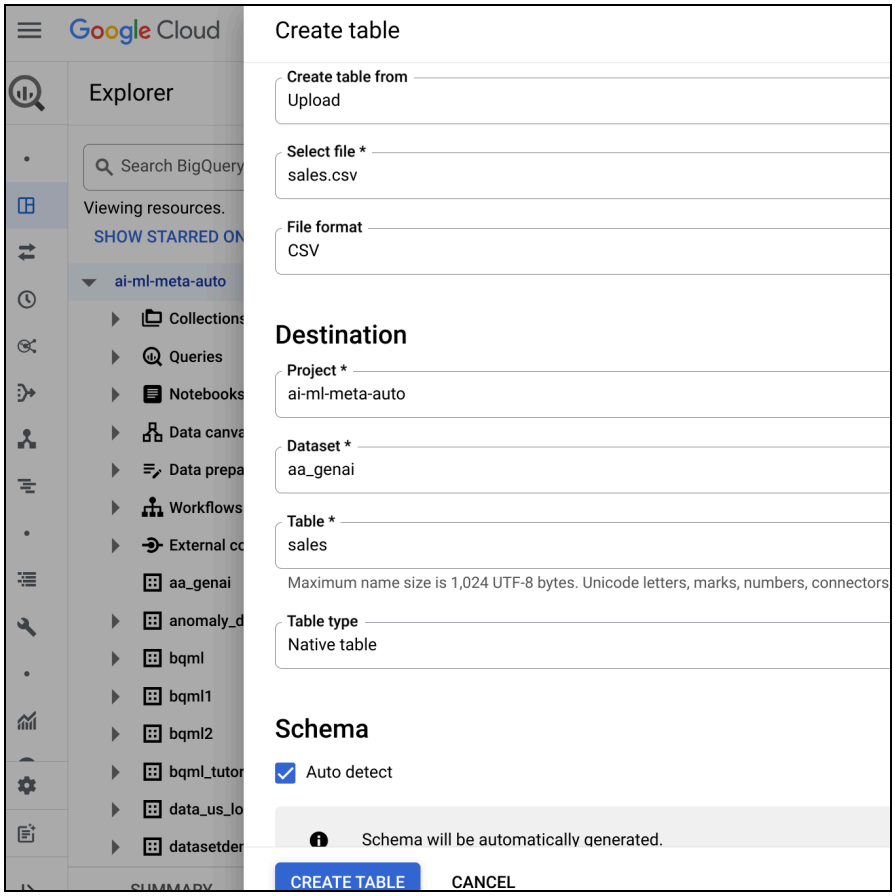


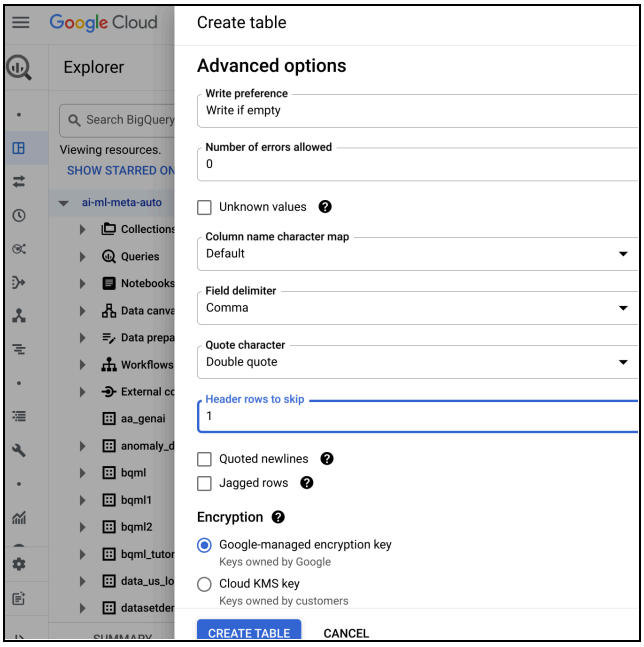
3. Click on the three vertical dots on the right of the dataset that was created in the previous step ***aa_genai***



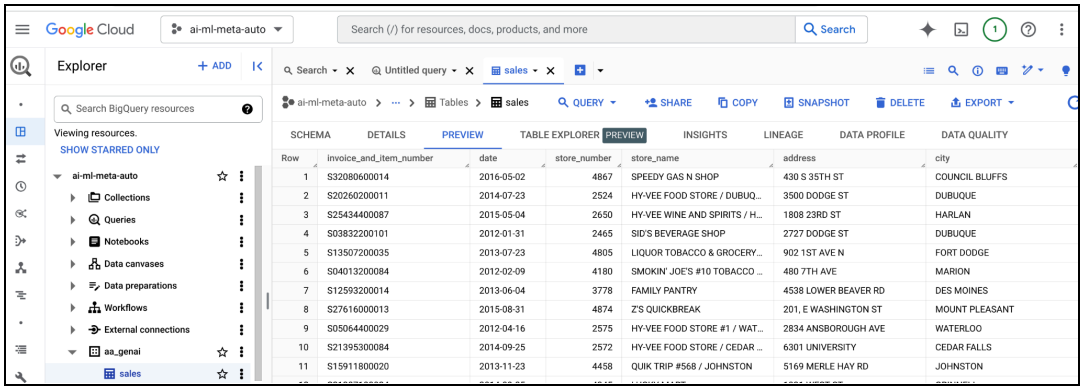
4. Create a table with the sales.csv that was downloaded from the github repository.

Please see the settings for creating a table. Please ensure the “header rows to skip” value set to 1 in the Advanced setting, as shown below.


Google Cloud
Explorer
Search BigQuery
Viewing resources.
SHOW STARRED ON
ai-ml-meta-auto
Collections
Queries
Notebooks
Data canva
Data prepa
Workflows
External cc
aa_genai
anomaly_d
bqml
bqml1
bqml2
bqml_tutor
data_us_lo
datasetder
SUMMARY
Create table
Create table from
Upload
Select file *
sales.csv
File format
CSV
Destination
Project *
ai-ml-meta-auto
Dataset *
aa_genai
Table *
sales
Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors
Table type
Native table
Schema
Auto detect
Schema will be automatically generated.
CREATE TABLE
CANCEL


Google Cloud
Explorer
Search BigQuery
Viewing resources.
SHOW STARRED ON
ai-ml-meta-auto
Collections
Queries
Notebooks
Data canva
Data prepa
Workflows
External cc
aa_genai
anomaly_d
bqml
bqml1
bqml2
bqml_tutor
data_us_lo
datasetder
SUMMARY
Create table
Advanced options
Write preference
Write if empty
Number of errors allowed
0
Unknown values
Column name character map
Default
Field delimiter
Comma
Quote character
Double quote
Header rows to skip
1
Quoted newlines
Jagged rows
Encryption
Google-managed encryption key
Keys owned by Google
Cloud KMS key
Keys owned by customers
CREATE TABLE
CANCEL

5. Validate the table is created



Task 3. Create a tool to list the tables from the BigQuery

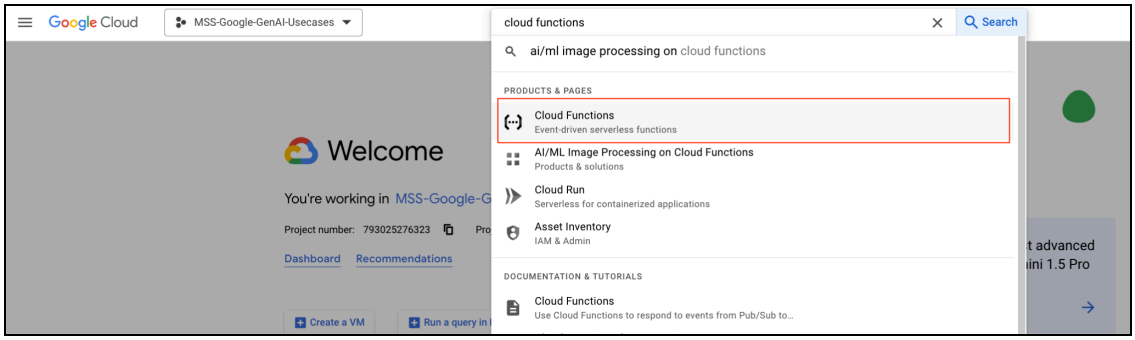
Create a tool to list the tables from the BigQuery

To create a tool, 1st go to the cloud console, and create a function named **list_tables_new**.

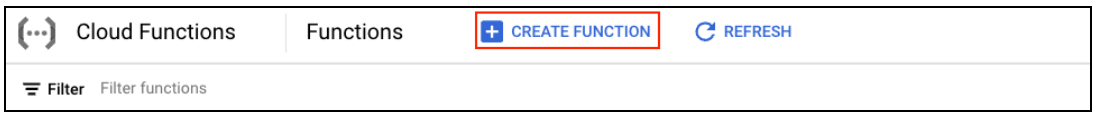
Note: pass the Flask endpoint in this function, to get the table names from BigQuery

Follow the steps below:

6. Search for the **cloud functions** in the GCP console, and select Cloud Functions from the search result



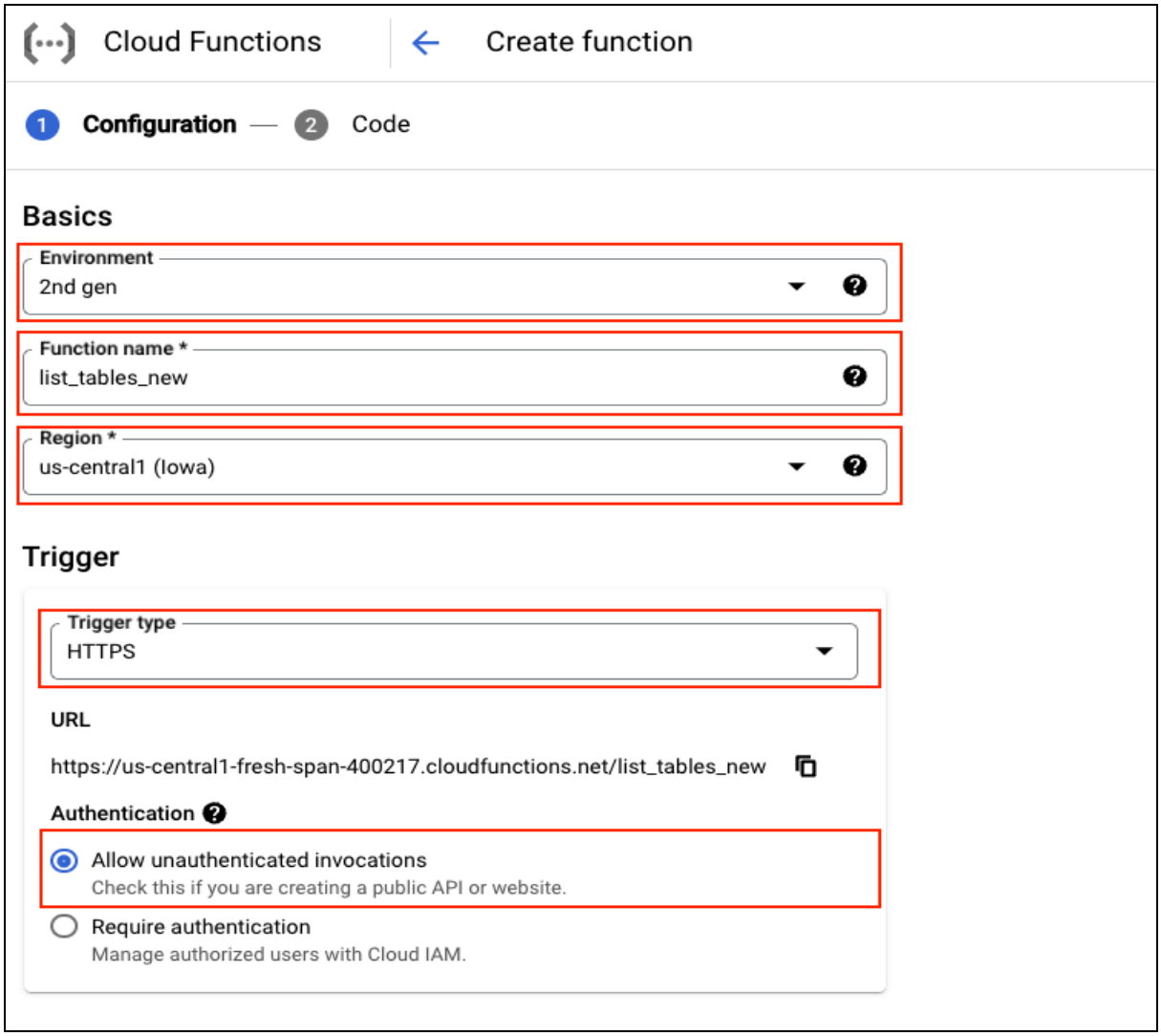
7. Create the Cloud Function
    a. Click on CREATE FUNCTION



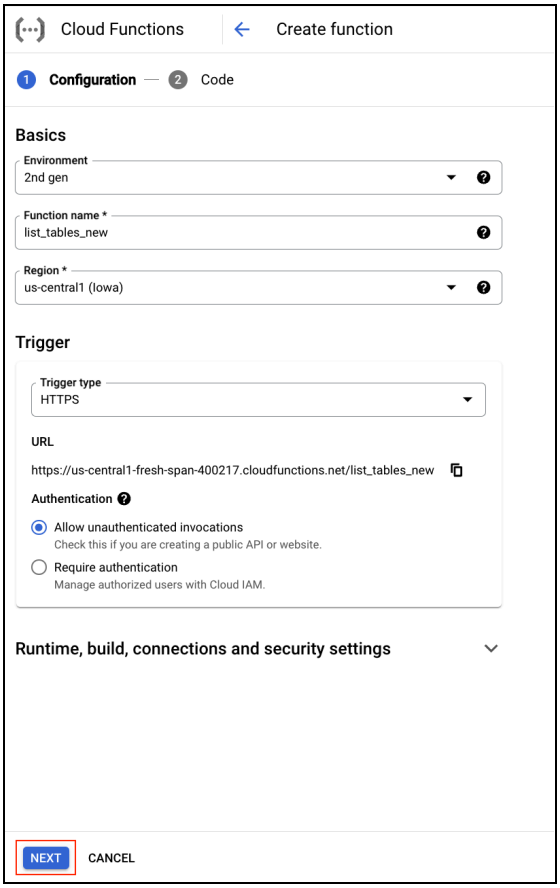
3. Update the parameters, and set the parameters as marked in **RED.**

Note: Function name is a name entered by the user

a. Environment : **Cloud Run Function (Default)**
b. Function name : **list_tables_new**
c. Region : **us-central1 (Iowa)**
d. Trigger type : **HTTPS**
e. Select "**Allow unauthenticated invocations**"



4. Click on NEXT



5. Update the code to point to a Flask endpoint.
(Note that the Function name from previous step, Entry point and the flask endpoint should be same)

a. Select Python 3.12 in Runtime
b. Update Entrypoint to list_tables_new
c. Update the python code in main.py

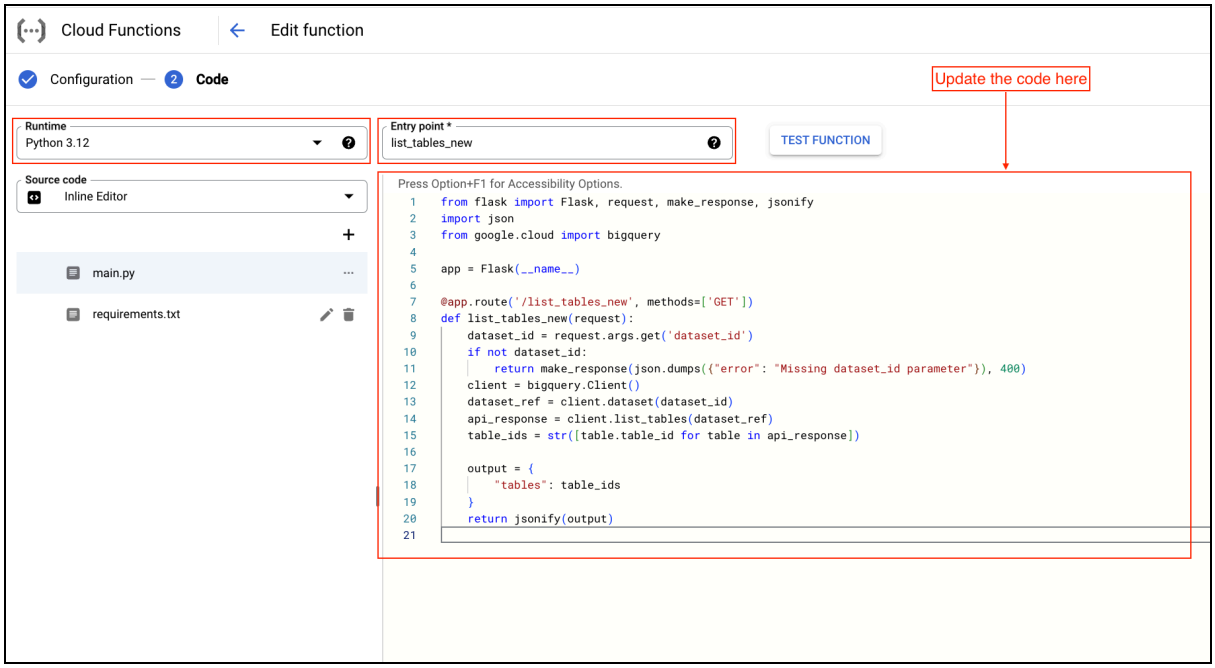
This code lists all the tables present in the input BigQuery Database.

Code for main.py: Note: "Please verify code indentation after copying."

—----------

```
from flask import Flask, request, make_response, jsonify
import json
from google.cloud import bigquery

app = Flask(__name__)
```

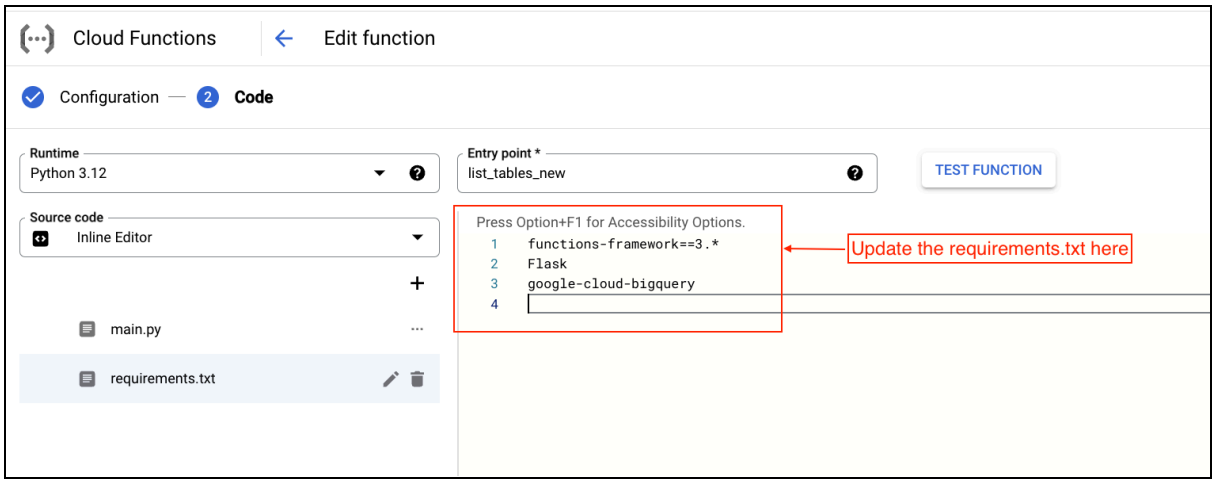
```
@app.route('/list_tables_new', methods=['GET'])
def list_tables_new(request):
    dataset_id = request.args.get('dataset_id')
    if not dataset_id:
        return make_response(json.dumps({"error": "Missing dataset_id parameter"}), 400)
    client = bigquery.Client()
    dataset_ref = client.dataset(dataset_id)
    api_response = client.list_tables(dataset_ref)
    table_ids = str([table.table_id for table in api_response])

    output = {
        "tables": table_ids
    }
    return jsonify(output)
```



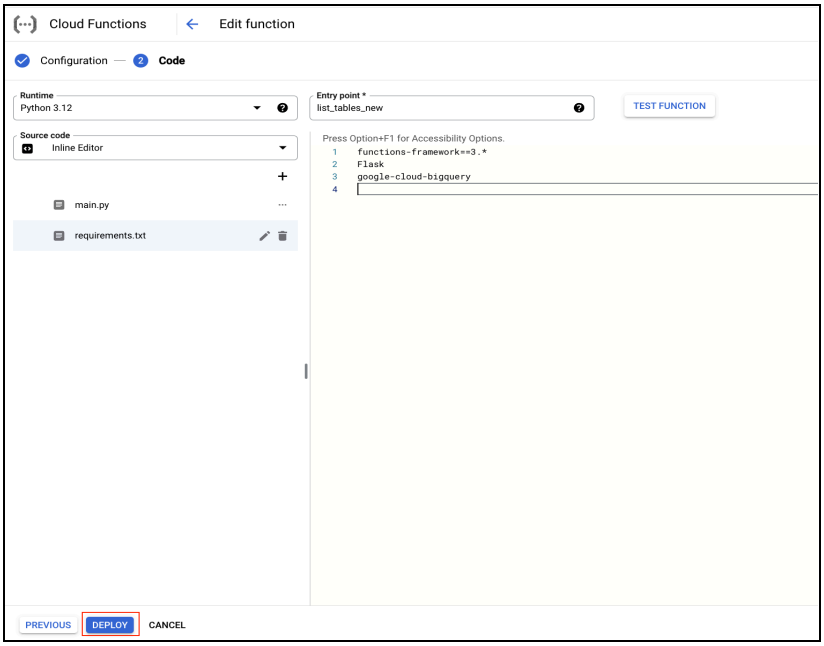
d. Update the requirements.txt file
e. **Requirements.txt**

functions-framework==3.*

Flask
google-cloud-bigquery



6. Click on Deploy



This will deploy the cloud function that can list all the tables present in the BigQuery database
Debug Steps: Make sure the Entry point and function name are same

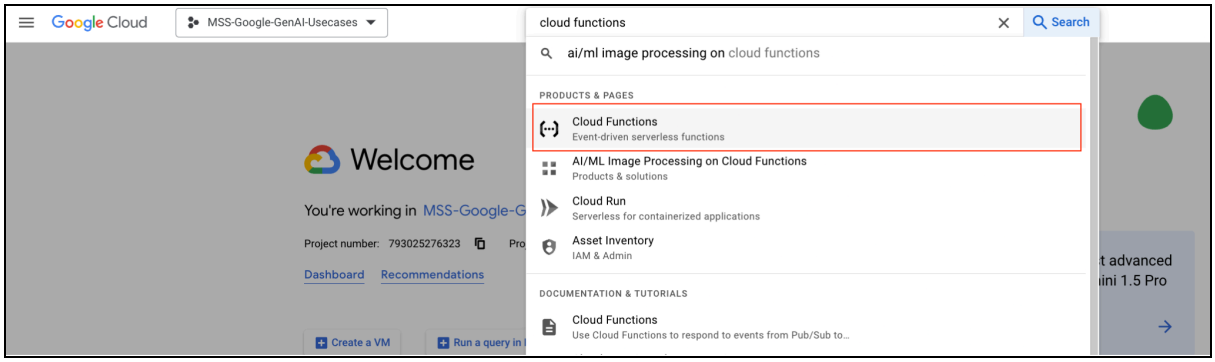
Step 2: Create a tool to get the table schema of a specific table from the BigQuery

To create a tool, 1st go to the cloud console, and create a function named **get_table_new**.

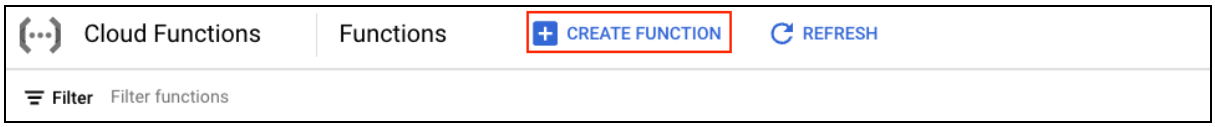
Note: pass the Flask endpoint in this function, to get the table schema from BigQuery

Follow the steps below:

1. Search for the cloud functions in the GCP console, and select **Cloud Run Functions** from the search result



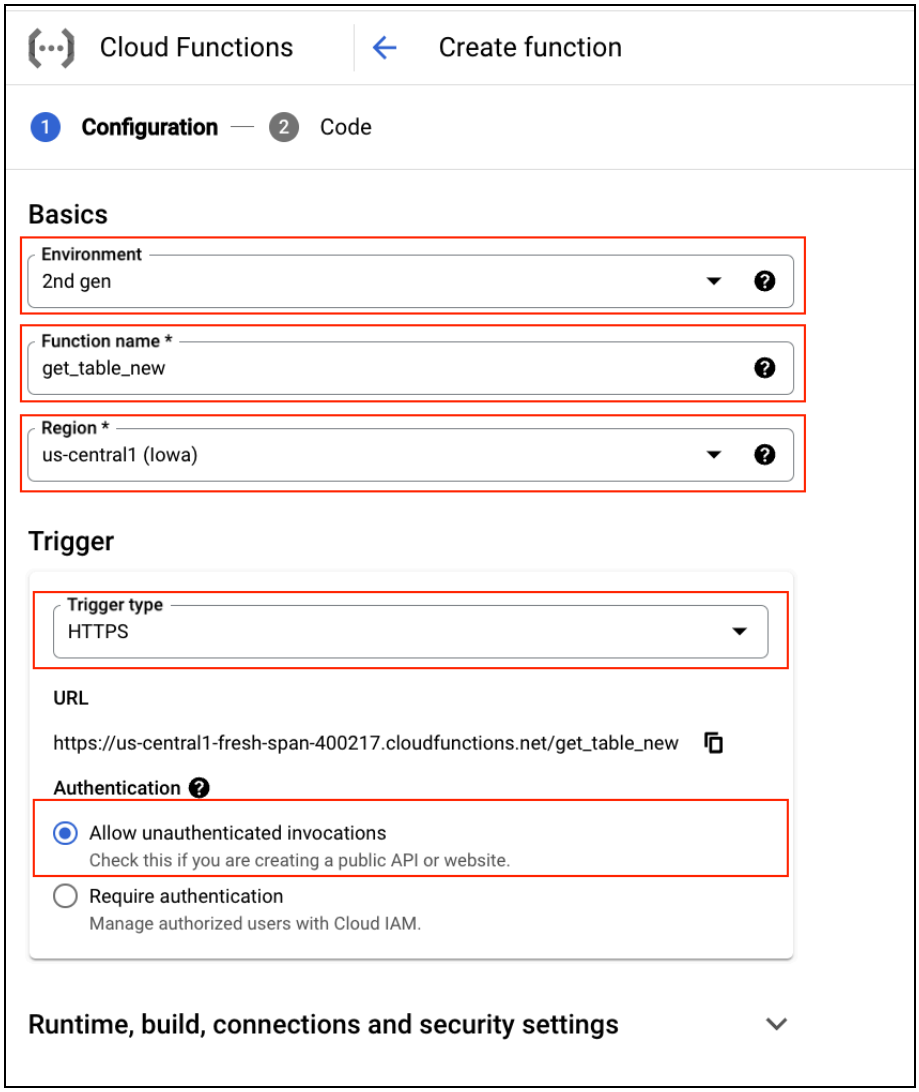
2. Create the Cloud Function
    f. Click on CREATE FUNCTION



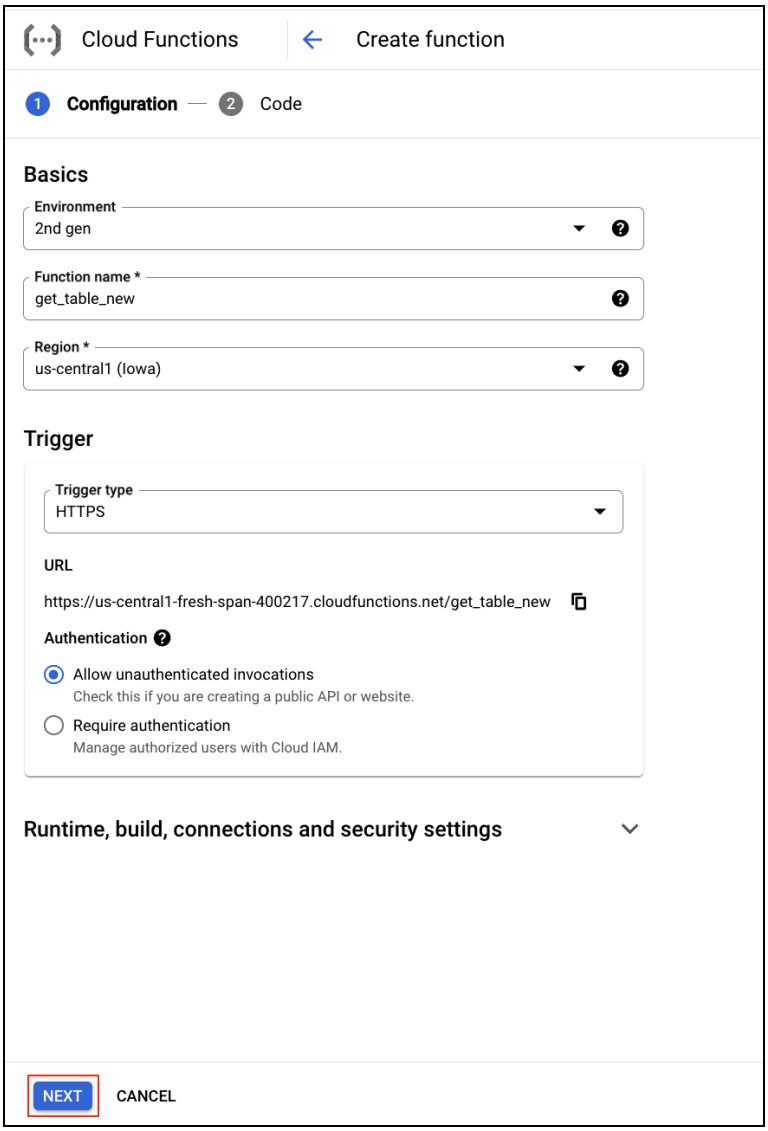
3. Update the parameters and set the parameters as marked in **RED.**

Note: Function name is a name entered by the user

b. Environment : **Cloud Run Function (Default)**
g. Function name : **get_table_new**
h. Region : **us-central1 (Iowa)**
i. Trigger type : **HTTPS**
j. Select "**Allow unauthenticated invocations**"



4. Click on NEXT



5. Update the code to point to a Flask endpoint.
Note that the Function name from previous step, Entry point and the flask endpoint should be same

f. Select Python 3.12 in Runtime
g. Update Entrypoint to get_table_new
h. Update the python code in main.py

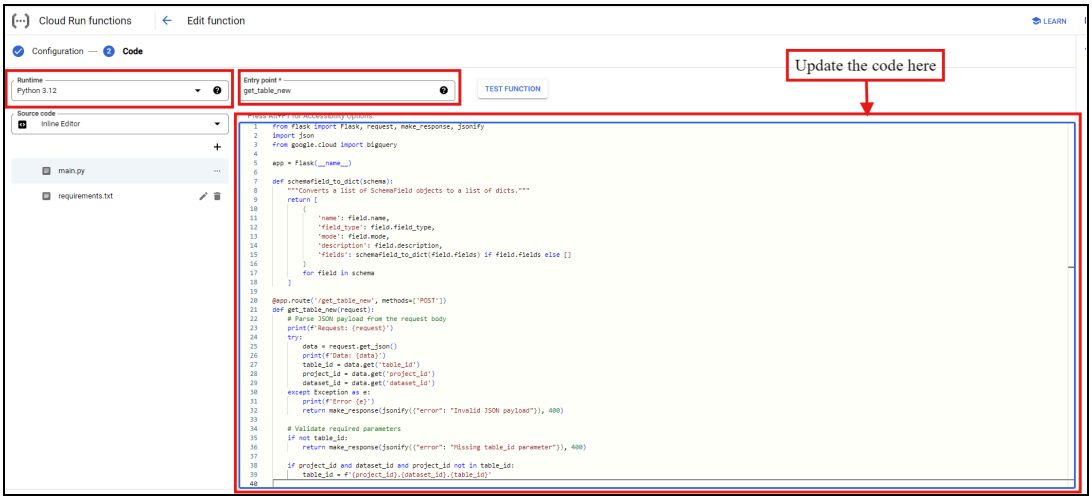
Code for main.py: Note: "Please verify code indentation after copying."

—----------

```
from flask import Flask, request, make_response, jsonify
import json
```

```
from google.cloud import bigquery

app = Flask(__name__)

def schemafield_to_dict(schema):
    """Converts a list of SchemaField objects to a list of dicts."""
    return [
        {
            'name': field.name,
            'field_type': field.field_type,
            'mode': field.mode,
            'description': field.description,
            'fields': schemafield_to_dict(field.fields) if field.fields else []
        }
        for field in schema
    ]

@app.route('/get_table_new', methods=['GET'])
def get_table_new(request):
    # Parse JSON payload from the request body
    print(f'Request: {request}')
    try:
        table_id = request.args.get('table_id')
    except Exception as e:
        print(f'Error {e}')
        return make_response(jsonify({"error": "Invalid JSON payload"}), 400)

    # Validate required parameters
    if not table_id:
        return make_response(jsonify({"error": "Missing table_id parameter"}), 400)

    print(f'Table id: {table_id}')

    # Initialize BigQuery client
    client = bigquery.Client()

    # Attempt to retrieve the table reference
    try:
        table_ref = client.get_table(table_id)
```

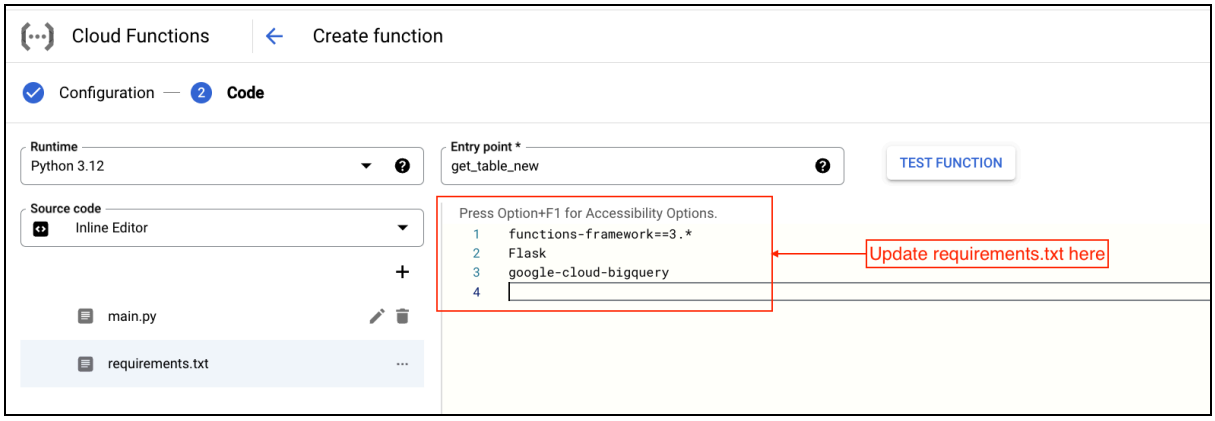
```
except Exception as e:
    return make_response(jsonify({"error": str(e)}), 404)

api_response = {
    "description": table_ref.description,
    "schema": schemafield_to_dict(table_ref.schema),
    "num_rows": table_ref.num_rows,
}

return jsonify(api_response)
```



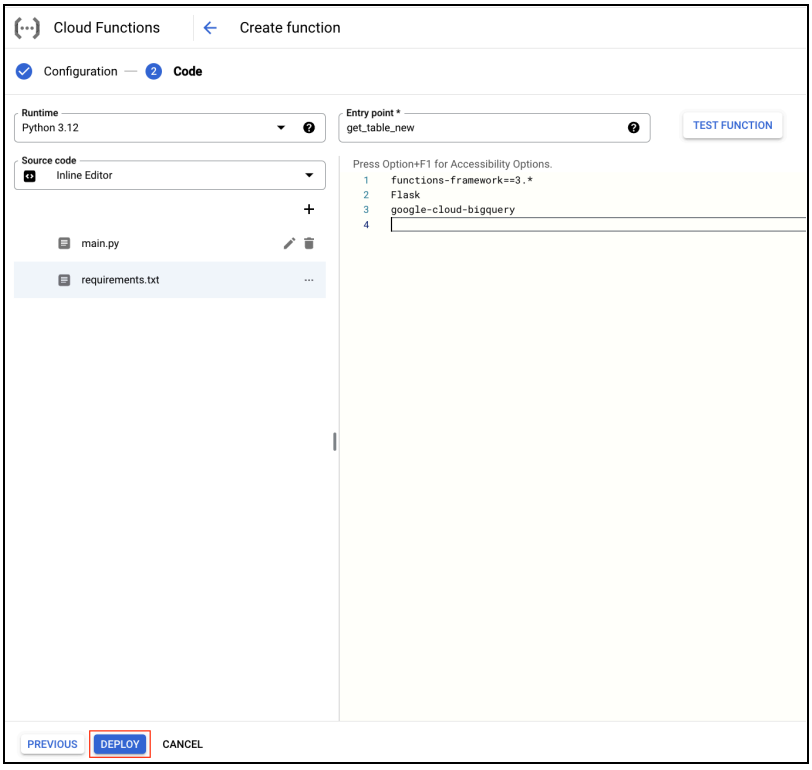
i. Update the requirements.txt file

Requirements.txt

functions-framework==3.*
Flask
google-cloud-bigquery



6. Click on Deploy



This will deploy the cloud function that will get the table schema from BigQuery

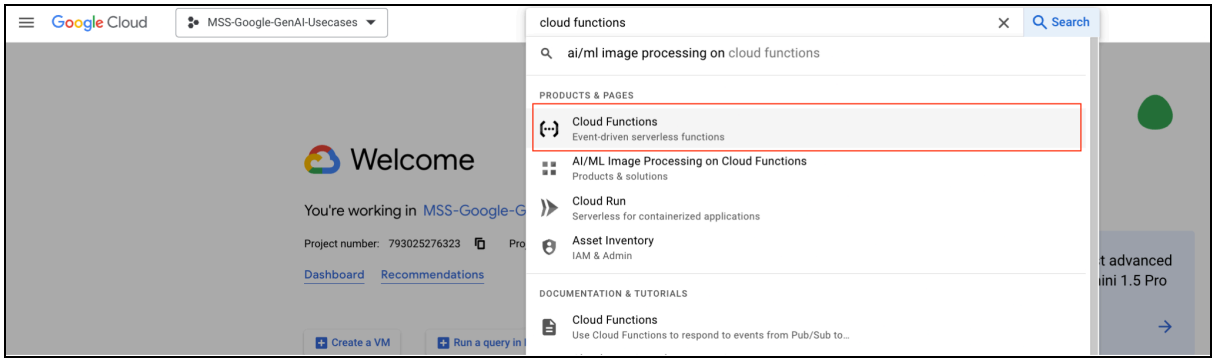
Step 3: Create a tool to execute the SQL query on BigQuery and return the corresponding result

To create a tool, 1st go to the cloud console, and create a function named **sql_query_new**.

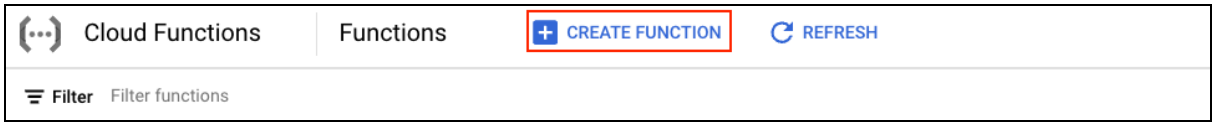
Note: pass the Flask endpoint in this function, to get the result of executing the SQL query on BigQuery

Follow the steps below:

1. Search for the cloud functions in the GCP console, and select Cloud Functions from the search result



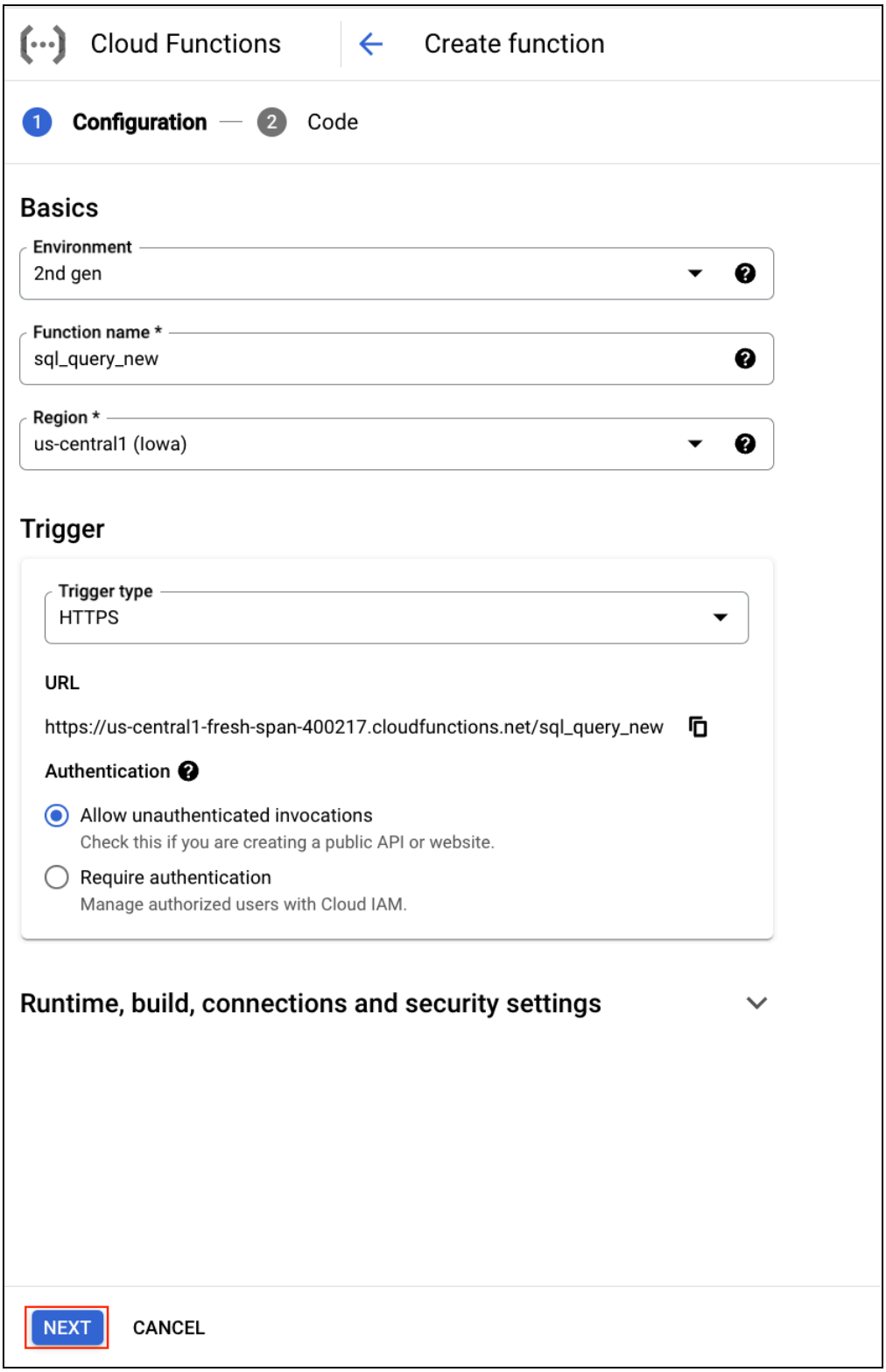
2. Create the Cloud Function
    k. Click on CREATE FUNCTION



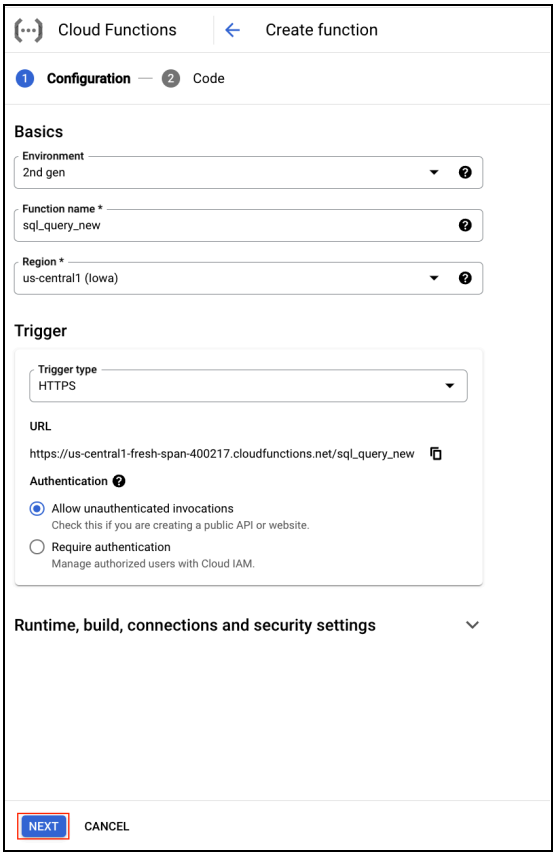
3. Update the parameters

Note: Function name is a name entered by the user

c. Environment : **Cloud Run Function (Default)**
l. Function name : **sql_query_new**
m. Region : **us-central1 (Iowa)**
n. Trigger type : **HTTPS**
o. Select "**Allow unauthenticated invocations**"



4. Click on NEXT



5. Update the code to point to a Flask endpoint.
Note that the Function name from previous step, Entry point and the flask endpoint should be same

j. Select Python 3.12 in Runtime
k. Update Entrypoint to **sql_query_new**
l. Update the python code in main.py

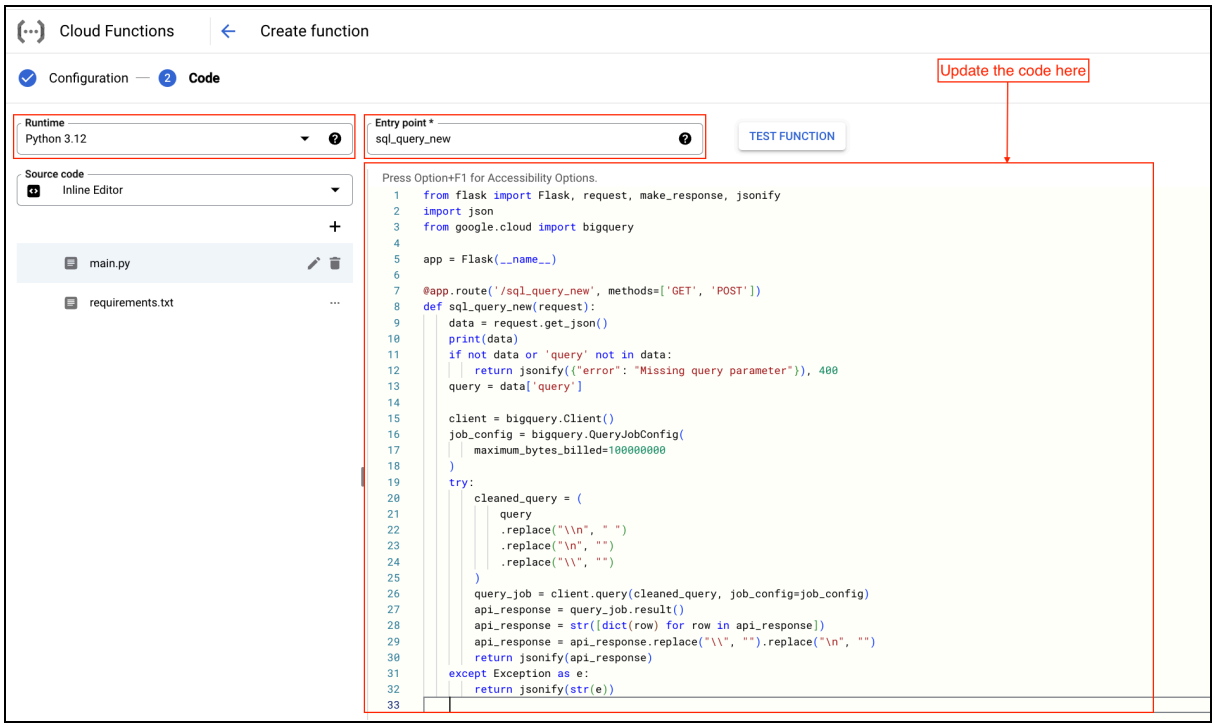
Code for main.py: Note: "Please verify code indentation after copying."

—----------

```
from flask import Flask, request, make_response, jsonify
import json
from google.cloud import bigquery

app = Flask(__name__)
```

```
@app.route('/sql_query_new', methods=['GET', 'POST'])
def sql_query_new(request):
    data = request.get_json()
    print(data)
    if not data or 'query' not in data:
        return jsonify({"error": "Missing query parameter"}), 400
    query = data['query']

    client = bigquery.Client()
    job_config = bigquery.QueryJobConfig(
        maximum_bytes_billed=100000000
    )
    try:
        cleaned_query = (
            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", " ")
        return jsonify(api_response)
    except Exception as e:
        return jsonify(str(e))
```

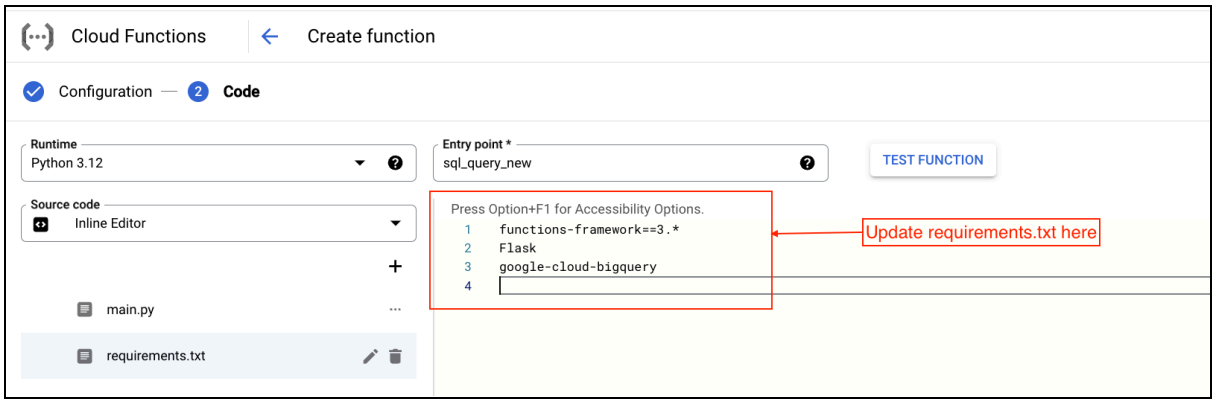
—----------



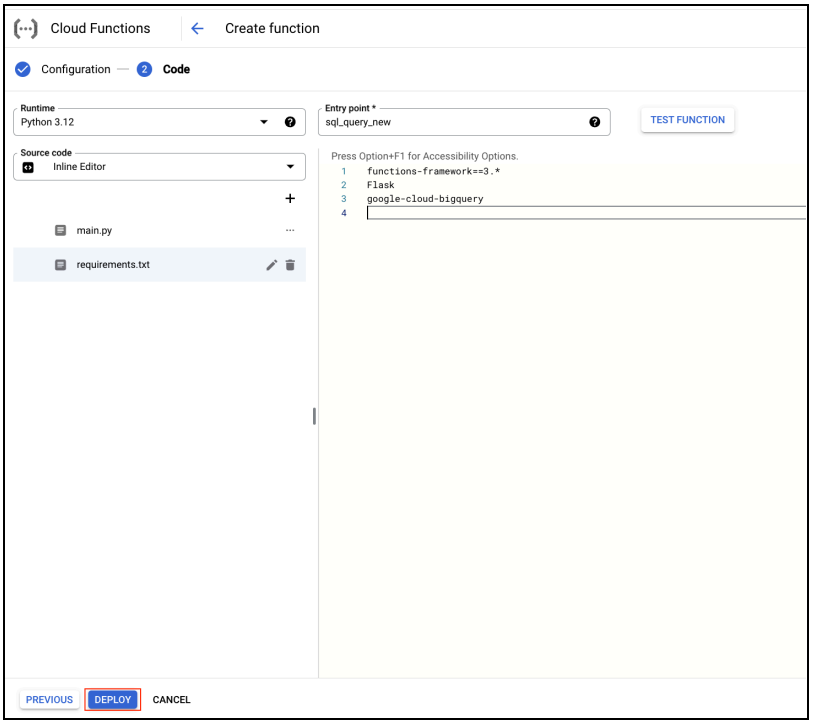
m. Update the requirements.txt file

Requirements.txt

functions-framework==3.*
Flask
google-cloud-bigquery



6. Click on Deploy



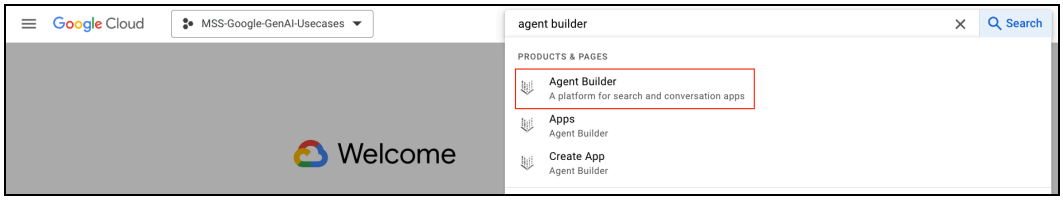
This will deploy the cloud function that will execute the SQL query on BigQuery and get the corresponding result.

Task 4. Create an Agent in Agent Builder

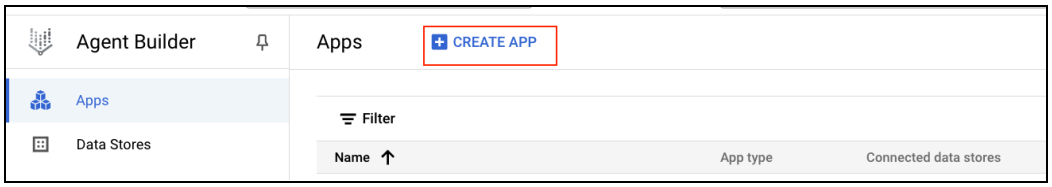
Step 4: Create an OpenAPI tool for the list_tables_new cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of list_tables_new cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

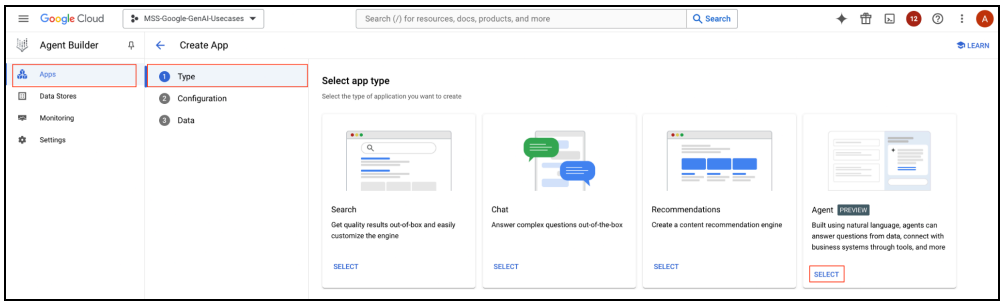
1. Go to Agent Builder on Google cloud console



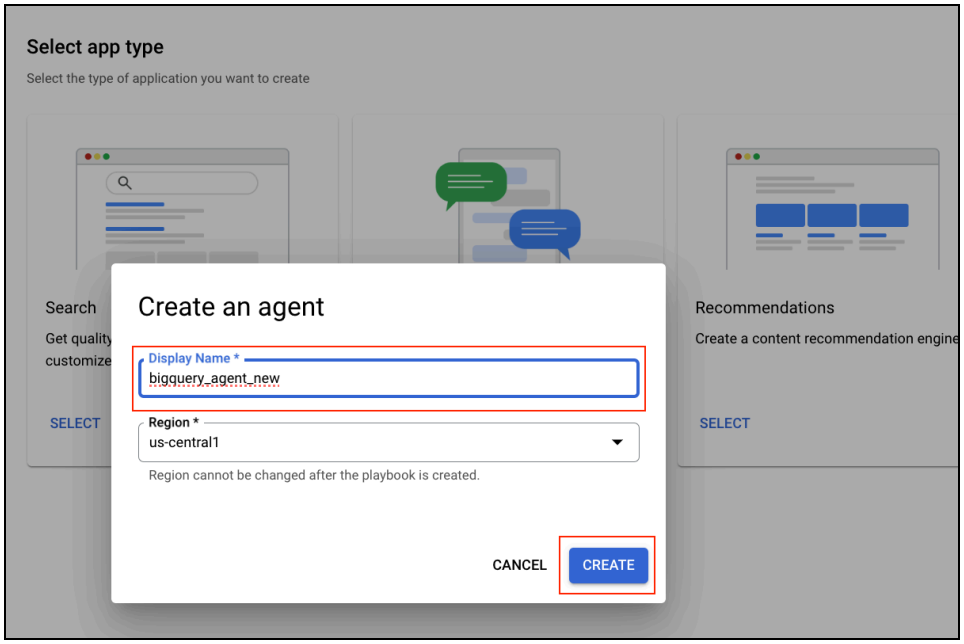
2. Click on Create App to start building a new Agent builder



3. Select Apps > Type > Agent

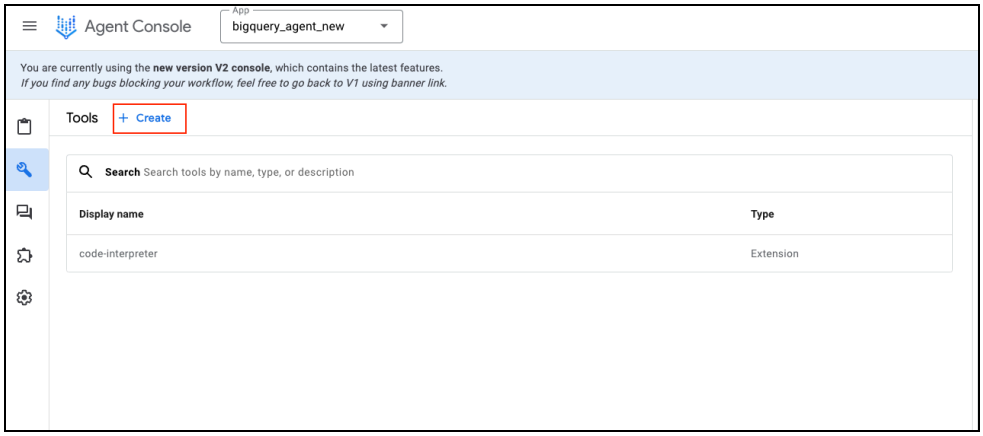


4. Update the Display name of the agent to **bigquery_agent_new** and click on Create



5. Select Tools from the left

6. Click on Create



7. Update the following parameters:
    a. Tool name : list_tables_new

b. Type : OpenAPI
c. Update the Description to:
*Note: Replace the project id, dataset id and table id with the correct values

This tool can be used to get the list of tables in the dataset
Project_id: '<PROJECT_ID>'
Dataset_id: '<Dataset_id>'
Table_id: '<Table_id>'

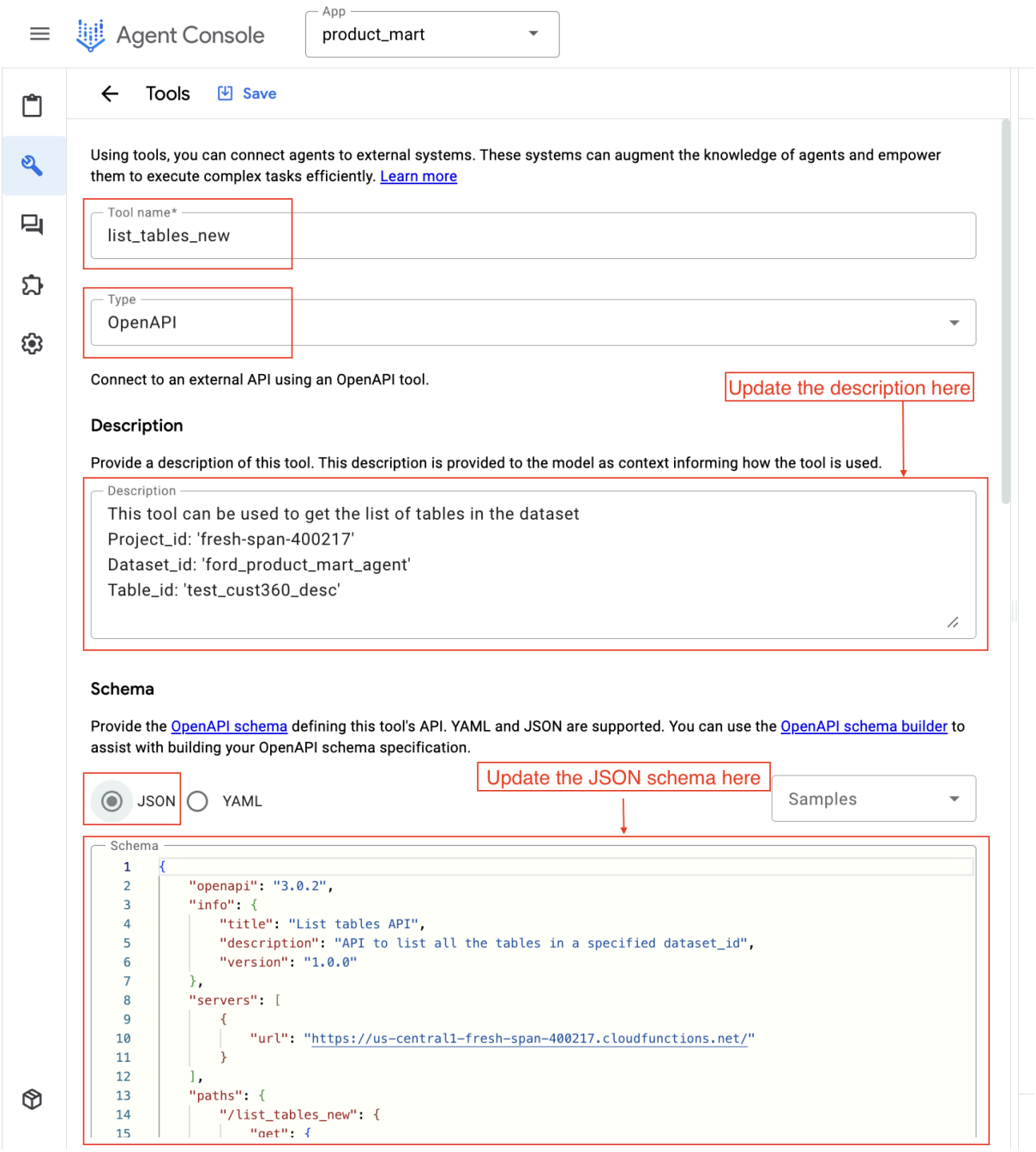
Example for your reference:
Project_id: 'fresh-span-400217'
Dataset_id: 'aa_genai'
Table_id: 'sales'

d. Select JSON under schema
e. Update the schema to:
* Note: Use the url corresponding to list_tables_new cloud function created previously in the schema. Set the paths to list_tables_new.
Please change the "URL" below- it will be the url of your cloud functions.

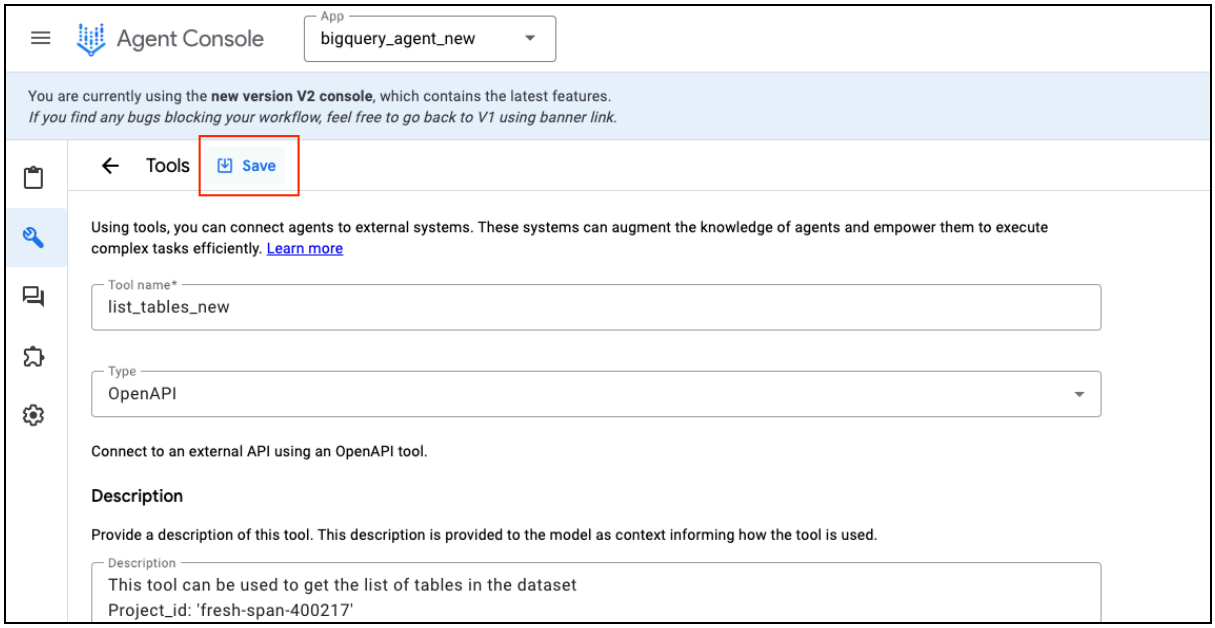
```
{
 "openapi": "3.0.2",
 "info": {
  "title": "List tables API",
  "description": "API to list all the tables in a specified dataset_id",
  "version": "1.0.0"
 },
 "servers": [
  {
   "url": "https://us-central1-fresh-span-400217.cloudfunctions.net/"
  }
 ],
 "paths": {
  "/list_tables_new": {
   "get": {
    "summary": "List all tables present in dataset_id",
    "description": "Returns the names of all tables for the specified dataset_id",
    "operationId": "list_tables",
    "parameters": [
```

```
        {
          "in": "query",
          "name": "dataset_id",
          "description": "The identifier of the dataset",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "List of all tables",
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "dataset_id": {
                    "type": "string",
                    "description": "The identifier of the dataset"
                  },
                  "tables": {
                    "type": "array",
                    "items": {
                      "type": "string"
                    },
                    "description": "The list of all table names in dataset_id"
                  }
                }
              }
            }
          }
        },
        "400": {
              "description": "Bad request - Invalid query parameter or missing required parameter"
        },
        "404": {
```

```
          "description": "Dataset not found"
        }
      }
    }
  }
 }
}
```



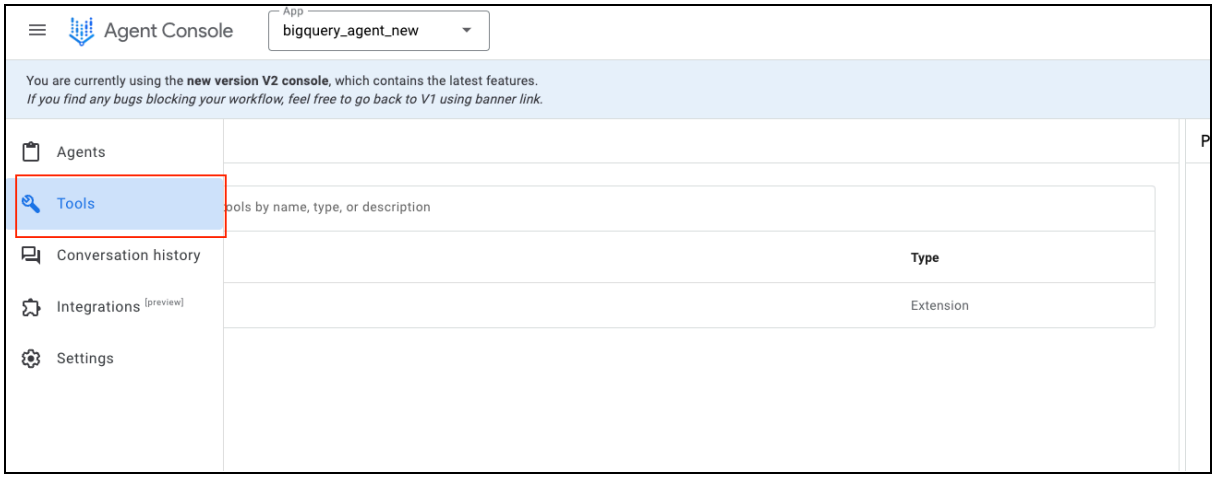
8. Save the Tool



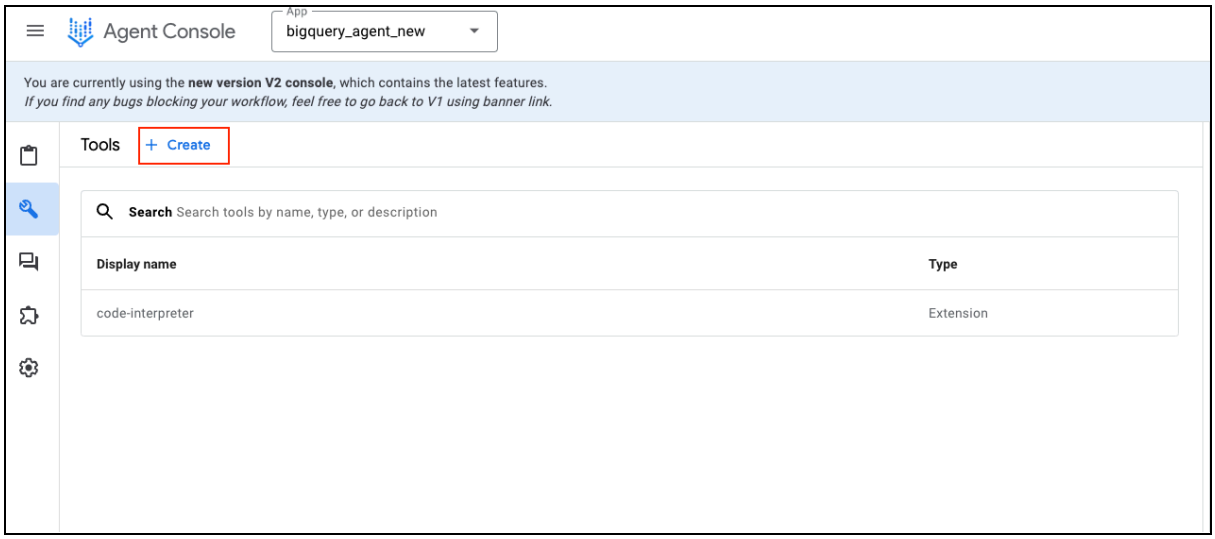
Step 5: Create an OpenAPI tool for the get_table_new cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of get_table_new cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

1. Select Tools from the left



2. Click on Create



3. Update the following parameters:
    a. Tool name : get_table_new
    b. Type : OpenAPI
    c. Update the Description to :

    Use this tool to get information about a table, including the description, schema, and number of rows that will help answer the user's question. Always include the Project_id and Dataset_id in Table_id You must pass the exact complete table ID (as provided below) along with the project ID and dataset ID for accurate results. use `fresh-span-400217.ford_product_mart_agent.cust_desc_flat` table

    Project_id: '<PROJECT_ID>'
    Dataset_id: '<Dataset_id>'
    Table_id: '<Table_id>'

    Example for your reference:
    Project_id: 'fresh-span-400217'
    Dataset_id: 'aa_genai'
    Table_id: 'sales'

    *Note: Replace the project id, dataset id and table id with the correct values

    d. Select JSON under schema

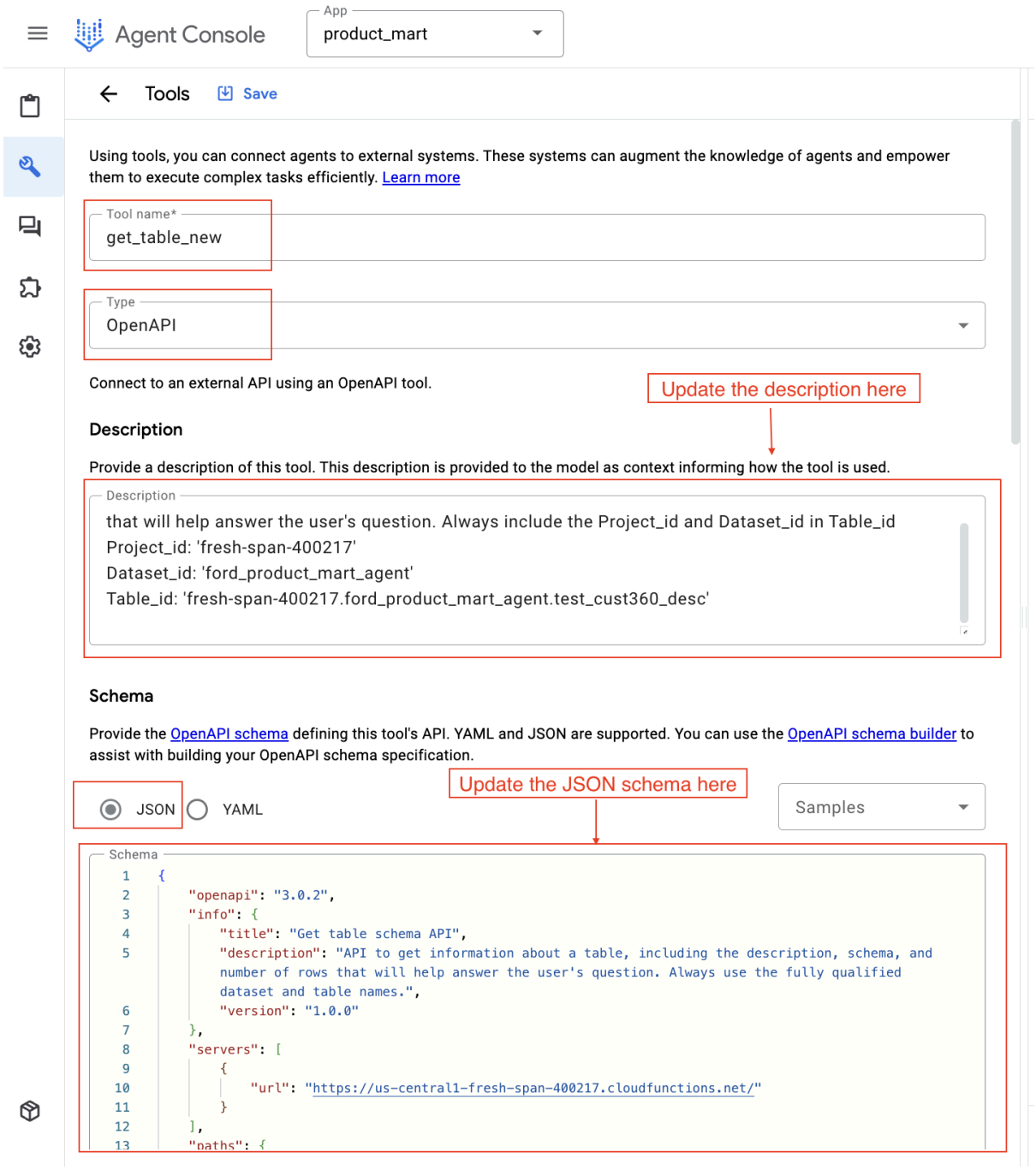
e. Update the schema to:

*Note: Use the url corresponding to **get_table_new** cloud function created previously in the schema. Set the paths to get_table_new.

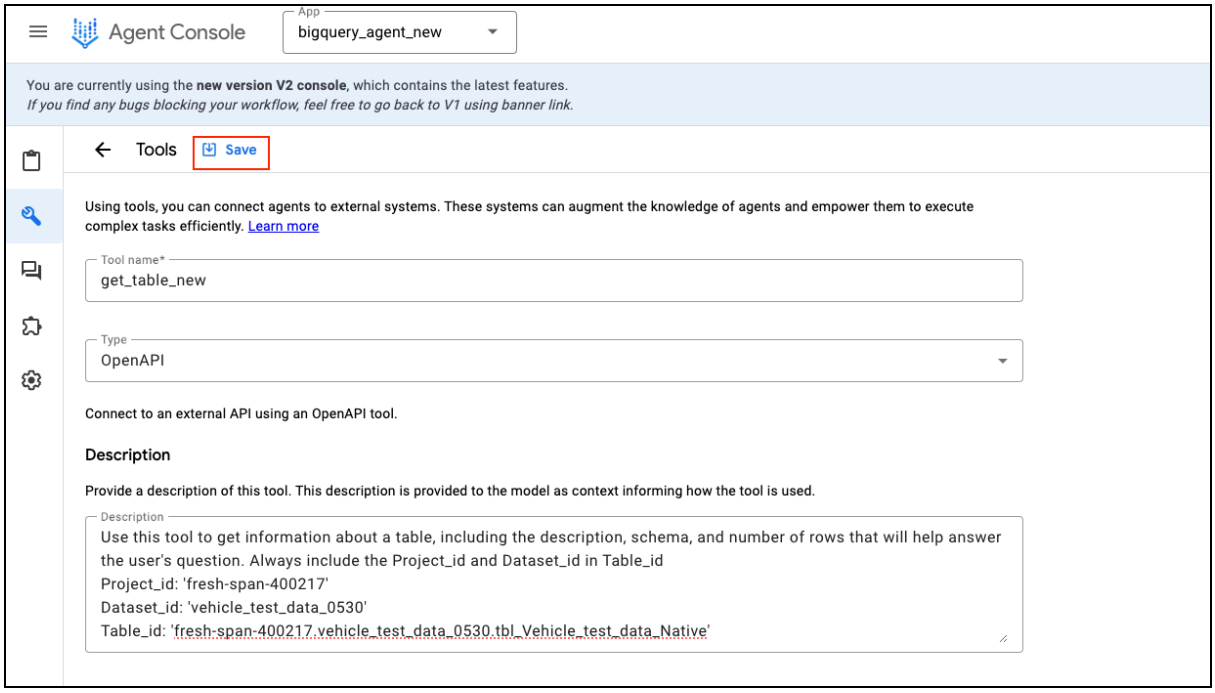
```
{
 "openapi": "3.0.2",
 "info": {
  "title": "Get table schema API",
      "description": "API to get information about a table, including the description, schema, and number of rows that will help answer the user's question. Always use the fully qualified dataset and table names.",
  "version": "1.0.0"
 },
 "servers": [
  {
   "url": "https://us-central1-fresh-span-400217.cloudfunctions.net/"
  }
 ],
 "paths": {
  "/get_table_new": {
   "get": {
    "summary": "Get information about a table",
     "description": "Retrieves metadata for the specified table_id, including the description, schema, and number of rows.",
    "operationId": "get_table_new",
    "parameters": [
     {
      "in": "query",
      "name": "table_id",
       "description": "The identifier of the table along with the Project_id and Dataset_id",
       "required": true,
       "schema": {
        "type": "string"
       }
     }
    ],
    "responses": {
     "200": {
```

```
"description": "Detailed information about the table",
"content": {
 "application/json": {
  "schema": {
   "type": "object",
   "properties": {
    "description": {
     "type": "string",
     "description": "A brief description of the table"
    },
    "schema": {
     "type": "array",
     "items": {
      "type": "object",
      "properties": {
       "name": {
        "type": "string",
        "description": "Name of the field in the table"
       },
       "type": {
        "type": "string",
        "description": "Data type of the field"
       },
       "mode": {
        "type": "string",
        "description": "Mode of the field (e.g., NULLABLE, REQUIRED)"
       },
       "description": {
        "type": "string",
        "description": "Description of the field"
       }
      }
     },
     "description": "Schema details of the table"
    },
    "num_rows": {
     "type": "integer",
     "description": "Total number of rows in the table"
    }
```

```
            }
           }
          }
         }
        },
        "400": {
                "description": "Bad request - Invalid query parameter or missing required parameter"
        },
        "404": {
          "description": "Table not found"
        }
       }
     }
   }
 }
}
```



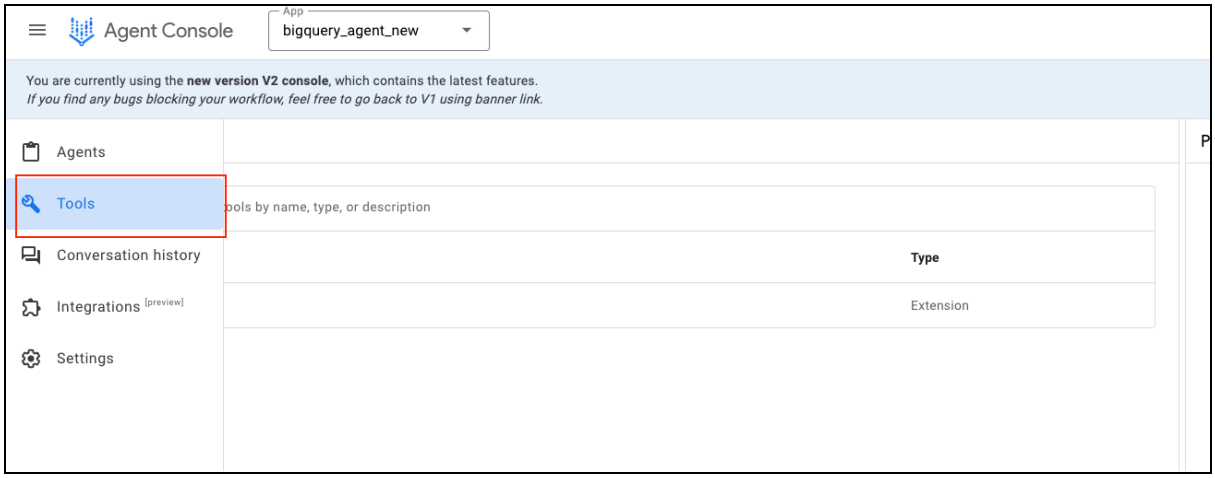
4. Save the Tool



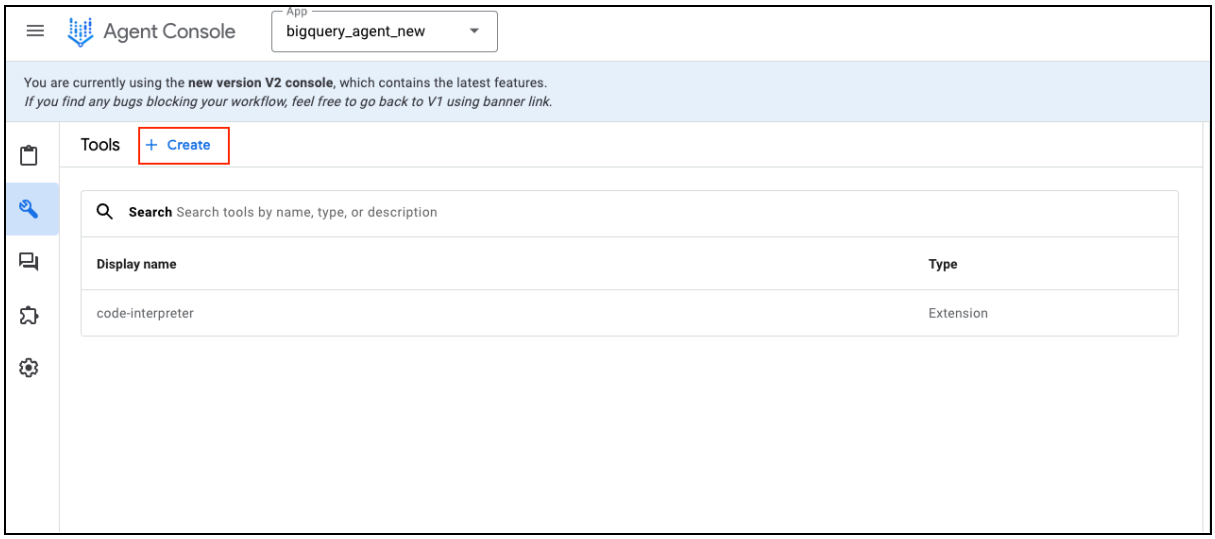
Step 6: Create an OpenAPI tool for the get_sql_query_new cloud function

This tool will be used to build the agent. A description of the tool and the corresponding schema should be provided here. The output format of get_sql_query_new cloud function and the input of this tool should be the same. If there is a mismatch, there will be errors.

1. Select Tools from the left



2. Click on Create



3. Update the following parameters:
    f. Tool name : sql_query_new
    g. Type : OpenAPI
    h. Update the Description to :
    *Note: Replace the project id, dataset id and table id with the correct values

    Use this tool to get information about a table, including the description, schema, and number of rows that will help answer the user's question. Always include the Project_id and Dataset_id in Table_id

    Project_id: '<PROJECT_ID>'
    Dataset_id: '<Dataset_id>'
    Table_id: '<PROJECT_ID>.<Dataset_id>.<Table_id>'

    Example for your reference:
    Project_id: 'fresh-span-400217'
    Dataset_id: 'aa_genai'
    Table_id: 'fresh-span-400217.aa_genai.sales'

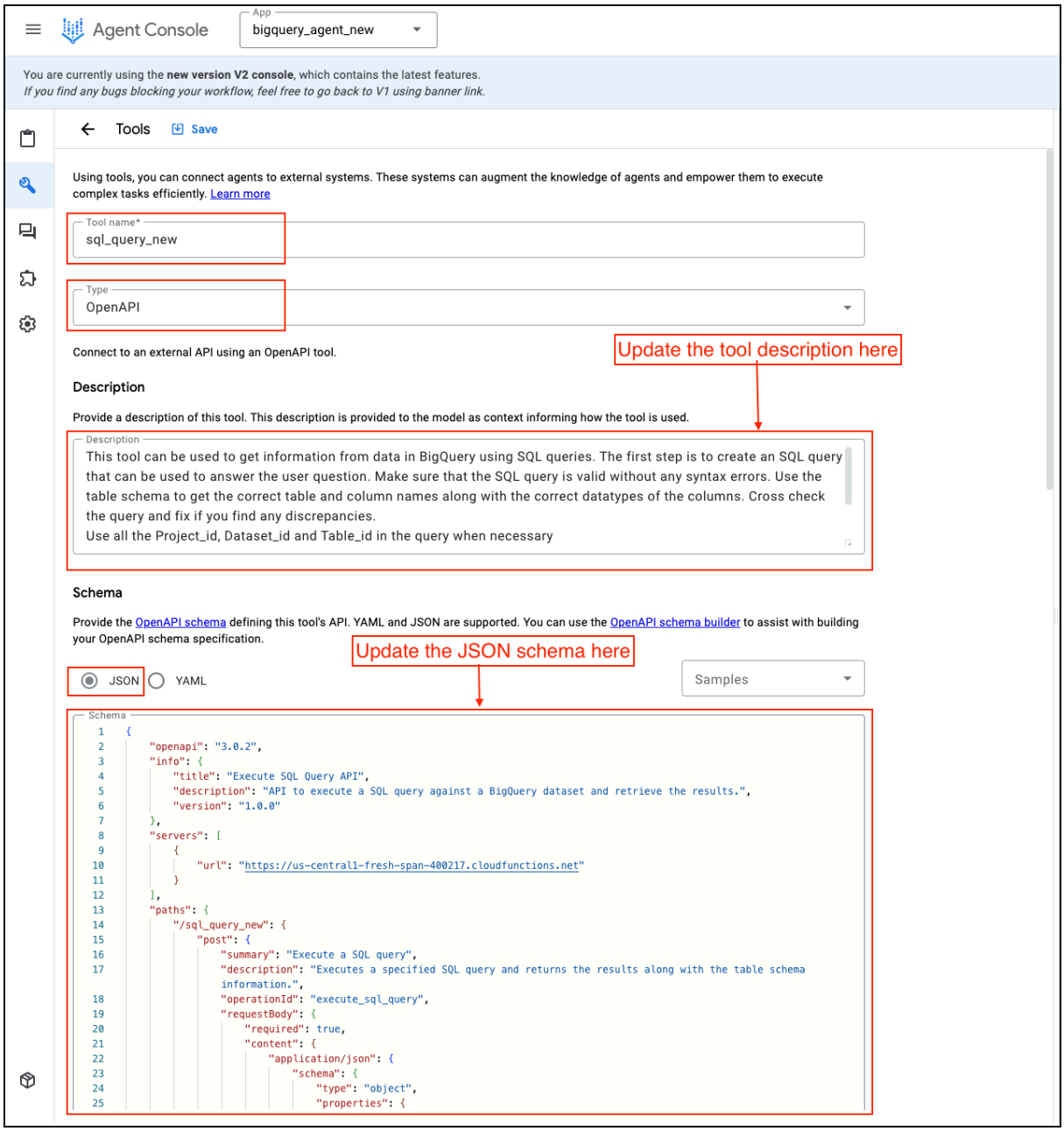
    i. Select JSON under schema
    j. Update the schema to:
    *Note: Use the url corresponding to **sql_query_new** cloud function created previously in the schema. Set the paths variable to sql_query_new

    ```
    {
     "openapi": "3.0.2",
    ```

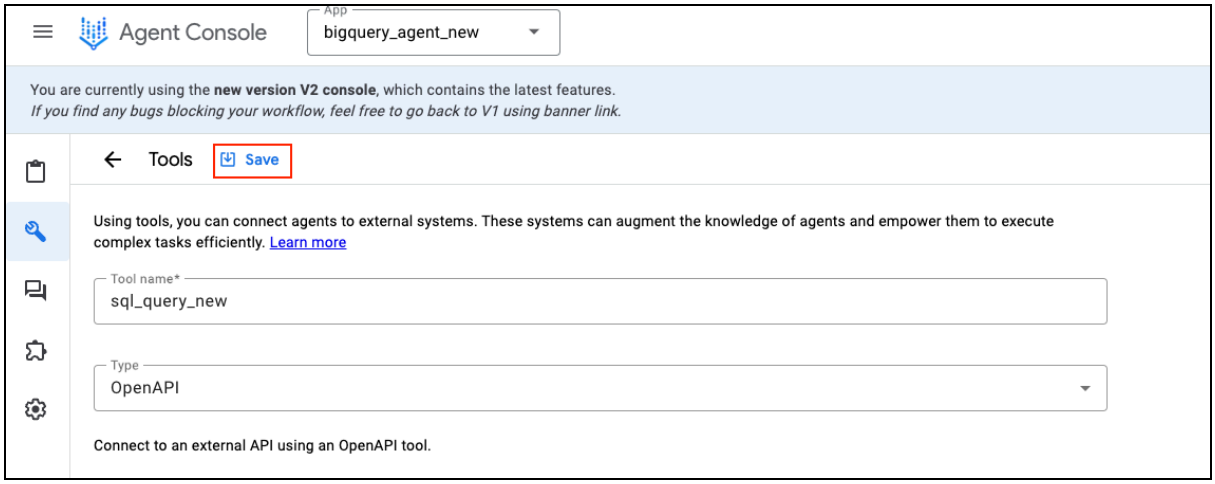
```
 "info": {
  "title": "Execute SQL Query API",
    "description": "API to execute a SQL query against a BigQuery dataset and retrieve the results.",
  "version": "1.0.0"
 },
 "servers": [
  {
   "url": "https://us-central1-fresh-span-400217.cloudfunctions.net"
  }
 ],
 "paths": {
  "/sql_query_new": {
   "post": {
    "summary": "Execute a SQL query",
       "description": "Executes a specified SQL query and returns the results along with the table schema information.",
    "operationId": "execute_sql_query",
    "requestBody": {
     "required": true,
     "content": {
      "application/json": {
       "schema": {
        "type": "object",
        "properties": {
         "query": {
          "type": "string",
          "description": "The SQL query to be executed."
         },
         "tableSchema": {
          "type": "object",
           "description": "The schema of the table that needs to be queried or the expected schema of the result.",
          "properties": {
           "schema": {
            "type": "array",
            "items": {
             "type": "object",
             "properties": {
```

```
                "name": {
                 "type": "string",
                 "description": "Name of the field in the table"
                },
                "type": {
                 "type": "string",
                 "description": "Data type of the field"
                },
                "mode": {
                 "type": "string",
                 "description": "Mode of the field (e.g., NULLABLE, REQUIRED)"
                },
                "description": {
                 "type": "string",
                 "description": "Description of the field"
                }
               }
              },
              "description": "Schema details of the table"
             },
             "num_rows": {
              "type": "integer",
              "description": "Total number of rows in the table"
             }
            }
           }
          },
          "required": [
           "query",
           "tableSchema"
          ]
         }
        }
       }
      },
      "responses": {
       "200": {
        "description": "Successful execution of SQL query",
        "content": {
```

```
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "results": {
                  "type": "string",
                      "description": "The JSON string containing the results of the SQL query."
                },
                "schema": {
                  "type": "object",
                    "description": "The schema of the result table as processed based on the query."
                }
              }
            }
          }
        },
        "400": {
          "description": "Bad request - Missing or invalid parameters"
        },
        "500": {
          "description": "Internal server error - Issues with executing the query or processing the result"
        }
      }
    }
  }
}
```

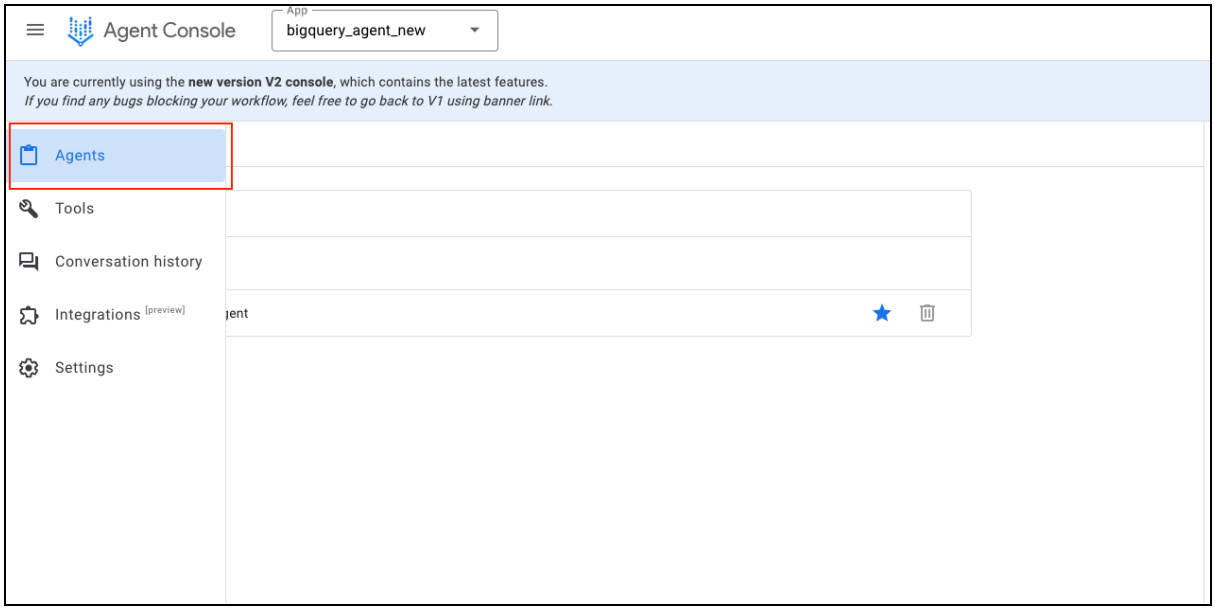


4. Save the Tool

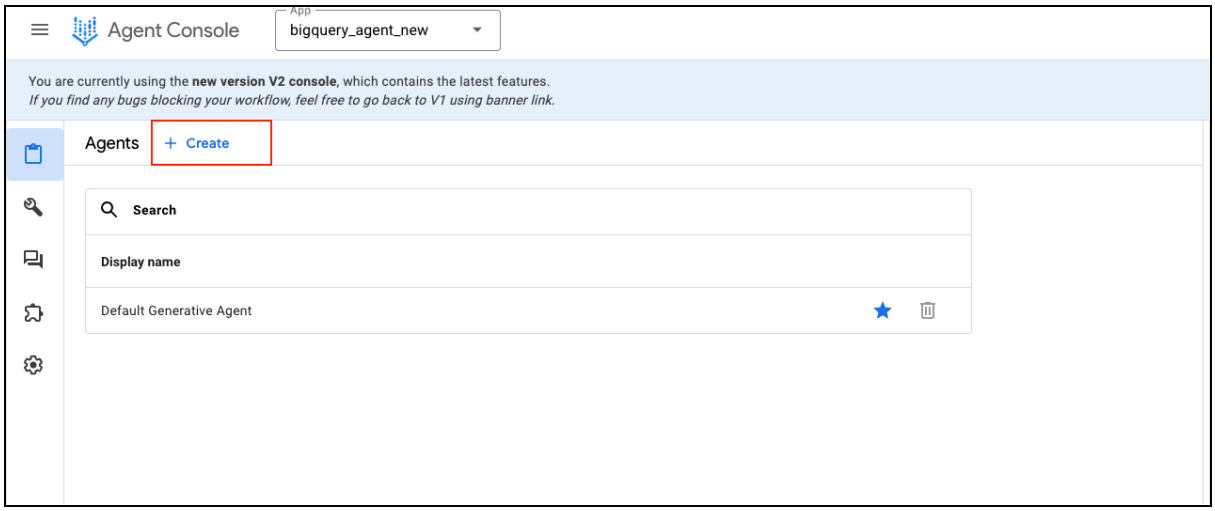


Step 7: The tools created in Steps 4, 5 and 6 will be used to construct the Agent Builder

1. Click on Agent



2. Click in Create to create a new Agent



3. Update the following fields:

   a. Agent name : Product mart *Note: Change the name of the agent as per requirement
   b. Set the **Goal** of the Agent to :

   The objective is to systematically utilize available tools in a specific sequence and answer the user question. First you need to use list_tables to get the list of tables in the dataset. Next use get_table tool to get the schema of a table. Construct an SQL query only using the table schema obtained from get_table tool. Finally use sql_query tool to run SQL queries on the data. Do not make up information. make sure you use all the tools available to generate the answer

   c. Specify the **instructions** to be followed by the Agent. Use the syntax **${TOOL: tool name}** to reference a tool :

   - User questions might be directly related to the data in BigQuery. Here is the step by step plan you MUST follow to answer the question: You must use the tools in the following order:

   - 1. You must Use ${TOOL:list_tables_new} to get the list of tables in the dataset

   - 2. Use ${TOOL:get_table_new} to get the table schema. You MUST ALWAYS pass the entire table id along with the project id and dataset id to this tool for accurate results:

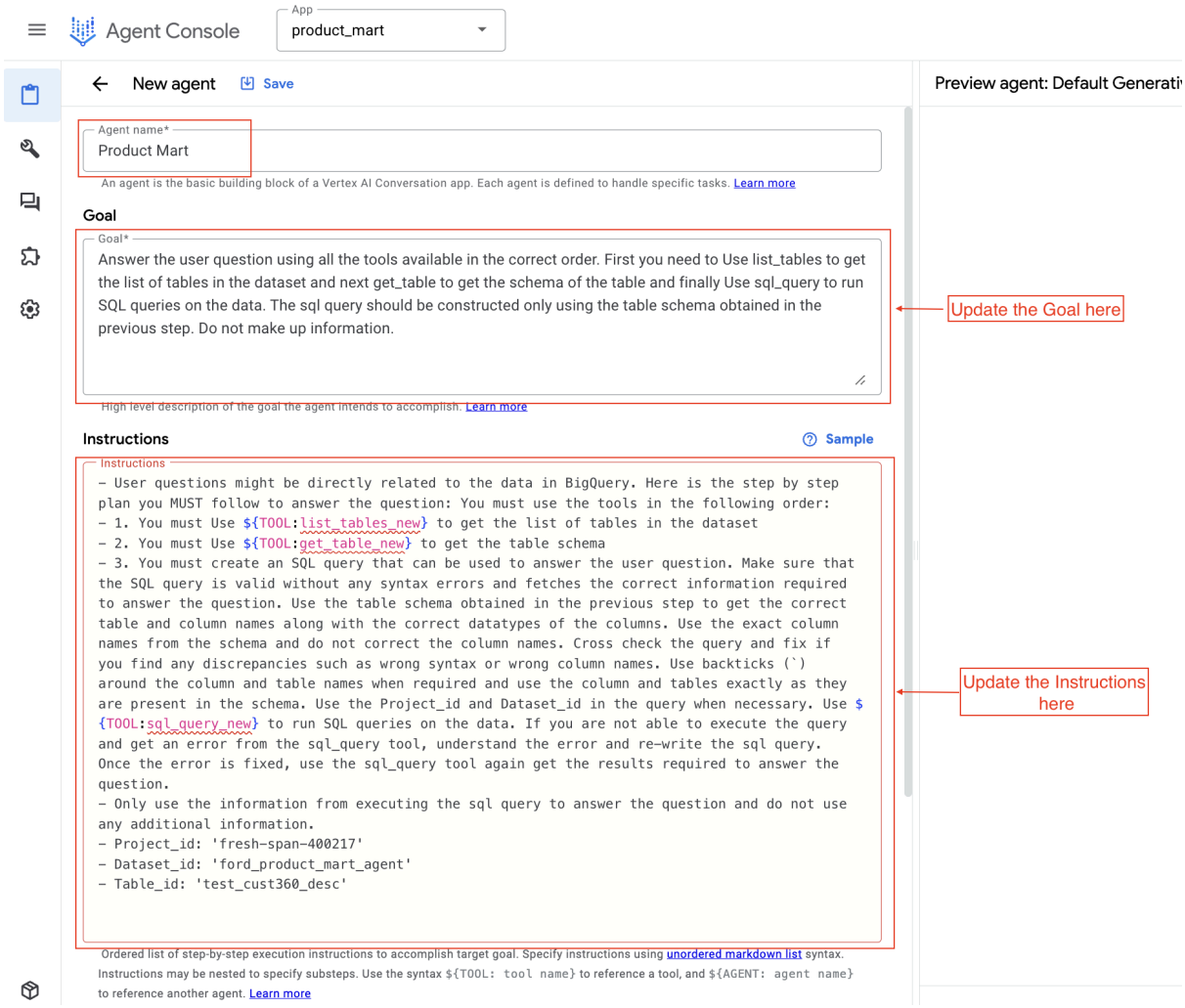
   - 3. You must create an SQL query that can be used to answer the user question. Make sure that the SQL query is valid without any syntax errors and fetches the correct information required to answer the question. Use the table schema obtained in the previous step to get the correct table and column names along with the correct datatypes of the columns. Use the exact column

names from the schema and do not correct the column names. Cross check the query and fix if you find any discrepancies such as wrong syntax or wrong column names. Use backticks (`) around the column and table names when required and use the column and tables exactly as they are present in the schema. Use the Project_id and Dataset_id in the query when necessary. Use ${TOOL:sql_query_new} to run SQL queries on the data. If you are not able to execute the query and get an error from the sql_query tool, understand the error and re-write the sql query. Once the error is fixed, use the sql_query tool again to get the results required to answer the question.
- Only use the information from executing the sql query to answer the question and do not use any additional information.

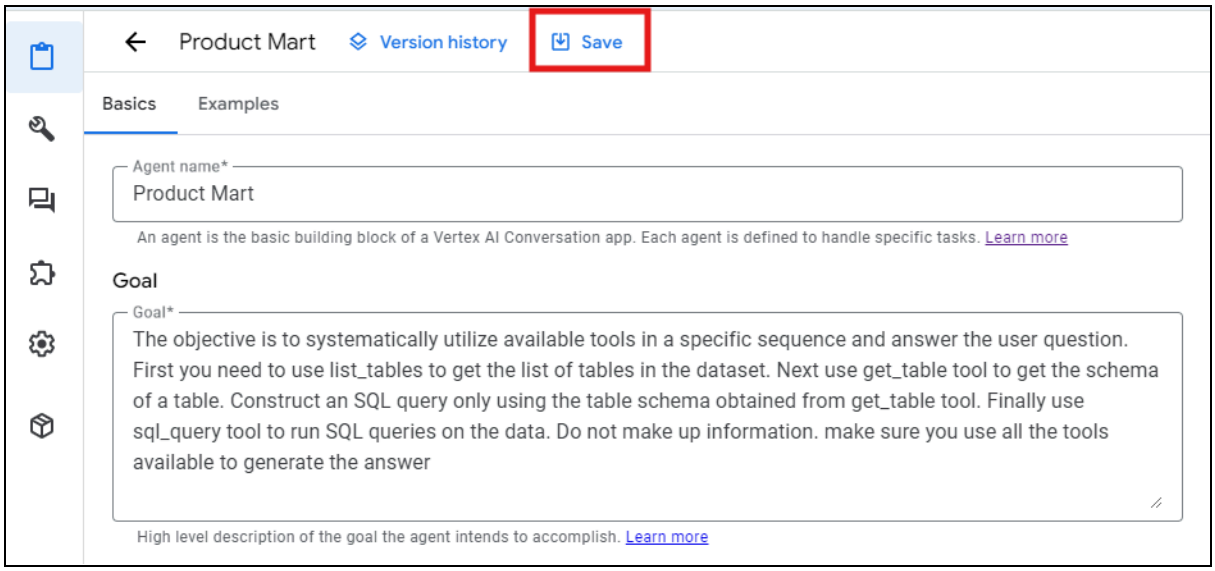
Project_id: '<PROJECT_ID>'
Dataset_id: '<Dataset_id>'
Table_id: '<Table_id>'

Example for your reference:
Project_id: 'fresh-span-400217'
Dataset_id: 'aa_genai'
Table_id: 'sales'

*Note: Replace the project id, dataset id and table id with the correct values



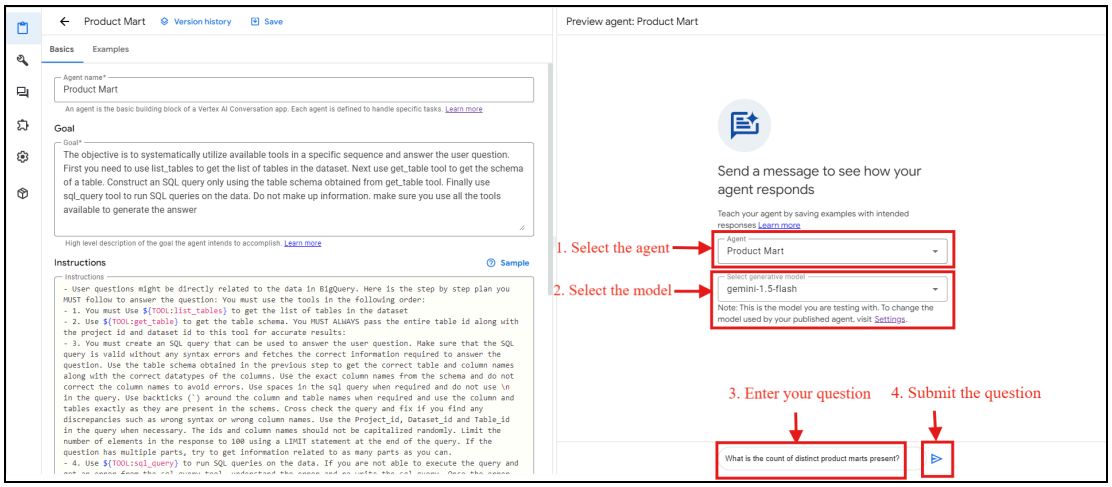
4. Save the Agent



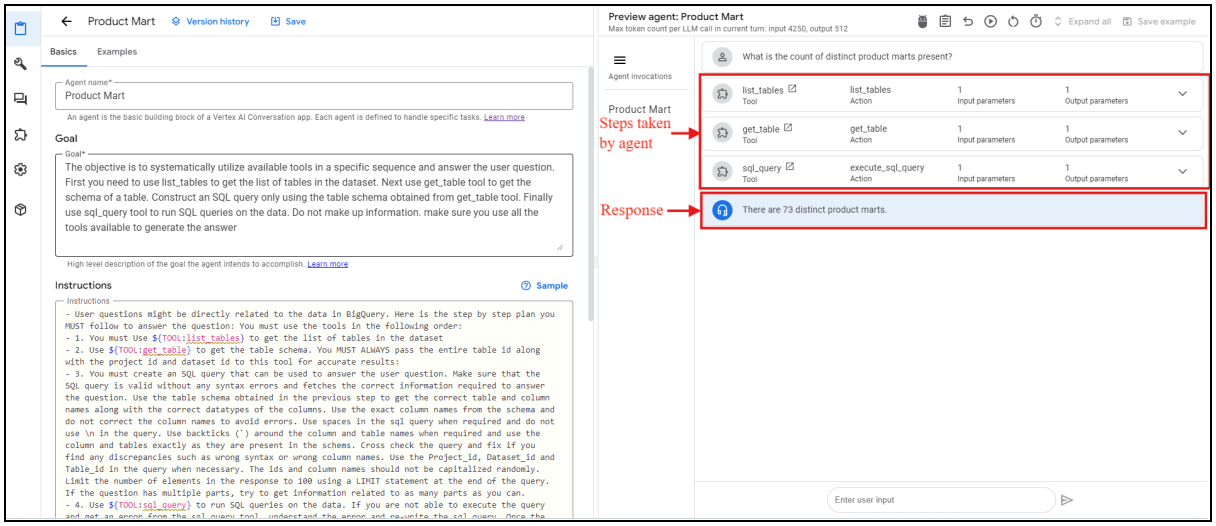
Step 8: Testing the Agent created

Follow these steps to test the Agent Builder:

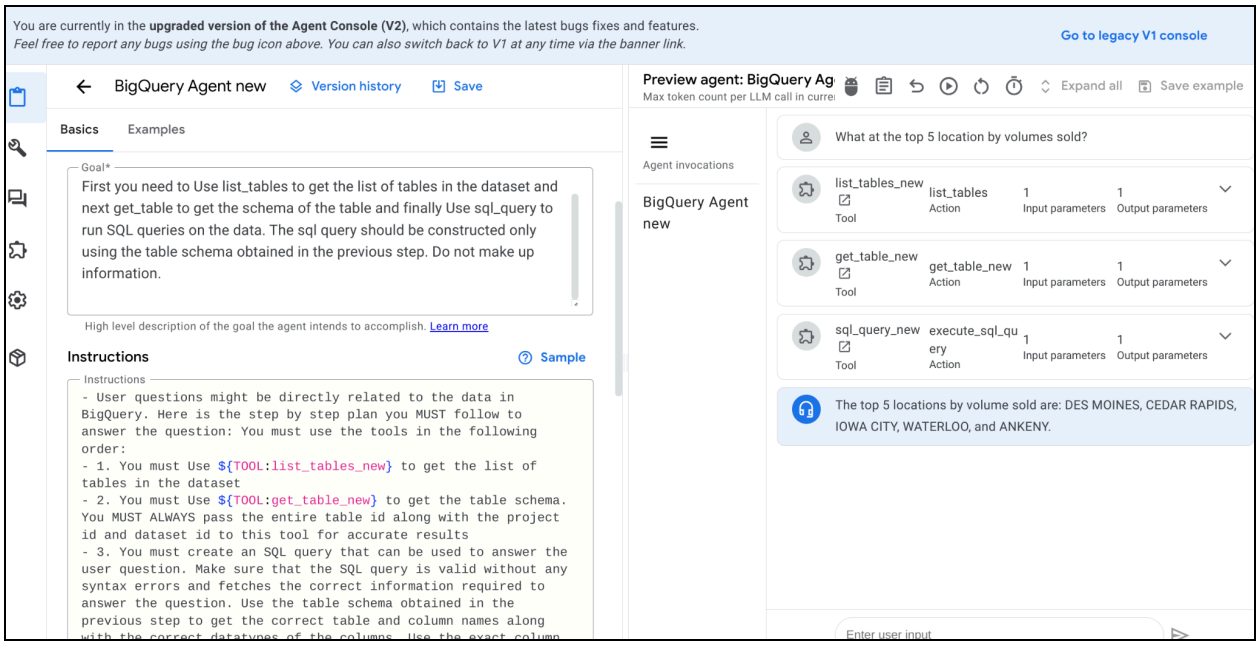
1. Select the agent created, Product Mart. *Note: Select the correct agent
2. Select the GenAI model, **gemini-1.5-flash**
3. Enter the question in the text box. Example question: ***'What are the top 5 locations by volume sold?'***
4. Click the **Submit** button or press Enter

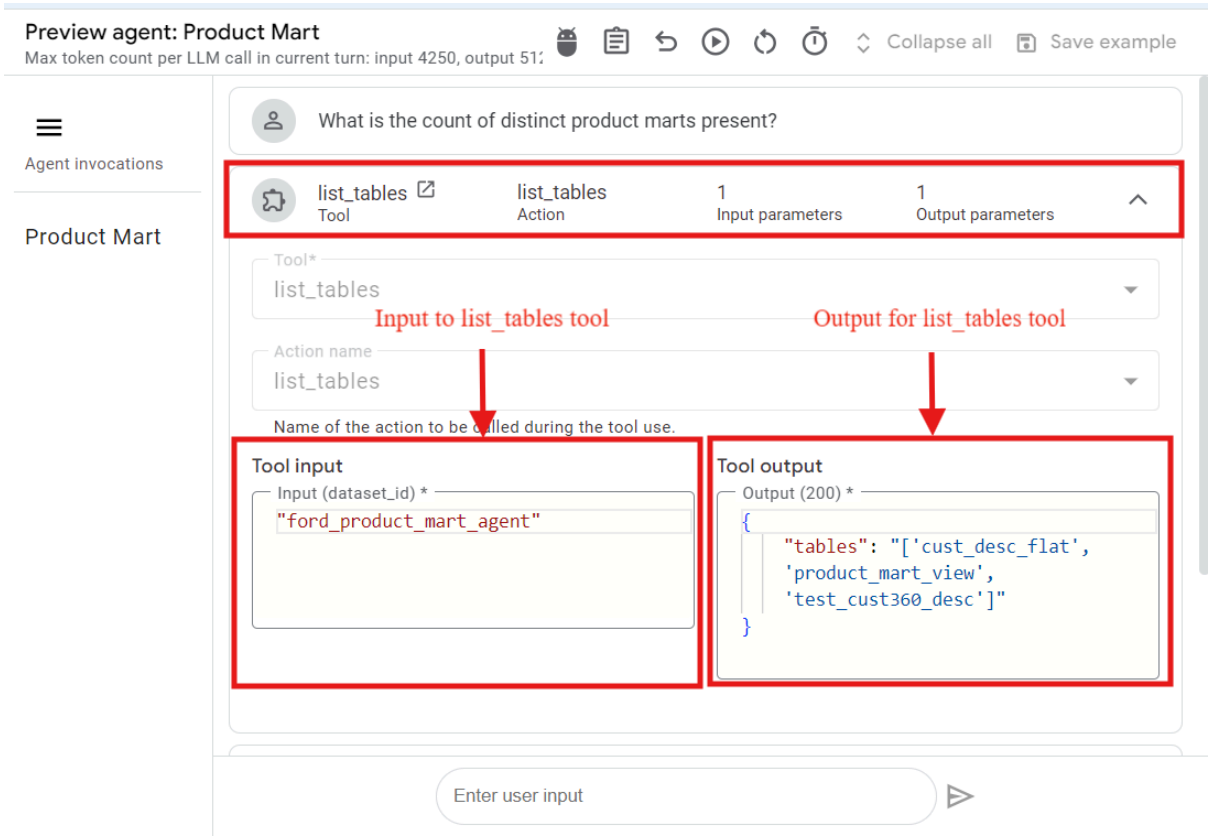


The response and the steps taken by the Agent can be seen in the UI.



Check the input and output for each tool by expanding the tool block



1. 

In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.

Congratulations!

In this lab, you have successfully learnt about how to build an agent using the Agent Builder on structured BigQuery datasets.