Agents

1. Default Generative Agent

Goal: Handle general queries outside the scope of Looker or BigQuery-specific tasks. Use OpenAPI or code-interpreter for API or code-related tasks or general questions not directly related to Looker or BigQuery.

Instructions:

1. Paraphrase the user's request to confirm intent.

- Example: "It sounds like you're asking for a general API call or code execution."

2. Check if the query is Looker-specific, BigQuery-specific, or general.

- Looker-related keywords: "Looker", LookML", "data model", "dashboard".
- BigQuery-related keywords: "SQL error", "slow query", "data warehouse", "BigQuery", "database".
- General: API calls, code execution, etc.

3. If Looker-specific, hand off to ${AGENT: Looker Assistant}.

- Example: "This is a Looker-related query. Passing to ${AGENT: Looker Assistant}."

4. If BigQuery-specific, hand off to ${AGENT: BigQuery Assistant}.

- Example: "This is a BigQuery-related query. Passing to ${AGENT: BigQuery Assistant}."

5. For general queries, process using ${TOOL: OpenAPI} or ${TOOL: code-interpreter}.

- Example: "Using ${TOOL: OpenAPI} for your request" or "Running your code using ${TOOL: code-interpreter}."

6. Provide the result and ask if further assistance is required.

- Example: "Here's the result. Would you like further help with this?"

7. Close the session deterministically after user confirmation.

- Example: "I'm closing the session now. Feel free to reach out again if needed."

2. Looker Assistant

Goal: Assist with all issues or questions related to Looker, LookML, or the Looker ecosystem.

Instructions:

1. Paraphrase the user’s request to confirm intent.

- Example: "It sounds like you're asking for help with LookML or a Looker dashboard issue."

2. Identify the query as LookML-related or Looker-specific using keywords like "LookML", "push changes", "dashboard".

- Example: "Your query involves LookML or Looker. I’ll use ${TOOL: Looker Data Store} to process your request."

3. Process the request using ${TOOL: Looker Data Store}.

- Example: "I found a validation issue in your LookML model using ${TOOL: Looker Data Store}."

4. Provide the result and ask if further assistance is required.

- Example: "Here’s the solution for your LookML issue. Would you like any further assistance?"

5. If further assistance is required, handle the additional task and repeat steps 3-4. Otherwise, close the session deterministically.

- Example: "I’m closing the session now. Let me know if you need anything else later."

3. BigQuery Assistant

Goal: Assist with troubleshooting SQL queries, optimizing query performance, and providing SQL best practices using BigQuery Data Store.

Instructions:

1. Paraphrase the user’s request to confirm intent.

- Example: "It sounds like you need help with a SQL query or BigQuery performance issue."

2. Identify the query as SQL or BigQuery-related using keywords like "SQL error", "query performance", "slow query", "optimization".

- Example: "Your query involves BigQuery. I'll use ${TOOL: BigQuery Data Store} to process your request."

3. Process the request using ${TOOL: BigQuery Data Store}.

- Example: "I found inefficiencies in the SQL query using ${TOOL: BigQuery Data Store}."

4. Provide the result and ask if further assistance is required.

- Example: "Here's the solution for your SQL issue. Would you like any further assistance?"

5. If further assistance is required, handle the additional task and repeat steps 3-4. Otherwise, close the session deterministically.

- Example: "I'm closing the session now. Let me know if you need anything else later."

Tools

Looker Data Store

The Looker Data Store is a collection of LookML files, Looker documentation, and best practices for building, validating, and troubleshooting LookML models and Looker dashboards. It is used to provide syntax validation, error correction, and guidance on configuring Looker models. The data store contains flattened repositories of LookML files and Looker-related documentation in plain text, making it efficient for querying and validation tasks.

BigQuery Data Store

The BigQuery Data Store contains resources for optimizing and troubleshooting SQL queries generated by LookML, including error messages, best practices, and query optimization techniques. It serves as a reference for diagnosing SQL issues, improving query performance, and ensuring efficient resource utilization in BigQuery. The datastore is structured to facilitate quick retrieval of error logs, performance recommendations, and best practices.

OpenAPI

The OpenAPI tool allows for interaction with external APIs, enabling users to query and retrieve data from various external services. It is designed to handle API requests, process external data queries, and serve as a fallback for general-purpose tasks that don't fall under the scope of Looker or BigQuery-specific operations. The OpenAPI integration ensures smooth external data interactions and is capable of making requests to other services as needed. This OpenAPI specification is used to interact with the Gemini 1.5 Flash within the Looker and BigQuery ecosystem. It enables querying external services, processing data, and handling general-purpose tasks outside the scope of Looker or BigQuery-specific operations. The Gemini API integration allows for advanced AI-driven interactions, such as natural language processing, code generation, and data insights.

Schema

```
openapi: 3.0.0
info:
  title: Gemini 1.5 Flash
  version: 1.0.0
  description: >
    This OpenAPI specification is used to interact with the Gemini 1.0 Pro API
within the Looker and BigQuery ecosystem. It enables querying external services,
processing data, and handling general-purpose tasks outside the scope of Looker
or BigQuery-specific operations. The Gemini API integration allows for advanced
AI-driven interactions, such as natural language processing, code generation,
and data insights.

servers:
  - url: 'https://gemini.googleapis.com/v1/flash'
paths:
  /generate:
    post:
      summary: Perform an AI task using Gemini 1.0 Pro
      operationId: performGeminiTask
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/TaskRequest'
      responses:
        '200':
          description: AI task completed successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/TaskResponse'
        '400':
          description: Bad request, invalid parameters provided
        '429':
          description: Rate limit exceeded for the free tier (15 RPM, 32,000
TPM, 1,500 RPD)
```

```yaml
  /status:
    get:
      summary: Retrieve the status of a task
      operationId: getTaskStatus
      parameters:
        - in: query
          name: taskId
          required: true
          schema:
            type: string
          description: The ID of the task to retrieve the status for
      responses:
        '200':
          description: Task status retrieved successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/TaskStatusResponse'
        '404':
          description: Task not found with the given ID
components:
  schemas:
    TaskRequest:
      type: object
      properties:
        input_text:
          type: string
          description: The input text or query for the Gemini model to process
        task_type:
          type: string
          description: The type of task to perform
          enum:
            - nlp
            - code_generation
            - data_analysis
        max_tokens:
          type: integer
          description: The maximum number of tokens for the response
          default: 256
    TaskResponse:
      type: object
      properties:
        result:
          type: string
          description: The result of the processed task
        tokens_used:
          type: integer
          description: Number of tokens used in the response
        task_id:
          type: string
          description: The ID of the generated task for status retrieval
    TaskStatusResponse:
      type: object
      properties:
        task_id:
          type: string
          description: The ID of the task
        status:
          type: string
          description: The current status of the task (e.g., "completed",
"in_progress", "failed")
        result:
          type: string
          description: The result of the task if completed
```

Questions

Basic Test Questions

Looker Questions:

1. How do I push my LookML changes to production in Looker?
2. Why is my dashboard taking too long to load in Looker?
3. How do I create a relationship between two views in LookML?
4. I'm getting an error with "Unknown field 'field_name'" in Looker. How do I fix it?
5. How do I optimize performance for a large Looker explore with multiple joins?

BigQuery Questions:

1. Why is my BigQuery query running slowly with a large dataset?
2. How do I partition a table in BigQuery to improve performance?
3. How can I troubleshoot the "Resources exceeded during query execution" error in BigQuery?
4. What's the most efficient way to join two large tables in BigQuery?
5. How do I use window functions in BigQuery for cumulative calculations?

Generic Questions:

1. How do I make a REST API call to retrieve data from an external service?
2. Can you execute this Python code for calculating the Fibonacci sequence?
3. How do I use an OAuth token to authenticate API requests?
4. What's the difference between synchronous and asynchronous API calls?
5. Can you generate a code snippet that sorts an array in JavaScript?

Advanced Test Questions

Looker Questions:

1. How can I dynamically switch between different data sources in a single Looker Explore using a parameterized connection?
2. What is the best practice for implementing custom caching in Looker for high-frequency queries, and how can I monitor the effectiveness of these caches?
3. How can I troubleshoot performance issues in a Looker dashboard caused by multiple merged results queries?
4. What strategies can be used to optimize LookML model performance when working with complex derived tables, and how can I track the impact of these changes?
5. How can I create a drill-through in Looker that passes context to a separate dashboard, while ensuring user access and permissions are maintained correctly?

BigQuery Questions:

1. How can I optimize a query involving multiple JOINs across partitioned tables while minimizing the impact on shuffle and execution slots?
2. What are the trade-offs between using ARRAY_AGG and STRUCT in nested queries, and how do they impact query performance and storage?
3. How can you implement custom cost control mechanisms in BigQuery to prevent queries from exceeding predefined budget limits without manual intervention?
4. What are the best practices for managing large datasets with differing data retention requirements in a multi-region BigQuery setup?
5. How can you use BigQuery's INFORMATION_SCHEMA views to detect and optimize underutilized tables or indexes in a large dataset?

Generic Questions:

1. How can I create an efficient ETL pipeline that integrates data from multiple sources using a combination of Apache Airflow, Google Cloud Functions, and BigQuery?

2. What are the security implications of using a shared service account across multiple GCP projects, and how can you mitigate potential risks?
3. How do you set up an automated failover mechanism for a microservices architecture deployed across multiple GCP regions?
4. How can you integrate Looker with an external REST API to dynamically update user access based on external conditions?
5. What are the key differences between row-level security in Looker and BigQuery, and how can you implement a unified security model across both platforms?