Quick tip: Review the prerequisites before you run the lab

Start Lab 01:00:00

Storing and Querying Embeddings with AlloyDB for PostgreSQL

Lab 1 hour No cost Intermediate

This lab may incorporate AI tools to support your learning.

Previous Next

Overview

This lab demonstrates how to easily integrate generative AI features into your

We will build a sample Python application together that will be able to understand and respond to human language queries about the relational data stored in your PostgreSQL database. In fact, we will further push the creative limits of the application by teaching it to generate new content based on our existing dataset.

This lab utilizes an example of an e-commerce company that operates an online marketplace for buying and selling children's toys. The company aims to incorporate new generative AI experiences into its e-commerce applications for both buyers and sellers on the platform.

The goals are:

(Usecase 1) For buyers: Build a new AI-powered hybrid search, where users can describe their needs in simple English text, along with regular filters (like price, etc.)
(Usecase 2) For sellers: Add a new AI-powered content generation feature, where sellers will get auto-generated item description suggestions for new products that they want to add to the platform.

Dataset: The dataset for this lab has been sampled and created from a larger public retail dataset available at Kaggle. The sampled dataset used in this lab has only about

Objective

At the end of this lab:

- You will have a good understanding of how to use the pgvector extension to store and search vector embeddings in PostgreSQL. Learn more about vector embeddings.
- You will get a hands-on experience with using the open-source LangChain framework to develop applications powered by large language models. LangChain makes it easier to develop and deploy applications against any LLM model in a vendor-agnostic manner.
- You will learn about the powerful features in Google PaLM models made available through Vertex AI.

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made

starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

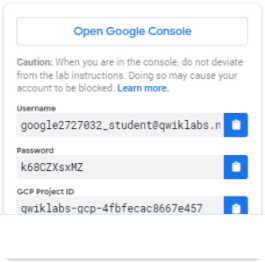
To complete this lab, you need:

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

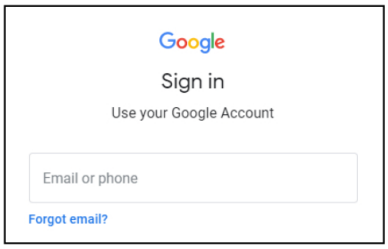
Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

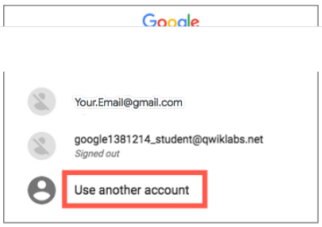


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.

Task 1. Open the notebook in Vertex AI Workbench and install packages

1. In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI > Workbench**. Alternatively, you can type "Vertex AI Workbench" in the top search bar and click the first result.

JupyterLab button.

The JupyterLab interface for your Workbench instance will open in a new browser tab.

3. On the **Launcher**, under **Notebook**, click on **Python 3** to open a new python notebook.

4. Install the required packages by running the following command in the first cell of the notebook. Either click the play ▶ button at the top or click **SHIFT+ENTER** keys on your keyboard to execute the cell.

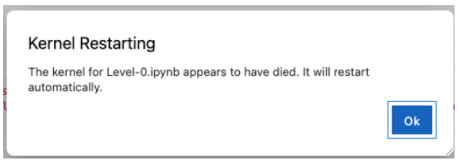
```
!pip install numpy pandas
!pip install pgvector
!pip install langchain langchain_google_vertexai
transformers
!pip install google-cloud-aiplatform
!pip install psycopg2-binary
!pip install protobuf
!pip install shapely
```

clicking the refresh button ↻ at the top, followed by clicking **Restart** button.

```
import IPython

app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Output:



After the restart is complete, click **Ok** on the prompt to continue.

Task 2. Download and load the dataset

An AlloyDB cluster named `AlloyDB Cluster` is configured in this lab. To begin, let's locate the AlloyDB cluster's IP address.

1. On the Google Cloud console title bar, type "AlloyDB" in the **Search** field, then click **AlloyDB** in the **Products & Pages** section.

2. Locate the cluster named `AlloyDB Cluster`, and the primary instance named `AlloyDB Instance`. The private IP address of this instance serves as your access point for utilizing AlloyDB throughout the lab.

3. Back in Vertex AI Workbench Notebook, import necessary libraries.

```
import os
import pandas as pd
import vertexai
```

```
from langchain_google_vertexai import VertexAIEmbeddings
import vertexai
```

```
PROJECT_ID = "PROJECT_ID"  # @param {type:"string"}
REGION = "Lab Region"  # @param {type:"string"}

# Initialize Vertex AI SDK
vertexai.init(project=PROJECT_ID, location=REGION)
```

4. Run the following code snippet to import the `psycopg2` library, which allows Python to interact with PostgreSQL databases, reads the CSV dataset into a pandas DataFrame, and finally saves the DataFrame to a table named `products` in the AlloyDB cluster.

```
import psycopg2

# Replace with your AlloyDB cluster credentials
cluster_ip_address = "Cluster-ip-address"
database_user = "postgres"
database_password = "postgres"
```

```
os.environ["PGHOST"] = cluster_ip_address
os.environ["PGUSER"] = database_user
os.environ["PGPASSWORD"] = database_password

# Establish a connection to the database
try:
    conn = psycopg2.connect(
        host=cluster_ip_address,
        user=database_user,
        password=database_password
    )
    print("Connected to the database successfully!")
except Exception as e:
    print("Connection error:", e)
exit(1)

# Read the dataset from the URL
DATASET_URL =
"https://github.com/GoogleCloudPlatform/python-docs-
samples/raw/main/cloud-
sql/postgres/pgvector/data/retail_toy_dataset.csv"
df = pd.read_csv(DATASET_URL)

# Select desired columns and drop missing values
```

```
df = df.dropna()

# Save the DataFrame to the AlloyDB cluster
df.to_sql('products',
con=f'postgresql://{cluster_ip_address}',
if_exists='replace', index=False)

# Retrieve data from the 'products' table
cur = conn.cursor()
cur.execute("SELECT * FROM products")
results = cur.fetchall()

# Close the connection
conn.close()
print(results[5])
```

Output:

```
Connected to the database successfully!
('74a695e3675efc2aad11ed73c46db29b', 'Slip N Slide Triple Racer with
```

Task 3. Generate Vector Embeddings using a Text Embedding Model

In this section, let's preprocess product descriptions, generate vector embeddings for them, and store the embeddings along with other relevant data in a PostgreSQL database table for downstream analysis or applications.

1. Run the following code snippet to import the `RecursiveTextSplitter` class from the LangChain library, which is used for splitting text into smaller chunks. Iterate through each row in the DataFrame `df` and extract the **product ID** and **description** from each row.

   Then, we will split each description into smaller chunks and will create a dictionary for each chunk.

```
from langchain.schema import Document
```

```
# Set up the text splitter
text_splitter = RecursiveCharacterTextSplitter(
    separators=[".", "\n"],
    chunk_size=500,
    chunk_overlap=0,
    length_function=len,
)

# Define the maximum number of documents to process
max_documents = 50  # Reduced limit to further control API
usage
documents = []

# Create Document objects with product_id as metadata
for index, row in df.iterrows():
    product_id = row["product_id"]
    desc = row["description"]
    documents.append(Document(page_content=desc, metadata=
{"product_id": product_id}))

# Use the text splitter on a subset of documents (e.g., 40-
```

```
docs =
text_splitter.split_documents(documents[40:max_documents])

# Collect split content along with product_id
for doc in docs:
    chunked.append({"product_id":
doc.metadata["product_id"], "content": doc.page_content})

print(len(chunked))
```

2. Run the following code snippet to process product descriptions from a dataset by splitting them into smaller chunks, sending them to Vertex AI for embedding generation, and storing the retrieved embeddings back into the data structure.

```
from langchain_google_vertexai import VertexAIEmbeddings
from google.cloud import aiplatform
import time

embeddings_service =
VertexAIEmbeddings(model_name="textembedding-gecko")

# Helper function to retry failed API requests with
```

```
backoff_factor=2.5, **kwargs):  # Increased delay and
backoff factor
    max_attempts = 10
    retries = 0
    for i in range(max_attempts):
        try:
            return func(*args, **kwargs)
        except Exception as e:
            print(f"error: {e}")
            retries += 1
            wait = retry_delay * (backoff_factor**retries)
            print(f"Retry after waiting for {wait}
seconds...")
            time.sleep(wait)

# Reduced batch size for API calls to manage quota limits
batch_size = 3
for i in range(0, len(chunked), batch_size):
    request = [x["content"] for x in chunked[i : i +
batch_size]]
    response =
retry_with_backoff(embeddings_service.embed_documents,
request)
```

```
        x["embedding"] = e

# Store the generated embeddings in a pandas dataframe.
product_embeddings = pd.DataFrame(chunked)
product_embeddings.head()
```

Output:

	product_id	content	embedding
0	8a6d71be41e01b284294ec488508b414	All of our productsWalmartply with internation...	[-0.0019801377784460783, -0.037444762885570526...
1	8a6d71be41e01b284294ec488508b414	. Holds Up to 6 Decks Fun for the whole family...	[0.008327648043632507, -0.038024611777110672, -...
2	9648838f5badebb9fc0b07f89cc29394	Better circulate water through your pool with ...	[-0.045468609780073166, -0.000837682629935443 4...
3	9648838f5badebb9fc0b07f89cc29394	.25-inch fitting (11070), 2 strainer grids (11...	[-0.020975708961486816, 0.010413266718387604, ...
4	9648838f5badebb9fc0b07f89cc29394	. Circulate water through your pool with the h...	[-0.042180399661645889, -0.014087582007050514, ...

3. Run the following command to enable AlloyDB integration.

```
!PROJECT_ID=$(gcloud config get-value project) && \
PROJECT_NUMBER=$(gcloud projects list --
filter="name=$PROJECT_ID" --format="value(PROJECT_NUMBER)")
&& \
```

```
alloydb.iam.gserviceaccount.com  \
--role="roles/aiplatform.user"
```

Output:

```
Updated IAM policy for project [qwiklabs-gcp-03-02b1c7e4b6a5].
bindings:
- members:
  - serviceAccount:399096163907-compute@developer.gserviceaccount.com
  role: roles/aiplatform.admin
- members:
  - serviceAccount:service-399096163907@gcp-sa-alloydb.iam.gserviceaccount.com
  role: roles/aiplatform.user
- members:
  - serviceAccount:service-399096163907@gcp-sa-alloydb.iam.gserviceaccount.com
  role: roles/alloydb.serviceAgent
- members:
  - serviceAccount:399096163907-compute@developer.gserviceaccount.com
  role: roles/artifactregistry.admin
- members:
  - serviceAccount:399096163907-compute@developer.gserviceaccount.com
  - serviceAccount:qwiklabs-gcp-03-02b1c7e4b6a5@qwiklabs-gcp-03-02b1c7e4b6a5.iam.gserviceaccount.com
  role: roles/bigquery.admin
- members:
  - serviceAccount:399096163907@cloudbuild.gserviceaccount.com
  role: roles/cloudbuild.builds.builder
- members:
  - serviceAccount:399096163907-compute@developer.gserviceaccount.com
  role: roles/cloudbuild.builds.editor
- members:
  - serviceAccount:399096163907-compute@developer.gserviceaccount.com
  role: roles/cloudbuild.integrations.editor
- members:
  - serviceAccount:service-399096163907@gcp-sa-cloudbuild.iam.gserviceaccount.com
  role: roles/cloudbuild.serviceAgent
```

`Cluster`, then select **AlloyDB Studio** from the left-hand side menu, enter the following values to sign in, and click on **AUTHENTICATE**.

Field	Value
Database	postgres
User	postgres
Password	postgres

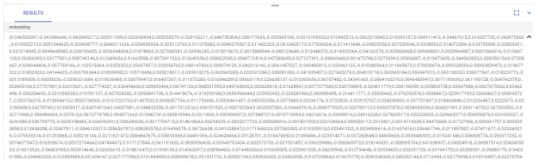
5. In the **AlloyDB Studio**, click on **Editor** tab at the top.

6. Enter the following command in the editor to grant the `postgres` user permission to execute the `embedding` function, install the `google_ml_integration` extension, and generate an embedding for the provided text using the `textembedding-gecko` model. Then, click on **Run** button at the top.

```
GRANT EXECUTE ON FUNCTION embedding TO postgres;
```

```
SELECT embedding('textembedding-gecko', 'AlloyDB is a
managed, cloud-hosted SQL database service.');
```

Output:



7. Once the embeddings are successfully generated, click the **Clear** button at the top to clear the contents of the editor. Then, run the following query in the editor to prepare the database.

```
CREATE EXTENSION IF NOT EXISTS vector;

DROP TABLE IF EXISTS product_embeddings;
```

8. Run the following query, to create the embeddings based on product descriptions.

```
CREATE TABLE product_embeddings(
        product_id VARCHAR(1024) NOT NULL PRIMARY KEY,
        content TEXT,
        embedding vector(768)
    );


insert into product_embeddings(product_id, content,
embedding)
SELECT
product_id,
description as content,
embedding('textembedding-gecko', description) as embedding
from products
where product_id not in (select product_id from
product_embeddings)
```

```
limit 10;
```

Task 4. Create Indexes for faster Similarity Search

Vector indexes can significantly speed up similarity search operations and avoid the brute-force exact nearest neighbor search that is used by default.

Pgvector comes with two types of indexes: `hnsw` and `ivfflat`.

1. In the Editor, run the following query to build the **HNSW index** on **product_embeddings** table using cosine similarity metric for faster search based on descriptions.

```
-- Create an HNSW index on the `product_embeddings` table
CREATE INDEX ON product_embeddings
USING hnsw(embedding vector_cosine_ops)
WITH (m = 24, ef_construction = 100);
```

2. Next, run the query to create an **IVFFLAT** index on the **product_embeddings** table using cosine similarity for swift similarity searches among product descriptions.

```
-- Create an IVFFLAT index on the `product_embeddings`
table
CREATE INDEX ON product_embeddings
USING ivfflat(embedding vector_cosine_ops)
WITH (lists = 100);
```

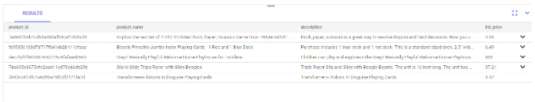
3. Now, we will conduct the **similarity search**. The provided code identifies products most relevant to the user's query based on their textual descriptions. To ensure relevant results.

```
with e as (
SELECT
    *
FROM
    product_embeddings
ORDER BY
```

```
LIMIT
    5
)
select
*
from products
where product_id in (select e.product_id from e);
```

Finally, the top matches are displayed as a `list`, making it easy to find products that fit your search and budget. You'll see the **product name**, **price**, and a brief **description** for each result, giving you a quick overview of your options.

Output:



Task 5. LLMs and LangChain

Use case 1: Building an AI-curated contextual hybrid search

Combine natural language query text with regular relational filters to create a powerful hybrid search.

Back in the **Jupyter Workbench instance**, run each of the code snippets below in the Jupyter notebook.

1. Build a user query with english text and the price filters.

```
# Please fill in these values.
user_query = "Do you have a toy set that teaches numbers
and letters to kids?"  # @param {type:"string"}
min_price = 20  # @param {type:"integer"}
max_price = 100  # @param {type:"integer"}
```

```
qe = embeddings_service.embed_query(user_query)
```

3. Use `pgvector` to find similar products. The pgvector similarity search operators provide powerful semantics to combine the vector search operation with regular query filters in a single SQL query.

```
import psycopg2
from psycopg2 import sql
from pgvector.psycopg2 import register_vector
import pandas as pd

def main(user_query, min_price, max_price):
    try:
        # AlloyDB cluster connection details (replace with
your actual values)
        cluster_ip_address = "Cluster-ip-address"
        database_user = "postgres"
        database_password = "postgres"

        # Connect to AlloyDB cluster
        conn = psycopg2.connect(
            user=database_user,
            password=database_password
        )

        # Register the vector type
        register_vector(conn)

        # Get the query embedding
        qe = embeddings_service.embed_query(user_query)

        # Check if qe is valid
        if not qe:
            print("Error: The query embedding is empty.")
            return

        # Perform the similarity search and filtering
        cur = conn.cursor()
        similarity_threshold = 0.5  # Increased threshold
for broader matching
        num_matches = 50

        # Modify the SQL query for indexed similarity
search
        cur.execute(
            """
            WITH vector_matches AS (
                SELECT product_id, embedding <=> %s::vector
AS distance
                FROM product_embeddings
                WHERE embedding <=> %s::vector < %s
                ORDER BY distance ASC
                LIMIT %s
            )
            SELECT product_name, list_price, description
            FROM products
            WHERE product_id IN (SELECT product_id FROM
vector_matches)
            AND list_price >= %s AND list_price <= %s
            """,
            (qe, qe, similarity_threshold, num_matches,
min_price, max_price)
        )
        results = cur.fetchall()

        # Check if any results are retrieved
        if not results:
            print("No results found. Try adjusting the
similarity threshold or checking the data.")
            return

        matches = []
        for r in results:
            try:
                list_price = round(float(r[1]), 2)  #
Attempt conversion and rounding
            except ValueError:
                list_price = r[1]  # Use original value if
conversion fails
            matches.append({
                "product_name": r[0],
                "list_price": list_price,
                "description": r[2]
            })

        # Display the results
```

```
        matches_df = pd.DataFrame(matches)
        print(matches_df.head(5))

    except Exception as e:
        print(f"Error during database operations: {e}")
    finally:
        # Close the connection
        if conn:
            conn.close()
```

```
main( Do you have a toy set that teaches numbers and
letters to kids?", 25, 100)
```

Output:

```
                              product_name  list_price  \
0             12"-20" Schwinn Training Wheels       28.17
1  Slip N Slide Triple Racer with Slide Boogies       37.21

                                      description
0  Turn any small bicycle into an instrument for ...
1  Triple Racer Slip and Slide with Boogie Boards...
```

4. Use LangChain to summarize and generate a high-quality prompt to answer the user query.

   After finding the similar products and their descriptions using pgvector, the next step is to use them for generating a prompt input for the LLM model. Since individual product descriptions can be very long, they may not fit within the specified input payload limit for an LLM model. The `MapReduceChain` from the `LangChain` framework is used to generate and combine short summaries of similarly matched products. The combined summaries are then used to build a high-quality prompt for an input to the LLM model.

```
from IPython.display import display, Markdown
from langchain_core.documents.base import Document
from langchain.chains.summarize import load_summarize_chain
from langchain_google_vertexai import VertexAI
from langchain_core.prompts import PromptTemplate

# Mock matches data
matches = [
    {"product_name": "Alphabet Learning Toy", "price": 30,
"features": "Teaches letters and numbers."},
    {"product_name": "Number Puzzle", "price": 20,
"features": "Interactive puzzle for number learning."},
]

# LangChain setup
llm = VertexAI(model_name="gemini-pro")

map_prompt_template = """
            You will be given a detailed description of a
toy product.
            This description is enclosed in triple
backticks (```).
            Using this description only, extract the name
of the toy,
            ```{text}```
            SUMMARY:
            """
map_prompt = PromptTemplate(template=map_prompt_template,
input_variables=["text"])

combine_prompt_template = """
                You will be given a detailed description of
different toy products
                enclosed in triple backticks (```) and a
question enclosed in
                double backticks(``).
                Select one toy that is most relevant to
answer the question.
                Using that selected toy description, answer
the following
                question in as much detail as possible.
                You should only use the information in the
description.
                Your answer should include the name of the
toy, the price of the toy
                and its features. Your answer should be
less than 200 words.
                Your answer should be in Markdown in a

                Description:
                ```{text}```

                Question:
                ``{user_query}``

                Answer:
                """
combine_prompt = PromptTemplate(
    template=combine_prompt_template, input_variables=
["text", "user_query"]
```

```
)

# Convert matches to LangChain documents
docs = [
    Document(page_content=f"Name: {match['product_name']},
Price: {match['price']}, Features: {match['features']}")
    for match in matches
]

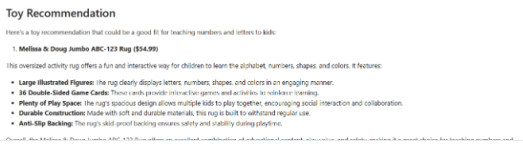
# Load and invoke the chain
chain = load_summarize_chain(
    llm, chain_type="map_reduce", map_prompt=map_prompt,
combine_prompt=combine_prompt
```

```
# User query
user_query = "Do you have a toy set that teaches numbers
and letters to kids?"

# Invoke the chain
output = chain.invoke({
    "input_documents": docs,
    "user_query": user_query,
})

# Extract and display the output
answer = output.get('output_text', ' ')
display(Markdown(answer))
```

Output:



Toy Recommendation

Here's a toy recommendation that could be a good fit for teaching numbers and letters to kids:

1. **Melissa & Doug Jumbo ABC-123 Rug ($54.99)**

This oversized activity rug offers a fun and interactive way for children to learn the alphabet, numbers, shapes, and colors. It features:

- **Large Illustrated Figures:** The rug clearly displays letters, numbers, shapes, and colors in an engaging manner.
- **36 Double-Sided Game Cards:** These cards provide interactive games and activities, to reinforce learning.
- **Plenty of Play Space:** The rug's spacious design allows multiple kids to play together, encouraging social interaction and collaboration.
- **Durable Construction:** Made with soft and durable materials, this rug is built to withstand regular use.
- **Anti-Slip Backing:** The rug's skid-proof backing ensures safety and stability during playtime.

Use case 2: Adding AI-powered creative content generation

Use knowledge from the existing dataset to generate new AI-powered content from an initial prompt.

1. A third-party seller on the retail platform wants to use the AI-powered content generation to create a detailed description of their new bicycle product.

```
# Please fill in these values.
creative_prompt = "A bicycle with brand name 'Roadstar
bike' for kids that comes with training wheels and helmet."
```

2. Leverage the `pgvector` similarity search operator to find an existing product description that closely matches the new product specified in the initial prompt.

```
import psycopg2
from pgvector.psycopg2 import register_vector
```

```
        # AlloyDB cluster connection details
        cluster_ip_address = "Cluster-ip-address"
        database_user = "postgres"
        database_password = "postgres"

        # Connect to AlloyDB cluster
        conn = psycopg2.connect(
            host=cluster_ip_address,
            user=database_user,
            password=database_password
        )

        # Register the vector type
        register_vector(conn)

        # Get the query embedding
        qe =
embeddings_service.embed_query(creative_prompt)

        # Check if qe is a valid embedding
        if not qe:
            print("Error: The query embedding is empty.")
            return
```

```
        matches = []

        # Perform the similarity search and filtering
        cur = conn.cursor()
        cur.execute(
            """
            WITH vector_matches AS (
```

```
                SELECT product_id, embedding <=> %s::vector
AS distance
                FROM product_embeddings
                WHERE embedding <=> %s::vector < %s
                ORDER BY distance ASC
                LIMIT 1
            )
            SELECT description FROM products
            WHERE product_id IN (SELECT product_id FROM
vector_matches)
            """,
            (qe, qe, similarity_threshold)
        )

        results = cur.fetchall()

        # Process the results
        matches = []
        for row in results:
            matches.append(row[0])

        if not matches:
            print("No matches found.")
        else:
            print("Matches found:", matches)

    except Exception as e:
        print(f"Error during database operations: {e}")
    finally:
        # Close the connection if it was established
        if conn:
            conn.close()

# Call the main function
main()
```

Output:

```
Matches found: ['Turn any small bicycle into an instrument for learning to ride with the Schwinn 12"-20" Training Wheels. They feature a
slotted design to fit 12" to 20" bikes. The training wheels are easy to assemble, install and remove, so that when your little one is ab
le to ride without assistance, you can take them off. These bicycle training wheels include steel brackets and rubber tires that can sta
nd up to heavy use. Training Wheels, fits 12 inches - 20 inches bicycles. Est. 1895. Durable Construction: Steel brackets stand up to he
avy use. Customizable: Two sets of wheel decals included. Features: Fits Most Childrens Bicycles: Intended for 12 inch - 20 inch bicycle
s. Steel Brackets: Offer increased durability. Includes two sets of wheel decals: Learn how to ride in style - see images below. Easy to
Adjust: Slotted design for size adjustment. Includes: One pair of training wheels, four decals, installation instructions, and all mount
ing hardware. Tools required: Adjustable wrench. www.schwinnbikes.com. Follow ride Schwinn on: Twitter. Facebook. Made in China. Trainin
g Wheels,Fits 12 inches - 20 inches bicycles. Est. 1895. Durable Construction: Steel brackets stand up to heavy use. Customizable: Two s
```

3. Use the existing matched product description as the prompt context to generate new creative output from the LLM.

```
from IPython.display import display, Markdown
from langchain_google_vertexai import VertexAI
from langchain_core.prompts import PromptTemplate
from langchain_core.runnables import RunnableSequence

# Define the template
template = """
            You are given descriptions about some similar
kind of toys in the context.
            This context is enclosed in triple backticks
(```).
            Combine these descriptions and adapt them to
match the specifications in
            the initial prompt. All the information from
the initial prompt must
            be included. You are allowed to be as creative
as possible,
            and describe the new toy in as much detail.
Your answer should be

            Context:
            ```{context}```

            Initial Prompt:
            {creative_prompt}

            Answer:
        """

prompt = PromptTemplate(
    template=template, input_variables=["context",
"creative_prompt"]
)

# Define the LLM
llm = VertexAI(model_name="gemini-pro", temperature=0.7)

# Example `matches` list
matches = [
    {"description": "This is a toy description 1."},
    {"description": "This is a toy description 2."},
    {},  # Missing `description`
    "Invalid item"  # Not a dictionary
]

context = "\n".join(
    item["description"] for item in matches if
isinstance(item, dict) and "description" in item
)
```
```

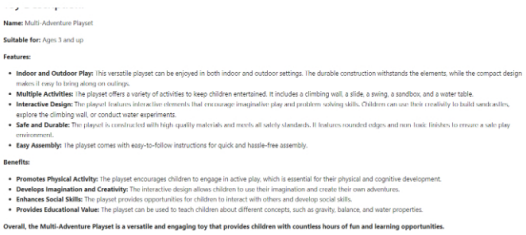
```
# Define the creative prompt
creative_prompt = "Describe a toy that is suitable for both
indoor and outdoor play."

# Use RunnableSequence for chaining
llm_chain = RunnableSequence(prompt | llm)

# Invoke the chain
answer = llm_chain.invoke({
    "context": context,
    "creative_prompt": creative_prompt,
})

# Display the answer in Markdown format
display(Markdown(answer))
```

Output:



Name: Multi-Adventure Playset

Suitable for: Ages 3 and up

Features:

- **Indoor and Outdoor Play:** This versatile playset can be enjoyed in both indoor and outdoor settings. The durable construction withstands the elements, while the compact design makes it easy to bring along on outings.
- **Multiple Activities:** The playset offers a variety of activities to keep children entertained. It includes a climbing wall, a slide, a swing, a sandbox, and a water table.
- **Interactive Design:** The playset features interactive elements that encourage imaginative play and problem-solving skills. Children can use their creativity to build sandcastles, explore the climbing wall, or conduct water experiments.
- **Safe and Durable:** The playset is constructed with high-quality materials and meets all safety standards. It features rounded edges and non-toxic finishes to ensure a safe play environment.
- **Easy Assembly:** The playset comes with easy-to-follow instructions for quick and hassle-free assembly.

Benefits:

- **Promotes Physical Activity:** The playset encourages children to engage in active play, which is essential for their physical and cognitive development.
- **Develops Imagination and Creativity:** The interactive design allows children to use their imagination and create their own adventures.
- **Enhances Social Skills:** The playset provides opportunities for children to interact with others and develop social skills.
- **Provides Educational Value:** The playset can be used to teach children about different concepts, such as gravity, balance, and water properties.

Overall, the Multi-Adventure Playset is a versatile and engaging toy that provides children with countless hours of fun and learning opportunities.

Congratulations!

At the end of the lab, you will have gained a good understanding of how to use pgvector to store and search vector embeddings in a PostgreSQL database hosted on an AlloyDB cluster. Additionally, you will have acquired hands-on experience with using LangChain to develop applications powered by Large Language Models (LLMs).

Manual Last Updated January 09, 2025

Lab Last Tested January 09, 2025

Your Next Steps