

# Oracle向量数据库与RAG应用

本节将实验向量数据库的一个典型应用场景：RAG。在RAG的解决方案中，组件要素主要包括：大语言模型（LLM）、向量嵌入模型（embedding model）、向量数据库以及 Rerank模型（非必要，根据实际情况可选，本实验不涉及Rerank模型）。

本实验中我们通过对比 直接与大模型（LLM）对话 和 使用RAG的方式与LLM对话 两者生成结果的区别来直观的了解向量数据库在这种场景中的作用。

## 大语言模型部署（仅讲师操作）

大语言模型是生成式AI的关键部分。本实验中，我们将选用开源的通义千问模型：Qwen2-7B-Instruct 考虑到硬件资源因素，本操作仅由讲师完成。

### 下载模型

从魔搭社区 (modelscope) 下载：[Qwen2-7B-Instruct](#)

### 用vLLM部署模型（GPU）

在GPU机器上可以采用vLLM来部署模型。vLLM是一个模型加速库，能大幅提升推理效率及并发。

安装Python环境及vLLM工具：

```
conda create -n vllm python=3.12

conda activate vllm

pip install vllm
```

启动运行：

```
python -m vllm.entrypoints.openai.api_server --port 8098 --model
/home/ubuntu/ChatGPT/Models/Qwen/Qwen2-7B-Instruct --served-model-name Qwen2-7B-
Instruct --device=cuda --dtype auto --max-model-len=2048
```

### 测试部署是否成功：

```
curl http://146.235.226.110:8098/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "Qwen2-7B-Instruct",
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Tell me something about large language
models."}
  ]
}'
```

## 用Ollama部署模型

开发测试也可以采用Ollama来部署模型，可以在CPU机器上运行模型。

安装 ollama:

```
curl -fsSL https://ollama.com/install.sh | sh
```

启动运行:

-- 手工启动ollama进程。这里指定了ollama服务的监听地址。如果 ollama 是以系统服务启动，则也需要将环境变量增加到系统服务中。

```
OLLAMA_HOST=0.0.0.0:8098 ollama serve
```

-- 运行模型

```
OLLAMA_HOST=127.0.0.1:8098 ollama run qwen2:7b-instruct
```

```
[root@ai23n2 ~]# export OLLAMA_HOST=0.0.0.0:8098
[root@ai23n2 ~]# cat /etc/systemd/system/ollama.service
[Unit]
Description=Ollama Service
After=network-online.target

[Service]
ExecStart=/usr/local/bin/ollama serve
User=ollama
Group=ollama
Restart=always
RestartSec=3
Environment="PATH=/opt/oracle/dcs/bin:/opt/oracle/oak/onecmd:/opt/oracle/dcs/bin:/opt/oracle/oak/bin:/opt/oracle/c
Environment="OLLAMA_HOST=0.0.0.0:8098"

[Install]
WantedBy=default.target
[root@ai23n2 ~]# █
```

## 测试部署是否成功:

```
curl http://146.235.226.110:8098/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "qwen2:7b-instruct",
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Tell me something about large language
models."}
  ]
}'
```

## 直接与LLM对话 (非RAG)

先运行如下语句，打开输出信息，这样dbms\_output就能在脚本输出窗口中输出打印信息了。

```
SET SERVEROUTPUT ON;
```

以下PL/SQL代码是直接调用LLM API的过程，也可以用其它语言实现，步骤或逻辑都一样。

```
declare
  l_question varchar2(500) := 'Oracle 23ai 新特性';
  l_input CLOB;
```

```

l_clob CLOB;
j apex_json.t_values;
l_embedding CLOB;
l_context CLOB;
l_rag_result CLOB;

begin
  apex_web_service.g_request_headers(1).name := 'Content-Type';
  apex_web_service.g_request_headers(1).value := 'application/json';
  l_input := '{"text": "' || l_question || '"}';

  -- 第一步：提示工程：给大语言模型明确的指示
  l_input := '{
    "model": "Qwen2-7B-Instruct",
    "messages": [
      {"role": "system", "content": "你是一个诚实且专业的数据库知识问答助手，请回答用户提出的问题。"},
      {"role": "user", "content": "' || l_question || '"}
    ]
  }';

  -- 第二步：调用大语言模型，生成RAG结果
  l_clob := apex_web_service.make_rest_request(
    p_url => 'http://146.235.226.110:8098/v1/chat/completions',
    p_http_method => 'POST',
    p_body => l_input
  );
  apex_json.parse(j, l_clob);
  l_rag_result := apex_json.get_varchar2(p_path =>
'choices[%d].message.content', p0 => 1, p_values => j);

  dbms_output.put_line('*** Result: ' || chr(10) || l_rag_result);
end;
/

```

运行结果：

The screenshot shows the Oracle SQL Developer interface. The top pane displays the PL/SQL script that was executed. The bottom pane shows the output of the script, which is a list of 7 features of Oracle 23ai. The output is as follows:

```

1. **智能分析改进**：Oracle 23ai 对其智能分析功能进行了增强，提供了更智能的决策支持和预测分析。这些改进包括更先进的预测模型和增强的数据预处理功能。
2. **自动化机器学习 (AutoML)**：此版本引入了支持自动化机器学习的工作流，使得非专业数据科学家也能快速构建、训练和部署预测模型。Oracle AI 提供了预训练的模型，用户只需提供数据，即可快速生成模型预测。
3. **增强的智能搜索**：Oracle 23ai 提升了其智能搜索功能，提供更精确和个性化的搜索结果，特别是在大型和复杂数据集上的表现更为出色。
4. **智能决策支持系统**：通过集成学习和决策树分析，Oracle 23ai 可以提供更深入的业务洞察，帮助决策者基于数据做出更明智的决策。
5. **AI 集成**：Oracle 提供了更紧密的 AI 集成，使得 AI 功能可以更自然地融入到现有的 Oracle 数据库环境之中，包括与 Oracle 的其他产品和服务的无缝集成。
6. **增强的安全性和合规性**：Oracle 23ai 在保持高效率的同时，也加强了数据安全性和合规性，提供了更强大的加密、访问控制和审计功能。
7. **优化的性能**：Oracle 23ai 继续优化其核心数据库性能，包括查询优化、索引管理、多租户和云部署方面的改进，以支持大量的数据分析和处理需求。

这些特性使得 Oracle 23ai 成为一个功能强大、易于使用且安全可靠的数据解决方案，尤其在企业级数据分析和人工智能应用领域。

PL/SQL procedure successfully completed.

```

## RAG方式与LLM对话

先运行如下语句，打开输出信息，这样dbms\_output就能在脚本输出窗口中输出打印信息了。

```
SET SERVEROUTPUT ON;
```

以下PL/SQL代码是执行 RAG 的过程，也可以用其它语言实现，步骤或逻辑都一样。

```
declare
    l_question varchar2(500) := 'Oracle 23ai 新特性';
    l_input CLOB;
    l_clob CLOB;
    j apex_json.t_values;
    l_embedding CLOB;
    l_context CLOB;
    l_rag_result CLOB;
begin
    apex_web_service.g_request_headers(1).name := 'Content-Type';
    apex_web_service.g_request_headers(1).value := 'application/json';
    l_input := '{"text": "' || l_question || '"}';

    -- 第一步：向量化用户问题
    l_clob := apex_web_service.make_rest_request(
        p_url => 'http://146.235.226.110:8099/workshop/embedding',
        p_http_method => 'POST',
        p_body => l_input
    );
    apex_json.parse(j, l_clob);
    l_embedding := apex_json.get_varchar2(p_path => 'data.embedding', p_values =>
j);
    -- dbms_output.put_line('*** embedding: ' || l_embedding);

    -- 第二步：从向量数据库中检索出与问题相似的内容
    for rec in (select document, json_value(cmetadata, '$.source') as src_file
        from lab_vecstore
        where dataset_name='oracledb_docs'
        order by VECTOR_DISTANCE(embedding, to_vector(l_embedding))
        FETCH APPROX FIRST 3 ROWS ONLY) loop
        l_context := l_context || rec.document || chr(10);
    end loop;

    -- 第三步：提示工程：将相似内容和用户问题一起，组成大语言模型的输入
    l_input := '{
        "model": "Qwen2-7B-Instruct",
        "messages": [
            {"role": "system", "content": "你是一个诚实且专业的数据库知识问答助手，请仅仅根据提供的上下文内容，回答用户的问题，且不要试图编造答案。\\n 以下是上下文内容: ' ||
replace(l_context, chr(10), '\\n') || '"},
            {"role": "user", "content": "' || l_question || ' (请仅根据提供的上下文内容回答)"}
        ]
    }';

    -- 第四步：调用大语言模型，生成RAG结果
```

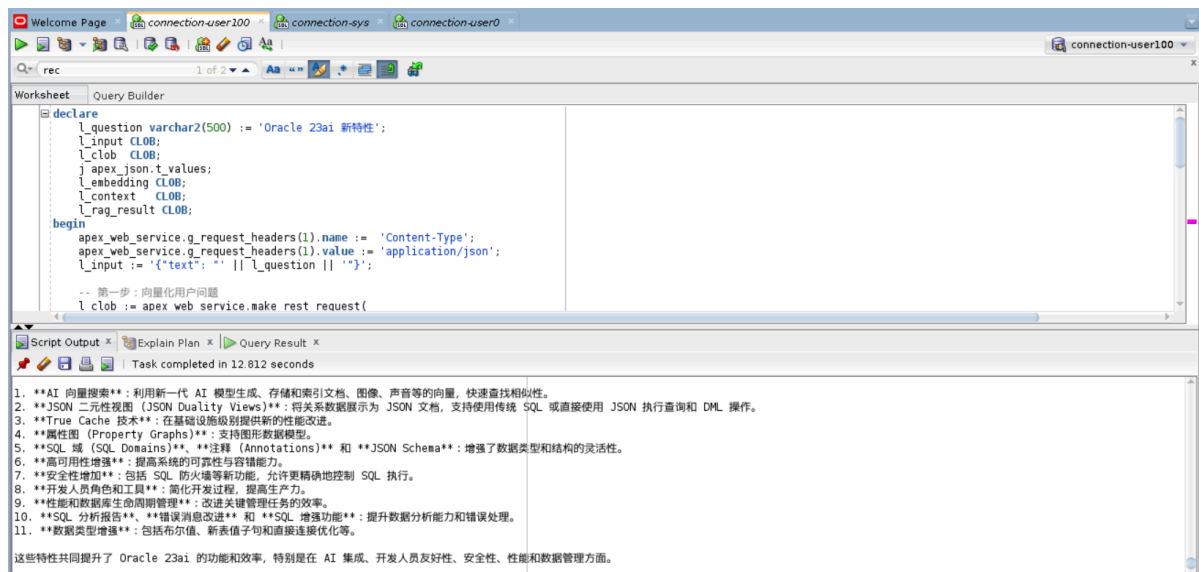
```

l_clob := apex_web_service.make_rest_request(
    p_url => 'http://146.235.226.110:8098/v1/chat/completions',
    p_http_method => 'POST',
    p_body => l_input
);
apex_json.parse(j, l_clob);
l_rag_result := apex_json.get_varchar2(p_path =>
'choices[%d].message.content', p0 => 1, p_values => j);

dbms_output.put_line('*** RAG Result: ' || chr(10) || l_rag_result);
end;
/

```

运行结果：



## Oracle 库内向量化流水线操作

Oracle数据库提供一系列工具，让用户可以用极简单的方式将源数据向量化并加载到数据库中。

本节主要目的在于：了解在Oracle库内实现一个完整的从源文件到生成向量数据 这样一个库内流水线操作：PDF文件 --> 文件文件 --> 文件分块 --> 生成向量数据。



Oracle 数据库提供了一系列的工具方法，以方便向量的操作。这些方法主要封装在 DBMS\_VECTOR / DBMS\_VECTOR\_CHAIN 这两个包中，可以直接调用。例如：

- dbms\_vector\_chain.utl\_to\_text: 将文件转换为文本格式，如PDF格式转换为文本格式。
- dbms\_vector\_chain.utl\_to\_chunks: 将文档以块的形式拆分成多个块
- dbms\_vector\_chain.utl\_to\_embeddings: 将文档块进行向量化（批量形式）。
- dbms\_vector\_chain.utl\_to\_generate\_text: 调用大语言模型，生成RAG结果。

对于【PDF文件 --> 文件文件 --> 文件分块 --> 向量化】这样一个复杂的过程，利用上面这些工具方法，在Oracle数据库中仅通过一条SQL语句即可实现。下面我们展示一下这个过程：

## 先准备数据表

```
-- 用来加载存储源文件
create table RAG_FILES (
    file_name varchar2(500),
    file_content BLOB
);

-- 用来存储文件块以及对象的向量
CREATE TABLE RAG_DOC_CHUNKS (
    "DOC_ID" VARCHAR2(500),
    "CHUNK_ID" NUMBER,
    "CHUNK_DATA" VARCHAR2(4000),
    "CHUNK_EMBEDDING" VECTOR
);
```

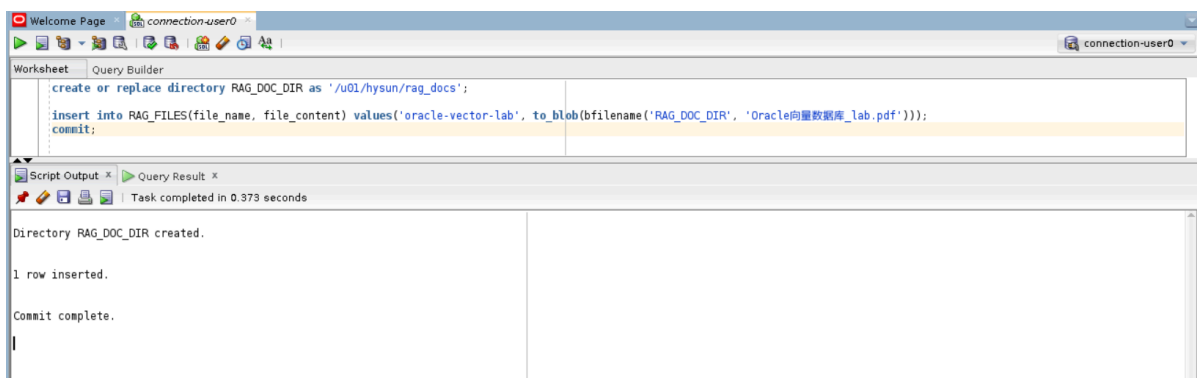
## 加载文件

加载文件有多种方式，比如从对象存储中加载、从文件服务器加载等等。为简单起见，本实验中预先将一个PDF文件上传到数据库服务器上，从本地目录加载文件。

```
-- 首先，将文件手工上传至 /u01/hysun/rag_docs 目录
-- 本实验中已经预先上传了一个PDF文件（内容就是本实验的PDF指导文件）

-- 然后再创建数据库目录，如下
create or replace directory RAG_DOC_DIR as '/u01/hysun/rag_docs';

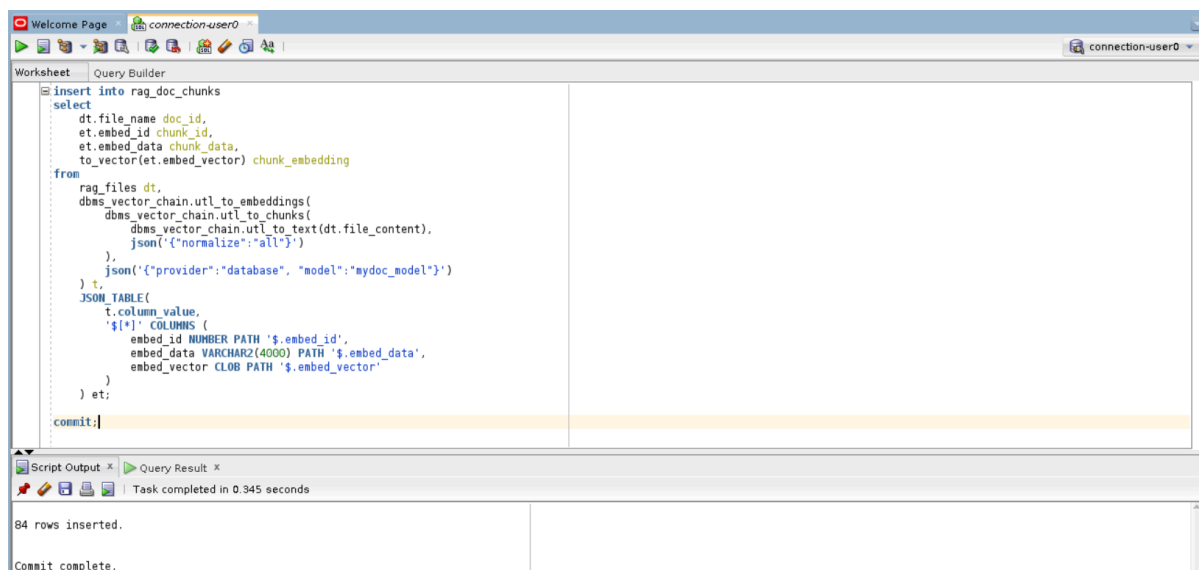
-- 从数据目录下加载源文件入库
insert into RAG_FILES(file_name, file_content) values('oracle-vector-lab',
to_blob(bfilename('RAG_DOC_DIR', 'Oracle向量数据库_lab.pdf')));
commit;
```



## 执行 文件转换-->文档拆分-->向量化

以下用一条SQL完成了【PDF格式 -> 文本格式 -> 文档分块 -> 向量化】这样一个比较复杂的流程：

```
insert into rag_doc_chunks
select
  dt.file_name doc_id,
  et.embed_id chunk_id,
  et.embed_data chunk_data,
  to_vector(et.embed_vector) chunk_embedding
from
  rag_files dt,
  dbms_vector_chain.utl_to_embeddings(
    dbms_vector_chain.utl_to_chunks(
      dbms_vector_chain.utl_to_text(dt.file_content),
      json('{"normalize":"all"}')
    ),
    json('{"provider":"database", "model":"mydoc_model"}')
  ) t,
  JSON_TABLE(
    t.column_value,
    '[$*]' COLUMNS (
      embed_id NUMBER PATH '$.embed_id',
      embed_data VARCHAR2(4000) PATH '$.embed_data',
      embed_vector CLOB PATH '$.embed_vector'
    )
  ) et;
commit;
```



## 向量相似度检索

源数据完成向量化后，就可以利用 VECTOR\_DISTANCE 进行向量相似度检索了。

```
select *
from rag_doc_chunks
order by VECTOR_DISTANCE(chunk_embedding, VECTOR_EMBEDDING(mydoc_model USING '本次实验的先决条件' as data), COSINE)
FETCH FIRST 3 ROWS ONLY;
```

The screenshot shows the Oracle SQL Developer interface. The top pane displays a SQL query: `select * from rag_doc_chunks order by VECTOR_DISTANCE(chunk_embedding, VECTOR_EMBEDDING(mydoc_model USING '本次实验的先决条件' as data), COSINE) FETCH FIRST 3 ROWS ONLY;`. The bottom pane shows the query results in a table with three columns: DOC\_ID, CHUNK\_ID, and CHUNK\_DATA. The results are as follows:

DOC_ID	CHUNK_ID	CHUNK_DATA
1 oracle-vector-lab	84 rag_doc_chunks	orderBy VECTOR_DISTANCE(chunk_embedding, VECTOR_EMBEDDING(mydoc_model USING '本次实验的先决条件' as data), COSINE) FETCH FIRST 1 ROWS ONLY;
2 oracle-vector-lab	25	5 精确检索和索引近似检索)、RAG。前提条件 1. 本实验重点在于动手操作, 非对向量数据库及Oracle向量数据库进行理论上的讲解, 因此, 需要参与者对向量数据库及Oracle向量数据库有一个基
3 oracle-vector-lab	10	掉任何一条记录(也就是说, 召回率始终能达到 100%)。由于结果的准确性, 毫无疑问, 在需要遍历的向量数据集较小时, 精确检索是较优的方式。在使用如Oracle

## 结合向量相似度检索做RAG

```
set serveroutput on;

declare
    l_question varchar2(500) := '完成本次实验的前提条件需要哪些';
    l_input CLOB;
    l_clob CLOB;
    j apex_json.t_values;
    l_context CLOB;
    l_rag_result CLOB;
begin
    -- 第一步: 从向量数据库中检索出与问题相似的内容
    for rec in (
        select
            chunk_data
        from rag_doc_chunks
        order by VECTOR_DISTANCE(chunk_embedding, VECTOR_EMBEDDING(mydoc_model
USING l_question as data), COSINE)
        FETCH FIRST 1 ROWS ONLY
    ) loop
        l_context := l_context || rec.chunk_data || chr(10);
    end loop;

    -- 第二步: 提示工程: 将相似内容和用户问题一起, 组成大语言模型的输入
    l_context := replace(replace(replace(l_context, ' ', ''), ' ', '\ '),
chr(10), '\n');

    l_input := '{
        "model": "Qwen2-7B-Instruct",
        "messages": [
            {"role": "system", "content": "你是一个诚实且专业的数据库知识问答助手, 请仅仅
根据提供的上下文内容, 回答用户的问题, 且不要试图编造答案。\\n 以下是上下文内容: ' ||
replace(l_context, chr(10), '\\n') || '"},
            {"role": "user", "content": "' || l_question || ' (请仅根据提供的上下文内
容回答)"}
        ]
    }';

    -- 第三步: 调用大语言模型, 生成RAG结果
    l_clob := apex_web_service.make_rest_request(
        p_url => 'http://146.235.226.110:8098/v1/chat/completions',
        p_http_method => 'POST',
        p_body => l_input
    );
```

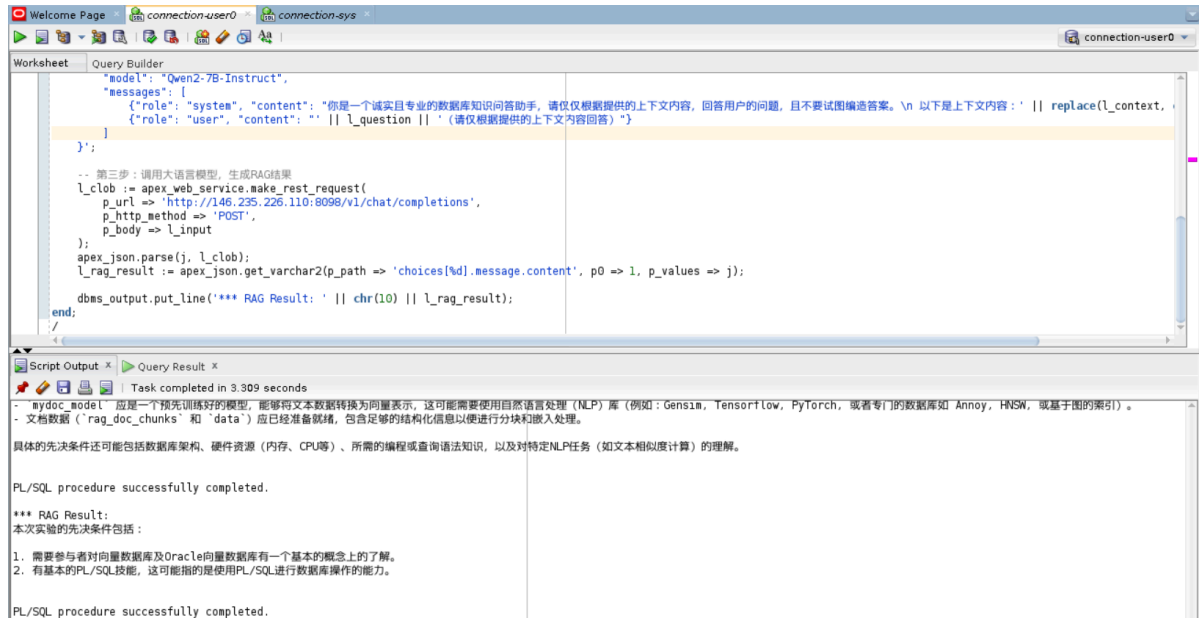


```

apex_json.parse(j, l_clob);
l_rag_result := apex_json.get_varchar2(p_path =>
'choices[%d].message.content', p0 => 1, p_values => j));

dbms_output.put_line('*** RAG Result: ' || chr(10) || l_rag_result);
end;
/

```



The screenshot displays the Oracle SQL Developer environment. The top pane shows a PL/SQL script in the Query Builder, which includes a REST client call to a Qwen2-7B-Instruct API and a PL/SQL procedure to parse the JSON response and output the RAG result. The bottom pane shows the Script Output window, which contains the execution log, including the completion message and the RAG result text.

**Script Output:**

```

Task completed in 3.309 seconds
- mydoc_model 应是一个预先训练好的模型，能够将文本数据转换为向量表示。这可能需要使用自然语言处理（NLP）库（例如：Gensim, TensorFlow, PyTorch, 或者专门的数据库如 Annoy, HNSW, 或基于图的索引）。
- 文档数据（rag_doc_chunks 和 data）应已经准备就绪，包含足够的结构化信息以便进行分块和嵌入处理。

具体的先决条件还可能包括数据库架构、硬件资源（内存、CPU等）、所需的编程或查询语法知识，以及对特定NLP任务（如文本相似度计算）的理解。

PL/SQL procedure successfully completed.

*** RAG Result:
本次实验的先决条件包括：
1. 需要参与者对向量数据库及Oracle向量数据库有一个基本的概念上的了解。
2. 有基本的PL/SQL技能，这可能指的是使用PL/SQL进行数据库操作的能力。

PL/SQL procedure successfully completed.

```