

Oracle 23ai 向量数据库

向量数据库动手实验(二)

技术文档

SE Hub

Hysun He

November, 2024

变更记录

日期	作者	版本	变更参考
11/12/2024	贺友胜 (Hysun He)	1.0	初始版本

审核

日期	审核人	版本	变更记录

概要

本实验是 Oracle 向量数据库动手实验的第二部分内容。

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）。

为方便拷贝粘贴，使用过程中也可以借助本文档的 Markdown 版本：

https://github.com/HysunHe/23ai_workshop_prep/blob/main/Oracle%E5%90%91%E9%87%8F%E6%95%B0%E6%8D%AE%E5%BA%93_lab2.md

预计时间：1 小时

目标

- 了解大语言模型(LLM)部署
- 了解 RAG 的作用
- 了解利用 Oracle 向量检索与 LLM 结合实现 RAG 的基本步骤
- 了解 Oracle 库内向量化流水线操作的主要函数
- 利用向量化流水线操作的结果数据，再次实现 RAG，进一步熟悉 RAG 的实现

前提条件

参与者已经熟悉 Oracle 数据库向量检索的基本操作，最好是已经参加了 Oracle 向量数据库动手实验的第一部分内容。

环境准备

本文档中涉及的账号密码，请参考《动手试验环境说明.pdf》文档 或 咨询现场技术人员。

目录

1 大语言模型部署（仅讲师操作）	1
1.1 下载模型.....	1
1.2 用 vLLM 部署模型	2
1.3 测试部署	2
2 直接与 LLM 对话（非 RAG）	3
3 RAG 方式与 LLM 对话	5
3.1 比较实验 2 和实验 3 的结果，理解导致差异的原因	6
4 Oracle 库内向量化流水线操作.....	7
4.1 准备数据表.....	8
4.2 加载文件	8
4.3 执行【文件转换-->文档拆分-->向量化】流水线.....	9
4.4 向量相似度检索	10
4.5 RAG.....	10
5 总结.....	13

1 大语言模型部署（仅讲师操作）

此节内容仅讲师动手操作及讲解。

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。

模型部署将部署到以下 GPU (A10) 机器上（外网可访问）：

- 机器 IP: 146.235.226.110

1.1 下载模型

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

1.2 用 vLLM 部署模型

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：

```
# 创建 Python 环境
conda create -n vllm python=3.12

# 进入新创建的环境
conda activate vllm

# 安装 vllm 依赖包
pip install vllm
pip install vllm-flash-attn
```

启动运行：

```
nohup python -u -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 > vllm.out
2>&1 &
```

1.3 测试部署

```
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."}
  ]
}'
```

2 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。

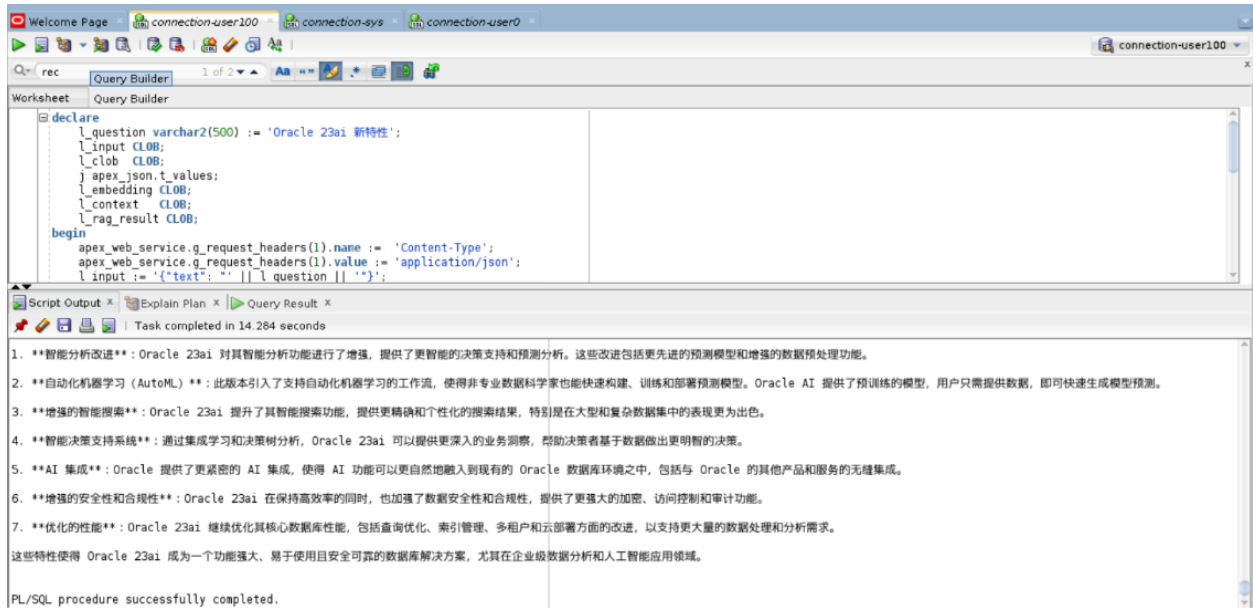
```
declare
    l_question varchar2(500) := 'Oracle 23ai 新特性';
    l_input CLOB;
    l_clob CLOB;
    j apex_json.t_values;
    l_embedding CLOB;
    l_context CLOB;
    l_rag_result CLOB;
begin
    apex_web_service.g_request_headers(1).name := 'Content-Type';
    apex_web_service.g_request_headers(1).value := '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 displays the Oracle SQL Developer environment. The top toolbar includes icons for running and saving. The 'Query Builder' tab is active, showing a PL/SQL procedure named 'Oracle 23ai 新特性'. The procedure declares several variables and uses the 'apex_web_service' package to make an API call. The 'Script Output' pane at the bottom shows the execution log, which includes a list of seven features of Oracle 23ai and a final confirmation message: 'PL/SQL procedure successfully completed.'

```
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 || '"}';
```

Task completed in 14.284 seconds

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，以支持更大量的数据处理和分析需求。

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.

3 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 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
```

```

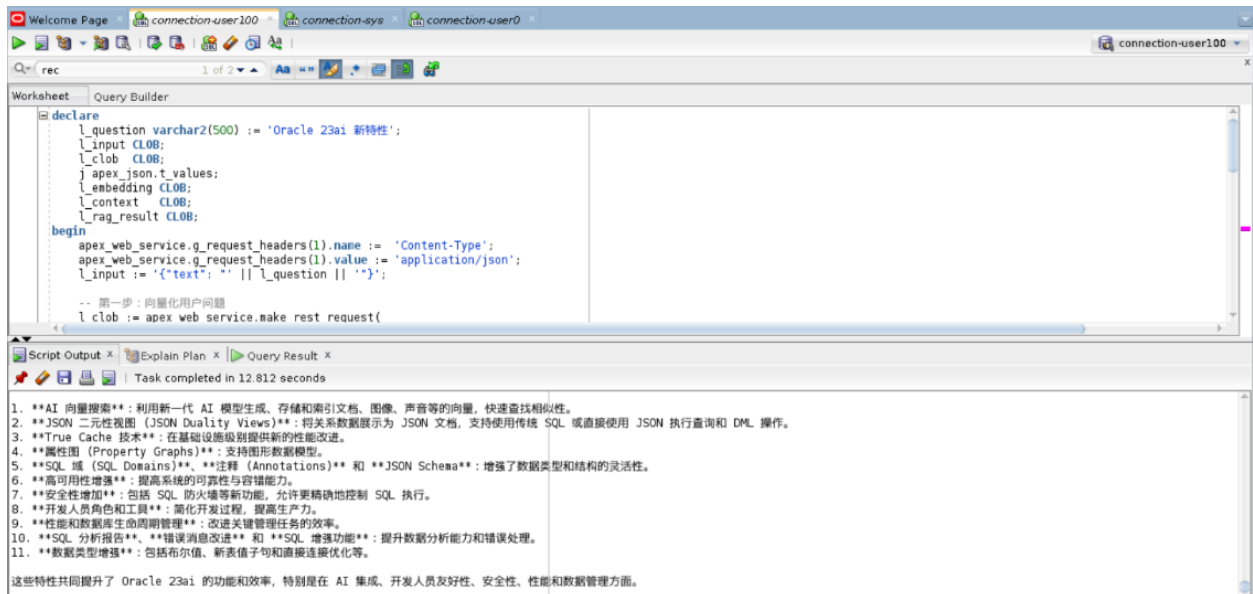
l_context := replace(replace(replace(l_context, '''', ''), '''', '\'),
chr(10), '\n');
l_input := '{
  "model": "Qwen2-7B-Instruct",
  "messages": [
    {"role": "system", "content": "你是一个诚实且专业的数据库知识问答助手, 请
根据提供的上下文内容, 回答用户的问题.\n 以下是上下文内容: ' || l_context || '"},
    {"role": "user", "content": "' ||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

-- 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：



The screenshot shows the Oracle SQL Developer interface. The top pane displays a PL/SQL script that defines variables for a REST API call to a Qwen2-7B-Instruct model. The script sets the URL to 'http://146.235.226.110:8098/v1/chat/completions' and constructs a JSON body with a system prompt and a user question. The bottom pane shows the execution results, which are a list of 11 items describing the features of Oracle 23ai, such as AI vector search, JSON duality views, True Cache, and enhanced security.

Script Output:

1. **AI 向量搜索**：利用新一代 AI 模型生成、存储和索引文档、图像、声音等的向量，快速查找相似性。
2. **JSON 二元性视图 (JSON Duality Views)**：将关系数据展示为 JSON 文档，支持使用传统 SQL 或直接使用 JSON 执行查询和 DML 操作。
3. **True Cache 技术**：在基础设施级别提供新的性能改进。
4. **属性图 (Property Graphs)**：支持图形数据模型。
5. **SQL 域 (SQL Domains)**、**注释 (Annotations)** 和 **JSON Schema**：增强了数据类型和结构的灵活性。
6. **高可用性增强**：提高系统的可靠性与容错能力。
7. **安全性增强**：包括 SQL 防火墙等新功能，允许更精确地控制 SQL 执行。
8. **开发人员角色和工具**：简化开发过程，提高生产力。
9. **性能和数据库生命周期管理**：改进关键管理任务的效率。
10. **SQL 分析报告**、**错误消息改进** 和 **SQL 增强功能**：提升数据分析能力和错误处理。
11. **数据类型增强**：包括布尔值、新表值子句和直接连接优化等。

这些特性共同提升了 Oracle 23ai 的功能和效率，特别是在 AI 集成、开发人员友好性、安全性、性能和数据管理方面。

3.1 比较实验 2 和实验 3 的结果，理解导致差异的原因

4 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 --> 文本文件 --> 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 --> 文本文件 --> 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。

4.1 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
);
```

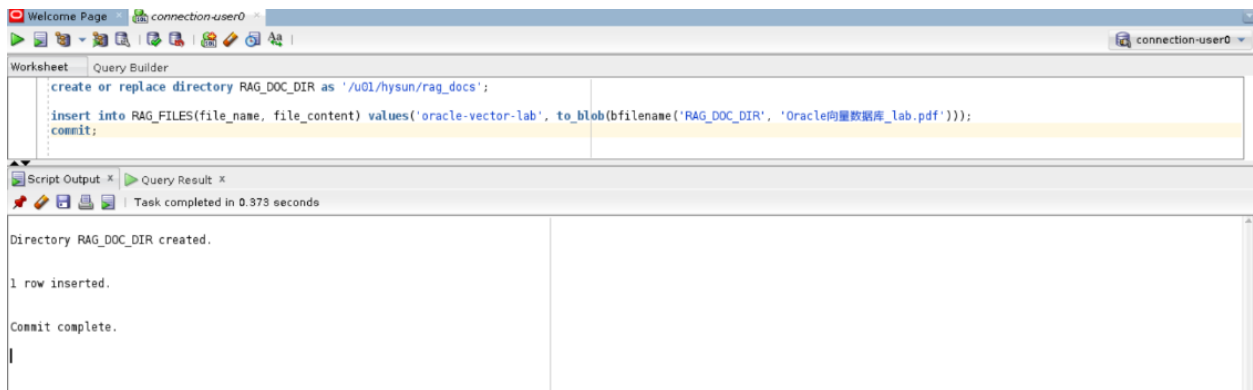
4.2 加载文件

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';

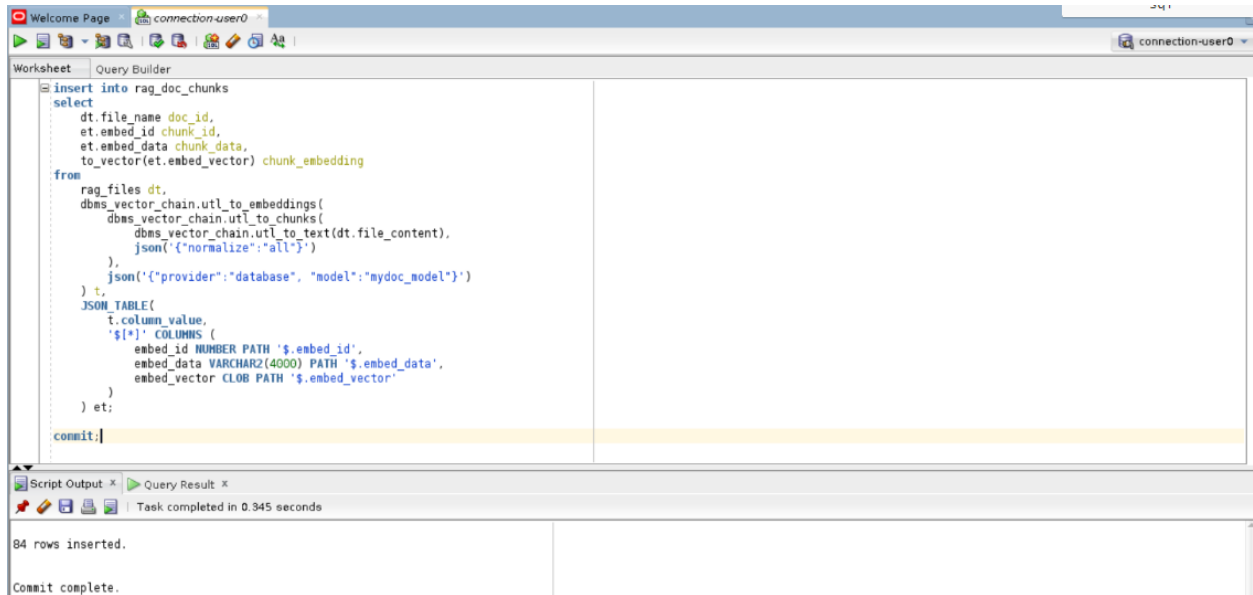
-- 从数据目录下加载源文件入库
insert into RAG_FILES(file_name, file_content) values('oracle-vector-lab',
to_blob(bfilename('RAG_DOC_DIR', 'Oracle 向量数据库_lab.pdf')));
commit;
```



4.3 执行【文件转换-->文档拆分-->向量化】流水线

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;
```



```

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;
  
```

84 rows inserted.
Commit complete.

4.4 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;
  
```



```

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;
  
```

DOC_ID	CHUNK_ID	CHUNK_DATA
1 oracle-vector-lab	84	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;
2 oracle-vector-lab	25	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，并非对向量数据库及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量较小时，精确检索是较优的方式。在使用如Oracle

4.5 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 3 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
根据提供的上下文内容，回答用户的问题。\\n 以下是上下文内容：' || l_context || '"},
            {"role": "user", "content": "' || l_question || ' (请仅根据提供的上下
文内容回答，不要试图编造答案)"}
        ]
    }';

    -- 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;
/
```

Worksheet | Query Builder

```

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,
  {role: "user", "content": "' || l_question || ' ' (请仅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://146.235.225.110:8098/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;

```

Script Output | Query Result

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'`）应已经准备好并结构化，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.

5 总结

至此，我们已经完成了 Oracle 向量数据库的动手实验第二部分。在本节中，我们重点实现了结合 Oracle 向量检索的 RAG 应用。本节中，我们 RAG 的实现用的是 PL/SQL，实际上它是不局限于用哪种编程语言的，比如 Python、Java、Go 等等都可以。

本实验的重点在于了解如何利用 Oracle 向量检索实现 RAG 应用的原理和方法。对于 RAG 应用，能否生成高质量的回答，除了大语言模型本身的能力外，还取决于高质量的输入文档和文档拆分技术或方式、构建相对合理的提示词（提示工程）、等等其它诸多方面，需要结合实际情况综合考虑。