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

# 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

## 下载模型

从魔搭社区 (modelscope) 下载：[Qwen2-7B-Instruct](https://www.modelscope.cn/models/qwen/Qwen2-7B-Instruct)

## 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 &

## 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."}

]

}'

# 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);

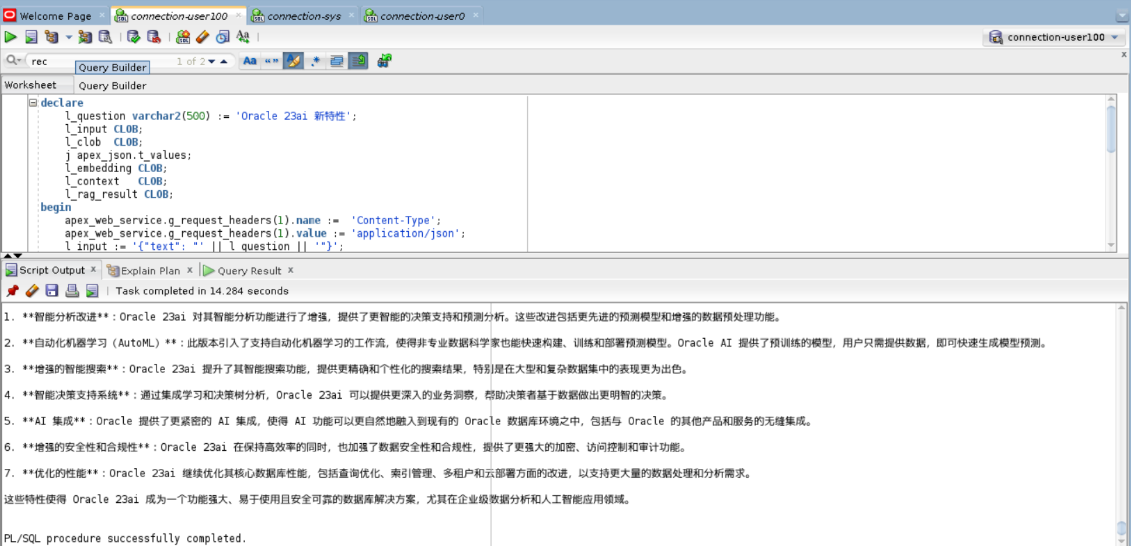
l\_rag\_result := apex\_json.get\_varchar2(p\_path => 'choices[%d].message.content', p0 => 1, p\_values => j);

dbms\_output.put\_line('\*\*\* Result: ' || chr(10) || l\_rag\_result);

end;

/

运行结果：



# 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：



## 比较实验2和实验3的结果，理解导致差异的原因

# 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。

## 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)

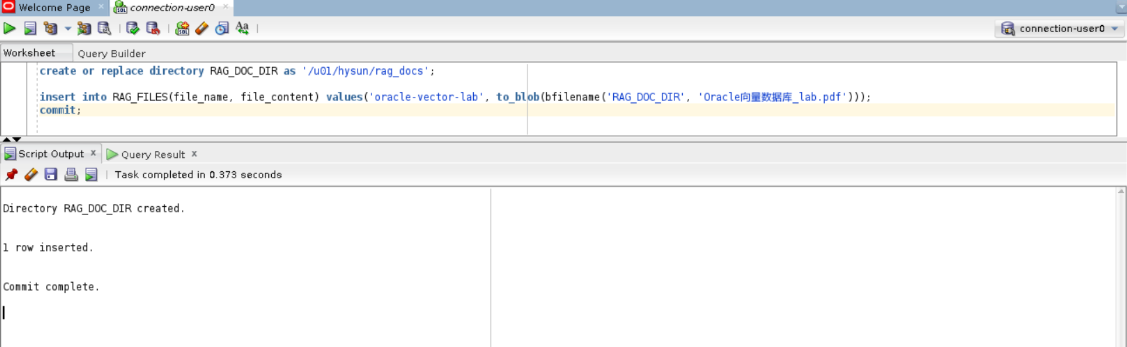
-- 然后再创建数据库目录，如下

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】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',

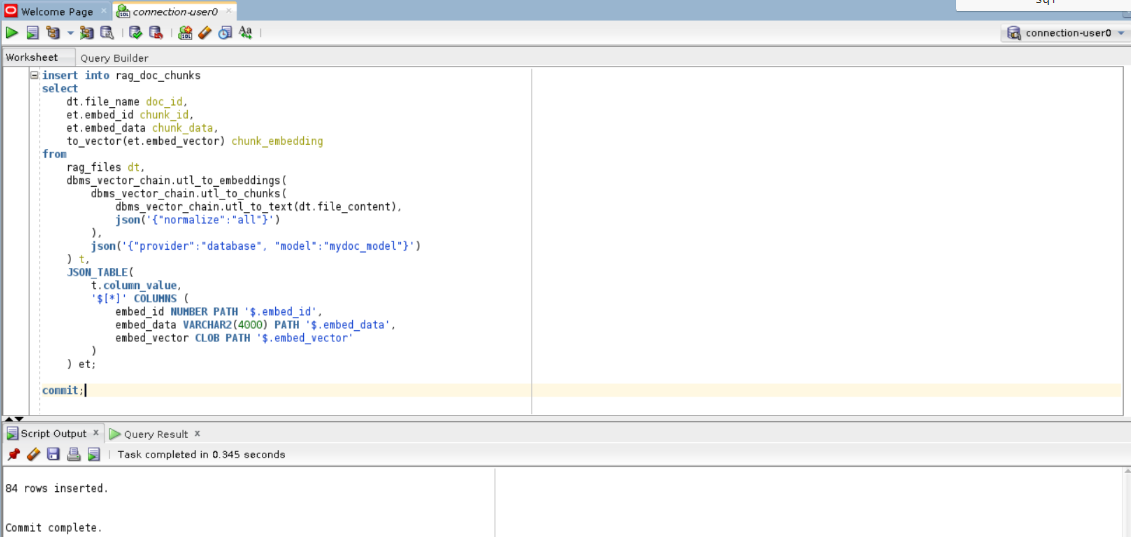
embed\_data VARCHAR2(4000) PATH '$.embed\_data',

embed\_vector CLOB PATH '$.embed\_vector'

)

) et;

commit;



## 向量相似度检索

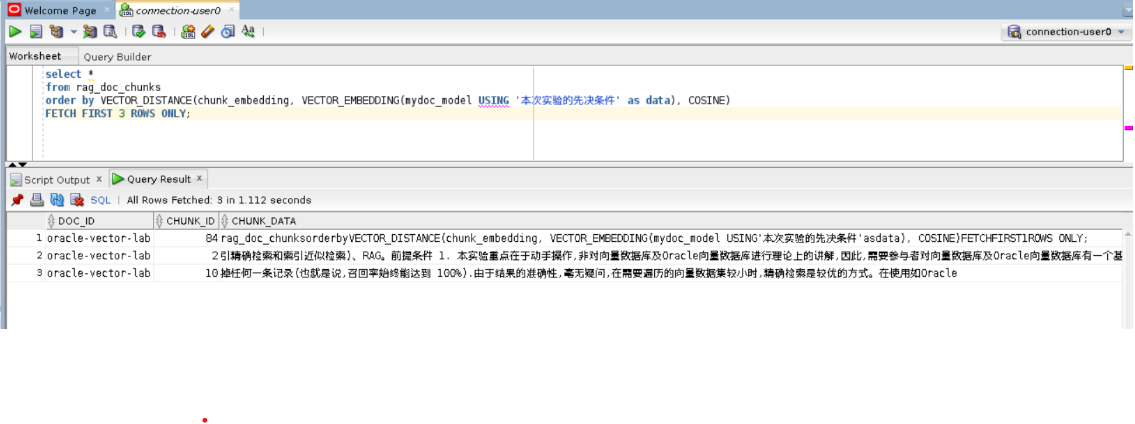
源数据完成向量化后，就可以利用 VECTOR\_DISTANCE 进行向量相似度检索了。

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;



## 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

);

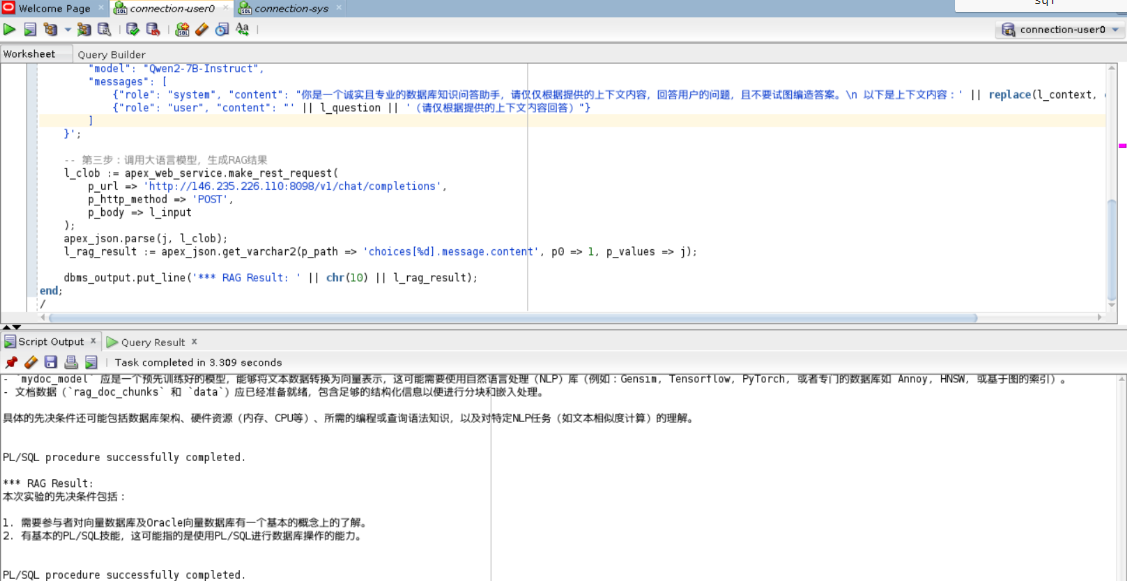
apex\_json.parse(j, l\_clob);

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;

/



# 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。