Oracle 23ai向量数据库

向量数据库动手实验(一)

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**

本实验以熟悉Oracle向量数据库的一些实际操作为主要目的，主要内容包括 Oracle向量数据类型、向量模型、数据向量化（库外向量化与库内向量化两种方式）、向量索引（HNSW和IVF）、向量检索（非索引精确检索和索引近似检索）。

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_lab1.md>

预计时间：1.5小时

**目标**

* 了解Oracle向量数据类型及基本操作
* 了解向量相似度检索，包括精确检索和近似检索
* 了解常用的向量索引类型
* 了解向量模型的部署以及Oracle数据库与外部向量模型的结合
* 了解向量模型的导入以及Oracle数据库的库内向量化操作
* 了解库外向量化及库内向量化的优劣现状及前景

**前提条件**

* 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，需要参与者已经参加过Oracle向量数据库的介绍或有所了解。
* 有基本的PL/SQL知识，能够看简单的PL/SQL示例代码，能够利用客户端（如sqlplus）等运行提供的PL/SQL示例代码。
* 最好有基本的Python知识，能够看懂简单的Python示例代码（非必需）。

**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 Oracle向量基本操作 1

1.1 字符串转换为向量 1

1.2 向量转换为字符串 1

1.3 向量间距离计算 1

1.3.1 利用欧氏距离(L2)策略计算两个向量之间的距离 1

1.3.2 利用余弦距离策略计算两个向量之间的距离 1

1.4 向量类型字段及样例表 2

1.5 样例数据 2

2 向量检索 3

2.1 向量精确检索 3

2.2 向量近似检索 4

2.2.1 向量内存池 4

2.2.2 向量内存池视图 5

2.2.3 创建HNSW索引 6

2.2.4 HNSW 近似检索 7

2.2.5 创建IVF索引 7

2.2.6 IVF 近似检索 8

3 部署向量嵌入模型（仅讲师操作） 10

3.1 向量嵌入模型部署 10

3.2 向量嵌入模型访问 10

3.3 库外向量化操作 11

3.3.1 数据加载 11

3.3.2 向量检索 13

4 库内向量化操作 16

4.1 导入向量嵌入模型 16

4.2 库内向量化及检索 18

4.2.1 准备数据 18

4.2.2 库内向量化 19

4.2.3 相似度检索 20

5 与第三方向量嵌入模型服务集成（仅演示） 21

5.1 开通第三方API服务 21

5.2 创建访问凭证 21

5.3 直接在SQL中调用外部Embedding服务 22

6 总结 24

# Oracle向量基本操作

## 字符串转换为向量

TO\_VECTOR()用来将字符串类型的数字数组转换为向量类型。

SELECT TO\_VECTOR( '[3,3]');

## 向量转换为字符串

FROM\_VECTOR()用来将向量类型转换为字符串类型。

SELECT FROM\_VECTOR( TO\_VECTOR( '[3,3]') );

## 向量间距离计算

VECTOR\_DISTANCE(v1, v2, 距离策略) 是向量检索的关键操作，用来比较两个向量的距离（相似度）。距离越大，说明相似度越小；反之，说明两个向量越相似。 Oracle支持的距离策略主要有：EUCLIDEAN, COSINE, DOT, HAMMING

### 利用欧氏距离(L2)策略计算两个向量之间的距离

SELECT VECTOR\_DISTANCE( vector('[2,2]'), vector('[5,6]'), EUCLIDEAN ) as distance;

注：欧几里得距离是指连接这两点的线段的长度（二维空间中），上述 [2,2] 和 [5,6] 两点间的距离由勾股定理可直接算出为 5

### 利用余弦距离策略计算两个向量之间的距离

SELECT VECTOR\_DISTANCE( vector('[2,2]'), vector('[5,5]'), COSINE) as distance;

注：余弦距离策略关注的是两个向量在方向上的一致性，上述 [2,2] 和 [5,5] 在方向上完全一致，因此，它们的距离为0，代表两个向量完全匹配。

## 向量类型字段及样例表

Oracle 23ai 引入了向量数据类型：VECTOR (dimentions, format)，该类型可指定两个参数，第一个是向量的维度，如 [2,2] 是一个二维向量；第二个是数据格式，如 FLOAT32。也可以不指定。

建立一个测试表 galaxies:

create table galaxies (

id number,

name varchar2(50),

doc varchar2(500),

embedding VECTOR

);

## 样例数据

向 galaxies 表中插入如下样例数据：

insert into galaxies values (1, 'M31', 'Messier 31 is a barred spiral galaxy in the Andromeda constellation.', '[0,1,1,0,0]');

insert into galaxies values (2, 'M33', 'Messier 33 is a spiral galaxy in the Triangulum constellation.', '[0,0,1,0,0]');

insert into galaxies values (3, 'M58', 'Messier 58 is an intermediate barred spiral galaxy in the Virgo constellation.', '[1,1,1,0,0]');

insert into galaxies values (4, 'M63', 'Messier 63 is a spiral galaxy in the Canes Venatici constellation.', '[0,0,1,0,0]');

insert into galaxies values (5, 'M77', 'Messier 77 is a barred spiral galaxy in the Cetus constellation.', '[0,2,2,0,0]');

insert into galaxies values (6, 'M91', 'Messier 91 is a barred spiral galaxy in the Coma Berenices constellation.', '[0,3,3,0,0]');

insert into galaxies values (7, 'M49', 'Messier 49 is a giant elliptical galaxy in the Virgo constellation.', '[0,0,0,1,1]');

insert into galaxies values (8, 'M60', 'Messier 60 is an elliptical galaxy in the Virgo constellation.', '[0,0,0,0,1]');

insert into galaxies values (9, 'NGC1073', 'NGC 1073 is a barred spiral galaxy in Cetus constellation.', '[0,3,3,0,0]');

commit;

数据准备好后，接下来，我们就可以根所数据进行检索了。

# 向量检索

## 向量精确检索

向量精确检索（Exact Search）类似于关系数据查询时的全表扫描，是指库中的每一个向量都与查询向量进行匹配，这样就能计算出每个向量与查询向量之间的相似度，从而精确的返回与查询向量最相似的 N 条记录，不会漏掉任何一条记录（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这类融合数据库时，很多情况下，可以使用关系数据的业务属性字段（标量字段）缩小需要进行向量匹配的数据，因此，结合关系数据库特征，可以很大程序上提高向量检索的精确性和性能。

SQL 查询语句：利用余弦策略检索出与向量 [0,1,1,0,0] 最相近的3条记录：

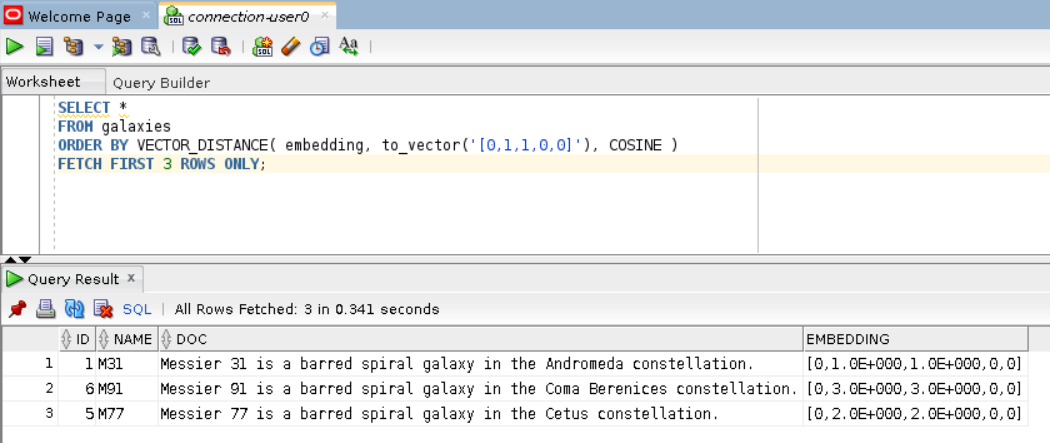
SELECT \*

FROM galaxies

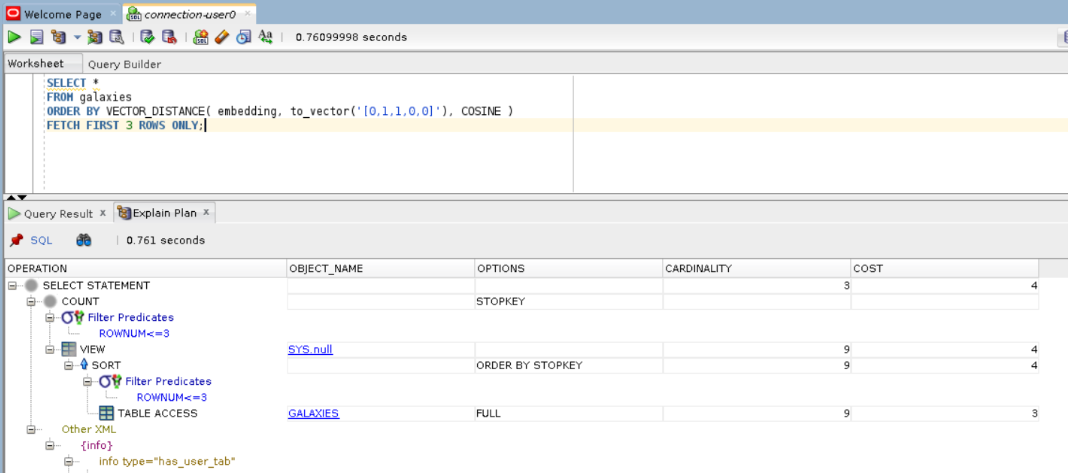
ORDER BY VECTOR\_DISTANCE( embedding, to\_vector('[0,1,1,0,0]'), COSINE )

FETCH FIRST 3 ROWS ONLY;

查询结果：



查看执行计划：



## 向量近似检索

向量近似检索（Approximate Search）

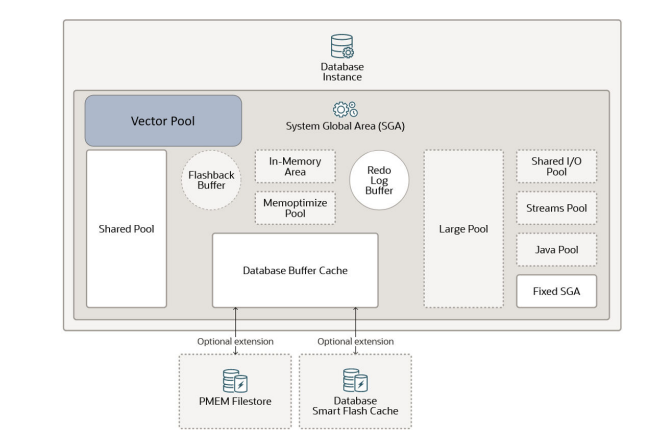
精确检索获得了最高的准确率，但需要遍历所有向量数据集，因此，在向量数据集比较大时，性能很可能会成为问题。向量检索中，准确率和性能之间，往往需要寻找一个平衡。在大数据集上，为了提高性能，利用索引进行向量近似检索是常用的方式。

常见的向量索引有HNSW和IVF两种。

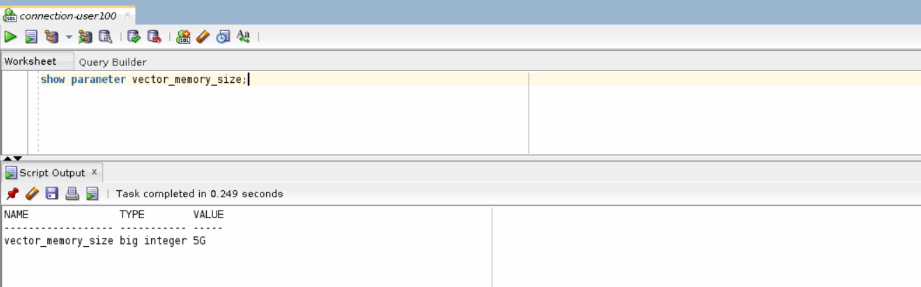
### 向量内存池

要允许创建向量索引，必须在SGA中启用一块新的内存区域，称为向量内存池。向量内存池用于存储HNSW向量索引和所有相关元数据，以及用于加速IVF索引创建和维护。

向量内存池参数可以由管理员用户进行修改。



show parameter vector\_memory\_size;



向量内存池大小估算公式： size of vector pool = 1.3 \* number of vectors \* number of dimensions \* size of vector dimension type

### 向量内存池视图

V$VECTOR\_MEMORY\_POOL视图包含了向量内存的分配和使用情况。比如：

SELECT

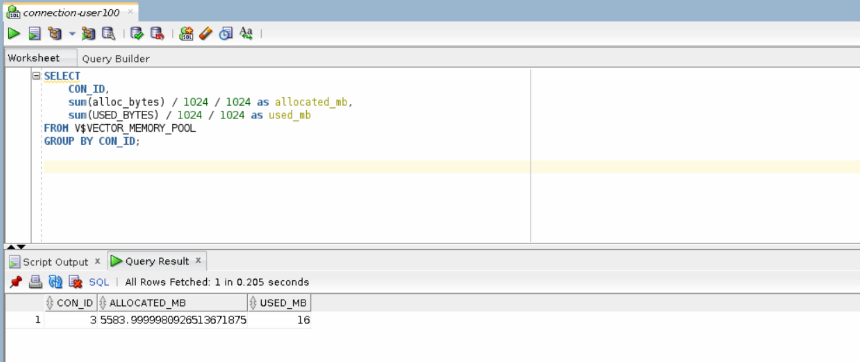
CON\_ID,

sum(alloc\_bytes) / 1024 / 1024 as allocated\_mb,

sum(USED\_BYTES) / 1024 / 1024 as used\_mb

FROM V$VECTOR\_MEMORY\_POOL

GROUP BY CON\_ID;



### 创建HNSW索引

创建索引语句：

CREATE VECTOR INDEX galaxies\_hnsw\_idx ON galaxies (embedding)

ORGANIZATION INMEMORY NEIGHBOR GRAPH

DISTANCE COSINE

WITH TARGET ACCURACY 90;

-- PARAMETERS (type HNSW, neighbors 32, efconstruction 200)

-- parallel 2;

创建 HNSW 索引时，我们可以指定目标准确率 target accuracy，并行执行；还可以指定 HNSW 的参数 M (即 neighbors) 和 efConstruction (如上面注释掉的 Parameters 一行)。关于 HNSW 相关参数的说明可以参考如下文档：

<https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-vector-search-users-guide.pdf> (184页)

<https://learn.microsoft.com/en-us/javascript/api/@azure/search-documents/hnswparameters?view=azure-node-latest>

### HNSW 近似检索

查询SQL:

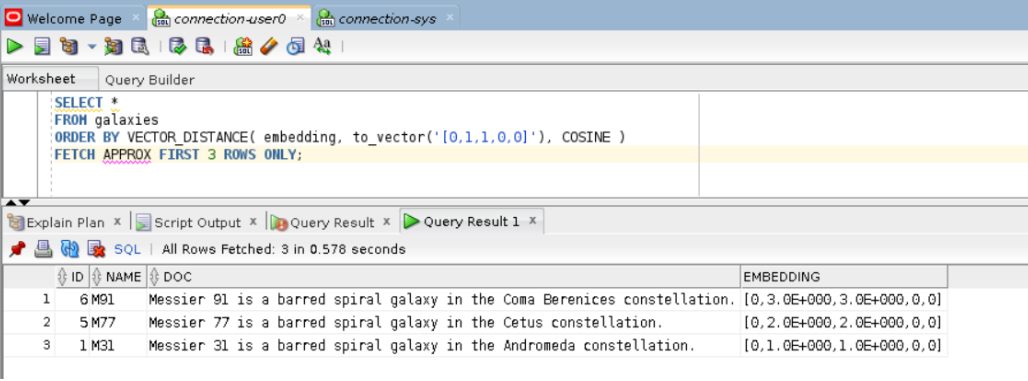
SELECT \*

FROM galaxies

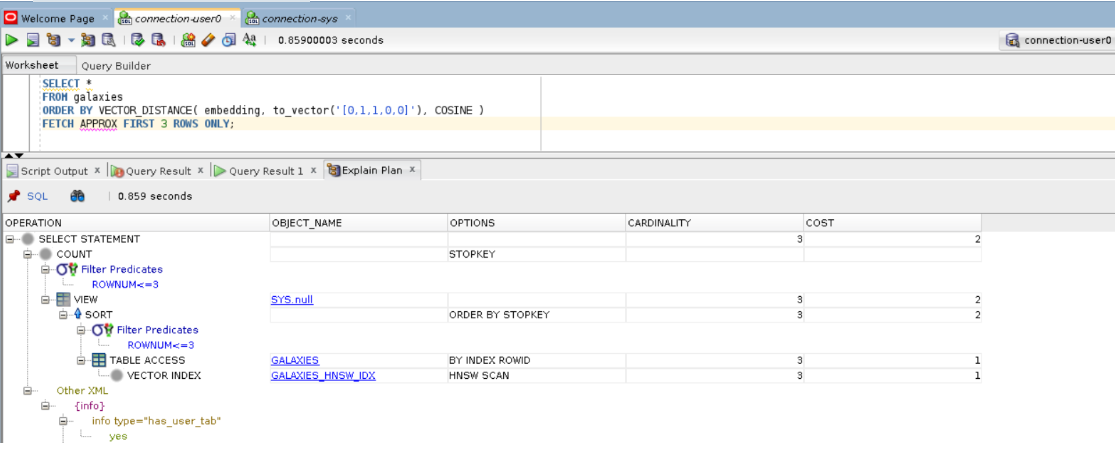
ORDER BY VECTOR\_DISTANCE( embedding, to\_vector('[0,1,1,0,0]'), COSINE )

FETCH APPROX FIRST 3 ROWS ONLY;

查询结果：



查看执行计划：



### 创建IVF索引

如果之前已经在对应的列上创建了向量索引，那么先将其删除，如：

drop index galaxies\_hnsw\_idx;

创建IVF索引语句：

CREATE VECTOR INDEX galaxies\_ivf\_idx ON galaxies(embedding)

ORGANIZATION NEIGHBOR PARTITIONS

DISTANCE COSINE

WITH TARGET ACCURACY 90;

-- PARAMETERS (type IVF, neighbor partitions 32)

-- parallel 2;

创建 IVF 索引时，我们可以指定目标准确率 target accuracy、并行执行参数，还可以指定 partition 数量等参数。关于 IVF 参数的说明，可以参考如下文档：

<https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-vector-search-users-guide.pdf> (196页)

### IVF 近似检索

创建了IVF索引之后，我们利用索引进行近似检索（注：由于我们的实验用的数据集很小，所以优化器很可能不会选择走IVF索引）

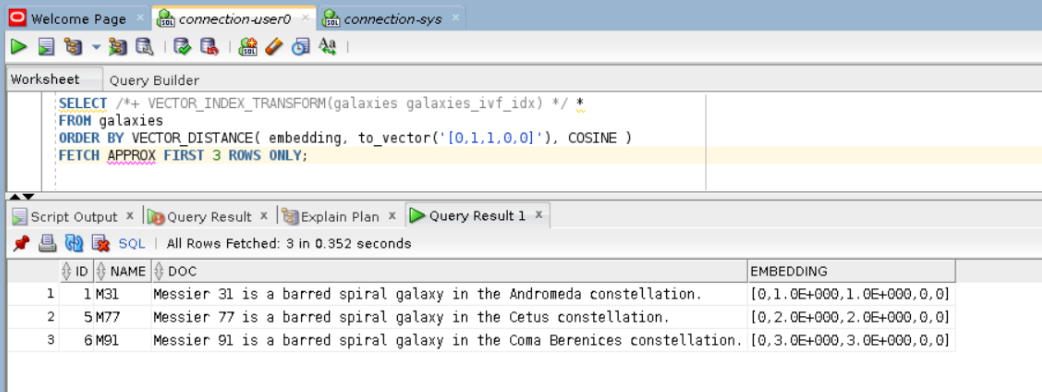
SELECT /\*+ VECTOR\_INDEX\_TRANSFORM(galaxies galaxies\_ivf\_idx) \*/ \*

FROM galaxies

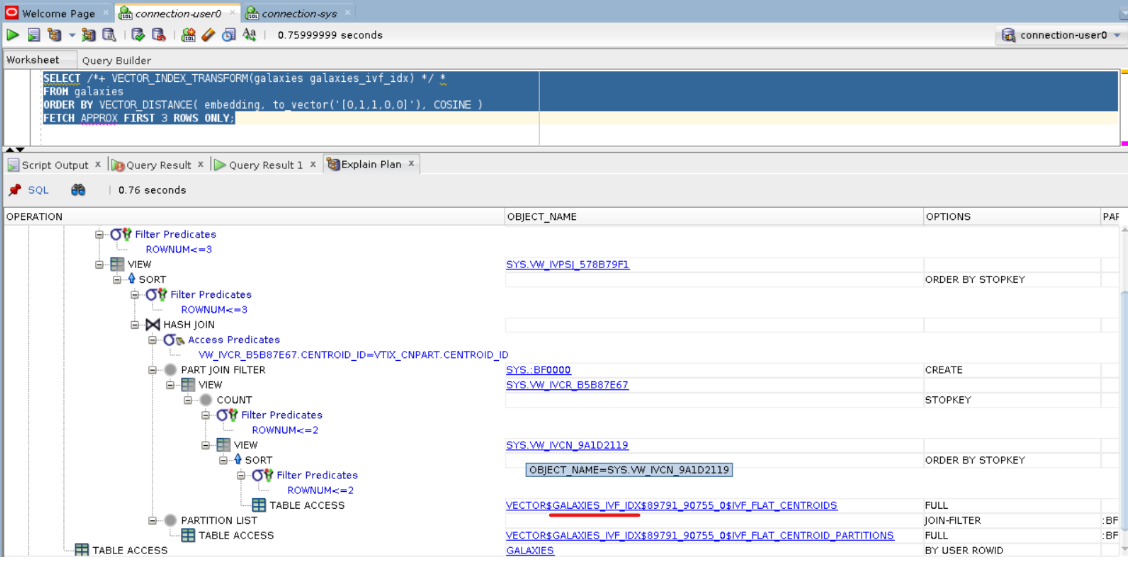
ORDER BY VECTOR\_DISTANCE( embedding, to\_vector('[0,1,1,0,0]'), COSINE )

FETCH APPROX FIRST 3 ROWS ONLY;

查询结果：



查看执行计划：



# 部署向量嵌入模型（仅讲师操作）

**此节内容仅讲师动手操作及讲解。**

以上我们介绍了向量的基本操作。在上面的例子中，我们的向量数据是手工造的，向量的维度也很小。那么，在现实环境中，向量数据是如何来的？答案是向量嵌入模型。

在本实验中，我们将使用开源的向量嵌入模型 text2vec-large-chinese

## 向量嵌入模型部署

考虑到硬件资源因素，没有足够的资源让每个人都部署一份模型，因此，本操作仅由讲师完成。讲师将向量嵌入模型部分为REST API 的方式，供大家调用；同时展示源代码并讲解。

源代码：<https://github.com/HysunHe/23ai_workshop_prep>

# 创建Python环境

conda create -n ws23ai python=3.12

# 进入新创建的Python环境

conda activate ws23ai

# 安装依赖

pip install -r requirements.txt

# 下载源码

git clone https://github.com/HysunHe/23ai\_workshop\_prep

# 启动模型

cd 23ai\_workshop\_prep

nohup python -u main.py > lab.out 2>&1 &

## 向量嵌入模型访问

向量嵌入模型部署完成后，就可以根据提供的REST API进行访问了。提供了如下两个API：

1. 文本向量化API（后续将用到）

curl -X 'POST' \

'http://<ip>:<port>/workshop/embedding' \

-H 'accept: application/json' \

-H 'Content-Type: application/json' \

-d '{

"text": "<需要向量化的文本>"

}'

1. 批量数据准备API（后续将用到）

curl -X 'POST' \

'http://<ip>:<port>/workshop/prepare-data' \

-H 'accept: application/json' \

-H 'Content-Type: application/json' \

-d '{

"db\_user": "<数据库用户名>",

"db\_password": "<数据库用户密码>",

"table\_name": "<表名>",

"dataset\_name": "<数据集名称>"

}'

## 库外向量化操作

### 数据加载

库外向量化指源数据由外部程序向量化之后，再插入或加载到数据库表中。在本例中，我们将使用 Python 程序将文本数据向量化之后，再调用Oracle客户包将数据插入到数据库中。这是常用的一种方法，操作方式也与平时的数据加载操作一致。

为了让接下来的实验更接近真实场景，我们将创建另一张表 lab\_vecstore：

CREATE TABLE lab\_vecstore (

id VARCHAR2(50) DEFAULT SYS\_GUID() PRIMARY KEY,

dataset\_name VARCHAR2(50) NOT NULL,

document CLOB,

cmetadata JSON,

embedding VECTOR(\*, FLOAT32)

);

这里我们没有指定向量的维度，但指定了数据类型格式是 FLOAT32，与向量模型的输出一致。下面我们将源数据文件（源数据集）加载进lab\_vecstore表。

源数据集：讲师展示源数据集。

接下来，请调用 批量数据准备API（API 会将上述源数据集进行向量化之后，再插入到数据库中）：

curl -X 'POST' \

'http://10.113.121.221:8099/workshop/prepare-data' \

-H 'accept: application/json' \

-H 'Content-Type: application/json' \

-d '{

"db\_user": "<userx>",

"db\_password": "<password>",

"table\_name": "lab\_vecstore",

"dataset\_name": "oracledb\_docs"

}'

注：如果没安装curl等api调用工具，也可以通过如下界面的方式执行：

1. 打开链接（注意IP地址为ODA数据库的IP） <http://x.x.x.x:8099/workshop/docs#/default/prepare_data_workshop_prepare_data_post>
2. 点击 "Try it out" 按钮
3. 在 "Request body" 输入框中，输入分配给你的 db\_user 和 db\_password 参数
4. 点击 "Execute" 按钮执行。

API 执行完成后，可以查看一下表中的数据：

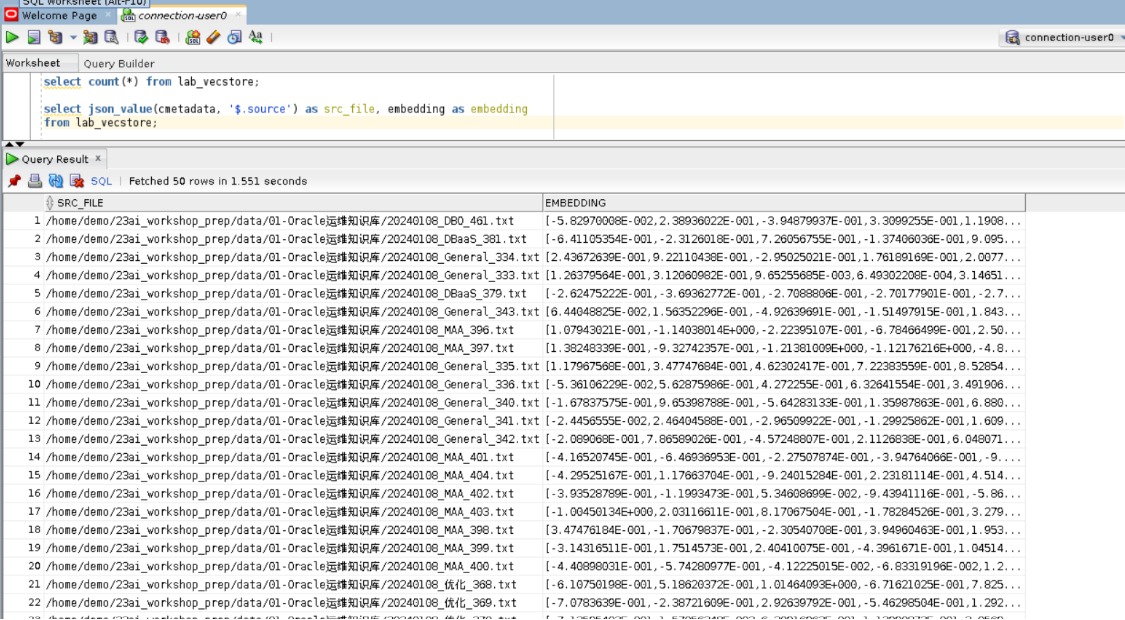
-- 本数据集总共有231条记录

select count(\*) from lab\_vecstore;

-- 查看数据

select t.cmetadata.source as src\_file, embedding as embedding

from lab\_vecstore t;



至此，源数据集已经向量化完成，并成功入库 (讲师展示并讲解外部向量化的源代码)

### 向量检索

本实验中，我们使用 “Oracle 23ai 新特性” 这个文本进行相似度检索。

第一步，先将要检索的文本在库外向量化。我们调用上述提供的API完成这一步。API将返回向量数据。

-- 第一步：向量化用户问题

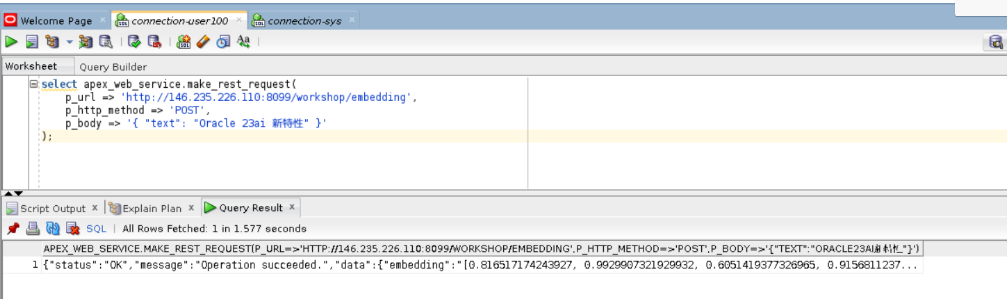
select apex\_web\_service.make\_rest\_request(

p\_url => 'http://146.235.226.110:8099/workshop/embedding',

p\_http\_method => 'POST',

p\_body => '{ "text": "Oracle 23ai 新特性" }'

);



第二步，执行 SQL 语句检索相似的数据，将上一步中返回的向量传入到VECTOR\_DISTANCE函数中：

set serveroutput on;

declare

l\_question varchar2(500) := 'Oracle 23ai 新特性';

l\_input CLOB;

l\_clob CLOB;

j apex\_json.t\_values;

l\_embedding 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：执行 SQL 语句检索相似的数据，将上一步中返回的向量传入到VECTOR\_DISTANCE函数中，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

DBMS\_OUTPUT.put\_line(chr(10) || '####################################');

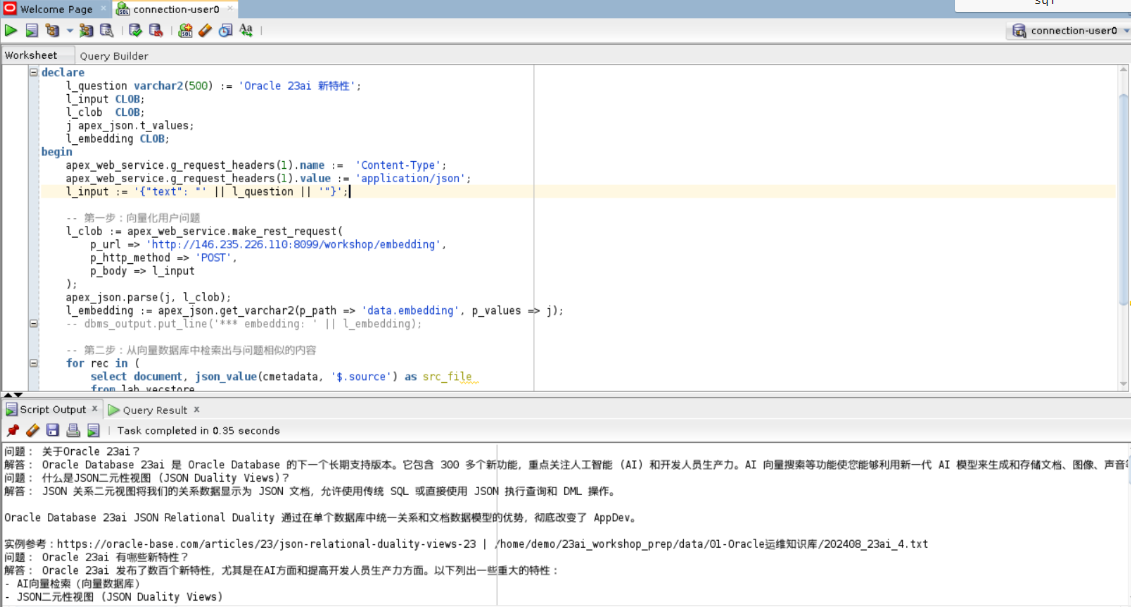
DBMS\_OUTPUT.put\_line(rec.document || ' | ' || rec.src\_file);

DBMS\_OUTPUT.put\_line('####################################' || chr(10));

end loop;

end;

/



# 库内向量化操作

Oracle 数据库提供了库内向量化的特性，其允许用户导入向量嵌入模型到数据库中，然后可以直接在SQL中对数据进行向量化操作，无需依赖外部的程序，这种方式很大程序的简化了向量数据的加载和检索，非常方便。

## 导入向量嵌入模型

需要加载进Oracle数据库的向量嵌入模型必须为标准的ONNX格式，且大小在1G之内。

-- 先将模型文件 bge-base-zh-v1.5.onnx 上传到/u01/hysun/models目录

-- 创建数据库目录指向模型文件所在目录

create or replace directory MODELS\_DIR as '/u01/hysun/models';

-- 导入模型

-- 先删除已经存在的模型（如果存在）：

EXEC DBMS\_VECTOR.DROP\_ONNX\_MODEL(model\_name => 'mydoc\_model', force => true);

-- 导入模型

BEGIN

DBMS\_VECTOR.LOAD\_ONNX\_MODEL(

directory => 'MODELS\_DIR',

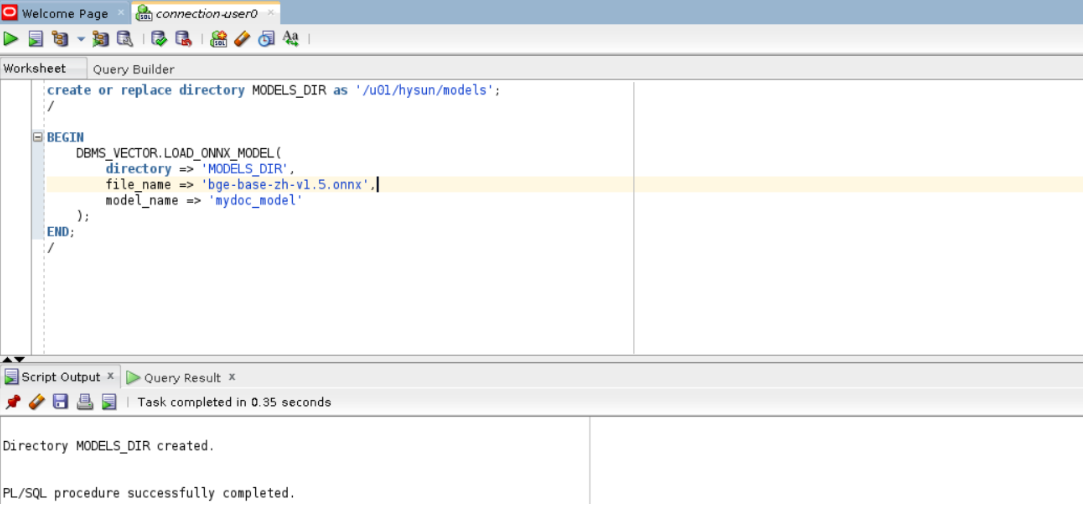
file\_name => 'bge-base-zh-v1.5.onnx',

model\_name => 'mydoc\_model'

);

END;

/



模型导入后，可以查看模型的属性：

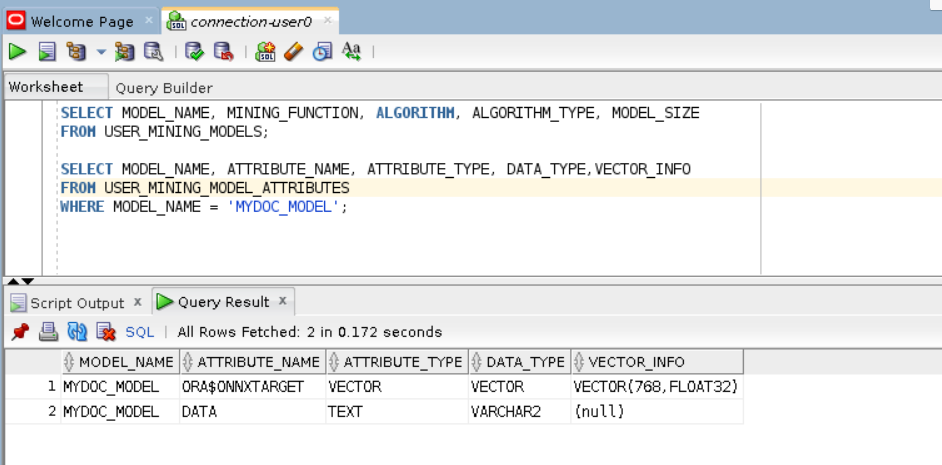
SELECT MODEL\_NAME, MINING\_FUNCTION, ALGORITHM, ALGORITHM\_TYPE, MODEL\_SIZE

FROM USER\_MINING\_MODELS;

SELECT MODEL\_NAME, ATTRIBUTE\_NAME, ATTRIBUTE\_TYPE, DATA\_TYPE,VECTOR\_INFO

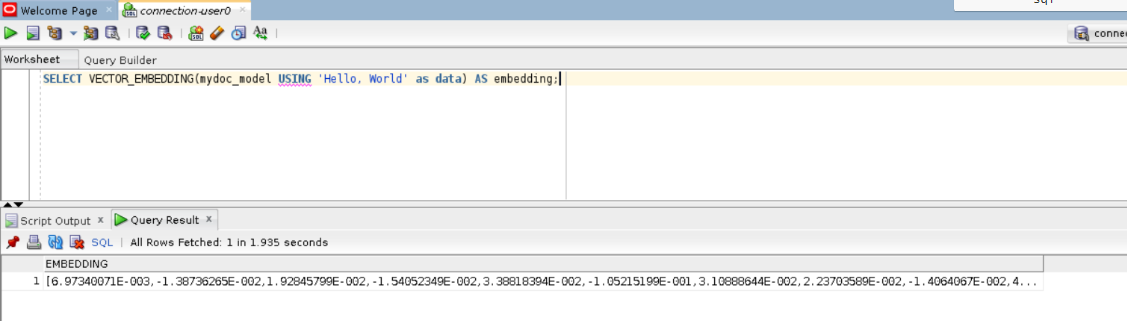
FROM USER\_MINING\_MODEL\_ATTRIBUTES

WHERE MODEL\_NAME = 'MYDOC\_MODEL';



做一个简单的向量化操作, 测试一下导入的模型是否如期工作：

SELECT VECTOR\_EMBEDDING(mydoc\_model USING 'Hello, World' as data) AS embedding;



## 库内向量化及检索

### 准备数据

为了排除干扰，我们新建同样的一张表 lab\_vecstore2：

CREATE TABLE lab\_vecstore2 (

id VARCHAR2(50) DEFAULT SYS\_GUID() PRIMARY KEY,

dataset\_name VARCHAR2(50) NOT NULL,

document CLOB,

cmetadata JSON,

embedding VECTOR(\*, FLOAT32)

);

然后从原来的表中拷贝几条数据（作为实验，建议不要拷贝太多数据，以避免造成资源紧张）：

insert into lab\_vecstore2(dataset\_name, document, cmetadata)

select dataset\_name, document, cmetadata

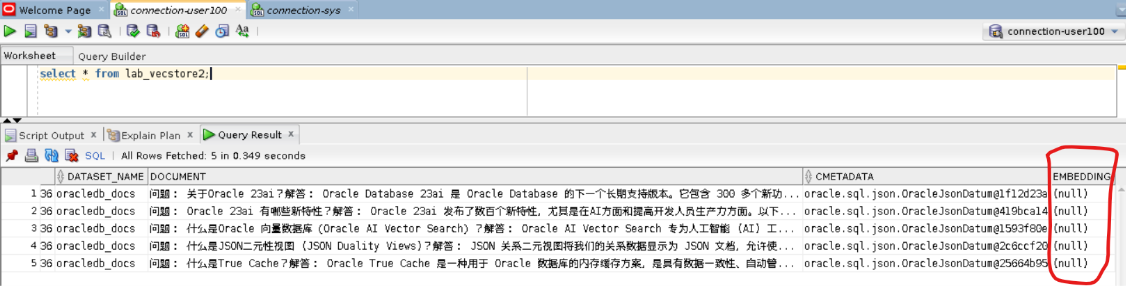
from lab\_vecstore --

where json\_value(cmetadata, '$.source') like '%202408\_23ai%';

commit;

select \* from lab\_vecstore2;

此时, lab\_vecstore2 表中的 Embedding 字段为空 (尚未做向量化操作).



### 库内向量化

-- 向量化之前，先查看一下表中的数据，此时 EMBEDDING 字段是空

select \* from lab\_vecstore2;

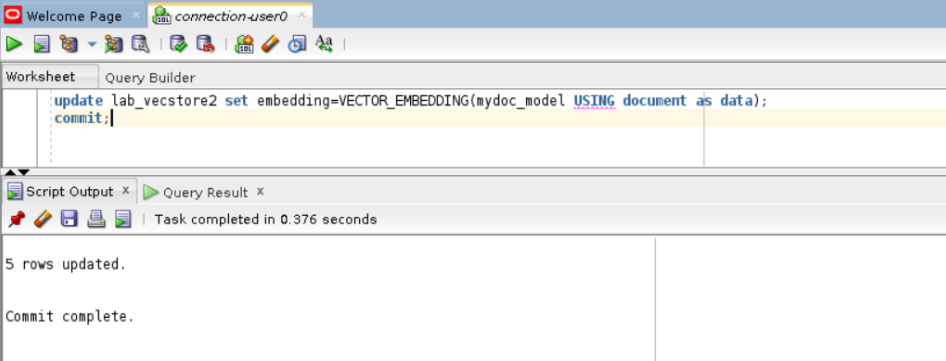
-- 执行SQL完成向量化

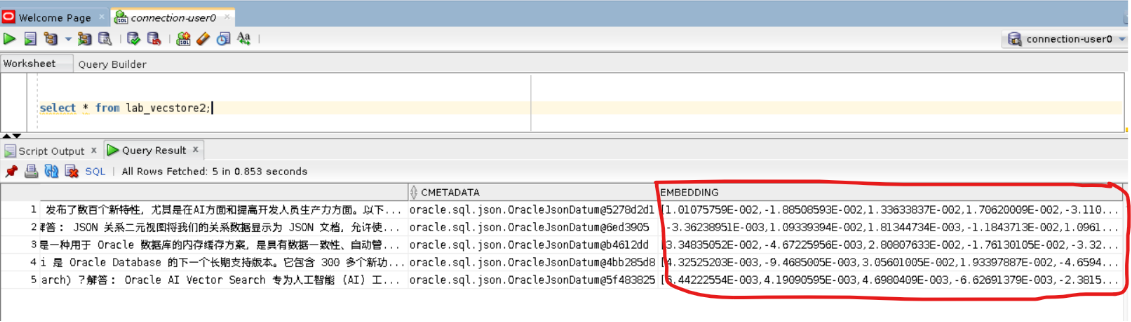
update lab\_vecstore2 set embedding=VECTOR\_EMBEDDING(mydoc\_model USING document as data);

commit;

-- 向量化之后，再次查看一下表中的数据，此时 EMBEDDING 字段是已经有值了。

select \* from lab\_vecstore2;





上述操作我们直接用标准的 SQL update 语句对表中的源数据进行了向量化。

### 相似度检索

由于我们已经在数据库中导入了向量嵌入模型，这里我们可以直接把文本传入 VECTOR\_EMBEDDING，进行相似度检索了。

select document,

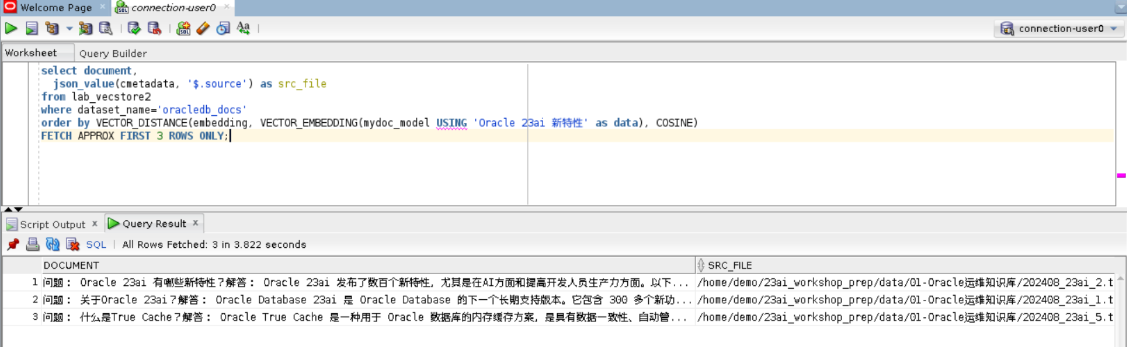
json\_value(cmetadata, '$.source') as src\_file

from lab\_vecstore2

where dataset\_name='oracledb\_docs'

order by VECTOR\_DISTANCE(embedding, VECTOR\_EMBEDDING(mydoc\_model USING 'Oracle 23ai 新特性' as data), COSINE)

FETCH APPROX FIRST 3 ROWS ONLY;



# 与第三方向量嵌入模型服务集成（仅演示）

Oracle数据库向量化操作能支持众多外部提供商提供的API，包括：

* OCIGenAI (Oracle OCI)
* OpenAI
* Cohere
* HuggingFace
* GoogleAI
* VertexAI
* 以及所有能兼容 OpenAI API 规范的其它服务接口。

本节以腾讯混元Embeddings模型为例，演示如何在Oracle中直接用简单的SQL调用腾讯混元Embedding模型，实现数据的向量化。对于其它的API提供商，做法上是一样的

## 开通第三方API服务

首先，开通腾讯混元大模型服务，并注册API Key：

<https://console.cloud.tencent.com/hunyuan/api-key> 。

## 创建访问凭证

利用刚才创建的API Key，在Oracle数据库中创建访问凭证。

declare

jo json\_object\_t;

begin

jo := json\_object\_t();

jo.put('access\_token', 'sk-IGiJxMkAxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');

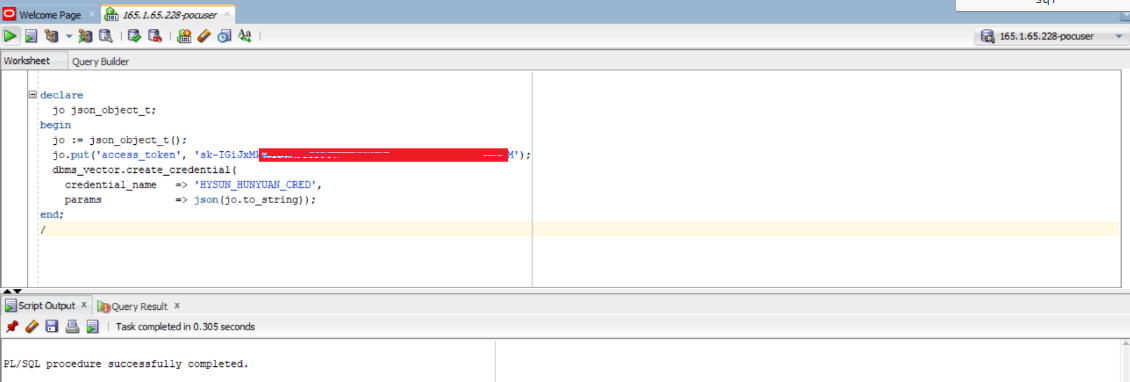
dbms\_vector.create\_credential(

credential\_name => 'HYSUN\_HUNYUAN\_CRED',

params => json(jo.to\_string));

end;

/



## 直接在SQL中调用外部Embedding服务

在SQL中直接调用dbms\_vector.utl\_to\_embedding或dbms\_vector.utl\_to\_embeddings将数据转化为向量.

首先，如果当前用户访问API的URL地址不被允许(ACL错误)，则先创建ACE:

BEGIN

DBMS\_NETWORK\_ACL\_ADMIN.APPEND\_HOST\_ACE(

host => 'api.hunyuan.cloud.tencent.com',

lower\_port => 443,

upper\_port => 443,

ace => xs$ace\_type(privilege\_list => xs$name\_list('http'),

principal\_name => '<数据库用户名>',

principal\_type => xs\_acl.ptype\_db)

);

END;

/

直接在SQL中调用混元 API Embedding 服务, 如:

SELECT

dbms\_vector.utl\_to\_embedding(

'Oracle向量数据库动手实验培训',

json('{

"provider": "OpenAI",

"credential\_name": "HYSUN\_HUNYUAN\_CRED",

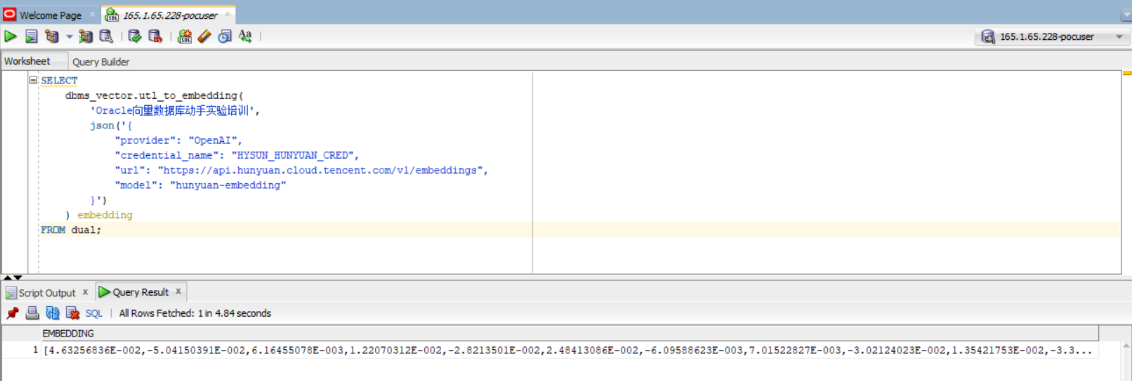
"url": "https://api.hunyuan.cloud.tencent.com/v1/embeddings",

"model": "hunyuan-embedding"

}')

) embedding

FROM dual;



# 总结

至此，我们已经完成了Oracle向量数据库的动手实验第一部分。

本节内容中，我们实现了利用向量检索的精确检索和近似检索两种方式。现实中，在相对较大的数据集中，精确检索往往只有在融合数据库中才能发挥出真正的优势。比如，在我们的实验中，我们使用标量字段dataset\_name='oracledb\_docs'将需要进行向量检索的数据集大幅度缩小了，有效弥补了精确检索的性能问题。

同时，我们还实现了Oracle库外向量化和库内向量化两种方式。库内向量化因其简单便捷的特点，有可能成为未来向量化的一个重要方向。然而，就目前而言，局限于数据库硬件资源现状，往往库外向量化方式使用更多。

最后，我们还介绍了如何通过与第三方Embedding API服务集成，在SQL中调用第三方服务完成向量化的过程。

下一节我们将进行第二部分的实验：结合Oracle向量检索的RAG应用。