

# Oracle向量数据库动手实验

---

- [Oracle向量数据库动手实验](#)
  - [介绍](#)
  - [前提条件](#)
  - [环境准备](#)
  - [实验1: Oracle向量基本操作](#)
  - [实验2: 向量检索](#)
    - [向量精确检索](#)
    - [向量近似检索](#)
  - [实验3: 向量嵌入模型部署 \(仅讲师操作\)](#)
  - [实验4: 库外向量化操作](#)
  - [实验5: 库内向量化操作](#)
    - [导入向量嵌入模型](#)
    - [库内向量化及检索](#)
  - [实验6: 与第三方向量嵌入模型服务集成 \(演示\)](#)
    - [开通第三方API服务](#)
    - [创建访问凭证](#)
    - [直接在SQL中Embedding](#)
  - [总结](#)

## 介绍

---

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](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)

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）。

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

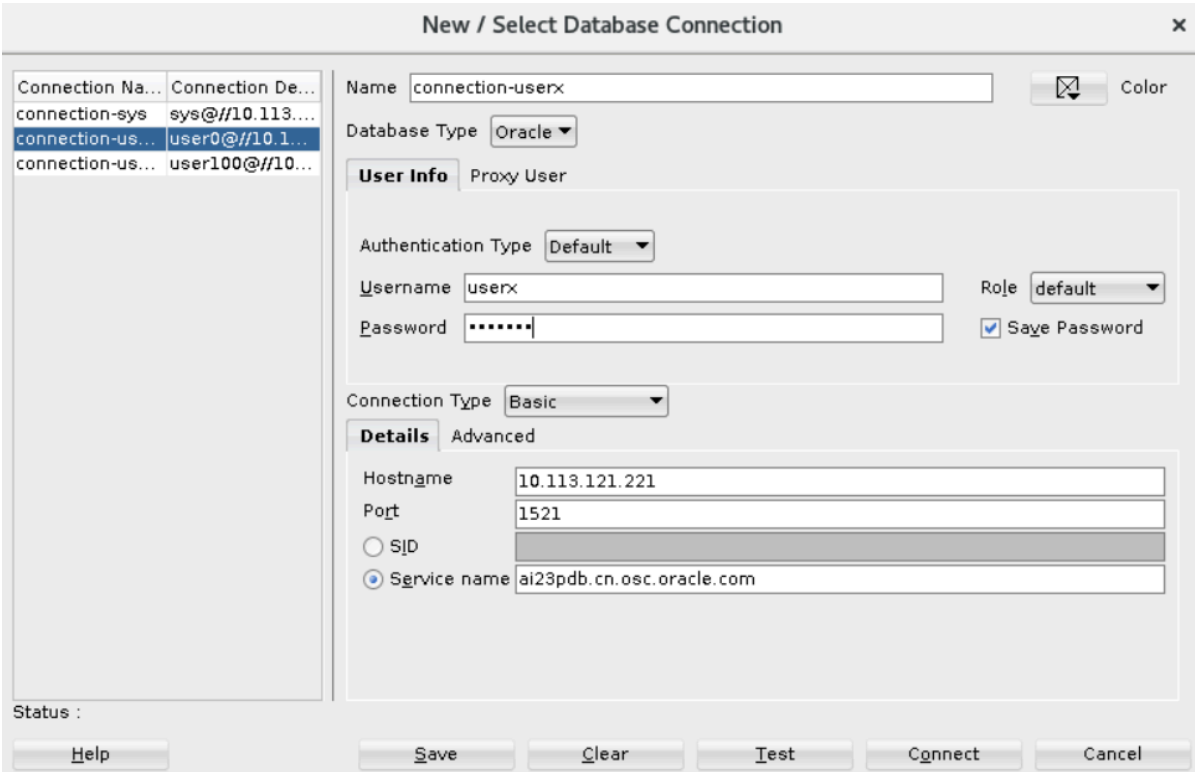
# 前提条件

- 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，需要参与者已经参加过Oracle向量数据库的介绍或有所了解。
- 2. 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。
- 3. 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

本实验需要的环境，请参考《动手试验环境说明.pdf》中“参与者”已分配的账号和密码。如 user1/user1。

例如，连接SQL Developer：



# 实验1：Oracle向量基本操作

让我们从基本的向量操作开始：

## 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。

## 向量类型字段及样列表

Oracle 23ai 引入了向量数据类型：VECTOR (dimensions, 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，我们就可以根据数据进行检索了。

## 实验2：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，可以很大程度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：

The screenshot shows a database interface with a 'Query Builder' tab. The SQL query is: `SELECT * FROM galaxies ORDER BY VECTOR_DISTANCE( embedding, to_vector('0.1,1.0,0'), COSINE ) FETCH FIRST 3 ROWS ONLY;`. Below the query, the 'Query Result' tab is active, showing a table with 3 rows and 4 columns: ID, NAME, DOC, and EMBEDDING. The results are as follows:

ID	NAME	DOC	EMBEDDING
1	M31	Messier 31 is a barred spiral galaxy in the Andromeda constellation.	[0.1.0E+000,1.0E+000,0,0]
2	M91	Messier 91 is a barred spiral galaxy in the Coma Berenices constellation.	[0.3.0E+000,3.0E+000,0,0]
3	M77	Messier 77 is a barred spiral galaxy in the Cetus constellation.	[0.2.0E+000,2.0E+000,0,0]

查看执行计划：

The screenshot shows the same database interface with the 'Explain Plan' tab active. The execution plan is as follows:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	4
COUNT		STOPKEY		
Filter Predicates ROWNUM<=3				
VIEW	SYS.null		9	4
SORT		ORDER BY STOPKEY	9	4
Filter Predicates ROWNUM<=3				
TABLE ACCESS	GALAXIES	FULL	9	3

## 向量近似检索

向量近似检索（Approximate Search）

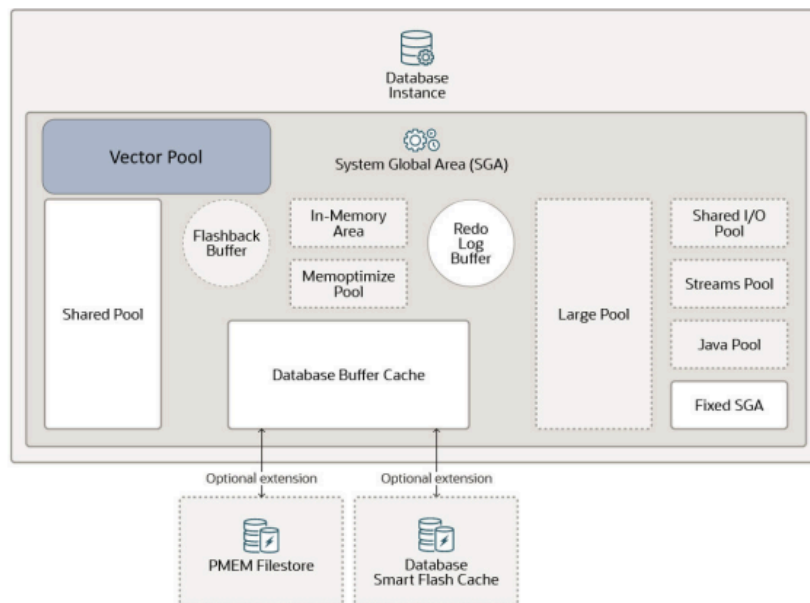
精确检索获得了最高的准确率，但需要遍历所有向量数据集，因此，在向量数据集比较大时，性能很可能会成为问题。向量检索中，准确率和性能之间，往往需要寻找一个平衡。在大数据集上，为了提高性能，利用索引进行向量近似检索是常用的方式。

常见的向量索引有HNSW和IVF两种。

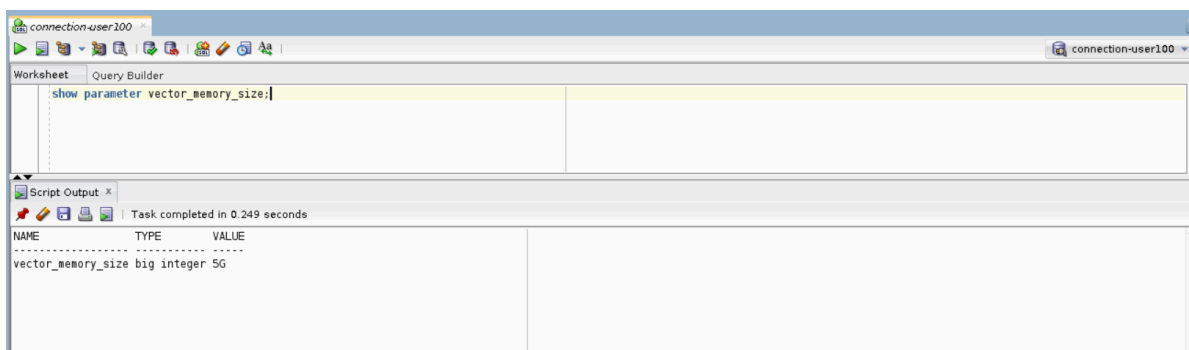
## 向量内存池

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;
```

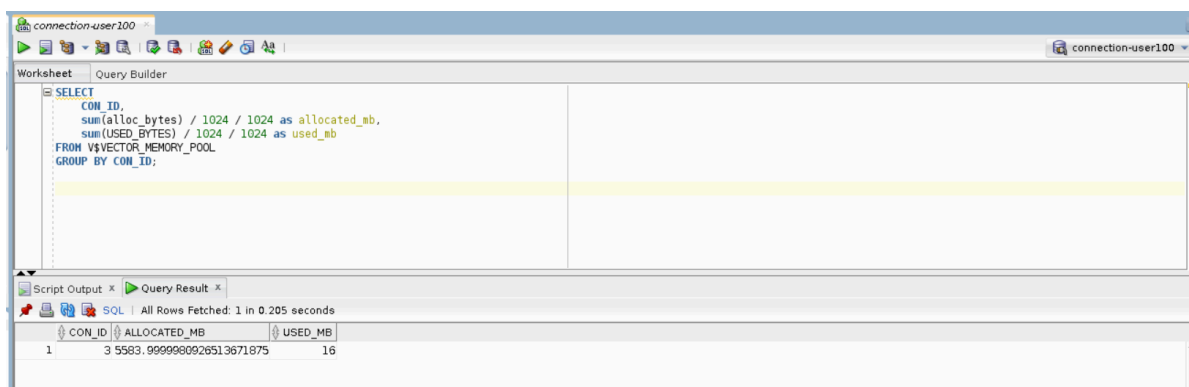


向量内存池大小估算公式：  $\text{size of vector pool} = 1.3 * \text{number of vectors} * \text{number of dimensions} * \text{size of vector dimension type}$

## 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IV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：

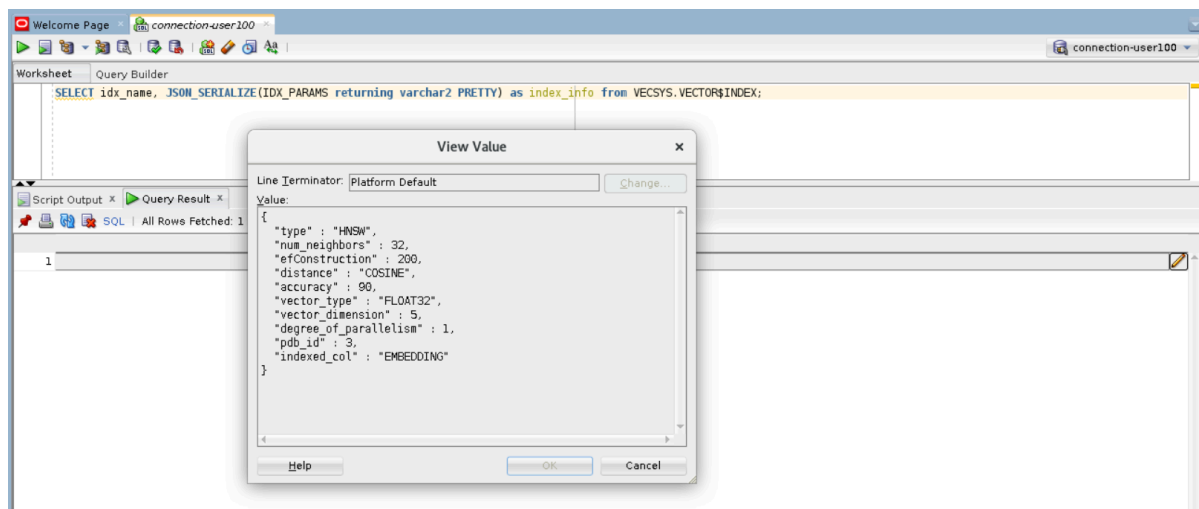
<https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/oracle-ai-vector-search-usage-guide.pdf> (184页)

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

## 查看索引信息

字典表 VEC\$SYS.VECTOR\$INDEX 记录了创建的索引的详细信息：

```
SELECT idx_name, JSON_SERIALIZE(IDX_PARAMS returning varchar2 PRETTY) as
index_info from VEC$SYS.VECTOR$INDEX;
```



## 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:

Worksheet | Query Builder

```
SELECT *
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('0.1,1.0,0') ), COSINE )
FETCH APPROX FIRST 3 ROWS ONLY;
```

Explain Plan | Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 3 in 0.578 seconds

ID	NAME	DOC	EMBEDDING
1	6 M91	Messier 91 is a barred spiral galaxy in the Coma Berenices constellation.	[0,3.0E+000,3.0E+000,0,0]
2	5 M77	Messier 77 is a barred spiral galaxy in the Cetus constellation.	[0,2.0E+000,2.0E+000,0,0]
3	1 M31	Messier 31 is a barred spiral galaxy in the Andromeda constellation.	[0,1.0E+000,1.0E+000,0,0]

查看执行计划：

Worksheet | Query Builder

```
SELECT *
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('0.1,1.0,0') ), COSINE )
FETCH APPROX FIRST 3 ROWS ONLY;
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.859 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	2
COUNT		STOPKEY		
Filter Predicates		ROWNUM<=3		
VIEW	SYS.null		3	2
SORT		ORDER BY STOPKEY	3	2
Filter Predicates		ROWNUM<=3		
TABLE ACCESS	GALAXIES	BY INDEX ROWID	3	1
VECTOR INDEX	GALAXIES_HNSW_IDX	HNSW SCAN	3	1

Other XML  
{info}  
info type="has\_user\_tab"  
yes

## 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IV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)

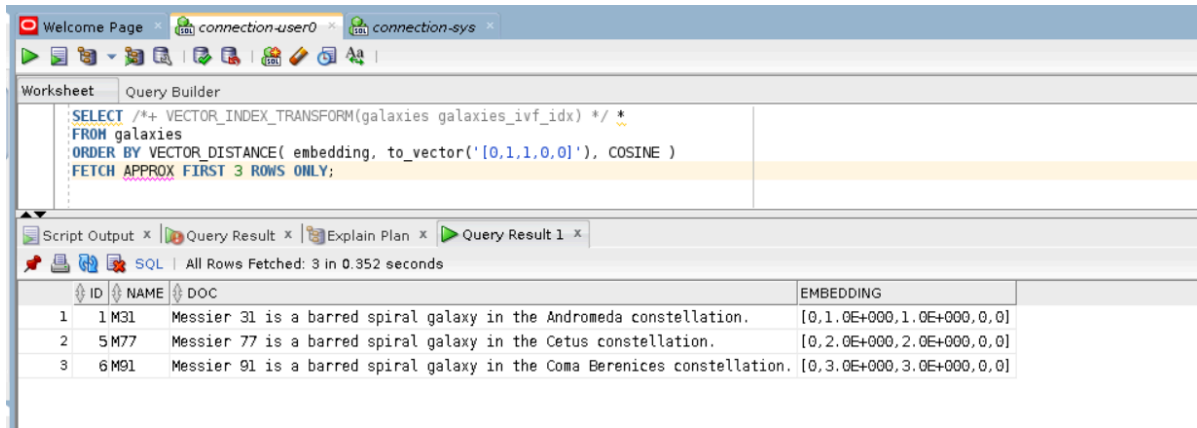
## IVF 近似检索

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）



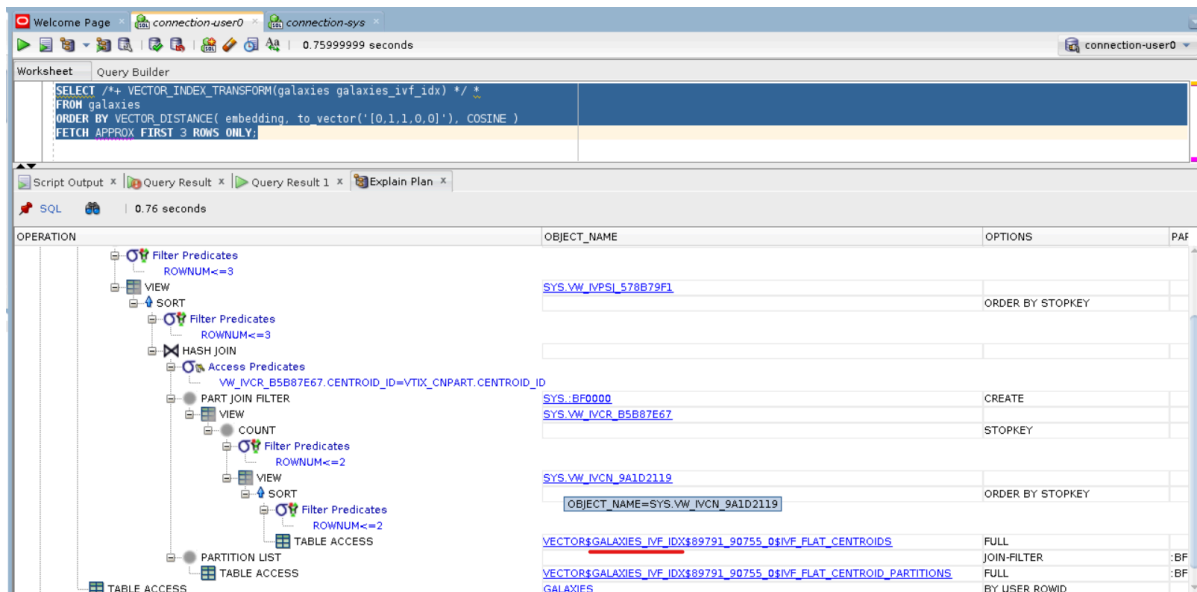
```
SELECT /*+ VECTOR_INDEX_TRANSFORM(galaxies galaxies_ivf_idx) */ *
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH APPROX FIRST 3 ROWS ONLY;
```

查询结果：



ID	NAME	DOC	EMBEDDING
1	M31	Messier 31 is a barred spiral galaxy in the Andromeda constellation.	[0,1.0E+000,1.0E+000,0,0]
2	M77	Messier 77 is a barred spiral galaxy in the Cetus constellation.	[0,2.0E+000,2.0E+000,0,0]
3	M91	Messier 91 is a barred spiral galaxy in the Coma Berenices constellation.	[0,3.0E+000,3.0E+000,0,0]

查看执行计划：



## 实验3：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](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>"
  }'
```

### 2. 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>"
  }'
```

## 实验4：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 [http://10.113.121.221:8099/workshop/docs#/default/prepare\\_data\\_workshop\\_prepare\\_data\\_post](http://10.113.121.221:8099/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 json_value(cmetadata, '$.source') as src_file, embedding as embedding
from lab_vecstore;
```

SRC_FILE	EMBEDDING
1 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_DBO_461.txt	[-5.82970008E-002, 2.39936022E-001, -3.94879937E-001, 3.3099255E-001, 1.1908...
2 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_DBaas_381.txt	[-6.41105354E-001, -2.3126018E-001, 7.26056755E-001, -1.37406036E-001, 9.095...
3 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_334.txt	[2.43672639E-001, 9.22110438E-001, -2.95025021E-001, 1.76189169E-001, 2.0077...
4 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_333.txt	[1.26379564E-001, 3.12060982E-001, 9.65255685E-003, 6.49302208E-004, 3.14651...
5 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_DBaas_379.txt	[-2.62475222E-001, -3.69362772E-001, -2.7088806E-001, -2.70177901E-001, -2.7...
6 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_343.txt	[6.44048625E-002, 1.56352296E-001, -4.92639691E-001, -1.51497915E-001, 1.843...
7 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_396.txt	[1.07943021E-001, -1.14038014E+000, -2.22395107E-001, -6.78466499E-001, 2.50...
8 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_397.txt	[1.38248339E-001, -9.32742357E-001, -1.21381009E+000, -1.12176216E+000, -4.8...
9 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_335.txt	[1.17967568E-001, 3.47747684E-001, 4.62302417E-001, 7.22383559E-001, 8.52854...
10 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_336.txt	[-5.36106229E-002, 5.62875986E-001, 4.272255E-001, 6.32641554E-001, 3.491906...
11 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_340.txt	[-1.67837575E-001, 9.65398788E-001, -5.64283133E-001, 1.35987863E-001, 6.880...
12 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_341.txt	[-2.4456555E-002, 2.46404588E-001, -2.96509922E-001, -1.29925862E-001, 1.609...
13 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_General_342.txt	[-2.089068E-001, 7.86589026E-001, -4.57248807E-001, 2.1126838E-001, 6.048071...
14 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_401.txt	[-4.16520745E-001, -6.46936953E-001, -2.27507874E-001, -3.94764066E-001, -9...
15 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_404.txt	[-4.29525167E-001, 1.17663704E-001, -9.24015284E-001, 2.23181114E-001, 4.514...
16 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_402.txt	[-3.93528789E-001, -1.1993473E-001, 5.34608699E-002, 9.43941116E-001, -5.06...
17 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_403.txt	[-1.00450134E+000, 2.03116611E-001, 8.17067504E-001, -1.78284526E-001, 3.279...
18 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_398.txt	[3.47476184E-001, -1.70679837E-001, -2.30540708E-001, 3.94960463E-001, 1.953...
19 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_399.txt	[-3.14316511E-001, 1.7514573E-001, 2.40410075E-001, -4.3961671E-001, 1.04514...
20 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_MAA_400.txt	[-4.40898031E-001, -5.74280977E-001, -4.12225015E-002, -6.83319196E-002, 1.2...
21 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_优化_368.txt	[-6.10750198E-001, 5.18620372E-001, 1.01464093E+000, -6.71621025E-001, 7.825...
22 /home/demo/23ai_workshop_prep/data/01-Oracle运维知识库/20240108_优化_369.txt	[-7.0783639E-001, -2.38721609E-001, 2.92639792E-001, -5.46296504E-001, 1.292...

至此，源数据集已经向量化完成，并且成功入库了。（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" }'
);
```

APEX_WEB_SERVICE_MAKE_REST_REQUEST.P_URL=>'HTTP://146.235.226.110:8099/WORKSHOP/EMBEDDING',P_HTTP_METHOD=>'POST',P_BODY=>'{\"TEXT\":\"ORACLE23AI新特性\"}'
1 {\"status\":\"OK\", \"message\":\"Operation succeeded.\", \"data\":{\"embedding\":[0.816517174243927, 0.9929907321929932, 0.6051419377326965, 0.9156811237...

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';
```

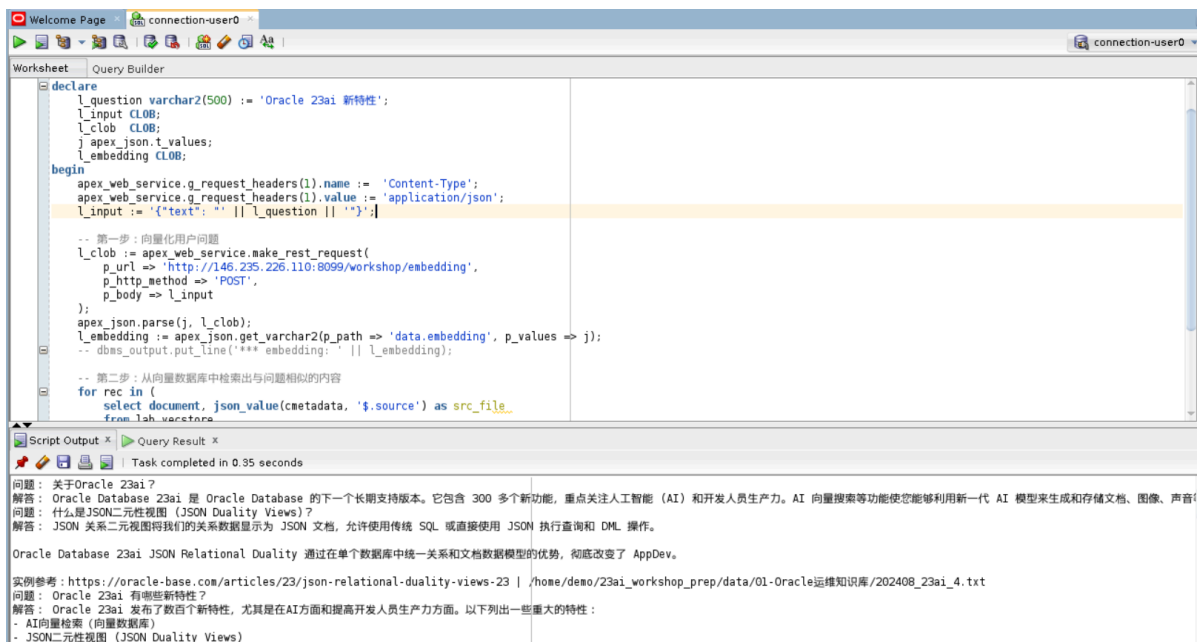
```

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;
/

```



## 实验5：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

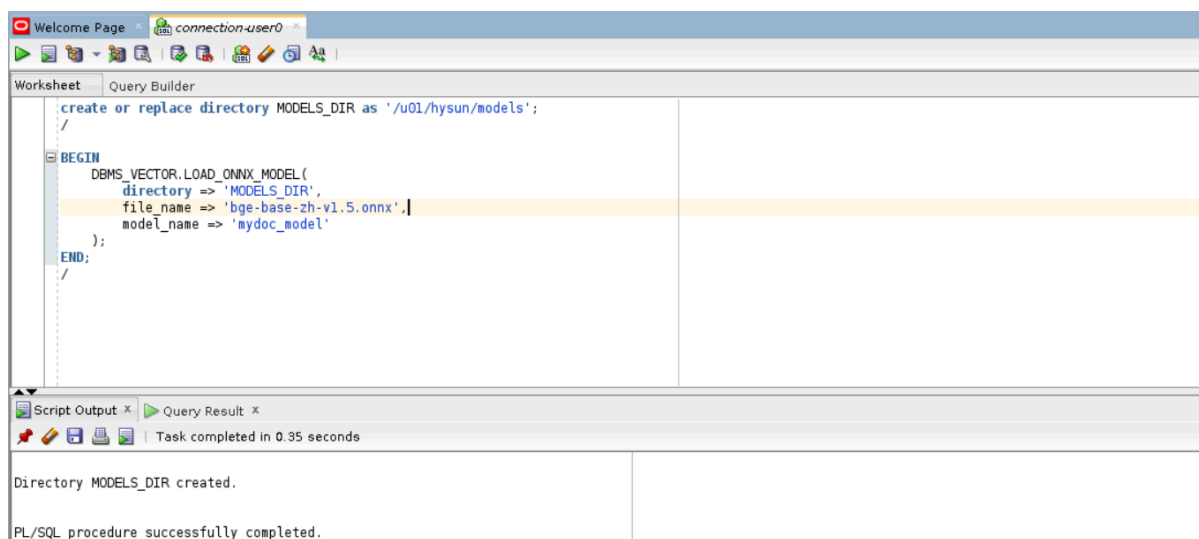
考虑到硬件资源因素，没有足够的资源让每个人都加载一份模型，因此，本操作仅由讲师完成。讲师展示加载操作，并提供讲解。

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);

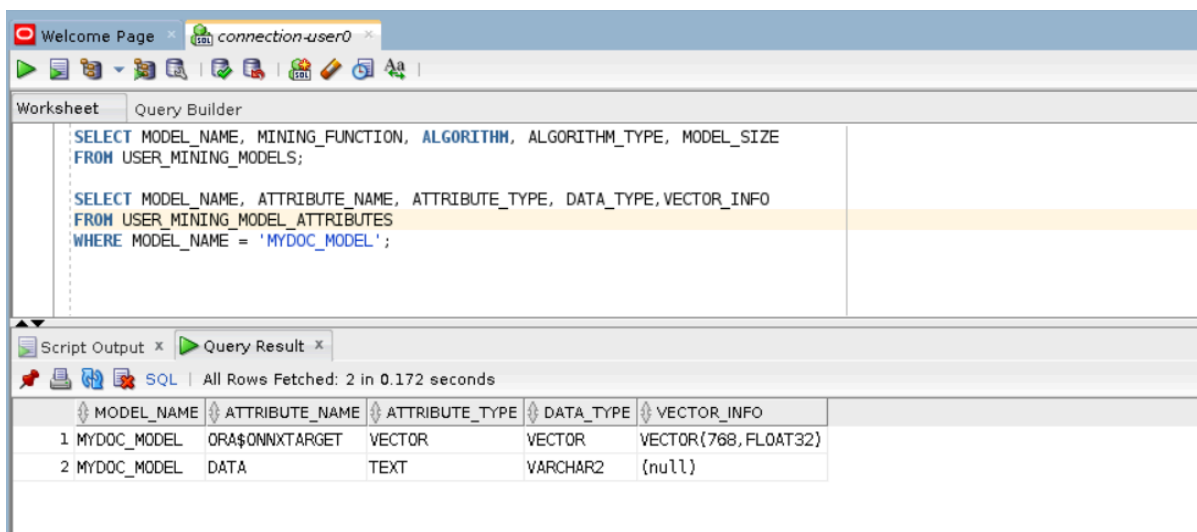
-- 导入模型
BEGIN
    DBMS_VECTOR.LOAD_ONNX_MODEL(
        directory => 'MODELS_DIR',
        file_name => 'bge-base-zh-v1.5.onnx',
        model_name => 'mydoc_model'
    );
END;
/
```



模型导入后，可以查看模型的属性：

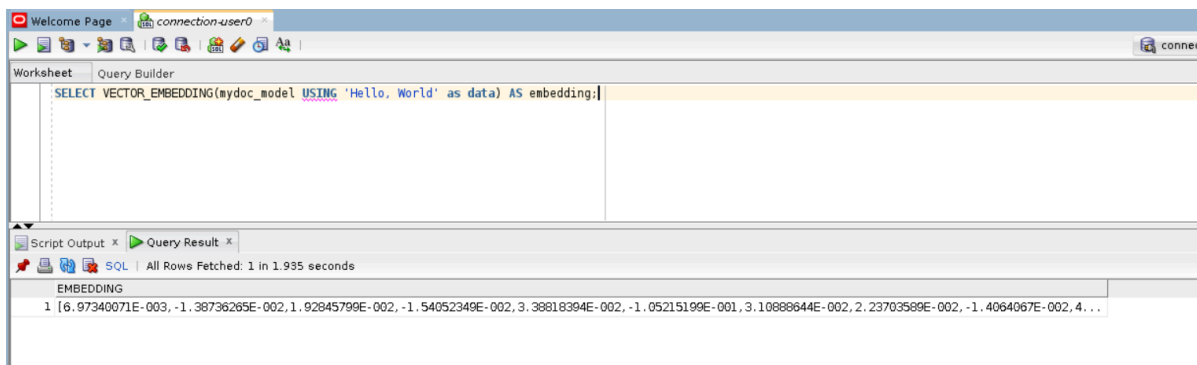
```
SELECT MODEL_NAME, MINING_FUNCTION, ALGORITHM, ALGORITHM_TYPE, MODEL_SIZE
FROM USER_MINING_MODELS;

SELECT MODEL_NAME, ATTRIBUTE_NAME, ATTRIBUTE_TYPE, DATA_TYPE, VECTOR_INFO
FROM USER_MINING_MODEL_ATTRIBUTES
WHERE MODEL_NAME = 'MYDOC_MODEL';
```



可以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;
```

DATASET_NAME	DOCUMENT	CMETADATA	EMBEDDING
1 36 oracledb_docs	问题：关于Oracle 23ai 是 Oracle Database 的下一个长期支持版本。它包含 300 多个新功...	oracle.sql.json.OracleJsonDatum@1f12d23a	(null)
2 36 oracledb_docs	问题：Oracle 23ai 有哪些新特性？解答：Oracle 23ai 发布了数百个新特性，尤其是在AI方面和提高开发人员生产力方面。以下...	oracle.sql.json.OracleJsonDatum@419bca14	(null)
3 36 oracledb_docs	问题：什么是Oracle 向量数据库 (Oracle AI Vector Search)？解答：Oracle AI Vector Search 专为人工智能 (AI) 工...	oracle.sql.json.OracleJsonDatum@1593f80e	(null)
4 36 oracledb_docs	问题：什么是JSON二元视图 (JSON Duality Views)？解答：JSON 关系二元视图将我们的关系数据显为 JSON 文档，允许使...	oracle.sql.json.OracleJsonDatum@2c6ccf20	(null)
5 36 oracledb_docs	问题：什么是True Cache？解答：Oracle True Cache 是一种用于 Oracle 数据库的内存缓存方案，是具有数据一致性、自动替...	oracle.sql.json.OracleJsonDatum@25664b95	(null)

## 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;
```

```
update lab_vecstore2 set embedding=VECTOR_EMBEDDING(mydoc_model USING document as data);
commit;
```

5 rows updated.

Commit complete.

The screenshot shows the SQL Developer interface. At the top, there's a 'Welcome Page' and a 'connection-user0' tab. Below that, a 'Worksheet' and 'Query Builder' tab are visible. The main area displays a SQL query: 

```
select * from lab_vecstore2;
```

 Below the query, the 'Script Output' and 'Query Result' tabs are shown. The 'Query Result' tab is active, displaying a table with columns: DATASET\_NAME, DOCUMENT, CMETADATA, and EMBEDDING. The 'EMBEDDING' column contains numerical values and is highlighted with a red box. The table data is as follows:

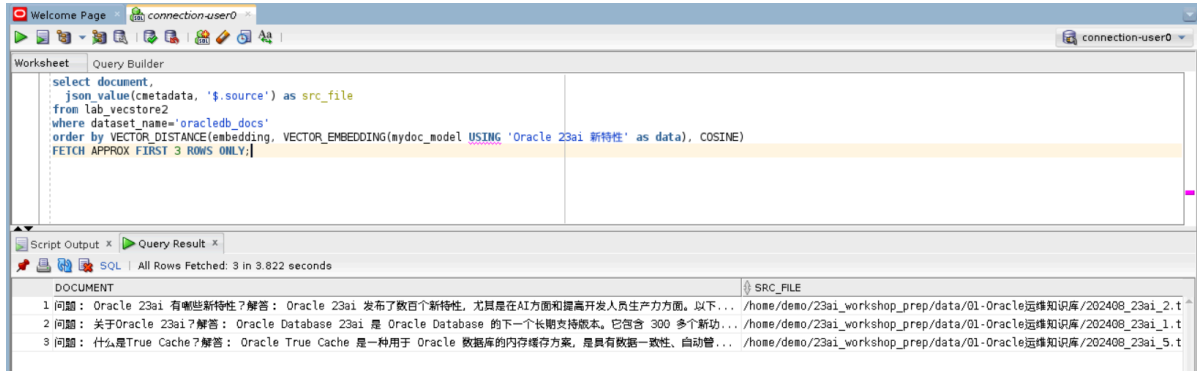
上述操作我们直接用标准的 SQL update 语句对表中的源数据进行了向量化。

## 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;
```



## 实验6：与第三方向量嵌入模型服务集成（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：

- OCI GenAI (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>。

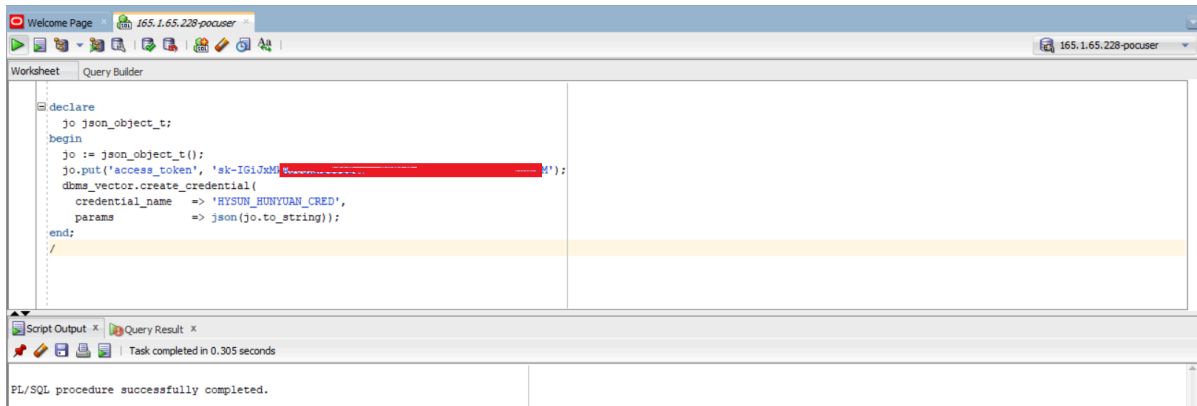
### 创建访问凭证

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Embedding

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：

```

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。