Oracle向量数据库动手实验

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）、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本的概念上的了解。
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. 有基本的Python知识，能够看懂简单的Python示例代码。

环境准备

1. Oracle 23ai 数据库
2. SQLcl 24.2+
3. API 调用工具，比如 curl。

快速一览

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。

Oracle向量数据库基本操作

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
);
```

插入样例数据

```
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;
```

向量精确检索（Exact Search）

向量精确检索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：

```
SQL> SELECT *
  2  FROM galaxies
  3  ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
  4* FETCH APPROX FIRST 3 ROWS ONLY;

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

SQL>
```

查看执行计划：

```
EXPLAIN PLAN FOR
SELECT *
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH FIRST 3 ROWS ONLY;

select plan_table_output from table(dbms_xplan.display('plan_table',null,'all'));
```

```
SQL> EXPLAIN PLAN FOR
  2  SELECT *
  3  FROM galaxies
  4  ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
  5* FETCH APPROX FIRST 3 ROWS ONLY;

Explained.

SQL> select plan_table_output from table(dbms_xplan.display('plan_table',null,'all'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 4057708865

-------------------------------------------------------------------------------
| Id  | Operation               | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |     3 | 13182 |     4  (25)| 00:00:01 |
|*  1 |  COUNT STOPKEY          |          |       |       |            |          |
|   2 |   VIEW                  |          |     9 | 39546 |     4  (25)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY|          |     9 | 39600 |     4  (25)| 00:00:01 |
|   4 |     TABLE ACCESS FULL   | GALAXIES |     9 | 39600 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
```

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。

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-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：

```
SQL> SELECT *
  2  FROM galaxies
  3  ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
  4* FETCH APPROX FIRST 3 ROWS ONLY;

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

SQL>
```

查看执行计划：

```
EXPLAIN PLAN FOR
SELECT *
FROM galaxies
ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
FETCH APPROX FIRST 3 ROWS ONLY;

select plan_table_output from table(dbms_xplan.display('plan_table',null,'all'));
```

```
SQL> EXPLAIN PLAN FOR
  2  SELECT *
  3  FROM galaxies
  4  ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
  5* FETCH APPROX FIRST 3 ROWS ONLY;

Explained.

SQL> select plan_table_output from table(dbms_xplan.display('plan_table',null,'all'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
Plan hash value: 1531424292

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                   |     3 | 13182 |     2  (50)| 00:00:01 |
|*  1 |  COUNT STOPKEY                |                   |       |       |            |          |
|   2 |   VIEW                        |                   |     9 | 39546 |     2  (50)| 00:00:01 |
|*  3 |    SORT ORDER BY STOPKEY      |                   |     9 | 39546 |     2  (50)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| GALAXIES          |     9 | 39546 |     1   (0)| 00:00:01 |
|   5 |      VECTOR INDEX HNSW SCAN   | GALAXIES_HNSW_IDX |     9 | 39546 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
```

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：

```
SQL> SELECT *
  2  FROM galaxies
  3  ORDER BY VECTOR_DISTANCE( embedding, to_vector('[0,1,1,0,0]'), COSINE )
  4* FETCH APPROX FIRST 3 ROWS ONLY;

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

SQL>
```

查看执行计划：

```
EXPLAIN PLAN FOR
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;

select plan_table_output from table(dbms_xplan.display('plan_table',null,'all'));
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                                                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                                                     |     1 |  4394 |   951   (1)| 00:00:01 |       |       |
|   1 |  VIEW                          |                                                     |     1 |  4394 |   951   (1)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                 |                                                     |     1 |  4422 |   951   (1)| 00:00:01 |       |       |
|   3 |    VIEW                        | VW_IVPSR_11E7D7DE                                   |     3 |    48 |   948   (1)| 00:00:01 |       |       |
|*  4 |     COUNT STOPKEY              |                                                     |       |       |            |          |       |       |
|   5 |      VIEW                      | VW_IVPSJ_578B79F1                                   |     7 |   126 |   948   (1)| 00:00:01 |       |       |
|*  6 |       SORT ORDER BY STOPKEY    |                                                     |     7 |   154 |   948   (1)| 00:00:01 |       |       |
|*  7 |        HASH JOIN               |                                                     |     7 |   154 |   947   (1)| 00:00:01 |       |       |
|   8 |         PART JOIN FILTER CREATE| :BF0000                                             |     4 |    12 |     4  (25)| 00:00:01 |       |       |
|   9 |          VIEW                  | VW_IVCR_B5B87E67                                    |     4 |    12 |     4  (25)| 00:00:01 |       |       |
|* 10 |           COUNT STOPKEY        |                                                     |       |       |            |          |       |       |
|  11 |            VIEW                | VW_IVCN_9A1D2119                                    |     5 |    65 |     4  (25)| 00:00:01 |       |       |
|* 12 |             SORT ORDER BY STOPKEY |                                                  |     5 |    45 |     4  (25)| 00:00:01 |       |       |
|  13 |              TABLE ACCESS FULL | VECTOR$GALAXIES_IVF_IDX$130178_130199_0$IVF_FLAT_CENTROIDS |     5 |    45 |     3   (0)| 00:00:01 |       |       |
|  14 |         PARTITION LIST JOIN-FILTER|                                                  |     9 |   171 |   236   (1)| 00:00:01 |:BF0000|:BF0000|
|  15 |          TABLE ACCESS FULL     | VECTOR$GALAXIES_IVF_IDX$130178_130199_0$IVF_FLAT_CENTROID_PARTITIONS |     9 |   171 |   236   (1)| 00:00:01 |:BF0000|:BF0000|
|  16 |    TABLE ACCESS BY USER ROWID  | GALAXIES                                            |     1 |  4406 |     1   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------------------------------
```

向量嵌入模型

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（仅讲师操作）

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。

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

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://146.235.226.110:8099/workshop/prepare-data' \
    -H 'accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
        "db_user": "<你的数据库用户名>",
        "db_password": "<密码>",
        "table_name": "lab_vecstore",
        "dataset_name": "oracledb_docs"
    }'
```

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;

-- 查看一条数据
select json_value(cmetadata, '$.source') as src_file, embedding
from lab_vecstore
where rownum < 2;
```

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。

```
curl -X 'POST' \
    'http://146.235.226.110:8099/workshop/embedding' \
    -H 'accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
        "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：

```
select document, json_value(cmetadata, '$.source') as src_file
from lab_vecstore
where dataset_name='oracledb_docs'
order by VECTOR_DISTANCE(embedding, to_vector('[0.8165184855461121,
0.9929913878440857, 0.60514235496521,...]'))
FETCH FIRST 3 ROWS ONLY;
```

库内向量化操作（仅讲师操作）

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。

```
-- 这一步需要有授权：
-- grant create mining model to <user>;
-- grant create any directory to <user>;

-- 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
-- 实验做完后，删除模型以释放资源：
-- EXEC DBMS_VECTOR.DROP_ONNX_MODEL(model_name => 'mydoc_model', force => true);
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;
```

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;
```

至此，我们已经完成了Oracle向量数据库的库内向量化操作。

RAG

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：

```
-- 设置 ollama 监听地址和端口。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。
export OLLAMA_HOST=0.0.0.0:8098

-- 手工启动ollama进程
ollama serve

-- 运行模型
ollama run qwen2:7b-instruct
```

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：

```
*** Result:
Oracle 23ai 是 Oracle 数据库的一个版本，它结合了 Oracle 的数据库技术与人工智能功能，提供了增强的数据管理能力。虽然具体的新特性会随着版本的更新而变化，以下是一些 Oracle 23ai 可能包含的潜在新特性和增强功能的概述：

1. **增强的AI和ML功能**：Oracle 23ai 可能包含更强大的内置机器学习（ML）和人工智能（AI）功能，帮助用户更轻松地从数据中提取价值，包括预测性分析、异常检测、分类和聚类等。
2. **自动化数据库管理**：可能包括自动化优化、故障检测和恢复，以及更高级的自动化配置和性能调优，减少人工干预的需要。
3. **智能数据管理**：改进的数据分类、安全性、合规性检查和数据治理功能，帮助组织更有效地管理其数据资产。
4. **AI驱动的查询优化**：使用AI技术改进查询执行计划的选择，以提高查询性能和响应时间。
5. **增强的图形数据库功能**：如果 Oracle 23ai 包含图形数据库特性，那么可能有改进的图形查询语言（GQL）支持，以及增强的图形分析能力。
6. **多云和混合云兼容性**：加强与不同云提供商的集成，包括对不同云环境的更广泛支持，以及更好的多云管理工具。
7. **更新的UI和开发工具**：提供更新的图形用户界面（GUI）和开发工具，以提高开发效率和用户体验。
8. **安全性增强**：包括数据加密、访问控制和威胁检测在内的安全功能的增强，以适应不断变化的安全威胁环境。

请注意，这些特性和增强功能的具体内容和细节可能会在 Oracle 的官方发布说明和文档中详细列出。建议访问 Oracle 官方网站或相关的技术文档以获取最新和最准确的信息。

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
根据提供的上下文信息内容，回答用户的问题，且不要试图编造答案。\n 以下是上下文信息：' ||
replace(l_context, chr(10), '\n') || '"},
            {"role": "user", "content": "' || l_question || '"}
        ]
    }';

    -- 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;
/
```

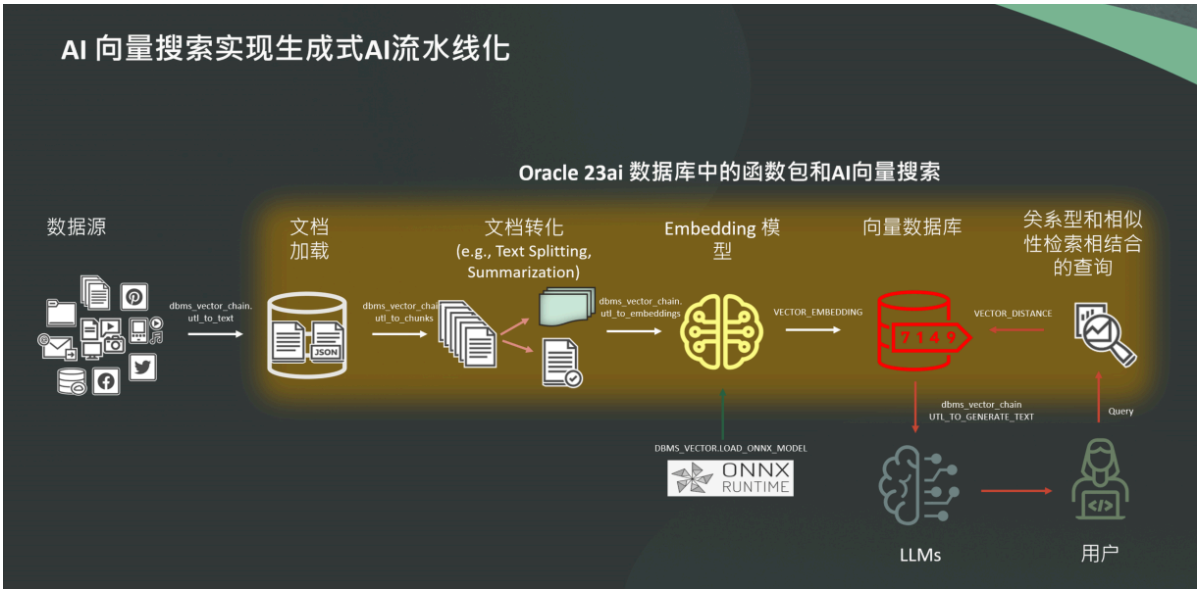
运行结果：

```
*** RAG Result:
Oracle 23ai 的新特性主要集中在人工智能（AI）、开发人员生产力、性能优化、安全性增强和数据库管理功能上。以下是一些重要的新特性：

### AI 和机器学习
- **AI 向量搜索**：使数据库能够利用新一代 AI 模型来生成和存储文本、图像、声音等的向量，进行索引和快速相似性搜索。
- **AI 优化的关键功能**：自动优化数据库功能以更准确地估算时间和资源成本。

### 开发人员生产力
- **JSON 二元性视图（JSON Duality Views）**：允许使用传统 SQL 或直接使用 JSON 执行查询和 DML 操作，结合关系和文档数据模型的优势。
- **微服务和消息传递支持**：改进了 Oracle Database 对微服务架构的支持。
- **JSON 支持**：引入新的数据类型和增强功能，支持 JSON 数据的存储和查询。

### 性能和数据库管理
- **True Cache**：在基础设施级别增加性能改进。
- **SQL 和 PL/SQL 改进**：包括新的数据类型、语言增强功能和 SQL 域功能，适用于 OLTP 和分析应用程序。
- **性能和数据库生命周期管理**：包括SQL分析报告、错误消息改进和性能增强功能。

### 数据库安全
- **SQL 防火墙**：提供更精确的 SQL 控制，增强安全性。
- **模式级别权限**：允许更细粒度的权限控制。

### 数据库可用性和管理
- **高可用性增强**：包括新的 RAFT 协议支持，简化数据库的分发和分片。
- **数据库管理简化**：降低复杂性，提高性能，并引入新功能简化管理任务，例如回收表空间中的可用空间。
- **回收表空间**：简化回收存储空间的任务。

### 其他改进
- **属性图（Property Graphs）**：用于复杂数据模型的存储和查询。
- **布尔值（Boolean）**：支持在 SQL 中使用布尔值类型。
- **DDL 的 IF Exists（IF Exists for DDL）**：在创建表等 DDL 语句中提供条件支持。
- **新表值子句**：允许在查询中直接生成表。
- **更新和删除的直接连接**：优化 DML 操作的执行效率。

这些新特性旨在提高 Oracle 数据库的灵活性、性能、安全性和与现代应用程序开发实践的兼容性。


PL/SQL procedure successfully completed.

SQL>
```

Oracle 库内向量化流水线操作（可选）

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）。

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_INDB_PIPELINE (
    id number,
    name varchar2(50),
    doc varchar2(500),
    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

-- 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
    chunk_data
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 1 ROWS ONLY;
```