ORACLE

Using Oracle Database Vectors in python-oracledb -Limited Availability Release

Driving AI/ML through Vector support in Python applications running Oracle Database

October 2023, Version 1.0

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without the prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

Table of contents

Introduction

Oracle Database 23.4 introduces a new VECTOR data type for advanced AI/ML operations. This data type is currently a homogeneous array of either 8-bit signed integers, or of 32-bit or 64-bit floating point numbers.

The python-oracledb extension module for Python is a database driver for high performance Oracle Database applications written in Python 3. Python-oracledb is a renamed, major upgrade of Oracle's popular cx_Oracle driver.

This interface lets you quickly develop applications that execute SQL or PL/SQL statements, allowing you to work with many data types including JSON and use Oracle's document storage SODA calls.

For the complete details on the python-oracledb driver, check the python-oracledb documentation.

The VECTOR data in Oracle Database

AI Vector Search is available in Oracle Database 23c Free (23.4.0.23.11) Limited Availability release.

VECTOR columns in Oracle Database can be created as

`VECTOR(<vectorDimensions>, <vectorDimensionFormat>),` where the attributes are:

- *vectorDimensions*: defines the number of dimensions for the vector data. For example, a point in 3D space is defined by vector data of 3 dimensions, i.e., the (x,y,z) coordinates;
- *vectorDimensionFormat*: one of the keywords INT8, FLOAT32, or FLOAT64 to define the storage format for each dimension value in the vector.

For example:

```
CREATE TABLE vecTab(dataVec VECTOR(3, FLOAT32));
```

For more information about vectors, refer to the Limited Availability documents included in the Limited Availability release ZIP file:

- Oracle Database AI Vector Search User's Guide

Vector support in python-oracledb

A Limited Availability release of Oracle Database 23c Free with vector support is provided to select customers. This release includes a python-oracledb package to support vectors.

The python-oracledb driver has direct access to Oracle Database through its default Thin mode.

In the Limited Availability release, only the Thin mode of python-oracledb natively supports vector data. Vectors are represented as Python objects of type *`array.array()`* or list. For testing vectors, Oracle recommends using Thin mode for performance and functionality reasons.

Using Vectors in python-oracledb Thick mode

In python-oracledb, an optional Thick mode can be enabled at runtime. It uses the Oracle Client libraries for connectivity to Oracle Database. Refer to the python-oracledb documentation.

Thick mode currently requires the use of strings when inserting vectors, and it returns vectors as Python lists of floating point numbers or integers.

Applications using Oracle Client libraries can insert vector data as string or CLOB directly.

```
INSERT INTO vecTab VALUES ('[1.1, 2.9, 3.14]');

SELECT dataVec FROM vecTab;
```

Installing the python-oracledb driver

Ensure that Python (version 3.7 or later) is installed on your machine and you have the connection details to an Oracle Database 23c Free (23.4.0.23.11) Limited Availability release that supports the VECTOR data type.

This Limited Availability python-oracledb driver is only included in the archive that was provided to you, it is not available on PyPI or GitHub.

To install the driver, use Python's pip module. Run the following in a command line terminal:

python -m pip install <filename>

Choose the right version of the python-oracledb wheel for your operating system and Python version. As a note, the Python binary might have a different name, depending on your platform.

For example, if you are on Linux x86_64 using Python 3.11, run the following:

python3.11 -m pip install oracledb-2.0.0.dev-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl*

This installs the driver as the 'oracledb' module.

For more details on installing the driver, refer to the python-oracledb installation manual.

APIs and Constants for Vectors in python-oracledb

Vectors can be fetched and inserted using standard python-oracledb APIs. When inserting and fetching vector data, columns are represented as Python `array.array()` values of integer (8-bit), float (32-bit), or double (64-bit). The vector data could also be converted to Python lists and NumPy *ndarrays*. For additional details on conversion, see the following examples.

After querying a vector, the metadata *Cursor.description* will show a new python-oracledb constant *oracledb.DB_TYPE_VECTOR* in the *FetchInfo.type_code* attribute. This constant can also be used as a type when calling *Cursor.setinputsizes()* or *Cursor.var()*.

New attributes *vector_dimensions* and *vector_type* have been added to the *FetchInfo* instance returned as part of fetch metadata:

- The *vector_dimensions* attribute is the number of dimensions of the VECTOR column. This attribute has the value *None* for non-vector columns and vector columns where the number of dimensions is flexible.
- The *vector_type* value will be one of the constants *oracledb.VECTOR_TYPE_INT8, oracledb.VECTOR_TYPE_FLOAT32*, or *oracledb.VECTOR_TYPE_FLOAT64*, representing 8-bit signed integer, 32-bit floating point, or 64-bit floating point numbers. This attribute has the value *None* for non-vector columns and for vector columns where the storage format is flexible.

Sample Python script using VECTOR data

The following is a sample Python script *vector.py*, which works with the VECTOR data type in Oracle Database using python-oracledb.

The script creates a table "*sample_vector_tab*" with three VECTOR columns:

- *V32* is a FLOAT32 format VECTOR column.
- *V64* is a FLOAT64 format VECTOR column.
- *V8* is an INT8 format VECTOR column.

Then data is inserted into the table. For inserting VECTOR columns, Python arrays (`array.array()`) of float (32-bit), double (64-bit), or int8_t (8-bit signed integer) are used as bind values.

When the data is queried and fetched from the table, VECTOR columns are fetched as a Python array `array.array()`.

```
# vector.py

import array
import oracledb

table_name = "sample_vector_tab"
vector_data_32 = array.array('f', [1.625, 2.5, 3.0])       # 32-bit float
vector_data_64 = array.array('d', [4.25, 5.75, 6.5])       # 64-bit float
vector_data_8  = array.array('b', [7, 8, 9])                # 8-bit signed integer

with oracledb.connect(user="un", password="pw", dsn="connection_string") as conn:
    cursor = conn.cursor()
    cursor.execute(f"drop table if exists {table_name}")
    cursor.execute(f"""
        create table {table_name} (
                    v32 vector(3, float32),
                    v64 vector(3, float64),
                     v8 vector(3, int8)
        )""")
    cursor.execute(f"insert into {table_name} values (:1, :2, :3)",
              [vector_data_32, vector_data_64, vector_data_8])
    cursor.execute(f"select * from {table_name}")
    for row in cursor:
        print(row)
```

Expect the following output after running the script:

```
$ python3 vector.py
(array('f', [1.625, 2.5, 3.0]), array('d', [4.25, 5.75, 6.5]), array('b', [7, 8,
9]))
```

The following output type handler can be created to perform a conversion to Python lists for vector data fetched from a connection:

```
def type_handler(cursor, metadata):
    if metadata.type_code is oracledb.DB_TYPE_VECTOR:
        return cursor.var(metadata.type_code, arraysize=cursor.arraysize,
                          outconverter=list)

connection.outputtypehandler = type_handler
```

If you are using NumPy, the following output type handler can be created to perform a conversion to the NumPy's *ndarray* for vector data fetched from a connection:

```
import numpy

def numpy_converter_out(value):
      if value.typecode == 'b':
          dtype = numpy.int8
      elif value.typecode == 'f':
          dtype = numpy.float32
      else:
          dtype = numpy.float64
      return numpy.array(value, copy=False, dtype=dtype)

def type_handler(cursor, metadata):
      if metadata.type_code is oracledb.DB_TYPE_VECTOR:
          return cursor.var(metadata.type_code, arraysize=cursor.arraysize,
                            outconverter=numpy_converter_out)

connection.outputtypehandler = type_handler
```

If your data is already in an NumPy *ndarray*, the following input type handler can be created that will perform the conversion:

```
import numpy

def numpy_converter_in(value):
      if value.dtype == numpy.int8:
          dtype = 'b'
      elif value.dtype == numpy.float32:
          dtype = 'f'
      else:
          dtype = 'd'
      return array.array(dtype, value)

def type_handler(cursor, value, arraysize):
      if isinstance(value, numpy.ndarray):
          return cursor.var(oracledb.DB_TYPE_VECTOR, arraysize=arraysize,
                            inconverter=numpy_converter_in)
connection.inputtypehandler = type_handler
```

Use the following sample code for inserting vector data in a table and fetching it using NumPy *ndarray* and type handlers to perform the conversions:

```
# vector_numpy.py

import array
import numpy
import oracledb

id_val = 4
vector_data_int = numpy.array([1, 2, 3, 4])
vector_data = numpy.array([15.625, 21.25, 18.75, 2.5])

def numpy_converter_in(value):
    if value.dtype == numpy.float64:
        dtype = 'd'
    elif value.dtype == numpy.float32:
        dtype = 'f'
    else:
        dtype = 'b'
    return array.array(dtype, value)

def input_type_handler(cursor, value, arraysize):
    if isinstance(value, numpy.ndarray):
        return cursor.var(oracledb.DB_TYPE_VECTOR, arraysize=arraysize,
            inconverter=numpy_converter_in)

def numpy_converter_out(value):
    if value.typecode == 'd':
        dtype = numpy.float64
    elif value.typecode == 'f':
        dtype = numpy.float32
    else:
        dtype = numpy.int8
    return numpy.array(value, copy=False, dtype=dtype)

def output_type_handler(cursor, metadata):
    if metadata.type_code is oracledb.DB_TYPE_VECTOR:
        return cursor.var(metadata.type_code, arraysize=cursor.arraysize,
                          outconverter=numpy_converter_out)

conn = oracledb.connect(user="un", password="pw", dsn="connection_string")
```

```
cursor = conn.cursor()
conn.inputtypehandler = input_type_handler
conn.outputtypehandler = output_type_handler

table_name = "sample_numpy_tab"
cursor.execute(f"drop table if exists {table_name}")
cursor.execute(f"""
    create table {table_name} (
        id number,
        v32 vector(4, float32),
        v64 vector(4, float64),
        v8  vector(4, INT8),
        primary key (id)
    )""")

cursor.execute(f"insert into {table_name} values (:1, :2, :3, :4)",
               [id_val, vector_data, vector_data, vector_data_int])

conn.commit()

cursor.execute(f"select * from {table_name}")
for row in cursor:
    print(row)
```

Expect the following output after running the script:

```
$ python3 vector_numpy.py
(4, array([15.625, 21.25 , 18.75 ,  2.5  ], dtype=float32), array([15.625, 21.25
, 18.75 ,  2.5  ]), array([1, 2, 3, 4], dtype=int8))
```

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

ORACLE