Using **OracleVectorDB** with Llama-index and LangChain.

Last update: 03/02/2024

Table of Contents

[Introduction. 1](#_Toc157847663)

[Database Schema. 1](#_Toc157847664)

[Metadata. 2](#_Toc157847665)

[Loading the data. 2](#_Toc157847666)

[Standalone usage. 3](#_Toc157847667)

[Integration with Llama-index. 3](#_Toc157847668)

[Integration with LangChain. 4](#_Toc157847669)

[Distance functions. 4](#_Toc157847670)

[How to change the schema. 4](#_Toc157847671)

## Introduction.

I have developed **OracleVectorDB** as an interface designed to integrate the capabilities of **Oracle DB 23.4 LA, to store and manage embeddings vectors,** with orchestration libraries for RAG, specifically **Llama-index** and **LangChain**.

It is not an official integration, and it will be replaced by the Oracle official integration, as soon as the DB Vector Store becomes GA.

The integration has been developed and tested for Llama-index and LangChain.

## Database Schema.

The biggest limitation in these demos, in my view, is that all the code now works with a fixed database schema. You can change the owner of the Schema, changing the values for the following properties, defined in the file config\_private.py

DB\_USER = "vector"

DB\_PWD = "xxxx"

DB\_HOST\_IP = "130.162.xxx.yyy"

DB\_SERVICE = "freepdb1"

The schema used is composed by 3 tables:

* BOOKS
* CHUNKS
* VECTORS

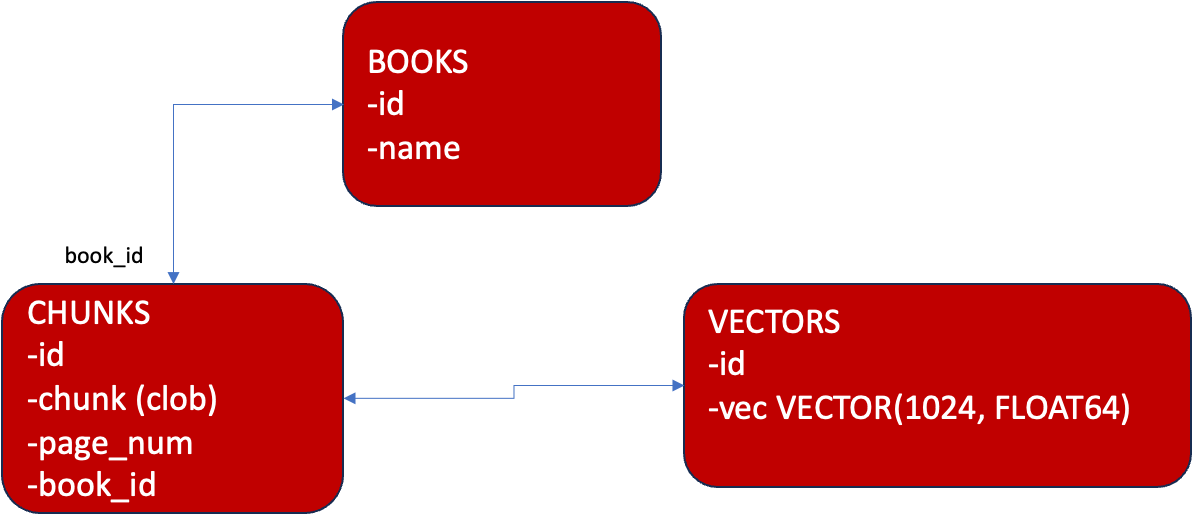


Figure 1: database schema

A chunk is a “portion of a document”. Several strategies can be used to do the “chunking” and, in a real production setting, this should be a key design decision. To keep things simpler, as part of the demo, for each book a chunk is a book page.

To assign a unique ID to each chunk (and therefore the associated embeddings vector) the ID is computed, during loading, as the HASH for the text.

This is the SQL script to create the schema:

<https://github.com/luigisaetta/llamaindex_oracle/blob/main/create_tables.sql>

## Metadata.

For now, only the following metadata are stored and managed:

* The book\_name (or file name)
* The number of the page (page\_num)

Consider that, we’re relying on Llama-index loaders. Therefore, you must ensure that the loader used is able to extract the page number.

## Loading the data.

Data loading (chunks and embeddings + metadata) should be done offline before querying.

Even if the OracleVectorDB class has some functionalities to write, I suggest that the **loading is always done using the Python program:**

* create\_and\_save\_embedding.py

The program reads the list of books to load from the config.py file. It can be run without any command-line parameter.

## Standalone usage.

If you want to use the OracleVectorDB standalone (not integrated in a Llama-index/LangChain chain), you’ll find a code example in the following Notebook:

<https://github.com/luigisaetta/llamaindex_oracle/blob/main/custom_vector_store_demo1.ipynb>

This is the code needed:

v\_store **=** OracleVectorStore(verbose**=True**)

question **=** (

"What is JSON Relational Duality in Oracle Database 23c? Explain with details"

)

*# embed the query using the selected embeddings model*

query\_embedding **=** embed\_model**.**embed\_documents([question])[0]

*# wrap in llama-index*

query\_obj **=** VectorStoreQuery(query\_embedding**=**query\_embedding, similarity\_top\_k**=**6)

q\_result **=** v\_store**.**query(query\_obj)

You should consider that **you need an Embedding model to compute embeddings**, even in the query phase, where it is needed to embed the text of the query.

## Integration with Llama-index.

In the Notebook

<https://github.com/luigisaetta/llamaindex_oracle/blob/main/rag_chain_demo5.ipynb>

it is shown how to create a complete chain (embeddings, vector db, reranker, llm)

service\_context **=** ServiceContext**.**from\_defaults(llm**=**llm\_oci, embed\_model**=**embed\_model)

index **=** VectorStoreIndex**.**from\_vector\_store(

vector\_store**=**v\_store, service\_context**=**service\_context

)

query\_engine **=** index**.**as\_query\_engine(

similarity\_top\_k**=**TOP\_K,

node\_postprocessors**=**[reranker],

)

## Integration with LangChain.

In the Python file

oracle\_vector\_db\_lc.py

there is a basic integration with LangChain. Only read functionalities are implemented.

Loading of data should again be done using create\_and\_save\_embeddings.

A demo is in this NB:

<https://github.com/luigisaetta/llamaindex_oracle/blob/main/demo_langchain2.ipynb>

## Distance functions.

As of now (Jan. 2024) only one distance function is supported in the code: DOT, which compute the **negated cosine distance**.

Since distance is negated (this is by design in Oracle Vector DB 23.4) the most relevant docs have negative similarity (are the first returned if sort is ASC).

## How to change the schema.

If you want to change the schema (and the query) you need to modify the code of the function

* oracle\_query

Now, this is the SQL code:

select = f"""select V.id, C.CHUNK, C.PAGE\_NUM,

ROUND(VECTOR\_DISTANCE(V.VEC, :1, DOT), 3) as d,

B.NAME

from VECTORS V, CHUNKS C, BOOKS B

where C.ID = V.ID and

C.BOOK\_ID = B.ID

order by d

FETCH FIRST {top\_k} ROWS ONLY"""

The function has for input only:

* the query embedding vector
* top\_k

If you change the list and the order of return values from the query, you need also to modify the following Python code, that builds the result (conforming to Llama-index/LangChain protocol).