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

Table of Contents

[Introduction. 1](#_Toc155361314)

[Database Schema. 1](#_Toc155361315)

[Metadata. 2](#_Toc155361316)

[Loading the data. 2](#_Toc155361317)

[Standalone usage. 2](#_Toc155361318)

[Integration with Llama-index. 3](#_Toc155361319)

## 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.It should work even in LangChain, due to the existing interoperability features.

## Database Schema.

The biggest limitation, in my view, is that OracleVectorDB now works with a fixed database schema. You can only 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

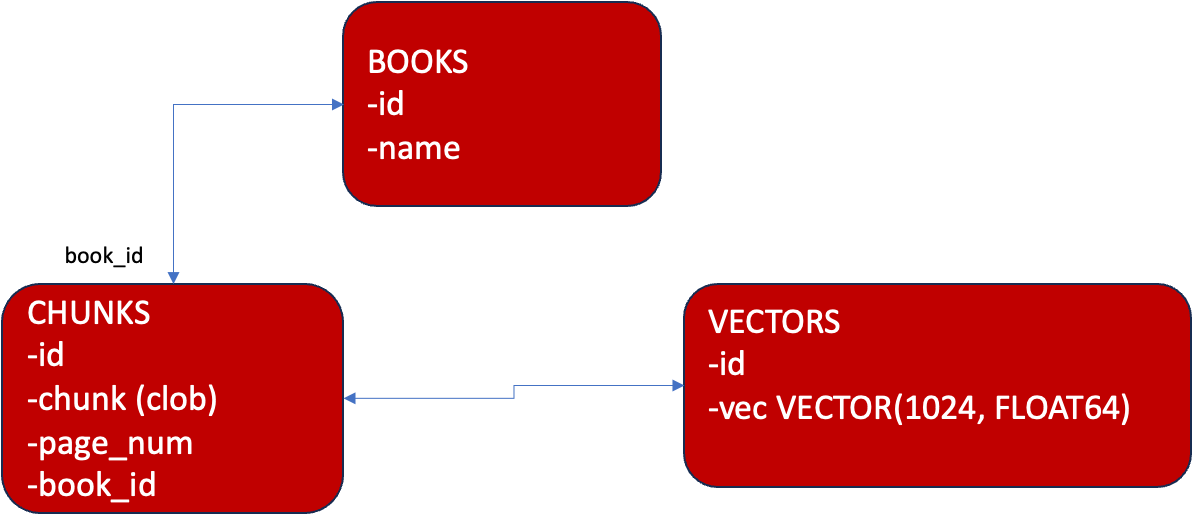


Figure 1: database schema

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
* 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 (chunks and embeddings + metadata) should be done offline, before querying.

Even if the OracleVectorDB class has some functionalities to write, the **loading should always be 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 commend-line parameter.

## Standalone usage.

If you want to use the OracleVectorDB standalone (not integrated in a Lalama-index 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],

)