

1

What's Our Vector, Victor! Navigating AI Vector Search

Presented on 24th October 2024

to

ITOUG

in Milan, Italy

by

Niall Mc Phillips - Long Acre sàrl

niall.mcphillips@longacre.ch

[@Niall_McP](https://twitter.com/Niall_McP)



2



3



4



Extensive Cloud Experience



Early Adopter

5

About me: Niall Mc Phillips

Owner - Long Acre sàrl

Irish 🇮🇪 / 🇨🇭 Swiss living in Geneva, Switzerland.



Oracle ACE
Pro

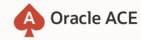
-SYMPOSIUM L2

- Oracle Developer and DBA for a long time
- Developing web applications with Oracle DB since 1995
- Developing with APEX since 2005 (HTML DB 1.6)

🐦 @Niall_McP

✉ niall.mcphillips@longacre.ch

6

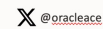
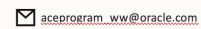


The Oracle ACE Program

400+ technical experts helping peers globally



- The Oracle ACE Program recognizes and rewards community members for their technical and community contributions to the Oracle community
- 3 membership levels: Director, Pro, and Associate
- Nominate yourself or a colleague at ace.oracle.com/nominate
- Learn more at ace.oracle.com



7



ACE Member Benefits



Key Benefits

Cool **swag***, Digital awards for social media, Oracle **CloudWorld pass***, & more



Direct Access to Product Management

Multiple **direct communication channels** to product management and fellow ACEs



Exclusive Content

Exclusive **monthly virtual meetings** with product development teams + engaging guest speakers



Networking

In-person & virtual networking opportunities for ACEs to **connect with product development** and each other.



Cloud Account

\$5k USD Cloud account*



Travel Support

ACE Directors are eligible for travel support to give presentations or lead workshops at conferences globally



8 * for Pro and Director levels

8

What is AI Vector Search

AI Vector Search enables you to search structured and unstructured data based on its semantics or meaning, in addition to its values.



10

Vectors

Vectors are derived from the **semantic content** of your data, rather than the underlying words or image parts.

For example, a search for the term **architecture** could also find results such as *buildings*, *construction* or even *a famous architect*.

In this respect they are quite different to other search options such as Oracle Search Indexes.



11

Vectors

From a simplistic point of view, a vector is simply a list of numbers.

The number of dimensions that a vector has is how many numbers it contains.

[0.8763, 1.8476, 0.2977, 2.8686,]



12

Vector Similarity

Vector similarity is calculated using the mathematical distance between two vectors

[0.8763, 1.8476, 0.2977, 2.8686,]

[0.7836, 1.6748, 0.7729, 2.6868,]



13

Where do I get my vectors – Vector Embeddings

The creation / calculation of a vector from data is called **EMBEDDING**

There are many different embedding models, including those provided by OpenAI, Cohere and many open-source models.

The model you choose will be influenced by the data that you are vectorising: text, images, video, sound,



14

Where do I get my vectors – Vector Embeddings

For the purposes of today's presentation, we'll be using an OpenAI model called: *text-embedding-3-large*

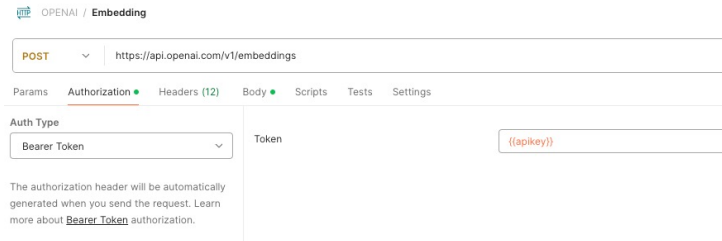
This model returns a vector of dimension 3000



15

Vector Embeddings – Using the OpenAI API

Calling the OpenAI API from Postman - Authentication



OPENAI / Embedding

POST ▼ | https://api.openai.com/v1/embeddings

Params | Authorization ● | Headers (12) | Body ● | Scripts | Tests | Settings

Auth Type: Bearer Token ▼

Token: {{apikey}}

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.



16

Vector Embeddings – Using the OpenAI API

Calling the OpenAI API from Postman specify text to be vectorised and which model to use



OPENAI / Embedding Save Share

POST ▼ | https://api.openai.com/v1/embeddings Send ▼

Params | Authorization ● | Headers (12) | Body ● | Scripts | Tests | Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```
1 {  
2   "input": "Geneva is situated at the end of the lake that shares its name 'Lake Geneva',  
3   "model": "text-embedding-3-large"  
4 }
```



17

Vector Embeddings – Using the OpenAI API

Calling the OpenAI API from Postman - Response

```
Body Cookies (2) Headers (25) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "object": "list",
3   "data": [
4     {
5       "object": "embedding",
6       "index": 0,
7       "embedding": [
8         -0.025723679,
9         -0.001198068,
10        -0.010066863,
11        0.01498899,
12        -0.035073247,
13        -0.04261721,
14        0.0186002,
15        0.0117920805,
16        0.0029263776,
17        -0.027826095,
```



18

Embeddings 1 – example of setting up in PL/SQL

An example of writing a PL/SQL procedure to :

- Make a call to a 3rd party embedding API
- Retrieve the response user the APEX_WEB_SERVICE package
- Parse the response and extract the vector



19

Embeddings 1 – setting up in PL/SQL

First declare some variables:

```
v_clob      clob;  
vj_json     json_object_t;  
vj_obj      json_object_t;  
vj_data     json_array_t;  
vj_embedding json_array_t;
```



20

Embeddings 1 – setting up the Request Header

We'll use apex_web_service.make_rest_request

First set the headers:

```
apex_web_service.g_request_headers(1).name := 'Content-Type';  
apex_web_service.g_request_headers(1).value := 'application/json';  
apex_web_service.g_request_headers(2).name := 'Authorization';  
apex_web_service.g_request_headers(2).value := 'Bearer '||getOpenaiAPIKey();
```

* note: getOpenaiAPIKey() is a function to retrieve our stored API key for the OpenAI API



21

Embeddings 1 – getting the response

Call and get the response from the OpenAI Embedding API:

```
...
-- Get the response from the web service.
v_clob := apex_web_service.make_rest_request
    (p_url          => 'https://api.openai.com/v1/embeddings',
      p_http_method => 'POST',
      p_body        => '{"model": "|||p_model
                        |||", "input": "||
                        ||trim(lower(p_input))||'|' }';
...
* note: p_model is 'text-embedding-3-large'
      p_input is the sentence that we want the embedding for
```



22

Embeddings 1 – extract the vector

Extract the vector from the response

```
...
-- Convert to JSON data type, then parse and extract embedd
vj_json := json_object_t(v_clob);
vj_data := vj_json.get_array('data'); -- extract the data object
vj_obj  := treat(vj_data.get(0) as json_object_t); -- it's a 1-element array
vj_embedding := vj_obj.get_array('embedding'); -- extract the embedding
return vj_embedding; -- this is the vector as a JSON object
...
```

A screenshot of a web browser's developer tools showing a JSON response. The response is a list containing one object, which is an embedding. The embedding is a list of 15 floating-point numbers. The JSON is formatted with syntax highlighting and line numbers.

```
1 {
2   "object": "list",
3   "data": [
4     {
5       "object": "embedding",
6       "index": 0,
7       "embedding": [
8         -0.025723679,
9         -0.001198068,
10        -0.010066863,
11         0.014988899,
12        -0.035073247,
13        -0.04261721,
14         0.0186802,
15         0.0117920805,
16         0.0029263776,
17        -0.027826895,
```



23

Embeddings 1 – convert the JSON vector to the new 23ai Vector data type

Convert using the new 23ai *to_vector* function

```
...  
    vv_vector      vector;  
...  
    vv_vector := to_vector(vj_embedding.to_clob());  
...
```



24

Embeddings 1 - use the embedding vector

Store it in a table

```
...  
update sentences s  
    set s.the_vector = vv_vector  
    where id = p_id;  
...
```



25


[illegible]

27

Let's look at the new data type: VECTOR

With 23ai VECTOR is a new data type fully integrated into the database.

```
create table my_tab  
(my_id integer,  
my_text varchar2(500),  
my_vector vector);
```

The logo for Long Acre Solutions Today, featuring a stylized green grass icon above the text "long acre" in a serif font, with "solutions today" in a smaller sans-serif font below it, all enclosed in a thin horizontal line.

27

VECTOR data type

Possible Declaration Format	Explanation
<code>VECTOR(*, *)</code>	Vectors can have an arbitrary number of dimensions and formats. <code>VECTOR</code> and <code>VECTOR(*,*)</code> are equivalent.
<code>VECTOR(number_of_dimensions, *)</code> equivalent to <code>VECTOR(number_of_dimensions)</code>	Vectors must all have the specified number of dimensions or an error is thrown. Every vector will have its dimensions stored without format modification.
<code>VECTOR(*, dimension_element_format)</code>	Vectors can have an arbitrary number of dimensions, but their format will be up-converted or down-converted to the specified dimension element format (<code>INT8</code> , <code>FLOAT32</code> , or <code>FLOAT64</code>).

A vector can be `NULL` but its dimensions cannot (for example, you cannot have a `VECTOR` with a `NULL` dimension such as `[1.1, NULL, 2.2]`).



28

VECTOR examples – define columns

```
CREATE TABLE my_vect_tab
(v1 VECTOR(3, FLOAT32),
 v2 VECTOR(2, FLOAT64),
 v3 VECTOR(1, INT8),
 v4 VECTOR(1, *),
 v5 VECTOR(*, FLOAT32),
 v6 VECTOR(*, *),
 v7 VECTOR);
```

Table created.



29

VECTOR examples – show

```
DESC my_vect_tab;
Name      Null?   Type
-----
V1                VECTOR(3 , FLOAT32)
V2                VECTOR(2 , FLOAT64)
V3                VECTOR(1 , INT8)
V4                VECTOR(1 , *)
V5                VECTOR(* , FLOAT32)
V6                VECTOR(* , *)
V7                VECTOR(* , *)
```



30

New Vector function – VECTOR_DISTANCE

VECTOR_DISTANCE calculates the distance between two vectors

vector_distance (vector1, vector2, method)

returns a number containing the distance between vectors 1 and 2, calculated according to the method specified in the 3rd parameter.



31

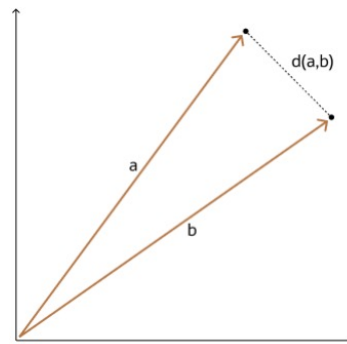
VECTOR_DISTANCE - methods

- COSINE *(function alias COSINE_DISTANCE)*
- EUCLIDEAN *(a.k.a. L2 – function alias L2_DISTANCE)*
- EUCLIDEAN_SQUARED
- MANHATTAN *(a.k.a. L1 – function alias L1_DISTANCE)*
- DOT *(a.k.a. INNER_PRODUCT – function alias INNER_PRODUCT)*
- HAMMING



32

VECTOR_DISTANCE – EUCLIDEAN and EUCLIDEAN_SQUARED (L2)

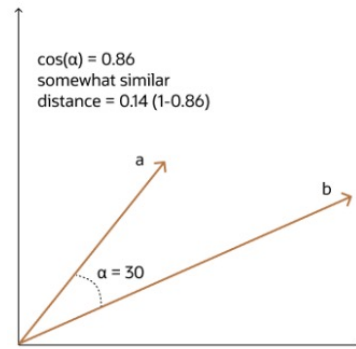


Measure the distance between the vector co-ordinates that are being compared.



33

VECTOR_DISTANCE – COSINE



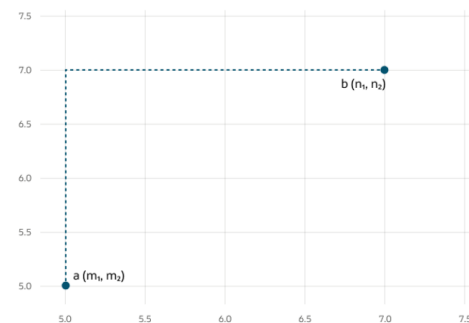
Measures the cosine of the angle between the vectors being compared.

This is the most commonly used method.



34

VECTOR_DISTANCE – MANHATTAN (L1)



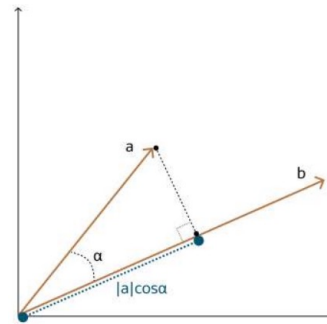
a.k.a. taxi cab distance

Sums the distance between the dimensions of the two vectors



35

VECTOR_DISTANCE – DOT product (INNER PRODUCT)



Multiplies the size of each vector by the cosine of their angle.



36

VECTOR_DISTANCE – Hamming

A	1	0	0	1	0	1	1	0
B	1	1	0	1	1	0	1	0
XOR Bit Operations								
	0	1	0	0	1	1	0	0

Hamming Distance = 3

Hamming is like an XOR between the two vectors that measures the number of differences.

Often used to detect network anomalies.



37

VECTOR_DISTANCE – Example

```
select vector_distance
      (s.the_vector, P_SEARCH_VECTOR, COSINE) as vdist,
      s.the_text,
      s.id
  from sentences s
 order by vdist asc
fetch first 8 rows only;
```



38

Some other Vector functions

TO_VECTOR / VECTOR
Constructs a vector from character data

FROM_VECTOR/VECTOR_SERIALIZE
Converts a vector to VARCHAR2 or CLOB

...



39

AI Vector Search - DEMOS



40

AI Vector Search – PL/SQL Packages

- **DBMS_VECTOR**

Simplifies common operations with Oracle AI Vector Search, such as extracting chunks or embeddings from user data, generating text for a given prompt, or creating vector indexes.

- **DBMS_VECTOR_CHAIN**

Enables advanced operations with Oracle AI Vector Search, such as chunking and embedding data along with text generation and summarization capabilities.

It is more suitable for text processing with similarity search, using functionality that can be pipelined together for an end-to-end search.



41

Embeddings 2 – DBMS_VECTOR

Example using the DBMS_VECTOR package

- Create credentials for a 3rd party embedding API



42

Embeddings 2 – create credentials

```
declare
  jo json_object_t;
begin
  jo := json_object_t();
  jo.put('access_token', 'sk-*****'); -- api key
  dbms_vector.create_credential(
    credential_name => 'NIALLOPENAI_CRED',
    params          => json(jo.to_string));
end;
/
```



43

Embeddings 2 – create credentials

```
SQL>
SQL> set echo on
SQL> declare
  2   jo json_object_t;
  3   begin
  4     jo := json_object_t();
  5     jo.put('access_token', embeddings.getOpenaiAPIKey());
  6     dbms_vector.create_credential(
  7       credential_name => 'NIALLOPENAI_CRED',
  8       params          => json(jo.to_string));
  9   end;
 10  /
```

retrieve the API
key from table

PL/SQL procedure successfully completed.



44

Embeddings 2 – *utl_to_embedding* function

- Create credentials for a 3rd party embedding API
- Use the *dbms_vector.utl_to_embedding* function to embed and return a vector

```
DBMS_VECTOR.UTL_TO_EMBEDDING  
  (DATA IN CLOB,  
   PARAMS IN JSON default NULL )  
  return VECTOR
```



45

Embeddings 2 – *utl_to_embedding* function parameters example

```
{  
  "provider": "openai",  
  "credential_name": "NIALLOPENAI_CRED",  
  "url": "https://api.openai.com/v1/embeddings",  
  "model": "text-embedding-3-large"  
}
```



46

Embeddings 2 – *utl_to_embedding* function usage example

```
declare  
  v_vector vector;  
begin  
  v_vector := dbms_vector.utl_to_embedding  
    (data => 'Buongiorno Italia!',  
     params => json(embeddings.getOpenaiAPIEmbeddingParams()));  
  
  dbms_output.put_line(embeddings.serializeMyVector(v_vector));  
end;  
/
```



47

Embeddings 2 – *utl_to_embedding* function usage example

```
SQL> declare
2   v_vector vector;
3   begin
4   v_vector := dbms_vector.utl_to_embedding
5   (data => 'Buongiorno Italia!',
6    params => json(embeddings.getOpenaiAPIEmbeddingParams()));
7
8   dbms_output.put_line(embeddings.serializeMyVector(v_vector));
9   end;
10  /
[-2.33546775E-002,3.92329413E-003,-2.01128423E-002,1.43358689E-002,-;
```

PL/SQL procedure successfully completed.



48

“Chunking” - splitting your input into smaller pieces

Models usually have a limit for how many tokens they can vectorise at a time. So we often have to breakup a text or a document into “chunks” before vectorising it.

This process is called “chunking”



49

Chunking with DBMS_VECTOR *utl_to_chunks*

We can use the *utl_to_chunks* function which returns an array of vectors along with some metadata in JSON format

```
DBMS_VECTOR.UTL_TO_CHUNKS  
  (DATA          IN CLOB (or VARCHAR2)  
   PARAMS        IN JSON default NULL)  
return VECTOR_ARRAY_T;
```



51

Chunking with DBMS_VECTOR *utl_to_chunks*

Parameters example:

```
{ "by": "words",  
  "max": "100",  
  "overlap": "0",  
  "split": "recursively",  
  "normalize": "all" }
```



52

Chunking with DBMS_VECTOR utl_to_chunks

```
select t.text_key doc,
       json_value(c.column_value, '$.chunk_id' returning number) as id,
       json_value(c.column_value, '$.chunk_offset' returning number) as offset,
       json_value(c.column_value, '$.chunk_length' returning number) as len,
       json_value(c.column_value, '$.chunk_data') as text
from texts_for_chunking t,
     dbms_vector.utl_to_chunks
      (t.text,
       json('{ "by": "words",
               "max": "100",
               "overlap": "0",
               "split": "recursively",
               "normalize": "all" }')) c;
```



53

Chunking with DBMS_VECTOR utl_to_chunks

```
select t.text_key doc,
       json_value(c.column_value, '$.chunk_id' returning number) as id,
       json_value(c.column_value, '$.chunk_offset' returning number) as offset,
       json_value(c.column_value, '$.chunk_length' returning number) as len,
       json_value(c.column_value, '$.chunk_data') as text
from texts_for_chunking t,
     dbms_vector.utl_to_chunks
      (t.text,
       json('{ "by": "words",
               "max": "100",
               "overlap": "0",
               "split": "recursively",
               "normalize": "all" }')) c;
```

Query Result x

SQL | All Rows Fetched: 17 in 0,495 seconds

DOC	ID	OFFSET	LEN	TEXT
BEEROVIAN-EMPIRE	1	1	511	The history of the Beerovian Empire is a tale
BEEROVIAN-EMPIRE	2	512	468	The earliest records of the Beerovian people c
BEEROVIAN-EMPIRE	3	980	455	The unification of these tribes began under th



54

Embedding the chunks with DBMS_VECTOR

Uses `utl_to_embedding` which accepts the output from `utl_to_chunks` as a parameter

```
select t.*,  
       et.column_value as vector_value  
from texts_for_chunking t,  
     dbms_vector.utl_to_embeddings  
  (dbms_vector.utl_to_chunks  
   (t.text,  
    json('{ "by": "words", "max": 100 }')),  
    json(embeddings.getOpenaiAPIEmbeddingParams())) et;
```



55

AI Vector Search – beyond the basics Importing pre-trained models

Import pre-trained embedding ML models for vector generation within the DB

- Open Neural Network Exchange (ONNX) is an open format built to represent machine learning models
- <https://onnx.ai/>



56

AI Vector Search – beyond the basics

Vector Indexes and Approximate Similarity

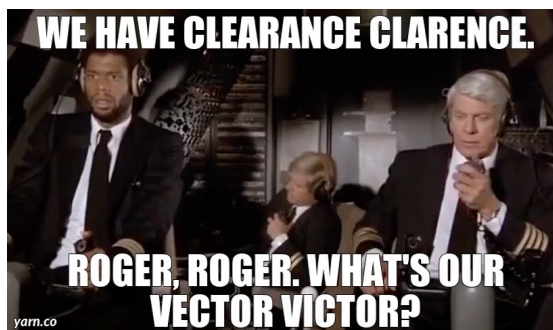
Vector Indexes and approximate similarity

Vector indexes are a class of specialized indexing data structures designed to accelerate similarity searches using high-dimensional vectors.

They use techniques such as clustering, partitioning, and neighbour graphs to group vectors representing similar items, which drastically reduces the search space, thereby making the search process quite efficient.



58



59

