Open in Colab

Postgres Vector Store

In this notebook we are going to show how to use Postgresql and pgvector to perform vector searches in LlamaIndex

If you're opening this Notebook on colab, you will probably need to install LlamaIndex 🦙 .

```
%pip install llama-index-vector-stores-postgres
```

```
!pip install llama-index
```

Running the following cell will install Postgres with PGVector in Colab.

```
!sudo apt update
!echo | sudo apt install -y postgresql-common
!echo | sudo /usr/share/postgresql-
common/pgdg/apt.postgresql.org.sh
!echo | sudo apt install postgresql-15-pgvector
!sudo service postgresql start
!sudo -u postgres psql -c "ALTER USER postgres PASSWORD
'password';"
!sudo -u postgres psql -c "CREATE DATABASE vector_db;"
```

```
# import logging
# import sys

# Uncomment to see debug logs
# logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
#
logging.getLogger().addHandler(logging.StreamHandler(stream=sys.stdo
ut))

from llama_index.core import SimpleDirectoryReader, StorageContext
from llama_index.core import VectorStoreIndex
from llama_index.vector_stores.postgres import PGVectorStore
```

```
import textwrap
import openai
```

Setup OpenAI

The first step is to configure the openai key. It will be used to created embeddings for the documents loaded into the index

```
import os

os.environ["OPENAI_API_KEY"] = "<your key>"
openai.api_key = os.environ["OPENAI_API_KEY"]
```

Download Data

```
!mkdir -p 'data/paul_graham/'
!wget 'https://raw.githubusercontent.com/run-llama/llama_index/main/docs/docs/examples/data/paul_graham/paul_graham_essay.txt' -O 'data/paul_graham/paul_graham_essay.txt'
```

```
--2024-07-29 15:29:26--  https://raw.githubusercontent.com/run-llama/llama_index/main/docs/docs/examples/data/paul_graham/paul_graham_essay.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 2606:50c0:8000::154, 2606:50c0:8002::154, 2606:50c0:8003::154, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|2606:50c0:8000::154|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75042 (73K) [text/plain]
Saving to: ‘data/paul_graham/paul_graham_essay.txt’

data/paul_graham/pa 100%[===================>]  73.28K  --.-KB/s    in 0.04s

2024-07-29 15:29:26 (1.75 MB/s) - ‘data/paul_graham/paul_graham_essay.txt’ saved [75042/75042]
```

Loading documents

Load the documents stored in the `data/paul_graham/` using the SimpleDirectoryReader

```
documents = SimpleDirectoryReader("./data/paul_graham").load_data()
print("Document ID:", documents[0].doc_id)
```

```
Document ID: e9a28a97-73af-42dd-8b40-9c585e222e69
```

Create the Database

Using an existing postgres running at localhost, create the database we'll be using.

```
import psycopg2

connection_string = "postgresql://postgres:password@localhost:5432"
db_name = "vector_db"
conn = psycopg2.connect(connection_string)
conn.autocommit = True

with conn.cursor() as c:
    c.execute(f"DROP DATABASE IF EXISTS {db_name}")
    c.execute(f"CREATE DATABASE {db_name}")
```

Create the index

Here we create an index backed by Postgres using the documents loaded previously. PGVectorStore takes a few arguments. The example below constructs a PGVectorStore with a HNSW index with m = 16, ef_construction = 64, and ef_search = 40, with the `vector_cosine_ops` method.

```
from sqlalchemy import make_url

url = make_url(connection_string)
vector_store = PGVectorStore.from_params(
    database=db_name,
    host=url.host,
    password=url.password,
    port=url.port,
    user=url.username,
    table_name="paul_graham_essay",
    embed_dim=1536,  # openai embedding dimension
    hnsw_kwargs={
        "hnsw_m": 16,
        "hnsw_ef_construction": 64,
```

```
        "hnsw_ef_search": 40,
        "hnsw_dist_method": "vector_cosine_ops",
    },
)

storage_context =
StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_documents(
    documents, storage_context=storage_context, show_progress=True
)
query_engine = index.as_query_engine()
```

Query the index

We can now ask questions using our index.

```
response = query_engine.query("What did the author do?")
```

```
print(textwrap.fill(str(response), 100))
```

```
The author worked on writing essays, programming, developing softwar
e, giving talks, and starting a
startup.
```

```
response = query_engine.query("What happened in the mid 1980s?")
```

```
print(textwrap.fill(str(response), 100))
```

```
In the mid-1980s, the context mentions the presence of a famous fund
manager who was not much older
than the author and was super rich. This sparked a thought in the aut
hor's mind about becoming rich
as well to have the freedom to work on whatever he wanted.
```

Querying existing index

```
vector_store = PGVectorStore.from_params(
    database="vector_db",
    host="localhost",
    password="password",
    port=5432,
```

```
    user="postgres",
    table_name="paul_graham_essay",
    embed_dim=1536,  # openai embedding dimension
    hnsw_kwargs={
        "hnsw_m": 16,
        "hnsw_ef_construction": 64,
        "hnsw_ef_search": 40,
        "hnsw_dist_method": "vector_cosine_ops",
    },
)

index =
VectorStoreIndex.from_vector_store(vector_store=vector_store)
query_engine = index.as_query_engine()
```

```
response = query_engine.query("What did the author do?")
```

```
print(textwrap.fill(str(response), 100))
```

```
The author worked on writing essays, programming, developing softwar
e, giving talks, and starting a
startup.
```

Hybrid Search

To enable hybrid search, you need to:

1. pass in `hybrid_search=True` when constructing the `PGVectorStore` (and optionally configure `text_search_config` with the desired language)
2. pass in `vector_store_query_mode="hybrid"` when constructing the query engine (this config is passed to the retriever under the hood). You can also optionally set the `sparse_top_k` to configure how many results we should obtain from sparse text search (default is using the same value as `similarity_top_k`).

```
from sqlalchemy import make_url

url = make_url(connection_string)
hybrid_vector_store = PGVectorStore.from_params(
    database=db_name,
    host=url.host,
    password=url.password,
    port=url.port,
```

```
    user=url.username,
    table_name="paul_graham_essay_hybrid_search",
    embed_dim=1536,  # openai embedding dimension
    hybrid_search=True,
    text_search_config="english",
    hnsw_kwargs={
        "hnsw_m": 16,
        "hnsw_ef_construction": 64,
        "hnsw_ef_search": 40,
        "hnsw_dist_method": "vector_cosine_ops",
    },
)

storage_context = StorageContext.from_defaults(
    vector_store=hybrid_vector_store
)
hybrid_index = VectorStoreIndex.from_documents(
    documents, storage_context=storage_context
)
```

```
hybrid_query_engine = hybrid_index.as_query_engine(
    vector_store_query_mode="hybrid", sparse_top_k=2
)
hybrid_response = hybrid_query_engine.query(
    "Who does Paul Graham think of with the word schtick"
)
```

```
print(hybrid_response)
```

```
Roy Lichtenstein
```

Improving hybrid search with QueryFusionRetriever

Since the scores for text search and vector search are calculated differently, the nodes that were found only by text search will have a much lower score.

You can often improve hybrid search performance by using `QueryFusionRetriever`, which makes better use of the mutual information to rank the nodes.

```
from llama_index.core.response_synthesizers import CompactAndRefine
from llama_index.core.retrievers import QueryFusionRetriever
from llama_index.core.query_engine import RetrieverQueryEngine

vector_retriever = hybrid_index.as_retriever(
```

```
    vector_store_query_mode="default",
    similarity_top_k=5,
)
text_retriever = hybrid_index.as_retriever(
    vector_store_query_mode="sparse",
    similarity_top_k=5,  # interchangeable with sparse_top_k in this
context
)
retriever = QueryFusionRetriever(
    [vector_retriever, text_retriever],
    similarity_top_k=5,
    num_queries=1,  # set this to 1 to disable query generation
    mode="relative_score",
    use_async=False,
)

response_synthesizer = CompactAndRefine()
query_engine = RetrieverQueryEngine(
    retriever=retriever,
    response_synthesizer=response_synthesizer,
)
```

```
response = query_engine.query(
    "Who does Paul Graham think of with the word schtick, and why?"
)
print(response)
```

```
Paul Graham associates the word "schtick" with artists who have a dis
tinctive signature style in their work. This signature style serves a
s a visual identifier unique to the artist, making their work easily
recognizable and attributed to them.
```

Metadata filters

PGVectorStore supports storing metadata in nodes, and filtering based on that metadata during the retrieval step.

Download git commits dataset

```
!mkdir -p 'data/git_commits/'
!wget 'https://raw.githubusercontent.com/run-llama/llama_index/main/docs/docs/examples/data/csv/commit_history.csv' -O 'data/git_commits/commit_history.csv'
```

```
import csv

with open("data/git_commits/commit_history.csv", "r") as f:
    commits = list(csv.DictReader(f))

print(commits[0])
print(len(commits))
```

```
{'commit': '44e41c12ab25e36c202f58e068ced262eadc8d16', 'author': 'Lakshmi Narayanan Sreethar<lakshmi@timescale.com>', 'date': 'Tue Sep 5 21:03:21 2023 +0530', 'change summary': 'Fix segfault in set_integer_now_func', 'change details': 'When an invalid function oid is passed to set_integer_now_func, it finds out that the function oid is invalid but before throwing the error, it calls ReleaseSysCache on an invalid tuple causing a segfault. Fixed that by removing the invalid call to ReleaseSysCache.  Fixes #6037 '}
4167
```

Add nodes with custom metadata

```
# Create TextNode for each of the first 100 commits
from llama_index.core.schema import TextNode
from datetime import datetime
import re

nodes = []
dates = set()
authors = set()
for commit in commits[:100]:
    author_email = commit["author"].split("<")[1][:-1]
    commit_date = datetime.strptime(
        commit["date"], "%a %b %d %H:%M:%S %Y %z"
    ).strftime("%Y-%m-%d")
    commit_text = commit["change summary"]
    if commit["change details"]:
        commit_text += "\n\n" + commit["change details"]
    fixes = re.findall(r"#(\d+)", commit_text, re.IGNORECASE)
    nodes.append(
        TextNode(
```

```
            text=commit_text,
            metadata={
                "commit_date": commit_date,
                "author": author_email,
                "fixes": fixes,
            },
        )
    )
    dates.add(commit_date)
    authors.add(author_email)

print(nodes[0])
print(min(dates), "to", max(dates))
print(authors)
```

```
Node ID: 50fdca05-b1ce-41d8-b771-b13aa3ad7df0
Text: Fix segfault in set_integer_now_func  When an invalid function
oid is passed to set_integer_now_func, it finds out that the function
oid is invalid but before throwing the error, it calls ReleaseSysCach
e
on an invalid tuple causing a segfault. Fixed that by removing the
invalid call to ReleaseSysCache.  Fixes #6037
2023-03-22 to 2023-09-05
{'dmitry@timescale.com', 'jguthrie@timescale.com', 'lakshmi@timescal
e.com', 'rafia.sabih@gmail.com', 'sven@timescale.com', 'konstantina@t
imescale.com', 'engel@sero-systems.de', 'nikhil@timescale.com', '3688
2414+akuzm@users.noreply.github.com', 'mats@timescale.com', 'satish.8
483@gmail.com', 'fabriziomello@gmail.com', 'me@noctarius.com', 'erik@
timescale.com', 'jan@timescale.com'}
```

```
vector_store = PGVectorStore.from_params(
    database=db_name,
    host=url.host,
    password=url.password,
    port=url.port,
    user=url.username,
    table_name="metadata_filter_demo3",
    embed_dim=1536,  # openai embedding dimension
    hnsw_kwargs={
        "hnsw_m": 16,
        "hnsw_ef_construction": 64,
        "hnsw_ef_search": 40,
        "hnsw_dist_method": "vector_cosine_ops",
    },
)

index =
```

```
VectorStoreIndex.from_vector_store(vector_store=vector_store)
index.insert_nodes(nodes)
```

```
print(index.as_query_engine().query("How did Lakshmi fix the segfault?"))
```

```
Lakshmi fixed the segfault by removing the invalid call to ReleaseSysCache that was causing the issue.
```

Apply metadata filters

Now we can filter by commit author or by date when retrieving nodes.

```
from llama_index.core.vector_stores.types import (
    MetadataFilter,
    MetadataFilters,
)

filters = MetadataFilters(
    filters=[
        MetadataFilter(key="author", value="mats@timescale.com"),
        MetadataFilter(key="author", value="sven@timescale.com"),
    ],
    condition="or",
)

retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("What is this software project about?")

for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-07', 'author': 'mats@timescale.com', 'fixes': []}
{'commit_date': '2023-08-27', 'author': 'sven@timescale.com', 'fixes': []}
{'commit_date': '2023-07-13', 'author': 'mats@timescale.com', 'fixes': []}
{'commit_date': '2023-08-07', 'author': 'sven@timescale.com', 'fixes': []}
{'commit_date': '2023-08-30', 'author': 'sven@timescale.com', 'fixe
```

```
s': []}
{'commit_date': '2023-08-15', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-23', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-10', 'author': 'mats@timescale.com', 'fixe
s': []}
{'commit_date': '2023-07-25', 'author': 'mats@timescale.com', 'fixe
s': ['5892']}
{'commit_date': '2023-08-21', 'author': 'sven@timescale.com', 'fixe
s': []}
```

```python
filters = MetadataFilters(
    filters=[
        MetadataFilter(key="commit_date", value="2023-08-15",
operator=">="),
        MetadataFilter(key="commit_date", value="2023-08-25",
operator="<="),
    ],
    condition="and",
)

retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("What is this software project
about?")

for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-23', 'author': 'erik@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-17', 'author': 'konstantina@timescale.com',
'fixes': []}
{'commit_date': '2023-08-15', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
{'commit_date': '2023-08-15', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
{'commit_date': '2023-08-24', 'author': 'lakshmi@timescale.com', 'fix
es': []}
{'commit_date': '2023-08-15', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-23', 'author': 'sven@timescale.com', 'fixe
s': []}
```

```
{'commit_date': '2023-08-21', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-20', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-21', 'author': 'sven@timescale.com', 'fixe
s': []}
```

Apply nested filters

In the above examples, we combined multiple filters using AND or OR. We can also combine multiple sets of filters.

e.g. in SQL:

```
WHERE (commit_date >= '2023-08-01' AND commit_date <= '2023-08-
15') AND (author = 'mats@timescale.com' OR author =
'sven@timescale.com')
```

```
filters = MetadataFilters(
    filters=[
        MetadataFilters(
            filters=[
                MetadataFilter(
                    key="commit_date", value="2023-08-01",
operator=">="
                ),
                MetadataFilter(
                    key="commit_date", value="2023-08-15",
operator="<="
                ),
            ],
            condition="and",
        ),
        MetadataFilters(
            filters=[
                MetadataFilter(key="author",
value="mats@timescale.com"),
                MetadataFilter(key="author",
value="sven@timescale.com"),
            ],
            condition="or",
        ),
    ],
    condition="and",
)
```

```
retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("What is this software project
about?")

for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-07', 'author': 'mats@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-07', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-15', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-10', 'author': 'mats@timescale.com', 'fixe
s': []}
```

The above can be simplified by using the IN operator. `PGVectorStore` supports `in`, `nin`, and `contains` for comparing an element with a list.

```
filters = MetadataFilters(
    filters=[
        MetadataFilter(key="commit_date", value="2023-08-01",
operator=">="),
        MetadataFilter(key="commit_date", value="2023-08-15",
operator="<="),
        MetadataFilter(
            key="author",
            value=["mats@timescale.com", "sven@timescale.com"],
            operator="in",
        ),
    ],
    condition="and",
)

retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("What is this software project
about?")
```

```
for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-07', 'author': 'mats@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-07', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-15', 'author': 'sven@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-10', 'author': 'mats@timescale.com', 'fixe
s': []}
```

```
# Same thing, with NOT IN
filters = MetadataFilters(
    filters=[
        MetadataFilter(key="commit_date", value="2023-08-01",
operator=">="),
        MetadataFilter(key="commit_date", value="2023-08-15",
operator="<="),
        MetadataFilter(
            key="author",
            value=["mats@timescale.com", "sven@timescale.com"],
            operator="nin",
        ),
    ],
    condition="and",
)

retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("What is this software project
about?")

for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-09', 'author': 'me@noctarius.com', 'fixes':
['5805']}
{'commit_date': '2023-08-15', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
{'commit_date': '2023-08-15', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
{'commit_date': '2023-08-11', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
{'commit_date': '2023-08-03', 'author': 'dmitry@timescale.com', 'fixe
```

```
s': []}
{'commit_date': '2023-08-09', 'author': 'konstantina@timescale.com',
'fixes': ['5923', '5680', '5774', '5786', '5906', '5912']}
{'commit_date': '2023-08-03', 'author': 'dmitry@timescale.com', 'fixe
s': ['5908']}
{'commit_date': '2023-08-01', 'author': 'nikhil@timescale.com', 'fixe
s': []}
{'commit_date': '2023-08-10', 'author': 'konstantina@timescale.com',
'fixes': []}
{'commit_date': '2023-08-10', 'author': '36882414+akuzm@users.norepl
y.github.com', 'fixes': []}
```

```python
# CONTAINS
filters = MetadataFilters(
    filters=[
        MetadataFilter(key="fixes", value="5680",
operator="contains"),
    ]
)

retriever = index.as_retriever(
    similarity_top_k=10,
    filters=filters,
)

retrieved_nodes = retriever.retrieve("How did these commits fix the
issue?")
for node in retrieved_nodes:
    print(node.node.metadata)
```

```
{'commit_date': '2023-08-09', 'author': 'konstantina@timescale.com',
'fixes': ['5923', '5680', '5774', '5786', '5906', '5912']}
```

PgVector Query Options

IVFFlat Probes

Specify the number of IVFFlat probes (1 by default)

When retrieving from the index, you can specify an appropriate number of IVFFlat probes (higher is better for recall, lower is better for speed)

```python
retriever = index.as_retriever(
    vector_store_query_mode="hybrid",
```

```
    similarity_top_k=5,
    vector_store_kwargs={"ivfflat_probes": 10},
)
```

HNSW EF Search

Specify the size of the dynamic candidate list for search (40 by default)

```
retriever = index.as_retriever(
    vector_store_query_mode="hybrid",
    similarity_top_k=5,
    vector_store_kwargs={"hnsw_ef_search": 300},
)
```