云数据库 PostgreSQL

AI 实践

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

AI 实践
使用 tencentdb_ai 插件构建 AI 应用

最近更新时间：2025-02-19 18:48:12

本文需要添加 `tencentdb_ai` 插件，该插件的使用请参考 使用 tencentdb_ai 插件调用大模型 。

集成 DeepSeek-V3 实现聊天

1. 创建 tencentdb_ai 插件。

```
damoxing=> CREATE EXTENSION tencentdb_ai CASCADE;
NOTICE:  installing required extension "pgcrypto"
CREATE EXTENSION
damoxing=> SELECT * FROM pg_extension;
  oid  |   extname    | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+--------------+----------+--------------+----------------+----
--------+-----------+--------------
 14275 | plpgsql      |       10 |           11 | f              | 1.0
|           |
 16631 | pgcrypto     |    16615 |         2200 | t              | 1.3
|           |
 16668 | tencentdb_ai |    16615 |         2200 | t              | 1.0
| {16670}   | {""}
(3 rows)
```

2. 添加 `DeepSeek-V3` 模型。

```
damoxing=> SELECT tencentdb_ai.add_model('deepseek-v3', '2024-05-22',
'ap-guangzhou', '$.Response.Choices[*].Message.Content');
 add_model
-----------

(1 row)

damoxing=> SELECT tencentdb_ai.update_model_attr('deepseek-v3',
'SecretId', 'AKID**************');
 update_model_attr
-------------------
```

```
(1 row)

damoxing=> SELECT tencentdb_ai.update_model_attr('deepseek-v3',
'SecretKey', '********************');
 update_model_attr
-------------------

(1 row)
```

3. 准备云数据库 `PostgreSQL` 对象。

```
damoxing=> CREATE TABLE chat_logs (
    id SERIAL PRIMARY KEY,
    user_input TEXT,
    system_response TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE
damoxing=>
```

4. 调用大模型聊天。

   创建存储过程调用 `deepseek-v3` 的 API 接口聊天：

```
damoxing=> CREATE OR REPLACE FUNCTION chat_with_deepseek(user_input
TEXT)
RETURNS TEXT AS $$
DECLARE
    system_response TEXT;
BEGIN
        SELECT tencentdb_ai.chat_completions('deepseek-v3',
user_input) INTO system_response;
    INSERT INTO chat_logs (user_input, system_response)
    VALUES (user_input, system_response);

    RETURN system_response;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
damoxing=>
```

调用存储过程聊天：

```
damoxing=> SELECT chat_with_deepseek('hi, deepseek');

chat_with_deepseek

--------------------------------------------------------------------------
-------------------------------------------------------------------
--------------------------------------------------------------------------
-----------
 "Hello! It seems like you're trying to reach DeepSeek, but I’m not
sure what you’re looking for. Could you clarify? Are you referring t
o a specific tool, service, or something else? Let me know how I can
assist!"
(1 row)

damoxing=>
```

5. **查询聊天结果。**

```
damoxing=> SELECT * FROM chat_logs ORDER BY timestamp DESC;
 id |  user_input  |
system_response

|         timestamp
----+--------------+-----------------------------------------------------
-------------------------------------------------------------------
--------------------------------------------------------------------------
-------------------------------+----------------------------
  3 | hi, deepseek | "Hello! It seems like you're trying to reach
DeepSeek, but I’m not sure what you’re looking for. Could you clarify?
 Are you referring to a specific tool, service, or something else? Let
me know how I can assist!" | 2025-02-18 16:02:39.145322
(1 row)

damoxing=>
```

集成 lke-text-embedding-v1 实现文本检索

1. **创建** `tencentdb_ai` **插件。**

```
damoxing_lke_embdding=> CREATE EXTENSION tencentdb_ai CASCADE;
```

```
NOTICE:  installing required extension "pgcrypto"
CREATE EXTENSION
damoxing_lke_embdding=> SELECT * FROM pg_extension;
  oid  |   extname    | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+--------------+----------+--------------+----------------+----
--------+-----------+--------------
 14275 | plpgsql      |       10 |           11 | f              | 1.0
|           |
 16631 | pgcrypto     |    16615 |         2200 | t              | 1.3
|           |
 16668 | tencentdb_ai |    16615 |         2200 | t              | 1.0
| {16670}   | {""}
(3 rows)
```

2. 添加 `lke-text-embedding-v1` 模型。

```
damoxing_lke_embdding=> SELECT tencentdb_ai.add_model('lke-text-
embedding-v1', '2024-05-22', 'ap-guangzhou', NULL);
-[ RECORD 1 ]
add_model |

damoxing_lke_embdding=> SELECT tencentdb_ai.update_model_attr('lke-
text-embedding-v1', 'SecretId', 'AKID**************');
-[ RECORD 1 ]-----+-
update_model_attr |

damoxing_lke_embdding=> SELECT tencentdb_ai.update_model_attr('lke-
text-embedding-v1', 'SecretKey', '********************');
-[ RECORD 1 ]-----+-
update_model_attr |

damoxing_lke_embdding=>
```

3. 准备云数据库 PostgreSQL 对象。

```
damoxing_lke_embdding=> CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    content TEXT,
    embedding FLOAT8[]
);
CREATE TABLE
```

```
damoxing_lke_embdding=>
```

4. 创建生成嵌入向量的存储过程。

```
damoxing_lke_embdding=> CREATE OR REPLACE FUNCTION
generate_embedding(doc_contents TEXT[])
RETURNS FLOAT8[][] AS $$
DECLARE
    embeddings FLOAT8[][] := '{}';
    embedding FLOAT8[];
    doc_content TEXT;
BEGIN
        SELECT * from tencentdb_ai.get_embedding('lke-text-embedding-
v1', doc_contents) INTO embedding;
    RETURN embedding;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
damoxing_lke_embdding=>
```

5. 创建插入文档及其嵌入向量的存储过程。

```
damoxing_lke_embdding=> CREATE OR REPLACE FUNCTION
insert_documents(doc_contents TEXT[])
RETURNS VOID AS $$
DECLARE
    embedding FLOAT8[];
    doc_content TEXT;
BEGIN
        doc_content := doc_contents[0];
        embedding :=  generate_embedding(doc_contents);
        INSERT INTO documents (content, embedding)
        VALUES (doc_contents, embedding);
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
damoxing_lke_embdding=>
```

6. 创建余弦相似度计算的存储过程。

```
damoxing_lke_embdding=> CREATE OR REPLACE FUNCTION
cosine_similarity(vec1 FLOAT8[], vec2 FLOAT8[])
RETURNS FLOAT8 AS $$
DECLARE
    dot_product FLOAT8 := 0;
    norm1 FLOAT8 := 0;
    norm2 FLOAT8 := 0;
    similarity FLOAT8;
BEGIN
    FOR i IN 1..array_length(vec1, 1) LOOP
        dot_product := dot_product + vec1[i] * vec2[i];
        norm1 := norm1 + vec1[i] * vec1[i];
        norm2 := norm2 + vec2[i] * vec2[i];
    END LOOP;

    similarity := dot_product / (sqrt(norm1) * sqrt(norm2));
    RETURN similarity;
END;
$$ LANGUAGE plpgsql;
damoxing_lke_embdding=>
```

7. 创建文本检索的存储过程。

```
damoxing_lke_embdding=> CREATE OR REPLACE FUNCTION
search_similar_documents(query TEXT, limit_count INT)
RETURNS TABLE(id INT, content TEXT, similarity FLOAT8) AS $$
DECLARE
    query_embedding FLOAT8[];
BEGIN
    query_embedding := generate_embedding(ARRAY[query]);

    RETURN QUERY
    SELECT d.id, d.content, cosine_similarity(d.embedding,
query_embedding) AS similarity
    FROM documents d
    ORDER BY similarity DESC
    LIMIT limit_count;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
damoxing_lke_embdding=>
```

8. 插入文档及其嵌入向量。

```
damoxing_lke_embdding=> SELECT insert_documents(ARRAY['This is a
sample document.']);
-[ RECORD 1 ]----+-
insert_documents |

damoxing_lke_embdding=> SELECT insert_documents(ARRAY['Another example
of a document.']);
-[ RECORD 1 ]----+-
insert_documents |

damoxing_lke_embdding=> SELECT insert_documents(ARRAY['This document
is about PostgreSQL and text embeddings.']);
-[ RECORD 1 ]----+-
insert_documents |

damoxing_lke_embdding=> SELECT insert_documents(ARRAY['Deep learning
models can generate text embeddings.']);
-[ RECORD 1 ]----+-
insert_documents |

damoxing_lke_embdding=>
```

9. 进行文本检索。

```
damoxing_lke_embdding=> SELECT * FROM
search_similar_documents('PostgreSQL text embeddings', 5);
-[ RECORD 1 ]----------------------------------------------------------
-
id         | 3
content    | {"This document is about PostgreSQL and text
embeddings."}
similarity | 0.8649945496222797
-[ RECORD 2 ]----------------------------------------------------------
-
id         | 4
content    | {"Deep learning models can generate text embeddings."}
similarity | 0.6824638514797066
-[ RECORD 3 ]----------------------------------------------------------
-
id         | 1
```

```
content    | {"This is a sample document."}
similarity | 0.66412244051794
-[ RECORD 4 ]----------------------------------------------------------
-
id         | 2
content    | {"Another example of a document."}
similarity | 0.6142928906219256

damoxing_lke_embdding=>
```

集成 lke-reranker-base 实现 RAG

1. 创建 tencentdb_ai 插件。

```
damoxing_rerank=> CREATE EXTENSION tencentdb_ai CASCADE;
NOTICE:  installing required extension "pgcrypto"
CREATE EXTENSION
damoxing=> SELECT * FROM pg_extension;
  oid  |   extname    | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+--------------+----------+--------------+----------------+----
--------+-----------+--------------
 14275 | plpgsql      |       10 |           11 | f              | 1.0
|           |
 16631 | pgcrypto     |    16615 |         2200 | t              | 1.3
|           |
 16668 | tencentdb_ai |    16615 |         2200 | t              | 1.0
| {16670}   | {""}
(3 rows)
```

2. 添加 lke-reranker-base 和 lke-text-embedding-v1 模型。

```
damoxing_rerank=> SELECT tencentdb_ai.add_model('lke-reranker-base',
'2024-05-22', 'ap-guangzhou', '$.Response.ScoreList');
 add_model
-----------

(1 row)

damoxing_rerank=> SELECT tencentdb_ai.update_model_attr('lke-reranker-
base', 'SecretId', 'AKID**************');
 update_model_attr
```

```
------------------

(1 row)

damoxing_rerank=> SELECT tencentdb_ai.update_model_attr('lke-reranker-
base', 'SecretKey', '********************');
 update_model_attr
------------------

(1 row)

damoxing_rerank=> SELECT tencentdb_ai.add_model('lke-text-embedding-
v1', '2024-05-22', 'ap-guangzhou', NULL);
 add_model
-----------

(1 row)

damoxing_rerank=> SELECT tencentdb_ai.update_model_attr('lke-text-
embedding-v1', 'SecretId', 'AKID**************');
 update_model_attr
------------------

(1 row)

damoxing_rerank=> SELECT tencentdb_ai.update_model_attr('lke-text-
embedding-v1', 'SecretKey', '********************');
 update_model_attr
------------------

(1 row)
```

3. 准备云数据库 PostgreSQL 对象。

```
damoxing_rerank=> CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    content TEXT,
    embedding FLOAT8[]
);
CREATE TABLE
damoxing_rerank=>
```

4. 创建生成嵌入向量的存储过程。

```
damoxing_rerank=> CREATE OR REPLACE FUNCTION
generate_embedding(doc_contents TEXT[])
RETURNS FLOAT8[][] AS $$
DECLARE
    embeddings FLOAT8[][] := '{}';
    embedding FLOAT8[];
    doc_content TEXT;
BEGIN
        SELECT * from tencentdb_ai.get_embedding('lke-text-embedding-
v1', doc_contents) INTO embedding;
    RETURN embedding;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
```

5. 创建插入文档及其嵌入向量的存储过程。

```
damoxing_rerank=> CREATE OR REPLACE FUNCTION
insert_documents(doc_contents TEXT[])
RETURNS VOID AS $$
DECLARE
    embedding FLOAT8[];
    doc_content TEXT;
BEGIN
        doc_content := doc_contents[0];
        embedding :=  generate_embedding(doc_contents);
        INSERT INTO documents (content, embedding)
        VALUES (doc_contents, embedding);
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
```

6. 创建召回和大模型重排的存储过程。

```
damoxing_rerank=> CREATE OR REPLACE FUNCTION retrieve_and_rerank(query
TEXT)
RETURNS TABLE (id INT, content TEXT, score FLOAT8) AS $$
DECLARE
    keyword_results RECORD;
    vector_results RECORD;
```

```
    query_vector VECTOR;
    rerank_content TEXT[];
    rerank_ids INT[];
    rerank_scores my_record;
    i INT;
BEGIN
    -- 生成查询向量
    SELECT generate_embedding(ARRAY[query])::vector INTO query_vector;

    -- 初始化重排序内容数组
    rerank_content := ARRAY[]::TEXT[];
    rerank_ids := ARRAY[]::INT[];

    -- 关键词召回
    FOR keyword_results IN
        SELECT d.id, d.content, d.embedding
        FROM documents d
        WHERE to_tsvector('english', d.content) @@
plainto_tsquery('english', query)
    LOOP
        -- 向量召回
        FOR vector_results IN
            SELECT d.id, d.content, d.embedding
            FROM documents d
            WHERE d.id = keyword_results.id
            ORDER BY d.embedding <-> query_vector
            LIMIT 10
        LOOP
            rerank_content := array_append(rerank_content,
vector_results.content);
            rerank_ids := array_append(rerank_ids, vector_results.id);
        END LOOP;
    END LOOP;

    -- 大模型重排序

    SELECT *  FROM tencentdb_ai.run_rerank('lke-reranker-base', query,
rerank_content) INTO rerank_scores;
    RETURN QUERY
    select text_field, numeric_field from rerank_scores;
END;
$$ LANGUAGE plpgsql;
```

7. 调用存储过程。

```
damoxing_rerank=>SELECT insert_documents(ARRAY['This is the first
document.']);
-[ RECORD 1 ]----+-
insert_documents |
damoxing_rerank=> SELECT * FROM retrieve_and_rerank('document');
("{""This is the first document.""}",-1.5107422)
```

使用 tencentdb_ai 插件调用大模型

最近更新时间：2025-02-19 18:48:12

当前无论是基础大模型还是 AI 应用开发都发展迅速，云数据库 PostgreSQL 提供了 tencentdb_ai 插件，方便您在云数据库 PostgreSQL 实例中轻松调用网络可通的大模型 API，完成各种场景的应用开发。本插件支持大版本为 PostgreSQL 17 的实例。

> **说明：**
> - 使用 `tencentdb_ai` 的前提是您已经创建完成了大版本为 PostgreSQL 17 的实例。
> - 创建 `tencentdb_ai` 插件会关联创建 `pgcrypto` 插件，请知悉。

tencentdb_ai 插件函数说明

新增模型

`add_model` 函数定义如下：

```
tencentdb_ai.add_model(
    model_name NAME,
    version TEXT,
    region TEXT,
    json_path JSONPATH
);
```

参数解释如下：

- model_name：模型名称，如 `hunyuan-lite`，`deepseek-r1` 等。
- version：该模型的公共参数 `version`，如该模型不需要 `version` 参数则可以不填写。
- region：该模型的公共参数 `region`，如该模型不需要 `region` 参数则可以不填写。
- json_path：为调用该模型时，指定如何对模型的 `JSON` 响应进行提取。相当于对模型输出调用 `jsonb_path_query_first` 函数。如果不指定该参数，默认原样输出模型的原始响应。

查看已注册模型

`list_models` 函数定义如下：

```
tencentdb_ai.list_models(void);
```

更新模型

`update_model_attr` 函数定义如下：

```
tencentdb_ai.update_model_attr(
    model_name NAME,
    attr_name text,
    attr_value text
);
```

参数解释如下：

- model_name：模型名称。
- attr_name：需要更新的模型属性。可更新的模型属性有 `version`、`region`、`json_path`、`SecretId`、`SecretKey`。
- attr_value：需要更新的属性的值。

具体 SecretId 和 SecretKey 请查看 访问管理 。

删除模型

`delete_model` 函数定义如下：

```
tencentdb_ai.delete_model(
    model_name NAME
);
```

参数解释如下：

- model_name：模型名称。

模型调用

call_model 函数定义如下：

```
tencentdb_ai.call_model(
    model_name NAME,
    common_params TEXT[],
    api_params TEXT[]
);
```

参数解释如下：

- model_name：模型名称。
- common_params：需要调用模型的公共参数，例如 `ARRAY['Action: ChatCompletions', 'Version: 2023-09-01']`

- api_params：需要调用模型的专有参数，例如

```
ARRAY['"Stream": false', '"Model": "hunyuan-lite"', '"Messages": [{"Role": "user", "Content": "您好"}]']
```

除了使用 `call_model` 函数来进行模型调用之外，您还可以使用如下几个接口来进行特定场景的模型调用。

固定场景接口

对话

`chat_completions` 函数定义如下：

```
tencentdb_ai.chat_completions(
    model_name  NAME,
    content TEXT,
    args TEXT[] default NULL
);
```

参数解释如下：

- model_name：模型名称。
- content：对话内容。
- args：调用模型的其他参数，默认为 NULL。

调用示例：

```
SELECT tencentdb_ai.chat_completions('hunyuan-lite', '您好');

         chat_completions

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
----------------------------------
 {"Response":{"RequestId":"*******-****-****-****-************","Note":"以上内容为AI生成，不代表开发者立场，请勿删除或修改本标记","Choices":[{"Index":0,"Message":{"Role":"assistant","Content":"您好呀！很高兴能和您聊天，今天有什么想和我分享的吗"},"FinishReason":"stop"}],"Created":1739786393,"Id":"*******-****-****-****-*********","Usage":{"PromptTokens":3,"CompletionTokens":17,"TotalTokens":20}}}
```

```
(1 row)
```

文本转向量

`get_embedding` 函数定义如下：

```
tencentdb_ai.get_embedding(
    model_name NAME,
    content TEXT[]
);
```

参数解释如下：

- model_name：模型名称。
- content：转化内容。

调用示例：

```
damoxing1=> SELECT tencentdb_ai.get_embedding('hunyuan-
embedding',ARRAY['您好', 'PostgreSQL']);
```

文本排序

`run_rerank` 函数定义如下：

```
tencentdb_ai.run_rerank(
    model_name NAME,
    query TEXT,
    documents TEXT[],
    args TEXT[] DEFAULT NULL
);
```

参数解释如下：

- model_name：模型名称。
- query：文本排序的问题。
- documents：待文本排序的内容。
- args：特定模型的指定参数。

```
damoxing1=> SELECT COUNT(*) FROM tencentdb_ai.run_rerank('lke-reranker-
base', '知识引擎大模型', ARRAY['混元大模型', '腾讯知识引擎']);
 count
-------
```

```
     2
(1 row)
```

插件使用示例

下面为您一一描述插件使用过程：

1. 创建插件

```
damoxing=> CREATE EXTENSION tencentdb_ai CASCADE;
NOTICE:  installing required extension "pgcrypto"
CREATE EXTENSION
```

检查插件创建完成

```
damoxing=> SELECT * FROM pg_extension;
  oid  |   extname    | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+--------------+----------+--------------+----------------+----
--------+-----------+--------------
 14275 | plpgsql      |       10 |           11 | f              | 1.0
|           |
 16440 | pgcrypto     |    16437 |         2200 | t              | 1.3
|           |
 16477 | tencentdb_ai |    16437 |         2200 | t              | 1.0
| {16479}   | {""}
(3 rows)
```

2. 添加大模型

```
damoxing=> SELECT tencentdb_ai.add_model('hunyuan-lite','2023-09-01',
NULL, NULL);
 add_model
-----------

(1 row)
```

3. 添加大模型 API 调用的 SecretId 和 SecretKey

具体 SecretId 和 SecretKey 请查看 访问管理 。

```
damoxing=> SELECT tencentdb_ai.update_model_attr('hunyuan-lite',
'SecretId', 'AKID***************');
 update_model_attr
-------------------

(1 row)
damoxing=> SELECT tencentdb_ai.update_model_attr('hunyuan-lite',
'SecretKey', '********************');
 update_model_attr
-------------------

(1 row)
```

4. 查看当前添加的模型列表

```
damoxing=> SELECT * FROM tencentdb_ai.list_models();
-[ RECORD 1 ]-------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
---------------------
model_name | hunyuan-lite
json_path  |
secretid   | ***************************
secretkey  | ***************************
version    |
region     |
id_random  | 516234453022
key_random | 134577899689
```

5. 调用大模型

聊天场景下，您可以使用 `call_model` 接口。

```
damoxing=> SELECT tencentdb_ai.call_model('hunyuan-lite',
                          ARRAY['Action: ChatCompletions', 'Version:
2023-09-01'],
                          ARRAY['"Stream": false', '"Model":
"hunyuan-lite"', '"Messages": [{"Role": "user", "Content": "您
好"}]']);damoxing(> damoxing(>


call_model
```

```
----------------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------------------------------
--
 {"Response":{"RequestId":"*******-****-****-****-
***********","Note":"以上内容为AI生成，不代表开发者立场，请勿删除或修改本标
记","Ch
oices":[{"Index":0,"Message":{"Role":"assistant","Content":"您好！很高兴
与您交流。请问有什么我可以帮助您的吗？无论是关于生活、工作、学
习还是其他方面的问题，我都会尽力为您提供帮
助。"},"FinishReason":"stop"}],"Created":1739793838,"Id":"*******-
****-****-****-***********
","Usage":{"PromptTokens":3,"CompletionTokens":33,"TotalTokens":36}}}
(1 row)

damoxing=>
```

结合 Supabase 快速构建基于云数据库 PostgreSQL 的后端服务

最近更新时间：2025-08-21 15:51:11

Supabase 提供基于 PostgreSQL 的全栈开发平台，集成实时数据库、身份认证、自动生成 API、对象存储和边缘函数，具备开箱即用的实时数据同步、细粒度权限控制及 Serverless 扩展能力，支持开发者快速构建 AI 集成的现代化应用，同时保持对 SQL 和开源生态的完全掌控。更详细的 Supabase 功能请参考 官方文档 。
本文将指引您快速部署 Supabase，并配置连接云数据库 PostgreSQL 作为后端数据库，快速搭建应用。

环境准备

您需要先创建同一 VPC 网络下的 云服务器实例 和 云数据库 PostgreSQL 实例 。

拉取 Supabase 代码

```
# Get the code
git clone --depth 1 https://github.com/supabase/supabase

# Make your new supabase project directory
mkdir supabase-project

# Tree should look like this
# .
# ├── supabase
# └── supabase-project

# Copy the compose files over to your project
cp -rf supabase/docker/* supabase-project

# Copy the fake env vars
cp supabase/docker/.env.example supabase-project/.env

# Switch to your project directory
cd supabase-project

# Pull the latest images
docker compose pull
```

修改 .env 文件

修改 `supabase-project/.env` 文件的如下几项，请确保云数据库 PostgreSQL 实例中已创建对应的账号。

```
POSTGRES_HOST=云数据库 PostgreSQL 实例 vip
POSTGRES_DB=postgres
POSTGRES_USER=supabase_admin
POSTGRES_PORT=5432
POSTGRES_PASSWORD=云数据库 PostgreSQL 实例账号密码
```

修改 docker-compose.yml 文件

修改 `supabase-project/docker-compose.yml` 文件，搜索 `external Postgres database` 关键字，文件中有提示，如果使用外部 PG，将对应部分注释掉：

```
depends_on:
      # db:
        # Disable this if you are using an external Postgres database
        # condition: service_healthy
```

在云数据库 PostgreSQL 中创建对象

在拉起 supabase 之前，我们需要创建一些对象（user、database、schema），否则会导致 supabase 无法拉起。此处的密码应该与POSTGRES_PASSWORD 配置的密码一致。

```
CREATE USER supabase_admin pg_tencentdb_superuser password '你的密码';
CREATE DATABASE _supabase;
create schema _analytics;
```

清除缓存

如果之前未启动过 Supabase，无需执行此步骤。

```
# Stop docker and remove volumes:
docker compose down -v
# Remove Postgres data:
rm -rf volumes/db/data/
```

```
[root@VM-0-13-centos supabase-project]# docker compose down -v
[+] Running 15/15
 ✔ Container supabase-auth                  Removed                                0.1s
 ✔ Container supabase-meta                  Removed                               10.2s
 ✔ Container supabase-storage               Removed                                0.1s
 ✔ Container supabase-pooler                Removed                                0.1s
 ✔ Container realtime-dev.supabase-realtime Remove...                              0.1s
 ✔ Container supabase-studio                Removed                                0.3s
 ✔ Container supabase-kong                  Removed                                0.3s
 ✔ Container supabase-edge-functions        Removed                               10.2s
 ✔ Container supabase-rest                  Removed                                0.1s
 ✔ Container supabase-imgproxy              Removed                                0.2s
 ✔ Container supabase-analytics             Removed                               10.3s
 ✔ Container supabase-db                    Removed                                1.2s
 ✔ Container supabase-vector                Removed                               10.1s
 ✔ Volume supabase_db-config                Removed                                0.0s
 ✔ Network supabase_default                 Removed                                0.1s
[root@VM-0-13-centos supabase-project]# rm -rf volumes/db/data/
```

启动 Supabase

上述配置完成后，即可启动 Supabase。

```
docker compose up -d
```

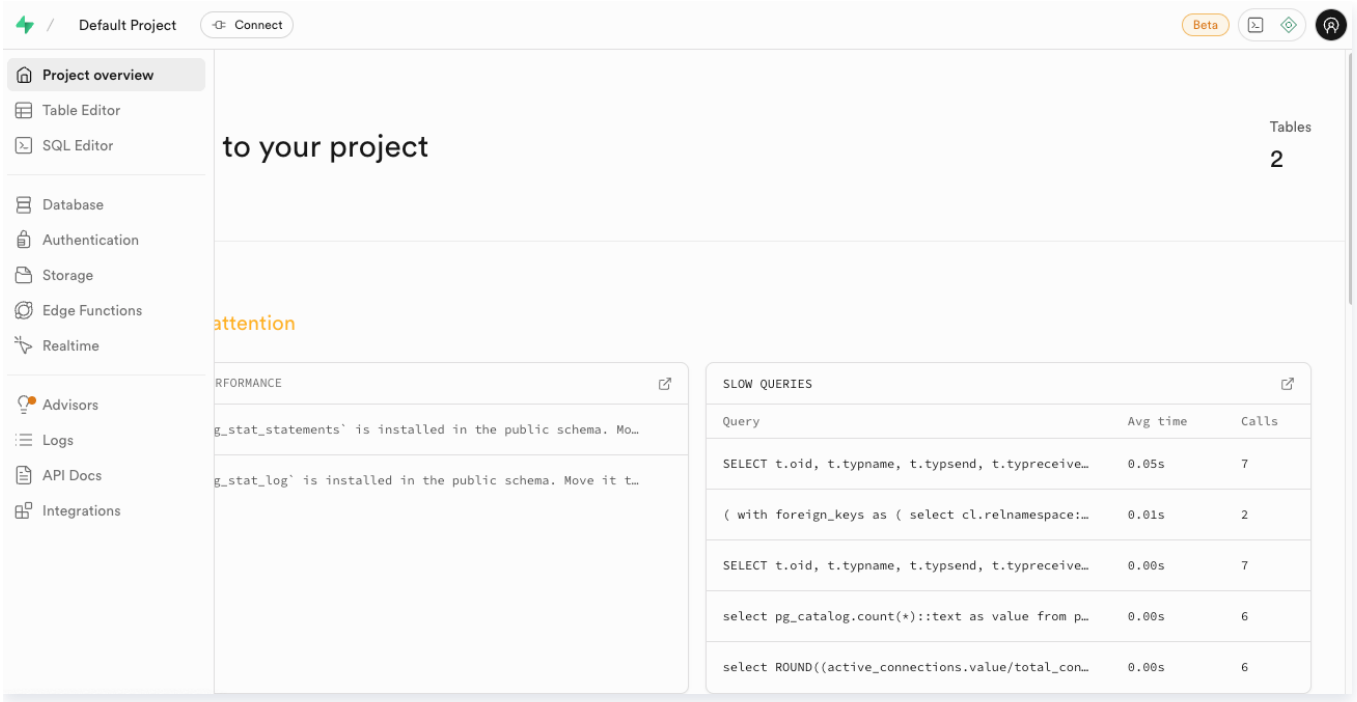
```
[root@VM-0-13-centos supabase-project]# docker compose up -d
[+] Running 15/15
 ✔ Network supabase_default                 Created                                0.1s
 ✔ Volume "supabase_db-config"              Created                                0.0s
 ✔ Container supabase-vector                Healthy                                6.1s
 ✔ Container supabase-imgproxy              Started                                0.6s
 ✔ Container supabase-db                    Healthy                               18.4s
 ✔ Container supabase-analytics             Healthy                               17.3s
 ✔ Container supabase-pooler                Started                               18.0s
 ✔ Container supabase-kong                  Started                               18.3s
 ✔ Container supabase-auth                  Started                               17.9s
 ✔ Container supabase-studio                Started                               18.4s
 ✔ Container supabase-edge-functions        Started                               18.3s
 ✔ Container supabase-meta                  Started                               18.1s
 ✔ Container supabase-rest                  Started                               17.8s
 ✔ Container realtime-dev.supabase-realtime Starte...                             18.1s
 ✔ Container supabase-storage               Started                               18.8s
[root@VM-0-13-centos supabase-project]#
```

登录 Supabase 界面

访问虚拟机实例外网 IP 对应的如下链接：http://虚拟机实例外网 IP :8000/，Supabase 提供的初始化账号密码为：

- supabase
- this_password_is_insecure_and_should_be_updated

成功登录后的界面：


Default Project
Connect
Beta
Project overview
Table Editor
SQL Editor
Database
Authentication
Storage
Edge Functions
Realtime
Advisors
Logs
API Docs
Integrations
to your project
Tables
2
attention
RFORMANCE
g_stat_statements` is installed in the public schema. Mo…
g_stat_log` is installed in the public schema. Move it t…
SLOW QUERIES
Query
Avg time
Calls
SELECT t.oid, t.typname, t.typsend, t.typreceive… 0.05s 7
(with foreign_keys as (select cl.relnamespace:… 0.01s 2
SELECT t.oid, t.typname, t.typsend, t.typreceive… 0.00s 7
select pg_catalog.count(*)::text as value from p… 0.00s 6
select ROUND((active_connections.value/total_con… 0.00s 6