Asyncpg

документация

Пример:

connectors/postgres/crud.py

```
import asyncio
import datetime
import os
from typing import Optional

import asyncpg


async def get_connection():
    return await asyncpg.connect(
        host=os.getenv("POSTGRES_HOST"),
        port=os.getenv("POSTGRES_PORT"),
        user=os.getenv("POSTGRES_USER"),
        password=os.getenv("POSTGRES_PASSWORD"),
        database=os.getenv("POSTGRES_DATABASE")
    )


async def get_connection_by_dsn():
    """
    POSTGRES_DSN = postgres://user:password@host:port/database
    """
    return await asyncpg.connect(os.getenv("POSTGRES_DSN"))


async def insert(conn) -> asyncpg.Record:
    res = await conn.fetchrow(
        "insert into users (first_name, last_name, is_tutor, created) values ($1, $2, $3, $4) returning *",
        'Alexander', 'Opryshko', True, datetime.datetime.now()
    )
    return res


async def select(conn, user_id) -> list[asyncpg.Record]:
    res = await conn.fetch("select * from users where id = $1", user_id)
    return res


async def update(conn, user_id) -> Optional[asyncpg.Record]:
    return await conn.fetchrow("update users set is_tutor = false where id = $1 returning *", user_id)


async def delete(conn, user_id) -> None:
    res = await conn.execute("delete from users where id = $1", user_id)
    print(type(res), res)


async def run():
    conn = await get_connection_by_dsn()
    res = await insert(conn)
    print('insert', res)

    user_id = res['id']
    res = await select(conn, user_id)
    print('select', res)

    res = await update(conn, user_id)
    print('update', res)

    await delete(conn, user_id)
    res = await select(conn, user_id)
    print('after delete', res)


asyncio.run(run())
```

Запустить пример можно в mercury:

```
python3 connectors/postgres/crud.py
```

Что делает этот код

Для выполнения запросов нужно создать подключение к базе данных. Сделать это можно с помощью функций *get_connection* и *get_connection_by_dsn*. Есть 2 способа получить объект подключения:

- указать параметры host, port, user, password, database
- указать dsn: postgres://<user>:<password>@<host>:<port>/<database>. По сути это то же самое, что и предыдущий способ, только запись более компактная

Подключение происходит в момет вызова *asyncpg.connect.* Поэтому, если параметры указаны неправильно, то будет вызвано исключение о том, что подключиться не удалось.

В функциях insert, select, update, delete используются методы подключения fetchrow, fetch, execute. Давайте рассмотрим их подробнее.

fetchrow. Выполнит sql-запрос и вернет только одну запись — даже если запрос вернул больше. Если запрос вернул запись, метод возвращает объект Record, если нет — объект None.

```
<Record id=12 first_name='Alexander' last_name='Opryshko' is_tutor=False created=datetime.datetime(2021, 11, 2, 21, 29, 4, 785353)>
```

fetch. Выполнит sql-запрос и вернет все записи, которые вернул запрос, даже если там были миллионы записей. Если запрос не вернул никакой записи, метод возвращает список объектов Record или пустой список.

execute. Выполнит sql-запрос и вернет строку, которую Postgres пишет после выполнения запроса.

```
# после выполнения
res = await conn.execute("delete from users where id = $1", user_id)
print(type(res), res)
# будет выведено
<class 'str'> DELETE 1
```

Общие свойства функций fetchrow, fetch, execute

Функции могут принимать переменные. Перечисленные аргументы после запроса попадут в запрос вместо $1, $2 и далее:

```
# запрос
res = await conn.fetchrow(
    "insert into users (first_name, last_name, is_tutor, created) values ($1, $2, $3, $4) returning *",
    'Alexander', 'Opryshko', True, datetime.datetime.now()
)
# превратится в
res = await conn.fetchrow("""
    insert into users (first_name, last_name, is_tutor, created)
    values ('Alexander', 'Opryshko', true, '2021-11-02 21:29:04.785353') returning *
""")
```

Важное примечание

НИКОГДА не форматируйте строки руками!
ВСЕГДА пользуйтесь форматированием библиотеки с помощью $1, $2...

Если форматировать руками, ваше приложение будет подвержено sql-инъекциям. Библиотека позволяет избежать этой уязвимости.

Библиотека конвертирует типы агрументов при запросе и строки при получении данных. То есть в функции insert мы использовали datetime.datetime.now(), а библиотека привела дату к формату, который понимает Postgres. При получении данных через функцию select библиотека тоже автоматически приведет дату к объекту datetime.datetime.