Concurrent

В предыдущих примерах мы получали подключения с помощью get_connection_by_dsn.

Объект, который мы получаем — Connection — «непотокобезопасный»: его методы нельзя вызывать из разных корутин одновременнно.

```
import asyncio
import os

import asyncpg


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


async def query(conn):
    print(await conn.fetch("select * from users"))


async def incorrect():
    conn = await get_connection_by_dsn()
    await asyncio.gather(query(conn), query(conn))


asyncio.run(incorrect())
```

Если запустить этот код, то возникнет исключение InterfaceError:

```
asyncpg.exceptions._base.InterfaceError: cannot perform operation: another operation is in prog
Task was destroyed but it is pending!
```

На каждую параллельную корутину нужно иметь собственное подключение. Есть два решения:

- **Создавать подключение на каждый запрос.** Решение плохое, так как создание подключения ресурсоемкий процесс. К тому же на каждое подключение Postgres создает дочерний процесс. Как мы уже знаем, количество параллельных корутин может быть сильно больше количества запущенных процессов, поэтому это решение плохо подходит
- **Создать несколько подключений к базе, по необходимости выдавать его корутинам, а после завершения работы возвращать обратно в общий чан.** Объект, который содержит в себе подключения, и из которого можно их брать, называется *pool*

Рассмотрим пример:

```
import asyncio
import os

import asyncpg


async def get_pool_by_dsn():
    return await asyncpg.create_pool(os.getenv("POSTGRES_DSN"))


async def query(conn):
    print(await conn.fetch("select * from users"))


async def pool_query(pool):
    async with pool.acquire() as conn:
        await query(conn)


async def correct():
    pool = await get_pool_by_dsn()
    await asyncio.gather(pool_query(pool), pool_query(pool))


asyncio.run(correct())
```

asyncpg.create_pool создает pool.

Функция create_pool принимает min_size, max_size. Это размеры pool, количество соединений, которое в нем находится. То есть количество параллельных запросов, которое можно выполнить к Postgres.

Асинхронный контекстный менеджер *pool.acquire()* достает из pool одно из свободных соединений. Если все подключения заняты, то корутина будет ждать, пока подключение не освободится. После выхода из контекстного менеджера подключение *conn* возвращается обратно в pool.

Запустить пример можно в mercury:

```
python3 connectors/postgres/pool.py
```