

Przedmiot: Organizacja Systemów Zarządzania Baz Danych

Laboratorium 3: PostgreSQL – Connection pooling

Autor: Bartłomiej Jamiołkowski, Adrianna Bodziony

### III Przygotowanie bazy danych

- a) Na potrzeby ćwiczenia utwórz z poziomu użytkownika postgres nową instancję / nowy klaster postgresowy w lokalizacji /tmp/test\_db

Polecenie:

```
postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ ./initdb -D /tmp/test_db
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /tmp/test_db ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Europe/Warsaw
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D /tmp/test_db -l logfile start
```

Ustawiono port 5434.

- b) Uruchom utworzoną instancję (pg\_ctl)

Polecenie:

```
postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ ./pg_ctl -D /tmp/test_db -l /tmp/test_db_log start
waiting for server to start.... done
server started
```

- c) Zainstaluj narzędzie pgbouncer. Jeśli narzędzie jest już zainstalowane – aby uniknąć pozostałości innych konfiguracji – usun go całkowicie (apt remove --purge) i zainstaluj od nowa

Polecenie:

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ sudo apt install pgbouncer
[sudo] password for postgres:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  libllvm14
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libc-ares2 libevent-2.1-7
Suggested packages:
  python3-psycpg2
The following NEW packages will be installed:
  libc-ares2 libevent-2.1-7 pgbouncer
0 upgraded, 3 newly installed, 0 to remove and 45 not upgraded.
Need to get 411 kB of archives.
After this operation, 1053 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 pgbouncer amd64 1.22.1-1.pgdg22.04+1 [217 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libc-ares2 amd64 1.18.1-1ubuntu0.22.04.3 [45.1 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 libevent-2.1-7 amd64 2.1.12-stable-1build3 [148 kB]
Fetched 411 kB in 1s (644 kB/s)
Selecting previously unselected package libc-ares2:amd64.
(Reading database ... 26633 files and directories currently installed.)
Preparing to unpack .../libc-ares2_1.18.1-1ubuntu0.22.04.3_amd64.deb ...
Unpacking libc-ares2:amd64 (1.18.1-1ubuntu0.22.04.3) ...
Selecting previously unselected package libevent-2.1-7:amd64.
Preparing to unpack .../libevent-2.1-7_2.1.12-stable-1build3_amd64.deb ...
Unpacking libevent-2.1-7:amd64 (2.1.12-stable-1build3) ...
Selecting previously unselected package pgbouncer.
Preparing to unpack .../pgbouncer_1.22.1-1.pgdg22.04+1_amd64.deb ...
Unpacking pgbouncer (1.22.1-1.pgdg22.04+1) ...
Setting up libc-ares2:amd64 (1.18.1-1ubuntu0.22.04.3) ...
Setting up libevent-2.1-7:amd64 (2.1.12-stable-1build3) ...
Setting up pgbouncer (1.22.1-1.pgdg22.04+1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/pgbouncer.service → /lib/systemd/system/pgbouncer.service.
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.7) ...

```

d) Konfigurujemy pgbouncer'a. Plik konfiguracyjny znajdziesz w

/etc/pgbouncer/pgbouncer.ini

- i. W pierwszej kolejności ustawiamy serwer/instancje serwera i baze danych do połączenia z którym ma służyć konfigurowane narzędzie. Ustawiamy tam baze danych na domyślną postgres, host na localhost, port na domyślny 5432 i auth\_user na testuser

Polecenie:

```

;; pool_mode= connect_query= application_name=
[databases]
* = host=localhost port=5434 dbname=postgres auth_user=testuser
;; feedb even Unix socket

```

- ii. Dodatkowo zmieniamy auth\_type na trust

Polecenie:

```

;;;
;;; Authentication settings
;;;

;; any, trust, plain, md5, cert, hba, pam
auth_type = trust
auth_file = /etc/pgbouncer/userlist.txt

```

- e) Łączymy się do instancji serwera (psql) i tworzymy superużytkownika testuser (create user..)

Polecenie:

```
postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ ./psql -U postgres
psql (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
Type "help" for help.

postgres=# CREATE USER testuser WITH SUPERUSER PASSWORD 'superuser';
CREATE ROLE
```

Ostatecznie utworzono użytkownika testuser3 oraz instancję pgbouncer na porcie 6434 -  
połączenie

```
postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ ./psql -p 6434 test_db -U testuser3
psql (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
Type "help" for help.

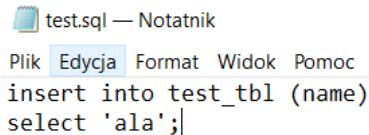
test_db=#
```

- f) Uruchamiamy narzędzie pgbouncer (polecenie pgbouncer)
- g) Łączymy się z serwerem postgres (polecenie psql) ale robimy to za pośrednictwem pgbouncera (który nasluchuje domyślnie na porcie 6432), łączymy się do bazy postgres ale jako nazwy używamy aliasu który zdefiniowaliśmy w pliku konfiguracyjnym bouncera, i łączymy się na użytkownika testuser
- h) Jeżeli wszystko przebiegło poprawnie, oczekiwany efekt na tym etapie powinien być następujący:

Polecenie:

```
2024-05-20 14:21:37.053 CEST [11469] LOG stats: 0 xacts/s, 0 queries/s, 0 client parses/s, 0 server parses/s, 0 binds/s, in 0 B/s, out 0 B/s, xa
ct 0 us, query 0 us, wait 0 us
2024-05-20 14:22:37.056 CEST [11469] LOG stats: 0 xacts/s, 0 queries/s, 0 client parses/s, 0 server parses/s, 0 binds/s, in 0 B/s, out 0 B/s, xa
ct 0 us, query 0 us, wait 0 us
2024-05-20 14:23:23.744 CEST [11469] LOG C-0x55eac97ed6c0: test_db/testuser3@unix(11878):6434 closing because: client close request (age=182s)
2024-05-20 14:23:37.054 CEST [11469] LOG stats: 0 xacts/s, 0 queries/s, 0 client parses/s, 0 server parses/s, 0 binds/s, in 0 B/s, out 0 B/s, xa
ct 0 us, query 0 us, wait 0 us
2024-05-20 14:24:35.650 CEST [11469] LOG C-0x55eac97ed6c0: test_db/testuser3@unix(12886):6434 login attempt: db=test_db user=testuser3 tls=no
2024-05-20 14:24:37.063 CEST [11469] LOG stats: 0 xacts/s, 0 queries/s, 0 client parses/s, 0 server parses/s, 0 binds/s, in 0 B/s, out 0 B/s, xa
ct 0 us, query 0 us, wait 0 us
2024-05-20 14:25:18.577 CEST [11469] LOG C-0x55eac97ed6c0: test_db/testuser3@unix(12886):6434 closing because: client close request (age=42s)
```

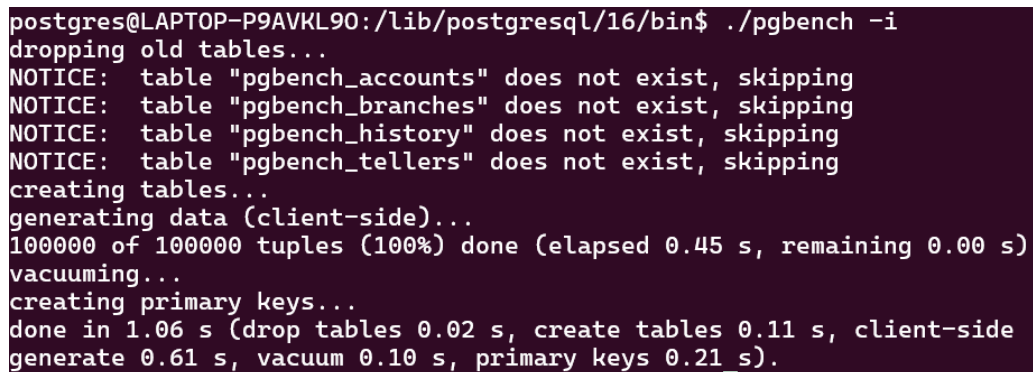
- i) .
- j) .
- k) Aby zweryfikować efektywność pgbouncera przygotuj prosty skrypt wykonujący jakąś pojedynczą / elementarną operację (wystarczy polecenie typu select 1).



```
test.sql — Notatnik
Plik Edycja Format Widok Pomoc
insert into test_tbl (name)
select 'ala';|
```

- 1) Do testów wykorzystamy standardowo dostępne narzędzie `pg_bench` w tym celu
- i. Aby „przygotować” bazę danych do przeprowadzenia benchmarku wykonujemy polecenie `pgbench -i` nazwa bazy danych na której chcemy wykonać benchmark (najprościej będzie zrobić to na domyślnie tworzonej bazie danych postgres)

Polecenie:



```
postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ ./pgbench -i
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.45 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 1.06 s (drop tables 0.02 s, create tables 0.11 s, client-side
generate 0.61 s, vacuum 0.10 s, primary keys 0.21 s).
```

- ii. Następnie uruchamiamy benchmark (polecenie `pgbench`) symulując działanie 20 aplikacji klienckich (-c) z których każda powinna wykonać po 1000 transakcji (-t), benchmark wykonamy na bazie postgres (-S), dla każdej transakcji istniejące połączenie powinno zostać zamknięte i utworzone nowe (-C), jako operacja wykonywana w ramach transakcji wskazujemy nasz wcześniej utworzony „skrypt” (-f) i najpierw wykonujemy nasz benchmark bezpośrednio na serwerze postgres (z pominięciem `pgbouncer`) a zatem port na który się łączymy to 5432 (-p)

Polecenie:

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000
-S -C -f /tmp/test.sql -p 5432 postgres
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.

transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 159.789 ms
average connection time = 7.893 ms
tps = 125.164693 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 9995 transactions (50.0% of total, tps = 62.551055)
- number of failed transactions: 0 (0.000%)
- latency average = 67.200 ms
- latency stddev = 102.148 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 10005 transactions (50.0% of total, tps = 62.613638)
- number of failed transactions: 0 (0.000%)
- latency average = 68.907 ms
- latency stddev = 112.225 ms

```

- iii. Po wykonaniu testu otrzymamy jego podsumowanie, w którym otrzymamy m.in. informacje o ilości realizowanych transakcji na sekundę. W moim przypadku w sytuacji kiedy działamy bez pośrednictwa pgbouncera i nowe połączenie jest tworzone i niszczone dla każdej pojedynczej transakcji serwer jest w stanie obsługiwać około 320 transakcji na sekundę, a średni czas połączenia to ponad 3 ms

Polecenie:

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -i -U postgres -p 5432
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.10 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.96 s (drop tables 0.00 s, create tables 0.06 s, client-side generate 0.45 s, vacuum 0.18 s, primary keys 0.27 s).

```

- iv. .
- v. Następnie powtarzamy taki sam test ale tym razem łącząc się za pośrednictwem pg\_bouncera. Jak widać przebieg połączeń przez pg\_bouncera i wprowadzenia pooli utrzymywanych przez niego połączeń pozwala zwiększyć liczbę realizowanych transakcji na sekundę w moim przypadku do 527 a czas średniego połączenia zmalał do 1.8 ms

Polecenie:

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.

transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 34.951 ms
average connection time = 1.689 ms
tps = 572.225616 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 10092 transactions (50.5% of total, tps = 288.745046)
- number of failed transactions: 0 (0.000%)
- latency average = 15.860 ms
- latency stddev = 13.075 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 9908 transactions (49.5% of total, tps = 283.480570)
- number of failed transactions: 0 (0.000%)
- latency average = 15.732 ms
- latency stddev = 13.359 ms

```

- vi. .
- vii. Powtórz benchmark zmieniając sposób zarządzania pulą połączeń z „per session” na „per transaction” (parametr pool\_mode w pliku konfiguracyjnym). Zmiana ta w moim przypadku zwiększyła liczbę transakcji na sekundę do ~750 a średni czas połączenia zmalał do ~1.3ms

Polecenie:

```

GNU nano 6.2 /etc/pgbouncer/pgbounc
;admin_users = user2, someadmin, otheradmin

;; comma-separated list of users who are just allowed to use SHOW command
;stats_users = stats, root

;;;
;;; Pooler personality questions
;;;

;; When server connection is released back to pool:
;; session - after client disconnects (default)
;; transaction - after transaction finishes
;; statement - after statement finishes
pool_mode = transaction

```

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 30.834 ms
average connection time = 1.488 ms
tps = 648.625747 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 9925 transactions (49.6% of total, tps = 321.880527)
- number of failed transactions: 0 (0.000%)
- latency average = 13.752 ms
- latency stddev = 10.713 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 10075 transactions (50.4% of total, tps = 326.745220)
- number of failed transactions: 0 (0.000%)
- latency average = 13.818 ms
- latency stddev = 10.879 ms

```

- viii. Powtórz benchmark zmieniając sposób zarządzania pulą połączeń z „per transaction” na „per statement” (parametr pool\_mode w pliku konfiguracyjnym).

Polecenie:

```

postgres@LAPTOP-P9AVKL90:/Lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 31.275 ms
average connection time = 1.511 ms
tps = 639.480885 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 10092 transactions (50.5% of total, tps = 322.682054)
- number of failed transactions: 0 (0.000%)
- latency average = 14.043 ms
- latency stddev = 11.224 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 9908 transactions (49.5% of total, tps = 316.798830)
- number of failed transactions: 0 (0.000%)
- latency average = 13.947 ms
- latency stddev = 11.149 ms

```

- ix. Stworz w bazie danych postgres tabele test\_tbl o strukturze (id int, name varchar).

Polecenie:


```

postgres=# create table test_tbl (id serial primary key ,name varchar );
CREATE TABLE

```

- x. Powtorz zestaw benchmarków jak powyżej dla operacji wstawienia pojedynczego wiersza do stworzonej tabeli)

Polecenie:

 test.sql — Notatnik

Plik Edycja Format Widok Pomoc

```

insert into test_tbl (name)
select 'ala';|

```

Wykonanie benchmarków dla różnych konfiguracji:

- statement

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 31.511 ms
average connection time = 1.493 ms
tps = 634.688963 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 9961 transactions (49.8% of total, tps = 316.106838)
- number of failed transactions: 0 (0.000%)
- latency average = 11.627 ms
- latency stddev = 9.934 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 10039 transactions (50.2% of total, tps = 318.582125)
- number of failed transactions: 0 (0.000%)
- latency average = 17.754 ms
- latency stddev = 11.284 ms

```

- transaction

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.

transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 33.897 ms
average connection time = 1.613 ms
tps = 590.017076 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 9879 transactions (49.4% of total, tps = 291.438935)
- number of failed transactions: 0 (0.000%)
- latency average = 12.926 ms
- latency stddev = 11.194 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 10121 transactions (50.6% of total, tps = 298.578141)
- number of failed transactions: 0 (0.000%)
- latency average = 19.059 ms
- latency stddev = 12.351 ms

```

- session

```

postgres@LAPTOP-P9AVKL90:/lib/postgresql/16/bin$ pgbench -c 20 -t 1000 -S postgres -C -f /tmp/test.sql -p 6434
pgbench (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1
query mode: simple
number of clients: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 1000
number of transactions actually processed: 20000/20000
number of failed transactions: 0 (0.000%)
latency average = 30.663 ms
average connection time = 1.458 ms
tps = 652.261515 (including reconnection times)
SQL script 1: <builtin: select only>
- weight: 1 (targets 50.0% of total)
- 10032 transactions (50.2% of total, tps = 327.174376)
- number of failed transactions: 0 (0.000%)
- latency average = 11.584 ms
- latency stddev = 10.383 ms
SQL script 2: /tmp/test.sql
- weight: 1 (targets 50.0% of total)
- 9968 transactions (49.8% of total, tps = 325.087139)
- number of failed transactions: 0 (0.000%)
- latency average = 17.223 ms
- latency stddev = 11.491 ms

```