

# Расширяемость Пул соединений



## Авторские права

© Postgres Professional, 2020 год.

Авторы: Егор Рогов, Павел Лузанов

## Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

## Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

## Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Для чего используется пул соединений  
Пул соединений в общей архитектуре системы  
Доступные варианты, PgBouncer  
Режимы работы  
Вопросы аутентификации  
Управление пулом  
Особенности разработки при наличии пула

# Один клиент, один процесс

Создание соединения обходится дорого

аутентификация, наполнение кешей

Большое количество соединений замедляет работу сервера

Каждое соединение расходует память



3

Вспомним архитектуру PostgreSQL. Когда клиент подключается к серверу, для него создается отдельный обслуживающий процесс, с которым и происходит вся дальнейшая работа этого клиента. Поэтому количество обслуживающих процессов на сервере равно количеству клиентов.

При большом количестве клиентов это вызывает проблемы.

- Создание соединений обходится дорого, в основном из-за того, что установка соединения требует аутентификации (особенно в случае шифрованных соединений) и наполнения кешей (например, кеша системного каталога).

- Большое количество процессов замедляет работу сервера. Основная причина в том, что для создания снимка данных (а это частая операция) приходится просматривать список всех имеющихся процессов. И такой просмотр требует блокировки внутренних структур данных.

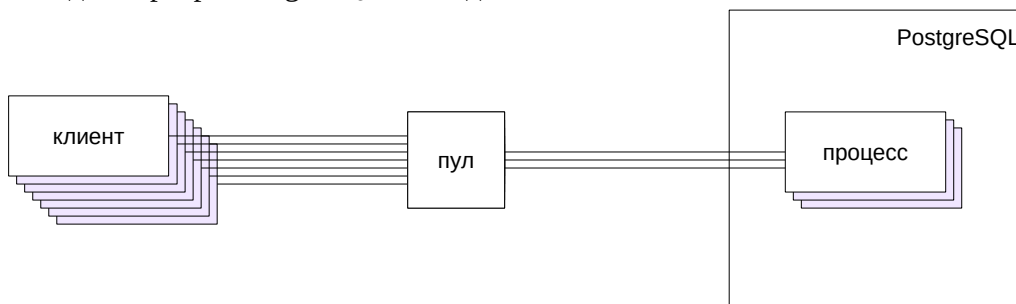
- Каждый процесс занимает определенную часть памяти сервера.

Количество соединений, не вызывающее проблем, зависит от многих факторов (наличие свободной памяти, количество ядер), но обычно лежит в диапазоне от нескольких десятков до нескольких сотен.

Обычно большую часть времени соединение простаивает

## Менеджер пула соединений

открывает и удерживает несколько соединений с сервером  
для клиентов выглядит как сервер PostgreSQL,  
для сервера PostgreSQL выглядит как клиент



4

Чтобы одновременно работать с большим количеством клиентов, используется *пул соединений*.

Специальная программа – менеджер пула – открывает и удерживает некоторое (небольшое) количество соединений с сервером БД. Для сервера БД все выглядит так, как будто имеются несколько долгоживущих соединений.

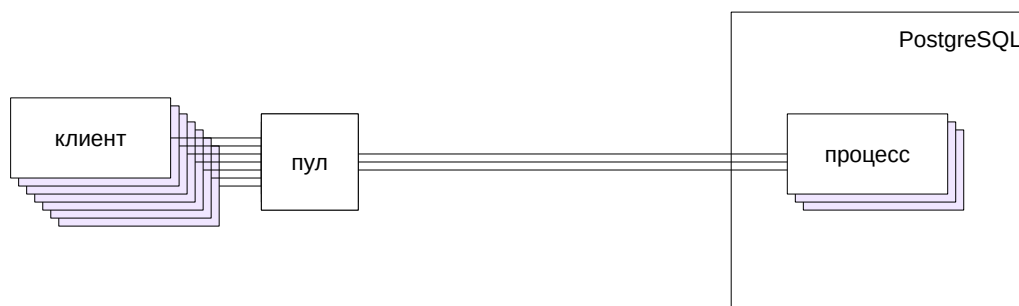
Клиенты подключаются не к серверу БД, а к менеджеру пула. Для этого он реализует тот же клиентский протокол, что и PostgreSQL, «притворяясь» для клиентов базой данных. Клиентские запросы переадресуются серверу БД, используя одно из имеющихся свободных соединений.

Идея здесь в том, что если клиент открывает соединение надолго, то большую часть времени оно все равно простаивает. (Можно провести некоторую аналогию с передачей данных по сети: между абонентами нет выделенного соединения, а данные передаются небольшими пакетами, используя имеющиеся каналы связи.)

## На сервере приложения

клиент использует быстрое локальное подключение

подключение к серверу устанавливается один раз и переиспользуется



5

Можно по-разному встроить пул соединений в архитектуру информационной системы.

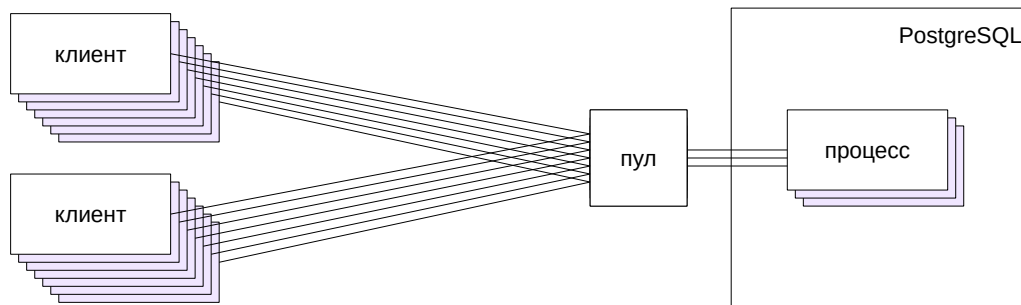
Один вариант – разместить менеджер пула на сервере приложения. Обычно это будет пул, предоставляемый самим сервером приложений или драйвером PostgreSQL, хотя при необходимости можно воспользоваться и сторонним решением.

При этом клиент получает возможность быстро подключаться к локальному менеджеру пула, а время установки соединения с сервером БД перестает играть роль, поскольку выполняется только один раз. (Это не совсем верно, поскольку реализация пула может предусматривать динамическое увеличение и уменьшение соединений с СУБД в зависимости от нагрузки, но идея остается той же самой – одно соединение используется многократно.)

На сервере баз данных

если используется несколько серверов приложений

Встроенного пула соединений нет



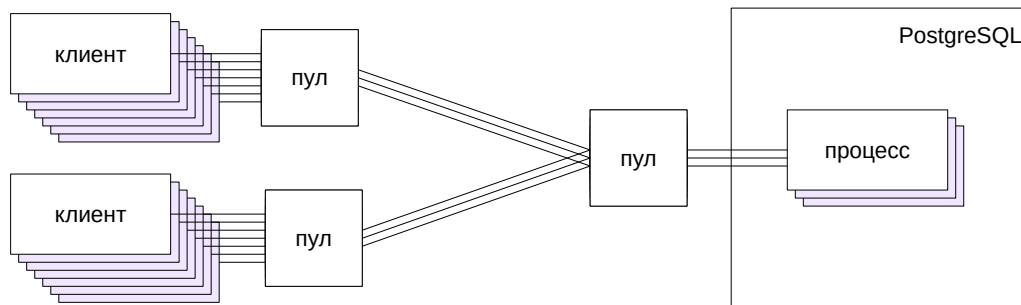
6

С другой стороны, если имеется несколько серверов приложений, то пул соединений имеет смысл устанавливать на сервере БД – иначе может не получиться ограничить количество соединений.

И на серверах приложений, и на сервере баз данных

полезные свойства комбинируются

накладные расходы растут, но обычно невелики



Вообще говоря, в этом случае может иметь смысл использовать менеджеры пулов и на серверах приложений, и на сервере БД, чтобы одновременно и сократить время подключения, и ограничить количество соединений.

Следует учитывать, что каждый дополнительный узел на пути от клиента к серверу будет вносить определенные накладные расходы, но они, как правило, незначительны.

## PgBouncer

стандарт де-факто  
используем далее как пример

## Pgpool-II

не только пул, но и балансировка нагрузки  
противоречивые отзывы

## Odyssey

многопоточный, высокопроизводительный

и др.

Как уже говорилось, многие, если не все, серверы приложений и просто драйверы, реализующие клиент-серверный протокол PostgreSQL, предоставляют возможность организации пула соединений на клиентской стороне.

Встроенного менеджера пула в PostgreSQL пока нет. Но есть сторонние продукты. Все представленные здесь менеджеры – проекты с открытым исходным кодом.

**PgBouncer** (<https://www.pgouncer.org/>) используется очень широко и является стандартом де-факто. Это легкий пул соединений, без дополнительных функций. Именно эту программу мы будем рассматривать дальше в этой теме.

**Pgpool-II** (<https://www.pgpool.net>) разрабатывает компания SRA OSS (Япония). Это не только пул соединений, но и балансировщик нагрузки. Он реализует интересный функционал (например, анализ трафика для того чтобы отличать пишущие транзакции от только читающих), но, к сожалению, есть противоречивые отзывы о качестве этого продукта.

**Odyssey** (<https://github.com/yandex/odyssey>) - недавно выпущенный пул соединений компании Яндекс. Основное преимущество состоит в многопоточной архитектуре, позволяющей добиться высокой производительности.

Это, конечно, не полный список.



## «Пул сеансов»

может иметь смысл только для коротких сеансов

## «Пул транзакций»

разные транзакции одного клиентского сеанса  
могут выполняться в разных соединениях с сервером

основной режим

есть особенности, которые надо учитывать при разработке

## «Пул операторов»

запрещает транзакции, состоящие более чем из одного оператора

Новый пул соединений создается для каждой пары  
(база данных, роль)

Существует несколько режимов, в которых может работать менеджер пула.

В режиме **«пул сеансов»** клиенту предоставляется выделенное соединение из числа доступных. Этот режим полезен только в случае короткоживущих соединений: клиент подключается, выполняет транзакцию и тут же отключается. Например, это может быть система мониторинга, посылающая запросы раз в секунду.

Режим **«пул транзакций»** является наиболее универсальным и полезным. В нем соединение предоставляется отдельно для каждой транзакции. Такой режим полезен и в случае, когда клиент устанавливает долгоживущее соединение. Но из-за такого режима возникают определенные особенности, которые необходимо учитывать при разработке приложений. Мы поговорим о них немного позже.

PgBouncer имеет еще один режим – **«пул операторов»**, но он фактически бесполезен, поскольку просто запрещает выполнение транзакций, состоящих более чем из одного оператора (работает только с режимом автофиксации).

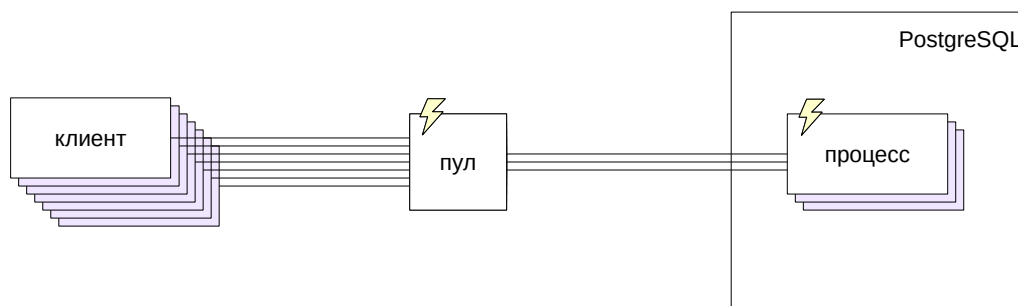
Любое соединение с сервером всегда выполняется к определенной базе данных и под определенной ролью. Поэтому на каждую пару {база данных, роль} менеджер пулов создает новый пул соединений.

Из-за этого схема с пулом соединений вряд ли имеет смысл, если используется много различных ролей и баз данных. Однако обычно все-таки используется одна база данных и ограниченное количество ролей (с аутентификацией конечных пользователей на уровне приложения).

Менеджер пула аутентифицирует соединения клиентов,  
сервер баз данных аутентифицирует соединения пула

требуется специальная настройка

можно использовать файл формата `pg_hba.conf`



10

Известно, что аутентификацию выполняет сервер БД. Но при наличии пула соединений нельзя полагаться только на нее.

Поскольку в данном случае менеджер пула является точкой входа для клиентов, он сам должен выполнять аутентификацию аналогично тому, как это делает сервер БД. Иначе любой клиент сможет получить несанкционированный доступ к уже открытому соединению (которое было аутентифицировано сервером БД один раз при открытии).

Таким образом, схема получается следующая:

1. Менеджер пула выполняет аутентификацию входящего соединения.

Для этого ему требуются настройки, более или менее аналогичные имеющимся в PostgreSQL. PgBouncer позволяет обойтись более простыми настройками, но допускает использование обычного файла формата `pg_hba.conf`. Необходимо учитывать, что менеджер пула может поддерживать не все методы аутентификации. Например, PgBouncer поддерживает SCRAM только с версии 1.11 (27.08.2019), не поддерживает GSSAPI и т. п..

2. При успешной аутентификации, если потребуется открыть новое соединение с базой данных, менеджер пула передает PostgreSQL информацию, полученную от клиента (например, имя и пароль), а PostgreSQL аутентифицирует входящее соединение обычным образом. (Хотя в общем случае менеджер пула может подключаться к PostgreSQL под другой ролью, игнорируя идентификацию клиента.)

## Минимальная настройка

Файл настроек PgBouncer:

```
postgres$ ls -l /etc/pgbouncer/pgbouncer.ini
```

```
-rw-r----- 1 postgres postgres 9377 дек 16 2020 /etc/pgbouncer/pgbouncer.ini
```

```
postgres$ grep '^[^;]' /etc/pgbouncer/pgbouncer.ini
```

```
[databases]
* = host=localhost port=5432
[users]
[pgbouncer]
logfile = /var/log/postgresql/pgbouncer.log
pidfile = /var/run/postgresql/pgbouncer.pid
listen_addr = 127.0.0.1
listen_port = 6432
unix_socket_dir = /var/run/postgresql
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
admin_users = student
pool_mode = transaction
```

- Секция [databases] позволяет переадресовывать обращения к разным БД на разные серверы PostgreSQL (в нашем случае это не используется);
- PgBouncer слушает порт 6432;
- Используется аутентификация md5;
- Роль student может выполнять администрирование PgBouncer;
- Используется режим пула транзакций.

---

Файл пользователей:

```
postgres$ cat /etc/pgbouncer/userlist.txt
```

```
"web"      "md568f4854080180847beec4cb3f9c61ed7"
"emp"      "md5d58d88ac9a362c82ebdae772f52dba14"
"student"  "md550d9482e20934ce6df0bf28941f885bc"
```

Чтобы не синхронизировать пароли, можно получать их непосредственно с сервера БД:

```
postgres$ grep 'auth_query' /etc/pgbouncer/pgbouncer.ini
```

```
;; use auth_user with auth_query if user not present in auth_file
;auth_query = SELECT username, passwd FROM pg_shadow WHERE username=$1
```

---

Попробуем подключиться. Поскольку PgBouncer настроен на парольную md5-аутентификацию, передадим пароль явно (можно было бы использовать файл ~/.pgpass):

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
=> SELECT pg_backend_pid();
```

```
pg_backend_pid
-----
17201
(1 row)
```

Подключение работает.

Теперь закроем соединение и откроем его заново:

```
=> \q
```

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
=> SELECT pg_backend_pid();
```

```
pg_backend_pid
-----
17201
(1 row)
```

Фактически мы продолжаем работать в том же самом сеансе, который PgBouncer удерживает открытым.

```
=> \q
```



## Консоль управления PgBouncer

подключение psql к псевдобазе pgbouncer

SHOW — ряд команд для вывода состояния и другой информации

PAUSE — приостановка клиентов (для перезагрузки базы данных)

RESUME — возобновление работы

RELOAD — перечитывание файла конфигурации

и другие возможности

Для управления PgBouncer используется консоль, к которой можно подключиться с помощью обычного psql, указывая специальную БД pgbouncer.

Консоль позволяет запрашивать состояние системы и выполнять ряд команд. Например, можно приостановить поток запросов от клиентов на время перезагрузки сервера БД и т. п.

## Консоль управления

Чтобы работать с консолью управления PgBouncer, необходимо подключиться к одноименной базе данных (это могут сделать пользователи, определенные в параметрах `admin_users` и `stat_users`).

```
student$ psql postgresql://student@localhost:6432/pgbouncer?password=student
```

Теперь мы работаем не с PostgreSQL — команды обрабатывает сам PgBouncer.

Например, можно получить информацию об используемых пулах:

```
=> SHOW POOLS;
```

database	user	cl_active	cl_waiting	sv_active	sv_idle	sv_used	sv_tested	sv_login	maxwait	maxwait_us	pool_mode
pgbouncer	pgbouncer	1	0	0	0	0	0	0	0	0	statement
student	student	0	0	0	1	0	0	0	0	0	transaction

(2 rows)

Приостановим работу клиентов:

```
=> PAUSE;
```

PAUSE

Теперь подключается новый клиент и пытается выполнить запрос:

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
| => SELECT now();
```

Для клиента ситуация выглядит так, как будто база данных не отвечает.

Теперь мы можем даже перезапустить PostgreSQL.

```
student$ sudo pg_ctlcluster 12 main restart
```

Возобновляем работу клиентов, и запрос успешно выполняется:

```
=> RESUME;
```

RESUME

```
|          now
| -----
| 2022-12-09 23:38:27.588279+03
| (1 row)
```

Таким образом можно выполнять работы на сервере БД (например, обновление), не обрывая клиентские соединения.

```
=> \q
```

```
| => \q
```

## Действия, локализованные в сеансе, а не в транзакции

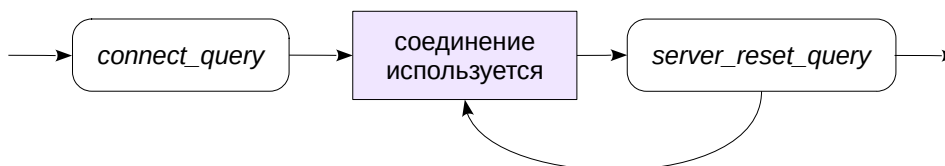
- подготовленные операторы (PREPARE/EXECUTE)
- временные таблицы (кроме ON COMMIT DROP)
- функция currval
- установка конфигурационных параметров (SET/RESET)
- рекомендательные блокировки на уровне сеанса
- межпроцессные уведомления (LISTEN/NOTIFY)
- курсоры с фразой WITH HOLD
- загрузка разделяемых библиотек (LOAD)
- расширения, сохраняющие состояние на уровне сеанса

14

Режим «пула транзакций» имеет для разработчика приложений особенности, связанные с тем, что две транзакции, выполняемые в *одном* клиентском сеансе, могут выполняться в *разных* сеансах сервера БД. Поэтому не будут работать (точнее, будут работать не так, как задумано) любые команды, действие которых распространяется на время сеанса, а не транзакции.

К ним относятся подготовленные операторы (если подготовкой управляет клиент), использование временных таблиц, функции currval, установка параметров и рекомендательных блокировок на время сеанса, межпроцессные уведомления, курсоры WITH HOLD, загрузка разделяемых библиотек в сеансе... А также расширения, реализующие «глобальные переменные» уровня сеанса, такие как pg\_variables (<https://postgrespro.ru/docs/postgrespro/12/pg-variables>).

## Начальные действия и изоляция



PgBouncer предоставляет параметры, позволяющие

- настраивать только что открытое соединение (перед тем, как выдать его клиенту), выполняя команду SQL,
- очищать состояние соединения после использования его клиентом.

Их можно использовать, чтобы принудительно изолировать состояние, видимое клиентам, если они выполняют операции, нарушающие изоляцию.



## Особенности работы

Рассмотрим на примере подготовленных операторов — в случае, когда подготовкой управляет клиент.

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
=> PREPARE hello AS SELECT 'Hello, world!';
```

```
PREPARE
```

Теперь откроем еще один сеанс.

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
| => BEGIN;
| BEGIN
|
| => SELECT * FROM pg_prepared_statements \gx
|
| -[ RECORD 1 ]---+-----
| name           | hello
| statement      | PREPARE hello AS SELECT 'Hello, world!';
| prepare_time   | 2022-12-09 23:38:27.717832+03
| parameter_types | {}
| from_sql       | t
|
| => SELECT pg_backend_pid();
|
| pg_backend_pid
| -----
|          17537
| (1 row)
```

Транзакция второго сеанса выполняется в том же соединении, поэтому ей доступен подготовленный оператор:

```
| => EXECUTE hello;
|
|      ?column?
| -----
| Hello, world!
| (1 row)
```

А какое соединение будет использоваться в первом сеансе?

```
=> BEGIN;
BEGIN
=> SELECT pg_backend_pid();
pg_backend_pid
-----
          17816
(1 row)
```

Уже другое, поскольку во втором сеансе транзакция не завершена.

```
=> EXECUTE hello;
ERROR:  prepared statement "hello" does not exist
Подготовленного оператора в памяти этого обслуживающего процесса нет.
=> END;
ROLLBACK
| => END;
| COMMIT
```

Чтобы пользоваться преимуществами и пула соединений, и подготовленными операторами, надо переносить управление подготовкой на сторону сервера. Самый простой и удобный способ — писать функции на языке PL/pgSQL. В этом случае интерпретатор автоматически подготавливает все запросы.



На каждое клиентское соединение создается обслуживающий процесс на сервере

Большое количество соединений приводит к проблемам

Пул соединений позволяет одновременно работать многим клиентам, ограничивая количество соединений с сервером

Транзакционный режим пула имеет особенности с точки зрения разработки приложений



1. Убедитесь, что при работе приложения через пул соединений (порт 6432) разные пользователи обычно видят одну и ту же корзину. Выясните причину ошибки и исправьте код хранимых функций так, чтобы приложение работало корректно как без пула соединений, так и с ним.
2. Использование пула мешает выводу в журнал сообщений сервера информации, относящейся только к одному клиенту. Наше приложение вызывает в начале каждой транзакции функцию `webapi.trace` (или `emrapi.trace`), если в служебной панели установлен признак трассировки. Реализуйте эти функции так, чтобы они устанавливали параметр сервера `log_min_duration_statements = 0` и облегчали идентификацию клиента. Проверьте работу по журналу сообщений.

18

Во всех темах курса, начиная с этой, мы будем использовать подключение через пул соединений на порту 6432 (настройка по умолчанию). PgBouncer настроен точно так же, как было показано в демонстрации.

1. Проблема воспроизводится следующим образом. Откройте две вкладки браузера с книжным магазином. В первой войдите как `alice` и положите в корзину одну или несколько книг. Во второй зайдите как `bob` и проверьте корзину.

Поскольку один пользователь видит корзину другого пользователя, можно предположить, что дело в разграничении доступа. Чтобы понять, в каком месте следует искать проблему, вспомните материал темы «Приложение 2.0».

2. Необходимые функции уже имеются в базе данных, но ничего не делают. Функция `webapi.trace` принимает на вход токен аутентификации (или `NULL`, если пользователь не вошел в систему).

Для идентификации можно использовать параметр `log_line_prefix` с маской `%a`, выводящей имя приложения. Имя приложения устанавливается параметром `application_name`, в которое можно записать имя пользователя, если оно известно.

```
student$ psql bookstore2
```

## 1. Корректная работа с пулом соединений

Проблема состоит в том, что функция, проверяющая токен, запоминает пользователя в параметре сервера на уровне сеанса:

```
=> \sf check_auth
```

```
CREATE OR REPLACE FUNCTION public.check_auth(auth_token uuid)
  RETURNS bigint
  LANGUAGE plpgsql
  STABLE
AS $function$
DECLARE
    user_id bigint;
BEGIN
    user_id := current_setting('check_auth.user_id', /* missing_ok */true);
    IF user_id IS NULL THEN
        SELECT s.user_id
        INTO STRICT user_id
        FROM sessions s
        WHERE s.auth_token = check_auth.auth_token;
        PERFORM set_config('check_auth.user_id', user_id::text, /* is_local */false);
    END IF;
    RETURN user_id;
END;
$function$
```

Это работает, когда для каждого клиента (в нашем случае — страницы приложения) используется собственный сеанс. Но в случае пула соединений все клиенты могут обслуживаться одним сеансом.

Исправление состоит в том, чтобы честно выполнять проверку каждый раз при вызове функции:

```
=> CREATE OR REPLACE FUNCTION check_auth(auth_token uuid) RETURNS bigint
AS $$
DECLARE
    user_id bigint;
BEGIN
    SELECT s.user_id
    INTO STRICT user_id
    FROM sessions s
    WHERE s.auth_token = check_auth.auth_token;
    RETURN user_id;
END;
$$ LANGUAGE plpgsql STABLE;

CREATE FUNCTION
```

## 2. Трассировка

Функция трассировки для админки:

```
=> CREATE OR REPLACE FUNCTION empapi.trace() RETURNS void
AS $$
    SELECT set_config(
        'log_min_duration_statement',
        '0',
        /* is_local */true
    );
$$ LANGUAGE sql SECURITY DEFINER;

CREATE FUNCTION
```

Обратите внимание, что параметр устанавливается на время транзакции (третий параметр).

Функция трассировки для магазина:

```
=> CREATE OR REPLACE FUNCTION webapi.trace(auth_token uuid) RETURNS void AS $$
    SELECT set_config(
        'log_min_duration_statement',
        '0',
        /* is_local */true
    ),
    set_config(
        'application_name',
        (SELECT 'client=' || u.username
         FROM users u JOIN sessions s ON u.user_id = s.user_id
         WHERE s.auth_token = trace.auth_token
        ),
        /* is_local */true
    );
$$ LANGUAGE sql SECURITY DEFINER;
```

CREATE FUNCTION

Чтобы имя пользователя попало в журнал сообщений, необходимо изменить параметр log\_line\_prefix, например, так:

```
=> ALTER SYSTEM SET log_line_prefix = '%m [%p] %q%u@%d (%a) ';
```

ALTER SYSTEM

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf
-----
t
(1 row)
```

Здесь к стандартному выводу добавлено имя приложения.

Проверим.

```
=> SELECT webapi.login('alice');
```

```
login
-----
3fab093f-9045-4ffb-98a4-1e679a6e528c
(1 row)
```

```
=> BEGIN;
```

BEGIN

```
=> SELECT webapi.trace('3fab093f-9045-4ffb-98a4-1e679a6e528c');
```

```
trace
-----
(1 row)
```

```
=> SELECT 2+2;
```

```
?column?
-----
4
(1 row)
```

```
=> COMMIT;
```

COMMIT

```
student$ tail -n 1 /var/log/postgresql/postgresql-12-main.log
```

```
2022-12-09 23:51:48.194 MSK [43986] student@bookstore2 (client=alice) LOG: duration: 0.116 ms statement: SELECT 2+2;
```

Теперь этим функционалом можно пользоваться, устанавливая в приложении признак трассировки в служебной панели.

1. В двух сеансах подключитесь к базе student ролью student через пул соединений. В третьем сеансе подключитесь к консоли pgbouncer и изучите вывод команд SHOW CLIENTS и SHOW SERVERS.
2. Перепишите следующий фрагмент SQL-кода так, чтобы он корректно выполнялся при использовании пула соединений:

```
INSERT INTO master(id, s)
    SELECT nextval('m_seq'), 'm1';
INSERT INTO detail(id, m_id, s)
    SELECT nextval('d_seq'), currval('m_seq'), 'd1';
```
3. Воспроизведите проблему с рекомендательными блокировками уровня сеанса при использовании пула соединений в режиме транзакций.

### 1. Консоль pgbouncer

Подключаемся к базе student полями student и web:

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

```
student$ psql postgresql://student@localhost:6432/student?password=student
```

Подключаемся к консоли pgbouncer:

```
student$ psql postgresql://student@localhost:6432/pgbouncer?password=student
```

```
=> SHOW CLIENTS;
```

type	user	database	state	addr	port	local_addr	local_port	connect_time	request_time	wait	wait_us	close_needed	ptr
C	student	pgbouncer	active	127.0.0.1	46070	127.0.0.1	6432	2022-12-09 23:51:51 MSK	2022-12-09 23:51:51 MSK	0	0	0	0x55de30ed4460
C	student	student	active	127.0.0.1	46064	127.0.0.1	6432	2022-12-09 23:51:50 MSK	2022-12-09 23:51:50 MSK	0	0	0	0x55de30ed4000
C	student	student	active	127.0.0.1	46068	127.0.0.1	6432	2022-12-09 23:51:50 MSK	2022-12-09 23:51:50 MSK	0	0	0	0x55de30ed4230
(3 rows)													

Здесь отображаются три соединения к клиентам pgbouncer: два сеанса для student, подключенных к базе student, и один — подключение к консоли.

```
=> SHOW SERVERS;
```

type	user	database	state	addr	port	local_addr	local_port	connect_time	request_time	wait	wait_us	close_needed	ptr	l
S	student	student	idle	127.0.0.1	5432	127.0.0.1	59886	2022-12-09 23:51:50 MSK	2022-12-09 23:51:50 MSK	0	0	0	0x55de30edaf20	1
(1 row)														

Здесь видно, что сейчас используется только одно соединение pgbouncer с сервером баз данных.

```
=> \q
```

```
| => \q
```

```
|| => \q
```

### 2. Функции nextval и currval

Все, что нужно сделать — заключить обе команды в одну транзакцию.

Но на практике не стоит напрямую вызывать функции управления последовательностями без особой необходимости. Лучше объявить автоматическую генерацию уникальных значений, как это определено стандартом SQL:

```
student$ psql
```

```
=> CREATE DATABASE ext_connpool;
```

```
CREATE DATABASE
```

```
=> \c ext_connpool
```

```
You are now connected to database "ext_connpool" as user "student".
```

```
=> CREATE TABLE master(
  id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  s text
);
```

```
CREATE TABLE
```

```
=> CREATE TABLE detail(
  id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  m_id integer REFERENCES master(id),
  s text
);
```

```
CREATE TABLE
```

А вставку можно организовать с помощью PL/pgSQL следующим образом:

```
=> DO $$
DECLARE
  m_id integer;
BEGIN
  INSERT INTO master(s) VALUES ('m1') RETURNING id INTO m_id;
  INSERT INTO detail(m_id, s) VALUES (m_id, 'd1');
END;
$$;
DO
=> \q
```

### 3. Рекомендательные блокировки на уровне сеанса

Первый клиент устанавливает рекомендательную блокировку на уровне сеанса:

```
student$ psql postgresql://student@localhost:6432/ext_connpool?password=student
```

```
=> BEGIN;
```

```
BEGIN
```

```
=> SELECT pg_backend_pid();
```

```
pg_backend_pid
-----
44923
```

```
(1 row)
```

```
=> SELECT pg_advisory_lock(42);
```

```
pg_advisory_lock
-----
```

```
(1 row)
```

```
=> END;
```

```
COMMIT
```

Блокировка удерживается после окончания транзакции:

```
=> SELECT objid FROM pg_locks WHERE locktype = 'advisory';
```

```
objid
-----
42
```

```
(1 row)
```

Теперь появляется транзакция второго клиента:

```
student$ psql postgresql://student@localhost:6432/ext_connpool?password=student
```

```
| => BEGIN;
```



```
| BEGIN
```

В это время первый клиент пытается освободить блокировку, но не может, поскольку выполняется в другом сеансе:

```
=> SELECT pg_advisory_unlock(42);
```

```
WARNING: you don't own a lock of type ExclusiveLock
pg_advisory_unlock
```

```
-----
f
(1 row)
```

```
=> SELECT pg_backend_pid();
```

```
pg_backend_pid
-----
45108
(1 row)
```

Зато может второй клиент, хоть он и не устанавливал эту блокировку:

```
| => SELECT pg_advisory_unlock(42);
```

```
pg_advisory_unlock
-----
t
(1 row)
```

```
| => COMMIT;
```

```
| COMMIT
```