МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Интеллектуальные системы информационной безопасности»

Политики безопасности баз данных

Практическая работа № 4

ОТЧЁТ

Выполнил студент группы
БСБО-07-20
Любовский С.В.

Москва 2023

Выполнение задания 1.

1. Создать новую базу данных с именем wal_db.

   CREATE DATABASE wal_db;

2. Создать таблицу orders с колонками:
   a. id: уникальный идентификатор заказа (целое число, первичный ключ, автоинкремент)
   b. customer_name: имя клиента (varchar(255))
   c. order_date: дата размещения заказа (date)
   d. total_amount: общая сумма заказа (numeric)

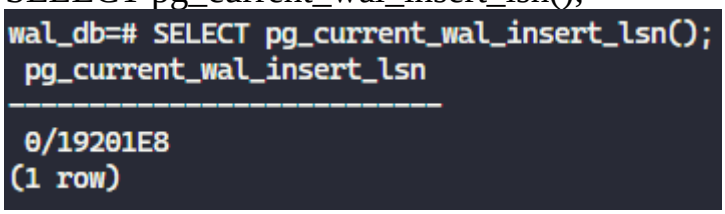
   ```
   CREATE TABLE orders (
      id serial primary key,
      customer_name varchar(255),
      order_date date,
      total_amount numeric(16, 2)
   );
   ```

3. Вставить в таблицу orders несколько примеров данных.

   ```
   INSERT INTO orders (customer_name, order_date, total_amount)
   VALUES
      ('ivan', '02-02-2022'::date, 1010.55),
      ('petr', '03-03-2022'::date, 999.99),
      ('nastya', '04-04-2022'::date, 949.49);
   ```

4. Узнать сколько байт занимают сгенерированные журнальные записи

   SELECT pg_current_wal_insert_lsn();

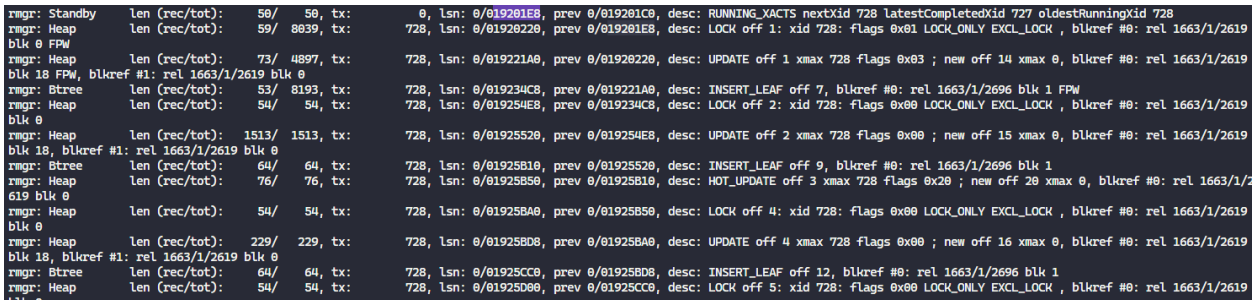
5. Изменить некоторые из существующих записей в таблице orders

   UPDATE orders SET total_amount = total_amount + 500 WHERE order_date > '01-03-2022'::date;

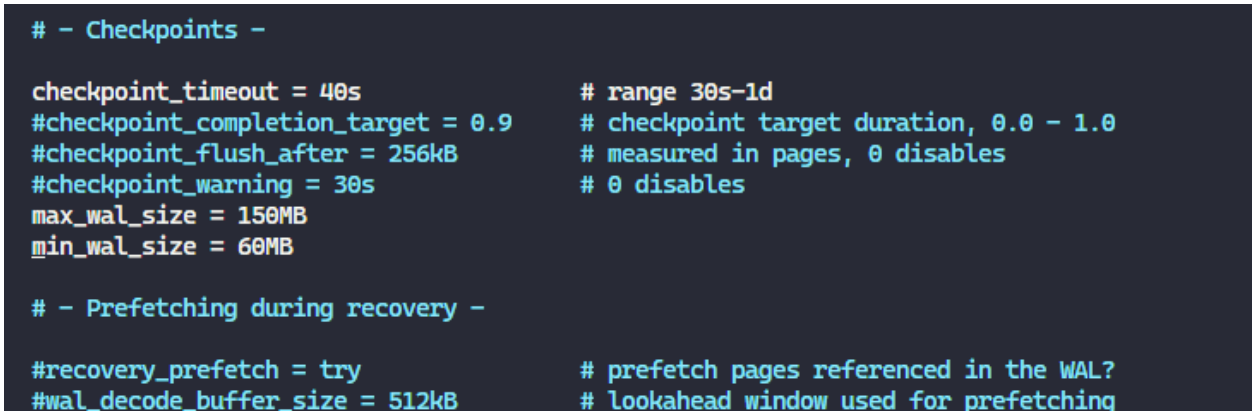
6. Удалить несколько записей из таблицы orders

   DELETE FROM orders WHERE customer_name = 'nastya';

7. Проверить содержимое файлов журнала WAL, чтобы увидеть записи операций, выполненных в шагах 4-6 (Для проверки содержимого файлов журнала WAL можно воспользоваться утилитой pg_waldump).

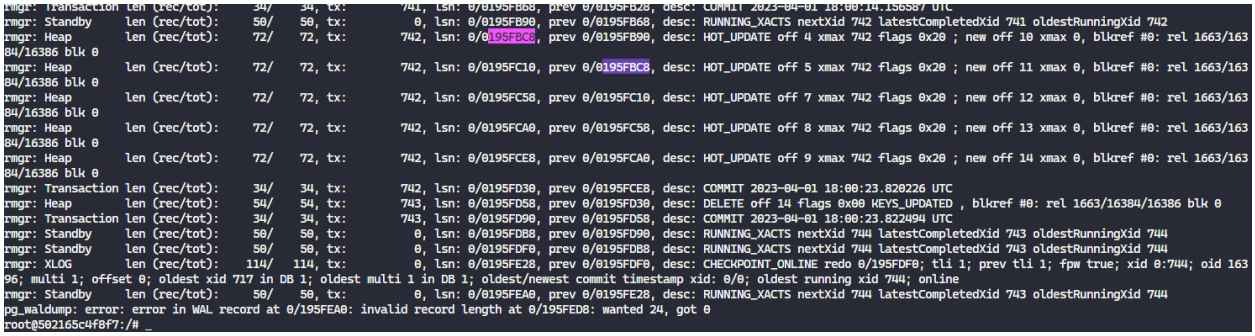


8. Настроить параметры, связанные с WAL, такие как max_wal_size и checkpoint_timeout в postgresql.conf.



9. Перезапустить сервер PostgreSQL, чтобы применить изменения в конфигурации

10. Повторить шаги 4-7, чтобы наблюдать изменения в файлах WAL из-за новых параметров конфигурации.



Выполнение задания 2.

1. Создайте новую базу данных benchmark в PostgreSQL, которая будет использоваться для тестирования производительности с помощью pgbench.

   CREATE DATABASE benchmark;

2. Настройте базу данных benchmark используя утилиту pgbench с параметрами -i и -s 50 (объясните что делают эти параметры)

   Ключ -i отвечает за инициализацию тестовых таблиц
   Ключ -s отвечает за масштабирование тестов, умножает количество генерируемых строк на этот коэффициент

```
root@382c8755f7a1:/# pgbench -i -s 50 benchmark;
pgbench: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL:  role "root" does not exist
pgbench: error: could not create connection for initialization
root@382c8755f7a1:/# pgbench -i -s 50 -U postgres benchmark;
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)...
5000000 of 5000000 tuples (100%) done (elapsed 9.37 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 12.33 s (drop tables 0.00 s, create tables 0.03 s, client-side generate 9.52 s, vacuum 0.23 s, primary keys 2.56 s).
```

3. Запустите команду pgbench на созданной таблице с разными параметрами, такими как количество клиентов, количество транзакций и длительность тестирования.

   Запуск с 100 клиентов и 30 транзакциями

```
root@382c8755f7a1:/# pgbench -c 100 -t 30 -U postgres benchmark
pgbench (15.2 (Debian 15.2-1.pgdg110+1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 100
number of threads: 1
maximum number of tries: 1
number of transactions per client: 30
number of transactions actually processed: 3000/3000
number of failed transactions: 0 (0.000%)
latency average = 39.985 ms
initial connection time = 436.484 ms
tps = 2500.912833 (without initial connection time)
```

   Запуск с 80 клиентов и 10 секундами продолжительности тестов

```
root@382c8755f7a1:/# pgbench -c 80 -T 10 -U postgres benchmark
pgbench (15.2 (Debian 15.2-1.pgdg110+1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 80
number of threads: 1
maximum number of tries: 1
duration: 10 s
number of transactions actually processed: 43028
number of failed transactions: 0 (0.000%)
latency average = 18.110 ms
initial connection time = 339.405 ms
tps = 4417.375228 (without initial connection time)
```

4. Проанализируйте результаты тестирования, смотрите на показатели, такие как количество транзакций в секунду, время выполнения транзакций и общее время работы

   В первом случае тесты были запущены для фиксированного количества транзакций (30 транзакций на клиент – 3000 в сумме). Среднее время на транзакцию у одного клиента – 39мс. Количество транзакций в секунду – 2500

   Во втором случае тесты были запущенны на фиксированное время и 80 клиентами. Среднее время одной транзакции составило – 18мс, количество в секунду – 4417.

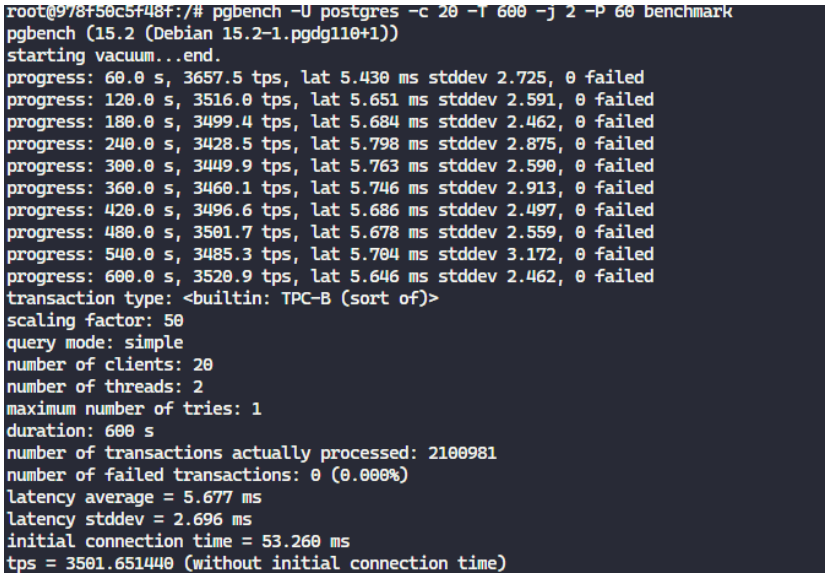
5. Попробуйте изменить параметры pgbench и повторить тестирование, чтобы понять, как они влияют на производительность базы данных. Используйте различные значения для параметров, а также изменяйте конфигурационные параметры PostgreSQL, такие как shared_buffers и work_mem, чтобы увидеть, как это влияет на производительность.

   Проведем тестирование на стандартной конфигурации. Будем использовать 20 подключений 2 потока и тест продолжительностью 10 минут, чтобы получить +- воспроизводимые результаты.

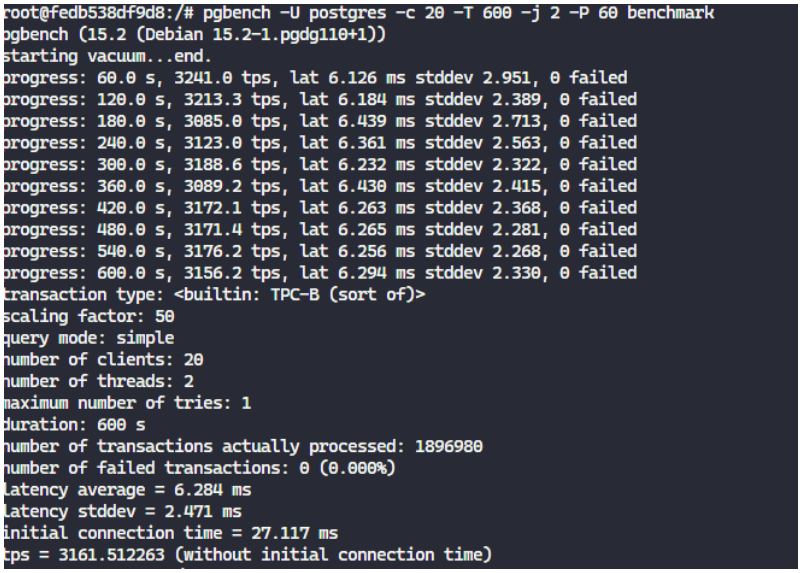
```
root@382c8755f7a1:/# pgbench -U postgres -c 20 -T 600 -j 2 -P 60 benchmark
pgbench (15.2 (Debian 15.2-1.pgdg110+1))
starting vacuum...end.
progress: 60.0 s, 2566.2 tps, lat 7.737 ms stddev 7.482, 0 failed
progress: 120.0 s, 3033.6 tps, lat 6.552 ms stddev 10.646, 0 failed
progress: 180.0 s, 3188.0 tps, lat 6.237 ms stddev 10.028, 0 failed
progress: 240.0 s, 3111.6 tps, lat 6.384 ms stddev 8.050, 0 failed
progress: 300.0 s, 3099.6 tps, lat 6.412 ms stddev 6.118, 0 failed
progress: 360.0 s, 3053.7 tps, lat 6.509 ms stddev 6.936, 0 failed
progress: 420.0 s, 2986.8 tps, lat 6.653 ms stddev 5.165, 0 failed
progress: 480.0 s, 2648.3 tps, lat 7.500 ms stddev 10.647, 0 failed
progress: 540.0 s, 2669.8 tps, lat 7.440 ms stddev 6.650, 0 failed
progress: 600.0 s, 2760.3 tps, lat 7.199 ms stddev 10.782, 0 failed
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 20
number of threads: 2
maximum number of tries: 1
duration: 600 s
number of transactions actually processed: 1747093
number of failed transactions: 0 (0.000%)
latency average = 6.824 ms
latency stddev = 8.490 ms
initial connection time = 44.406 ms
tps = 2911.900579 (without initial connection time)
```

   Получили 2911 tps. Теперь увеличим объем памяти который postgresql использует для кэширования. За это отвечает параметр shared_buffers.

По умолчанию он равен 128MB но сейчас большинство компьютеров имеют объемы памяти большие на порядки. Установим значения 512MB и посмотрим что получится.
Полученный результат на ~600 транзакций в секунду больше.



Попробуем увеличить еще в 2 раза до 1024MB
В этот раз результаты оказались неожиданными, tps просел на 400.



Возможно это связано с тем, что запуск производился на ноутбуке в WSL и условия могут меняться от количества зарядки и других факторов. Но в целом дальнейшее увеличение этого параметра не сильно улучшит ситуацию.

Выполнение задания 3.

1. Для базы данных из первого задания настройте выполнение контрольной точки раз в 30 секунд. Установите параметры min_wal_size и max_wal_size в 16 МБ.

```
# - Checkpoints -

checkpoint_timeout = 30s                # range 30s-1d
#checkpoint_completion_target = 0.9     # checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 256kB         # measured in pages, 0 disables
#checkpoint_warning = 30s               # 0 disables
max_wal_size = 16MB
min_wal_size = 16MB_

# - Prefetching during recovery -
```

min_wal_size и max_wal_size должны быть минимум в два раза больше чем wal_segment_size. Получаем ошибку.

`wal_segment_size` (integer)

Сообщает число блоков (страниц) в файле сегмента WAL. Общий размер файла сегмента WAL равняется произведению `wal_segment_size` и `wal_block_size`; по умолчанию это 16 мегабайт. За дополнительными сведениями обратитесь к Разделу 30.4.

```
Attaching to dsp-postgres
dsp-postgres  |
dsp-postgres  | PostgreSQL Database directory appears to contain a database; Skipping initialization
dsp-postgres  |
dsp-postgres  | 2023-04-02 19:18:31.238 UTC [1] FATAL:  "min_wal_size" must be at least twice "wal_segment_size"
dsp-postgres  | 2023-04-02 19:18:31.238 UTC [1] LOG:  database system is shut down
```

Установим 32MB

2. Несколько минут с помощью утилиты pgbench подавайте нагрузку 100 транзакций/сек.

```
root@2764eb932166:/# pgbench -U postgres -T 180 -R 100 wal_db
pgbench (15.2 (Debian 15.2-1.pgdg110+1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 180 s
number of transactions actually processed: 18210
number of failed transactions: 0 (0.000%)
latency average = 7.555 ms
latency stddev = 5.935 ms
rate limit schedule lag: avg 2.834 (max 102.727) ms
initial connection time = 3.210 ms
tps = 101.168016 (without initial connection time)
```

3. Измерьте, какой объем журнальных файлов был сгенерирован за это время. Оцените, какой объем приходится в среднем на одну контрольную точку.

```
-rw-------  1 postgres postgres  16M Apr  2 19:32 0000000100000007000000004F
-rw-------  1 postgres postgres  16M Apr  2 19:31 0000000100000007000000050
```

При весе каждого дампа 16мб суммарный их объем будет равен 32мб. Контрольные точки срабатывают каждые 30 секунд. Учитывая время работы бенчмарка равного 3 минуты, то выходит, за одну минуту приходится объем равный 16мб, тогда за 30 секунд: 16мб / 2, что будет равно 8мб.

4. Проверьте данные статистики: все ли контрольные точки выполнялись по расписанию? Как можно объяснить полученный результат?

```
-rw-------  1 postgres postgres  16M Apr  2 19:47 0000000100000007000000005D
-rw-------  1 postgres postgres  16M Apr  2 19:48 0000000100000007000000005E
```

данные в таблице до бенчмарка

```
postgres=# select checkpoints_timed, checkpoints_req from pg_stat_bgwriter;
 checkpoints_timed | checkpoints_req
-------------------+-----------------
                37 |              41
```

после

```
postgres=# select checkpoints_timed, checkpoints_req from pg_stat_bgwriter;
 checkpoints_timed | checkpoints_req
-------------------+-----------------
                40 |              56
(1 row)
```

Видно что за время выполнения бенчмарка было запрошено и выполнено 15 чекпоинтов, по времени было выполнено 3. Вероятно, все дело в том, что система не успевает собирать их в дамп вовремя из-за небольшого размера max_wal_size.

5. Сбросьте настройки к значениям по умолчанию.

```
# - Checkpoints -

#checkpoint_timeout = 30s               # range 30s-1d
#checkpoint_completion_target = 0.9     # checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 256kB         # measured in pages, 0 disables
#checkpoint_warning = 30s               # 0 disables
#max_wal_size = 32MB
#min_wal_size = 32MB

# - Prefetching during recovery -

#recovery_prefetch = try                # prefetch pages referenced in the WAL?
#wal_decode_buffer_size = 512kB         # lookahead window used for prefetching
                                        # (change requires restart)
```