



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();  
wal_db=# SELECT pg_current_wal_insert_lsn();  
pg_current_wal_insert_lsn  
-----  
0/19201E8  
(1 row)
```

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).

```
rmgr: Standby len (rec/tot): 58/ 58, tx: 0, lsn: 0/019201E8, prev 0/019201C8, desc: RUNNING_XACTS nextXid 728 latestCompletedXid 727 oldestRunningXid 728
rmgr: Heap len (rec/tot): 59/ 8839, tx: 728, lsn: 0/01920228, prev 0/019201E8, desc: LOCK off 1: xid 728: flags 0x01 LOCK_ONLY EXCL_LOCK , blkref #0: rel 1663/1/2619
blk 0 FPM
rmgr: Heap len (rec/tot): 73/ 4897, tx: 728, lsn: 0/019221A8, prev 0/01920228, desc: UPDATE off 1 xmax 728 flags 0x03 ; new off 14 xmax 0, blkref #0: rel 1663/1/2619
blk 18 FPM, blkref #1: rel 1663/1/2619 blk 0
rmgr: Btree len (rec/tot): 53/ 8193, tx: 728, lsn: 0/019234C8, prev 0/019221A8, desc: INSERT_LEAF off 7, blkref #0: rel 1663/1/2696 blk 1 FPM
rmgr: Heap len (rec/tot): 54/ 54, tx: 728, lsn: 0/019254E8, prev 0/019234C8, desc: LOCK off 2: xid 728: flags 0x00 LOCK_ONLY EXCL_LOCK , blkref #0: rel 1663/1/2619
blk 0
rmgr: Heap len (rec/tot): 1513/ 1513, tx: 728, lsn: 0/01925528, prev 0/019254E8, desc: UPDATE off 2 xmax 728 flags 0x00 ; new off 15 xmax 0, blkref #0: rel 1663/1/2619
blk 18, blkref #1: rel 1663/1/2619 blk 0
rmgr: Btree len (rec/tot): 64/ 64, tx: 728, lsn: 0/01925B18, prev 0/01925528, desc: INSERT_LEAF off 9, blkref #0: rel 1663/1/2696 blk 1
rmgr: Heap len (rec/tot): 76/ 76, tx: 728, lsn: 0/01925B58, prev 0/01925B18, desc: HOT_UPDATE off 3 xmax 728 flags 0x20 ; new off 20 xmax 0, blkref #0: rel 1663/1/2619
blk 0
rmgr: Heap len (rec/tot): 54/ 54, tx: 728, lsn: 0/01925BA8, prev 0/01925B58, desc: LOCK off 4: xid 728: flags 0x00 LOCK_ONLY EXCL_LOCK , blkref #0: rel 1663/1/2619
blk 0
rmgr: Heap len (rec/tot): 229/ 229, tx: 728, lsn: 0/01925BD8, prev 0/01925BA8, desc: UPDATE off 4 xmax 728 flags 0x00 ; new off 16 xmax 0, blkref #0: rel 1663/1/2619
blk 18, blkref #1: rel 1663/1/2619 blk 0
rmgr: Btree len (rec/tot): 64/ 64, tx: 728, lsn: 0/01925CC8, prev 0/01925BD8, desc: INSERT_LEAF off 12, blkref #0: rel 1663/1/2696 blk 1
rmgr: Heap len (rec/tot): 54/ 54, tx: 728, lsn: 0/01925D08, prev 0/01925CC8, desc: LOCK off 5: xid 728: flags 0x00 LOCK_ONLY EXCL_LOCK , blkref #0: rel 1663/1/2619
blk 0
```

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

```
# - Checkpoints -

checkpoint_timeout = 40s           # 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 = 150MB
min_wal_size = 60MB

# - Prefetching during recovery -

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

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.

```
rmgr: Transaction len (rec/tot): 34/ 34, tx: 741, lsn: 0/0195FB68, prev 0/0195FB28, desc: COMMIT 2023-04-01 18:08:14.156887 UTC
rmgr: Standby len (rec/tot): 59/ 59, tx: 0, lsn: 0/0195FB90, prev 0/0195FB68, desc: RUNNING_XACTS nextXid 742 latestCompletedXid 741 oldestRunningXid 742
rmgr: Heap len (rec/tot): 72/ 72, tx: 742, lsn: 0/0195FBC8, prev 0/0195FB90, desc: HOT_UPDATE off 4 xmax 742 flags 0x20 ; new off 10 xmax 0, blkref #0: rel 1663/16384/16386 blk 0
rmgr: Heap len (rec/tot): 72/ 72, tx: 742, lsn: 0/0195FC10, prev 0/0195FBC8, desc: HOT_UPDATE off 5 xmax 742 flags 0x20 ; new off 11 xmax 0, blkref #0: rel 1663/16384/16386 blk 0
rmgr: Heap len (rec/tot): 72/ 72, tx: 742, lsn: 0/0195FCC8, prev 0/0195FC10, desc: HOT_UPDATE off 7 xmax 742 flags 0x20 ; new off 12 xmax 0, blkref #0: rel 1663/16384/16386 blk 0
rmgr: Heap len (rec/tot): 72/ 72, tx: 742, lsn: 0/0195FCA0, prev 0/0195FCC8, desc: HOT_UPDATE off 8 xmax 742 flags 0x20 ; new off 13 xmax 0, blkref #0: rel 1663/16384/16386 blk 0
rmgr: Heap len (rec/tot): 72/ 72, tx: 742, lsn: 0/0195FCE8, prev 0/0195FCA0, desc: HOT_UPDATE off 9 xmax 742 flags 0x20 ; new off 14 xmax 0, blkref #0: rel 1663/16384/16386 blk 0
rmgr: Transaction len (rec/tot): 34/ 34, tx: 742, lsn: 0/0195FD30, prev 0/0195FCE8, desc: COMMIT 2023-04-01 18:08:23.828226 UTC
rmgr: Heap len (rec/tot): 54/ 54, tx: 743, lsn: 0/0195FD58, prev 0/0195FD30, desc: DELETE off 14 flags 0x00 KEYS_UPDATED , blkref #0: rel 1663/16384/16386 blk 0
rmgr: Transaction len (rec/tot): 34/ 34, tx: 743, lsn: 0/0195FD90, prev 0/0195FD58, desc: COMMIT 2023-04-01 18:08:23.822494 UTC
rmgr: Standby len (rec/tot): 59/ 59, tx: 0, lsn: 0/0195FDB8, prev 0/0195FD90, desc: RUNNING_XACTS nextXid 744 latestCompletedXid 743 oldestRunningXid 744
rmgr: Standby len (rec/tot): 59/ 59, tx: 0, lsn: 0/0195FDF0, prev 0/0195FDB8, desc: RUNNING_XACTS nextXid 744 latestCompletedXid 743 oldestRunningXid 744
rmgr: XLOG len (rec/tot): 114/ 114, tx: 0, lsn: 0/0195FE28, prev 0/0195FDF0, desc: CHECKPOINT_ONLINE redo 0/195FDF0; tli 1; prev tli 1; fpm true; xid 0:744; oid 16396; multi 1; offset 0; oldest xid 717 in DB 1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; oldest running xid 744; online
rmgr: Standby len (rec/tot): 59/ 59, tx: 0, lsn: 0/0195FEA0, prev 0/0195FE28, desc: RUNNING_XACTS nextXid 744 latestCompletedXid 743 oldestRunningXid 744
pg_waldump: error: error in WAL record at 0/195FEA0: invalid record length at 0/195FED8: wanted 24, got 0
root@502165c4f8f7:/#
```

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 запущены 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 = 2911.988579 (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 128МВ но сейчас большинство компьютеров имеют объемы памяти большие на порядки. Установим значения 512МВ и посмотрим что получится.

Полученный результат на ~600 транзакций в секунду больше.

```
root@978f50c5f48f:/# 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, 3657.5 tps, lat 5.430 ms stddev 2.725, 0 failed
progress: 120.0 s, 3516.0 tps, lat 5.651 ms stddev 2.591, 0 failed
progress: 180.0 s, 3499.4 tps, lat 5.684 ms stddev 2.462, 0 failed
progress: 240.0 s, 3428.5 tps, lat 5.798 ms stddev 2.875, 0 failed
progress: 300.0 s, 3449.9 tps, lat 5.763 ms stddev 2.590, 0 failed
progress: 360.0 s, 3460.1 tps, lat 5.746 ms stddev 2.913, 0 failed
progress: 420.0 s, 3496.6 tps, lat 5.686 ms stddev 2.497, 0 failed
progress: 480.0 s, 3501.7 tps, lat 5.678 ms stddev 2.559, 0 failed
progress: 540.0 s, 3485.3 tps, lat 5.704 ms stddev 3.172, 0 failed
progress: 600.0 s, 3520.9 tps, lat 5.646 ms stddev 2.462, 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: 2100981
number of failed transactions: 0 (0.000%)
latency average = 5.677 ms
latency stddev = 2.696 ms
initial connection time = 53.260 ms
tps = 3501.651440 (without initial connection time)
```

Попробуем увеличить еще в 2 раза до 1024МВ

В этот раз результаты оказались неожиданными, tps просел на 400.

```
root@fedb538df9d8:/# 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, 3241.0 tps, lat 6.126 ms stddev 2.951, 0 failed
progress: 120.0 s, 3213.3 tps, lat 6.184 ms stddev 2.389, 0 failed
progress: 180.0 s, 3085.0 tps, lat 6.439 ms stddev 2.713, 0 failed
progress: 240.0 s, 3123.0 tps, lat 6.361 ms stddev 2.563, 0 failed
progress: 300.0 s, 3188.6 tps, lat 6.232 ms stddev 2.322, 0 failed
progress: 360.0 s, 3089.2 tps, lat 6.430 ms stddev 2.415, 0 failed
progress: 420.0 s, 3172.1 tps, lat 6.263 ms stddev 2.368, 0 failed
progress: 480.0 s, 3171.4 tps, lat 6.265 ms stddev 2.281, 0 failed
progress: 540.0 s, 3176.2 tps, lat 6.256 ms stddev 2.268, 0 failed
progress: 600.0 s, 3156.2 tps, lat 6.294 ms stddev 2.330, 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: 1896980
number of failed transactions: 0 (0.000%)
latency average = 6.284 ms
latency stddev = 2.471 ms
initial connection time = 27.117 ms
tps = 3161.512263 (without initial connection time)
```

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 | 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

- 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)
```

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

```

drwx----- 1 postgres postgres 4.0K Apr 2 19:21 .
-rw----- 1 postgres postgres 16M Apr 2 19:32 0000000010000000700000004F
-rw----- 1 postgres postgres 16M Apr 2 19:31 00000000100000007000000050
drwx----- 1 postgres postgres 4.0K Apr 1 18:33 archive_status

```

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: $16\text{мб} / 2$, 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 0000000100000000700000005D
-rw----- 1 postgres postgres 16M Apr  2 19:48 0000000100000000700000005E
```

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 дампы вовремя из-за небольшого размера `max wal size`.

5. Сбросьте настройки к значениям по умолчанию.

[illegible]