МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**МИРЭА − Российский технологический университет**»

РТУ МИРЭА

Институт комплексной безопасности и цифровых технологий (ИКБ)
Кафедра КБ-14 «Цифровые технологии обработки данных»

Направление подготовки

09.03.02 Информационные системы и технологии

Практическая работа №6-8

ОТЧЕТ

Выполнил студент группы:

БСБО-07-20

Любовский С.В.

Москва 2023г.

Выполнение задания

Компания, в рамках развития своей IT-инфраструктуры, нуждается в улучшении и настройках политики безопасности баз данных и системе автоматических бэкапов

На основе эталонной модели внести следующие изменения или дополнения:

1. В системе должен присутствовать отдельный кластер для как минимум одной таблицы, которая будет часто использоваться (добавление, изменение и удаление данных).

```
CREATE INDEX printers_idx ON printers(printer_name);
CLUSTER  printers USING printers_idx;
```

2. Настроить транзакционные параметры и фактор заполнения для оптимизации производительности.

```
-- Установить транзакционный уровень READ COMMITTED
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-- Установить фактор заполнения 80% для таблицы printers
ALTER TABLE printers SET (fillfactor = 80);
```

3. Предоставить отдельные допуски для доступа к кластеру.

```
CREATE ROLE worker;
CREATE ROLE manager;

GRANT SELECT, INSERT ON printers TO worker;
GRANT SELECT, INSERT, UPDATE ON printers TO manager;
```

4. На основном кластере должна быть настроена автоматическая очистка и бэкапирование для как минимум 2 таблиц, которые являются важными для работы системы и могут меняться со временем.

```
-- Настройка автоматической очистки

-- Установить параметр autovacuum для таблицы important_table1
ALTER TABLE employees SET (autovacuum_enabled = on);
ALTER TABLE contact_persons SET  (autovacuum_enabled  = on);

-- Установить параметры max_worker_processes,
max_parallel_workers_per_gather и max_parallel_workers в конфигурационном
файле
-- для управления параллельным выполнением очистки таблиц
ALTER SYSTEM SET max_worker_processes = 6;
ALTER SYSTEM SET max_parallel_workers_per_gather = 2;
ALTER SYSTEM SET max_parallel_workers = 4;

-- Установить параметры autovacuum_vacuum_scale_factor и
autovacuum_analyze_scale_factor
-- для выполнения очистки и анализа таблицы important_table1
ALTER TABLE employees SET (autovacuum_vacuum_scale_factor = 0.1,
autovacuum_analyze_scale_factor = 0.05);
ALTER TABLE contact_persons SET (autovacuum_vacuum_scale_factor = 0.1,
autovacuum_analyze_scale_factor = 0.05);
```

Подготовим следующий shell скрипт для бэкапирования таблиц:

```
#!/bin/bash
pg_dump.exe -t employees -t contact_persons -U postgres -f "/backups/db.bak"
clients_database
echo Команда выполнена.
```

Далее в crontab -e выставляем следующие параметры:

```
0 3 * * * /sripts/db_back.sh
```

5. Настроить гибкую настраиваемую автоочистку и бэкапирование всей системы.

Пример настройки параметров очистки в конфигурационном файле:

```
autovacuum = on

autovacuum_naptime = 3min
autovacuum_vacuum_threshold = 100

autovacuum_analyze_threshold = 50
```

```
autovacuum_vacuum_scale_factor = 0.2
autovacuum_analyze_scale_factor = 0.1
```

Далее делаем то же самое:
Скрипт:

```
#!/bin/bash
pg_dumpall.exe -U postgres -f "/backups/db_all.bak"
echo Команда выполнена.
```

Crontab:

```
0 1 * * * /sripts/db_back_all.sh
```

6. Необходимо вести журналы посещения пользователей, отслеживая их активность в системе, и журналы использования самой нагруженной таблицы для анализа производительности

Выставляем следующие параметры в конф. файле:

```
log_connections = on
log_disconnections = on
log_statement = 'all'
```

Данные параметры отвечают за логирование действий, происходящих в базе данных. log_connections включает логирование подключений к базе данных, log_disconnections включает логирование отключений от базы данных, а log_statement включает логирование SQL-запросов, которые отправляются к базе данных.

7. Необходимо определить производительность системы при различных уровнях нагрузки. Для проведения нагрузочного тестирования базы данных использовать pgbench.

Подготовка конфигурации к тестам:

```
root@1af4168edd1b:/# pgbench -U postgres -i -s 50 benchmark -h
/usr/lib/postgresql/15/bin/pgbench: option requires an argument -- 'h'
pgbench: hint: Try "pgbench --help" for more information.
root@1af4168edd1b:/# pgbench -U postgres -i -s 50 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 7.97 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 10.79 s (drop tables 0.00 s, create tables 0.02 s, client-side generate 8.13 s, vacuum 0.23 s, primary keys 2.40 s).
```

Тестирование с разным уровнем нагрузки:

```
root@1af4168edd1b:/# pgbench -U postgres -c 10 -t 50 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: 10
number of threads: 1
maximum number of tries: 1
number of transactions per client: 50
number of transactions actually processed: 500/500
number of failed transactions: 0 (0.000%)
latency average = 8.090 ms
initial connection time = 39.588 ms
tps = 1236.026718 (without initial connection time)
root@1af4168edd1b:/# pgbench -U postgres -c 20 -t 200 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: 20
number of threads: 1
maximum number of tries: 1
number of transactions per client: 200
number of transactions actually processed: 4000/4000
number of failed transactions: 0 (0.000%)
latency average = 9.527 ms
initial connection time = 79.966 ms
tps = 2099.361742 (without initial connection time)
root@1af4168edd1b:/# pgbench -U postgres -c 50 -t 500 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: 50
number of threads: 1
maximum number of tries: 1
number of transactions per client: 500
number of transactions actually processed: 25000/25000
number of failed transactions: 0 (0.000%)
latency average = 13.845 ms
initial connection time = 203.939 ms
tps = 3611.439190 (without initial connection time)
root@1af4168edd1b:/# _
```

8. Разработать стратегию мониторинга и оповещения для контроля состояния системы и оперативного реагирования на возможные проблемы. Это включает мониторинг производительности, использования ресурсов и состояния журналов, а также оповещения при превышении пороговых значений или возникновении ошибок.

```
import psutil
import time

MEMORY_THRESHOLD = 80.0
CPU_THRESHOLD = 80.0
DISK_THRESHOLD = 80.0

while True:
    # мониторинг использования ресурсов процесса Postgres
    for proc in psutil.process_iter(["pid", "name", "cpu_percent",
"memory_percent"]):
        if proc.info["name"] == "postgres":
            pid = proc.info["pid"]
            cpu_percent = proc.info["cpu_percent"]
            mem_percent = proc.info["memory_percent"]
            mem_info = proc.memory_info()
            rss = mem_info.rss / (1024 * 1024)  # в мегабайтах
            vms = mem_info.vms / (1024 * 1024)  # в мегабайтах
            print(
                f"Postgres process (PID {pid}): CPU usage: {cpu_percent}%, "
                f"memory usage: {mem_percent}% ({rss:.2f} MB RSS, {vms:.2f}
MB VMS)"
            )

            if mem_percent > MEMORY_THRESHOLD:
                print(
                    f"Postgres process (PID {pid}) "
                    f"is using too much memory: {mem_percent}%"
                )

            if cpu_percent > CPU_THRESHOLD:
                print(
                    f"Postgres process (PID {pid}) "
                    f"is using too much CPU: {cpu_percent}%",
                    color="yellow",
                )

    cpu_percent = psutil.cpu_percent()
    mem = psutil.virtual_memory()
    mem_percent = mem.percent
    mem_used = mem.used / (1024 * 1024 * 1024)  # в гигабайтах
    disk = psutil.disk_usage("/")
    disk_percent = disk.percent
```

```
    disk_used = disk.used / (1024 * 1024 * 1024)  # в гигабайтах
    print(
        f"System usage: CPU usage: {cpu_percent}%, "
        f"memory usage: {mem_percent}% ({mem_used:.2f} GB used), "
        f"disk usage: {disk_percent}% ({disk_used:.2f} GB used)"
    )

    if mem_percent > MEMORY_THRESHOLD:
        print(f"System is using too much memory: {mem_percent}%")

    if cpu_percent > CPU_THRESHOLD:
        print(f"System is using too much CPU: {cpu_percent}%")

    if disk_percent > DISK_THRESHOLD:
        print(f"System is using too much disk: {disk_percent}%")

    time.sleep(60)
```

Данный код представляет собой бесконечный цикл, который мониторит использование ресурсов на уровне системы и процесса Postgres. Для мониторинга используется библиотека psutil. Если использование памяти, CPU или диска превышает заданные пороговые значения (MEMORY_THRESHOLD, CPU_THRESHOLD и DISK_THRESHOLD соответственно), то выводится сообщение с предупреждением о превышении порога. Для вывода цветного текста используется библиотека termcolor. Все проверки осуществляются в бесконечном цикле с интервалом в 60 секунд.