??????????????????

- MySQL
- Postgresql
- Архивация и отправка архивов

MySQL

Базы данных хранятся в /var/lib/mysql

```
sudo grep -R 'datadir' /etc/mysql/
```

Настройка сессии

Уровень изолированности	Черновое чтение	Неповторяющееся чтение	Фантомное чтение	Блокировка чтения
READ UNCOMMITTED	Да	Да	Да	Нет
READ COMMITTED	Нет	Да	Да	Нет
REPEATABLE READ	Нет	Нет	Да	Нет
SERIALIZABLE	Нет	Нет	Нет	Да

Можно установить на уровне сессии

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Взаимоблокировки: Innodb отказывает самой короткой транзакции
цели в трех ключевых областях: задержка, доступность и ошибки

Performance Schema

Это отдельный engine Проверить факт наличия:

```
SHOW ENGINES;
+--------------------+---------+----------------------------------+-------------+------
+------------+
| Engine             | Support | Comment                          | Transactions | XA   |
Savepoints |
+--------------------+---------+----------------------------------+-------------+------
+------------+
| PERFORMANCE_SCHEMA | YES     | Performance Schema               | NO           | NO   |
NO         |
```

включен или нет:

```
SHOW VARIABLES LIKE 'performance_schema';
```

Переменные окружения

Просмотр

```
SHOW VARIABLES LIKE 'performance_schema';
```

Поддерживает %

Настройки
Настройка при запуске

```
shell> mysql --max_allowed_packet=16M
```

Настройка при работе

```
mysql>SET GLOBAL max_connections = 1000; - потеря при перезагрузке
mysql>SET PERSIST max_connections = 1000; - сохранение после перезагрузки
```

Инструменты (setup_instruments) пишут в потребителя (таблицы данных,).
Схемой sys представления и хранимые подпрограммы над performance_schema.
включение инструментов вызывает дополнительный код

```
statement/sql/select;
wait/synch/mutex/innodb/autoinc_mutex
```

Крайний левый - тип, далее слева направо подсистемы от общей к частной.
Дайджест — агрегирование запросов путем удаления из них вариаций
Структура таблиц потребителей:

- *_current — события, происходящие на сервере в данный момент;
- *_history — последние 10 завершенных событий на поток;
- *_history_long — последние 10 000 завершенных событий на поток по всему миру

Ограничения:

- Инструментарий должен поддерживаться компонентом MySQL.
- Она собирает данные только после включения конкретного инструмента и потребителя.
- Трудно освободить память. Даже если позже отключите определенные инструменты или потребители, память не будет освобождена, пока вы не перезапустите сервер.

Настройка инструментов мониторинга

Список инструментов: https://mariadb.com/kb/en/performance-schema-setup_instruments-table/

Таблица setup_instruments - включенные инструменты.

Структура:

NAME	Название инструмента
ENABLED	Включен или нет
TIMED	Планировщик. Если disabled, работать не будет

CRUD db, tables, users

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newPassword';

CREATE DATABASE IF NOT EXISTS my_timeweb;
DROP DATABASE IF EXISTS my_timeweb;
CREATE USER 'sub_user'@'10.0.%' IDENTIFIED BY 'password';
SELECT * FROM mysql.user;
DROP USER 'user'@'localhost';
```

Привилегии:

ALTER	Менять структуру таблицы или БД
CREATE	Создавать новые БД и таблицы
DELETE	Удалять строки в таблице
INSERT	Добавлять строки в таблицу
SELECT	Читать данные из таблицы
UPDATE	Обновлять данные в таблице
DROP	Удалять БД
ALL PRIVILEGES	все, кроме GRANT;
USAGE PRIVILEGES	никаких привилегий;
FILE	разрешает читать файлы на сервере;
INDEX	создавать индексы для таблиц;

DROP	удалять таблицы;
EVENT	обработка событий;
TRIGGER	создание триггеров.

Привилегия на действия

GRANT	изменять права пользователей;
SUPER	суперпользователь;
PROCESS	получение информации о состоянии MySQL;
RELOAD	позволяет перезагружать таблицы привилегий;
SHUTDOWN	позволяет отключать или перезапускать базу данных;
SHOW DATABASES	просмотр списка баз данных;
LOCK TABLES	блокирование таблиц при использовании SELECT;
REFERENCES	создание внешних ключей для связывания таблиц;
CREATE USER	создание пользователей;

```
GRANT SELECT, INSERT ON my_timeweb.* TO 'user'@'localhost'; из-под root
GRANT ALL PRIVILEGES ON my_timeweb.* TO 'user'@'localhost';
REVOKE SELECT, INSERT ON my_timeweb.* FROM 'user'@'localhost';
```

применение изменений

```
flush privileges;
```

статус, приложения

mysqladmin version
mysqlshow mysql - список таблиц БД mysql

mysql_secure_installation
mysqladmin - административное
mysqlcheck - проверка
mysqldump - сохранение
mysqlimport - импорт текстовых файлов в нужную таблицу

mysqlpump - экспорт базы в sql файл
mysqlslap - статус загрузки сервера

--print-defaults выводит конфиг по умолчанию

Postgresql

Подключение:

```
psql
-U имя пользователя
-W пароль
    -d имя базы
```

Сброс пароля пользователя: в файле pg_dba.conf

Установка pg_dump pg_restore

```
sudo apt update && sudo apt upgrade
sudo apt install postgresql-client postgresql-client-common libpq-dev

pg_dump -h 127.0.0.1 -p 5437 -U dbuser -d dbname -F p > ~/tmpdb.sql
```

Команды

список пользователей

```
\du
SELECT usename, usesuper, usecreatedb FROM pg_catalog.pg_user;
```

создать пользователя

```
CREATE USER user_name WITH PASSWORD 'myPassword';
```

сменить пароль пользователя

```
ALTER USER user_name WITH PASSWORD 'new_password';
```

список баз данных

```
\l
SELECT datname FROM pg_database;
```

создать базу данных

```
CREATE DATABASE имя_БД;
drop database имя_БД
```

установить права пользователя на базу

```
GRANT ALL PRIVILEGES ON DATABASE database1 to dmosk;
```

список таблиц

```
\dt
```

Восстановление базы данных

```
CREATE DATABASE mainbase_restore WITH TEMPLATE template0 ENCODING 'UTF8';
-- Завершаем все подключения к mainbase
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE datname = 'mainbase' AND pid <> pg_backend_pid();

-- 3. Переименовываем оригинальную базу в mainbase_old
ALTER DATABASE mainbase RENAME TO mainbase_old;

-- ⚠ Также нужно завершить подключения к mainbase_restore, чтобы переименовать её
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE datname = 'mainbase_restore' AND pid <> pg_backend_pid();

-- 4. Переименовываем восстановленную базу в mainbase
ALTER DATABASE mainbase_restore RENAME TO mainbase;
```

????????? ? ????????? ???????

Задача: есть некритичный ко времени сервис с небольшим объемом данных. Необходимо настроить отправку резервных файлов раз в 15 минут на внешний сервис и отображение имен последних отправленных файлов на дашборде в Zabbix, при отсутствии файлов в течение 20 минут генерировать ошибку. При каждом получении файла отправлять сообщение в канал об успешном получении файлов.

В дальнейшем нужно добавить удаление старых архивов.

Решил сначала собрать образ и сохранить его на локальном хабе, затем запустить через compose.

Структура файлов проекта.

```
db_archiver (dir)
    app (dir)
        votes_db_arch (dir)
        .env (file)
        db_archiver (file)
    cron_job (file)
    Dockerfile (file)
    requirements.txt (file)
docker-compose.yml
```

Название	Назначение
db_archiver	Директория. Хранится исходный код, настройки для создания образа и архивы.
db_archiver/app	Директория. Исходный код и архивы
db_archiver/app/votes_db_arch	Директория. В ней локально будут сохраняться архивы перед отправкой на сервер.
db_archiver/app/.env	Переменные для скрипта
db_archiver/app/db_archiver.py	Скрипт архивации

Название	Назначение
db_archiver/cron_job	Настройка планировщика в образе. После сборки образа не нужен.
db_archiver/Dockerfile	После сборки образ не нужен.
db_archiver/requirements.txt	Дополнительные python пакеты. После сборки образа не нужен.
docker-compose.yml	

Состав файлов

.env

```
VOTE_DB_HOST=
VOTE_DB_PORT=
VOTE_DB_USER=
VOTE_DB_PASS=
VOTE_DB_NAME=

ARCHIVE_SERVER_IP=
ARCHIVE_SERVER_LOGIN=
ARCHIVE_SERVER_PASSWORD=
ARCHIVE_SERVER_DIR=

ZABBIX_SERVER_IP=
```

db_archiver.py

```
import subprocess
import os
from datetime import datetime
import logging
from dotenv import load_dotenv
import paramiko
from zabbix_utils import Sender
import time

# Константы и лог
LOCAL_ARCH_DIR = "votes_db_arch"

logging.basicConfig(level=logging.INFO, filename="votes_db_archive.log",filemode="w",
```

```
                    format="%(asctime)s %(levelname)s %(message)s")

# Загружаем переменные из .env
load_dotenv()

# Читаем переменные
host = os.getenv("VOTE_DB_HOST")
port = os.getenv("VOTE_DB_PORT", "5432")
user = os.getenv("VOTE_DB_USER")
password = os.getenv("VOTE_DB_PASS")
dbname = os.getenv("VOTE_DB_NAME")

arch_server = os.getenv("ARCHIVE_SERVER_IP")
arch_username = os.getenv("ARCHIVE_SERVER_LOGIN")
arch_password = os.getenv("ARCHIVE_SERVER_PASSWORD")
arch_dir = os.getenv("ARCHIVE_SERVER_DIR")

zabbix_ip = os.getenv("ZABBIX_SERVER_IP")

def create_file_name() -> str:
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    return f"{LOCAL_ARCH_DIR}/{timestamp}_backup.sql", f"{arch_dir}/{timestamp}_backup.sql"

def create_backup(fname):
    command = [
        "pg_dump",
        "-h", host,
        "-p", port,
        "-U", user,
        "-d", dbname,
        "-F", "p",
    ]
    # Подмешиваем пароль в окружение
    env = {
        **os.environ,
        "PGPASSWORD": password
    }
    is_ok = True
    try:
        with open(fname, "w", encoding="utf-8") as f:
```

```
            subprocess.run(command, env=env, stdout=f, check=True)

    except Exception as e:
        is_ok = False
        logging.error(f"Ошибка в создании бэкапа: {e}")
    return is_ok

def remove_transaction_timeout(outputfile) -> bool:
    """
    Удаляет строки 'SET transaction_timeout = 0;' из указанного файла.
    """
    is_ok = True
    try:
        # Читаем файл построчно
        with open(outputfile, "r", encoding="utf-8") as f:
            lines = f.readlines()

        # Фильтруем лишние строки
        cleaned_lines = [line for line in lines if line.strip() != "SET transaction_timeout =
0;"]

        # Перезаписываем файл
        with open(outputfile, "w", encoding="utf-8") as f:
            f.writelines(cleaned_lines)
    except Exception as e:
        is_ok = False
        logging.error(f"Ошибка в удалении строк: {e}")
    return is_ok

def send_file_via_ssh(outputfile, remote_path):
    is_ok = True
    try:
        # Подключаемся по SSH
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # автоматическое добавление
ключа
        ssh.connect(arch_server, port=22, username=arch_username, password=arch_password)

        # Используем SFTP для передачи файла
        sftp = ssh.open_sftp()
```

```
        sftp.put(outputfile, remote_path)
        sftp.close()

        ssh.close()
        logging.info(f"Бэкап успешно отправлен на {arch_server}:{remote_path}")
    except Exception as e:
        is_ok = False
        logging.error(f'Ошибка при отправке файла по SSH: {e}')
    return is_ok

def send_to_zabbix(local_output_file):
    sender = Sender(server=zabbix_ip, port=10051)
    d = datetime.now()
    unix_time = int(time.mktime(d.timetuple()))
    # Parameters: (host, key, value, clock)
    resp = sender.send_value('', 'filename', local_output_file, unix_time)
    if resp.failed == 0:
        # Print a success message along with the response time
        logging.info(f"Value sent successfully in {resp.time}")
    else:
        # Print a failure message
        logging.info("Failed to send value")
        logging.info(resp.details)

if __name__ == "__main__":
    local_output_file, remote_output_file = create_file_name()
    if not create_backup(local_output_file):
        raise SystemExit()
    if not remove_transaction_timeout(local_output_file):
        raise SystemExit()
    if not send_file_via_ssh(local_output_file, remote_output_file):
        raise SystemExit()
    if not send_to_zabbix(local_output_file):
        raise SystemExit()
```

cron_job

```
PATH=/usr/local/bin:/usr/bin:/bin
*/2 * * * * root cd /app && python3 db_archiver.py >> /var/log/cron.log 2>&1
```

* обязательно последняя пустая строка!

requirements.txt

```
dotenv
paramiko
zabbix-utils
```

Dockerfile

```
FROM python:3.11-slim-bookworm

WORKDIR /app
# Устанавливаем зависимости и zabbix-utils
RUN apt-get update && \
    apt-get install -y --no-install-recommends postgresql-client postgresql-client-common
libpq-dev cron && \
    rm -rf /var/lib/apt/lists/*
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt
# Копируем скрипт и настройки cron
COPY cron_job /etc/cron.d/zabbix-cron
# Даем права на выполнение cron-файла
RUN chmod 0644 /etc/cron.d/zabbix-cron && \
    touch /var/log/cron.log
CMD ["cron", "-f"]
```

Сборка образа:

```
docker build -t db-archiver:0.3.0 .

Сохранение в репозитории
docker login https://hub.bobrobotirk.ru
docker tag db-archiver:0.3.0 hub.bobrobotirk.ru/db-archiver:0.3.0
docker push hub.bobrobotirk.ru/db-archiver:0.3.0
```

Создание контейнера:

docker-compose.yml

```
db_archiver:
    image: hub.bobrobotirk.ru/db-archiver:0.3.0
    container_name: votes_archiver
    restart: unless-stopped  # Автоперезапуск при ошибках
    volumes:
      - ./db_archiver/app:/app
```

Теперь нужно настроить zabbix trapper и вывести его на дашборд. Все!