ИТМО

Итоговый проект
"Проектирование и создание базы данных на основе некоторого набора данных"

Кравченкова Елизавета
Bonustrack (Методы анализа данных)
Университет ИТМО. Высшая школа цифровой культуры
Санк-Петербург, Россия

30 июня 2024 г.

Оглавление

Цели и Задачи

Цель: Спроектировать полноценную, отказоусточивую базу данных.

Задачи:

1. Подготовка и проектирование
2. Развертывание СУБД PostgreSQL
3. Генерация данных
4. Создание сложных запросов
5. Оптимизация базы
6. Автоматизация бэкапов
7. Развертывание Patroni для достижения отказоустойчивости
8. Установка средств мониторинга

Этап 1. Подготовка и проектирование

Задача: Провести анализ предметной области и разработать начальную схему базы данных.

2.1 Предметная область. Функциональные требования

Тема, для которой я решила спроектировать базу данных: "Сервис для соревнований по бальным танцам". Он обеспечивает функции для организаторов, участников и зрителей.

2.1.1 Выделенные сущности:

Отделение - Соревнования могут идти несколько дней, каждый день имеет минимум 2 отделения. У отделения есть время начала и конца. В рамках отделения номер пары (неважно в скольких категориях она участвует) одинаковый. Зрители и участники покупают билеты на каждое отделение.

Категория - Категория начинается и заканчивается в одном отделении. Категория строго определяет набор исполняемых танцев (Программу и количество исполняемых танцев). Каждая категория может иметь несколько туров (1/8 финала, полуфинал, финал и тд). Число туров определяется исходя из количества зарегистрированных участников. Также категория строго ограничивает: тип участника (соло или пара), а также танцоры какого уровня могут принимать в ней участие. Допустимых уровней может быть несколько (Например в категории «до Д» разрешено участвовать танцорам H,A,D класса) или же всего одна (в категории «B класс» принимают участие только танцоры B класса). Пара не может выйти в категорию(уровень пары определяется классом партнерши и партнера) ниже классом. Также категория имеет ограничение по возрасту (возраст пары определяется старшим в ней).

Танцор - определяется именем, номером книжки федерации, уровнем класса по латине и стандарту. Танцор имеет возраст и пол.

Тип выступающего - определяется типом выступающего (чаще всего это пара человек, но бывает и один). Выступающий определяется клубом и тренером. Пара состоит из мужчины и женщины. Класс пары определяется классом партнера (или на один выше, если класс партнерши выше партнера на 2 уровня). Если тип высутпающего ProAm, то класс определяется опытом. Возраст пары определяется возрастом старшего в ней. Выступающий может принимать участие в нескольких категориях. Он имеет номер, который его идентифицирует на паркете в рамках отделения.

Судья - Судья имеет свой уникальный номер, который потом будет использоваться для публикации в протоколе. У судьи естть имя. Он может принимать участие в нескольких категориях.

Отборочные туры - Данные о выступлении пары, должны содержать оценку за каждый танец от каждого судьи для каждой пары в каждой категории, в каждом туре. Судьи ставят крест за танец, если считают, что пара достойна пройти в дальше. Чем больше крестов, тем лучше.

Финал - Данные о выступлении пары, должны содержать оценку за каждый танец от каждого судьи для каждой пары в каждой категории. В финале же судьи ставят место, на которое считают верным поставить танцующего. Чем меньше число, тем лучше. Также в финале иногда добавляется дополнительный танец с уровнем basic.

Прайс - Данные и стоимости входных билетов для участников и зрителей на конкретное отделение.

Билет - Определяется хэшом, по которому будет осуществлятся вход зрителей. Также хранятся данные об отделении и статусе в билете(для проверки на входе).

Музыка - Определятся ссылкой на песню. Имеет параметр для какого танца она и минимальным классом, которому можно ее ставить.

2.1.2 Функциональные требования:

Руководитель должен иметь возможность с помощью системы:

1. Получить полное расписание соревнований на каждый день. Оно должно включать отделения и входящие в него категории, время начала очного подтвержение регистрации на отделение, а также время начала и конца каждой категории.

2. Организовать регистрацию судей. Сбор их давнных (ФИО и номер категории). Проверка соответствия их категории требуемой.

3. Организовать регистрацию участников. Сбор данных предполагаемых участников (ФИО, номер книжки федерации, клуб, тренер, интересующие категории). При этом система дожна проверять подходят ли данные люди для участия. Это решается на основе соответствия участниками типу танцующего, возрастной категории, уровню мастерства.

4. Возможность подсчитать количество участников, в каждом отделении.

5. Возможность подсчитать количество участников, в каждой категории, исходя из этого по стандартным правилам уметь подсчитать количество требуемых туров.

6. Возможность присвоить номер спортсмену в день соревнований по факту его прибытия. Номер должен присваиваться на все категории в рамках отделения.

7. Определить количество туров для каждой категории по итогам регистрировавших участников. Количество туров определяется исходя из правил федерации: Таблица 8

8. Получить список номеров участников в конкретной категории и туре с распределением по заходам. Количество заходов определяется в зависимости от количества пар в каждом заходе. Количество пар в заходе должно быть одинаковое, плюс-минус одна пара. Количество пар в заходе зависит от размера площадки. Норматив определения количества пар в заходе из расчета на каждую пару - 25 кв. м для соревнований, включенных в ЕКП, и 20 кв. м для других соревнований. Максимальный лимит - не более 16 пар в заходе.
9. Получить список судей для каждой категории.
10. Получить отсортированный список номеров участников по их результатам в отборочном туре. Результат: сумма поставленных крестов.
11. Получить отсортированный список номеров участников по их результатам в финале. Результат высчитывается по скейтинг-системе.
12. Узнать данные участника по номеру.
13. Составить протокол для каждой категории и тура (который потом выкладывается в открытый доступ). В протоколе хранится следующая информация: отделение, категория, номер пары, номер тура, оценка каждого судьи за каждый танец, итоговый результат.
14. Получение музыки для конкретной категории и танца.

Судья должен иметь возможность с помощью системы:

1. Получить полное расписание соревнований на каждый день.
2. Зарегистрироваться на конкретную категорию.
3. Получить полное расписание категорий, на которые он зарегистрировался.
4. Поставить крест паре под номером x в заданном отделении, категории, туре, танце.
5. Поставить место паре под номером x в заданном отделении, категории, танце.

Участник должен иметь возможность с помощью системы:

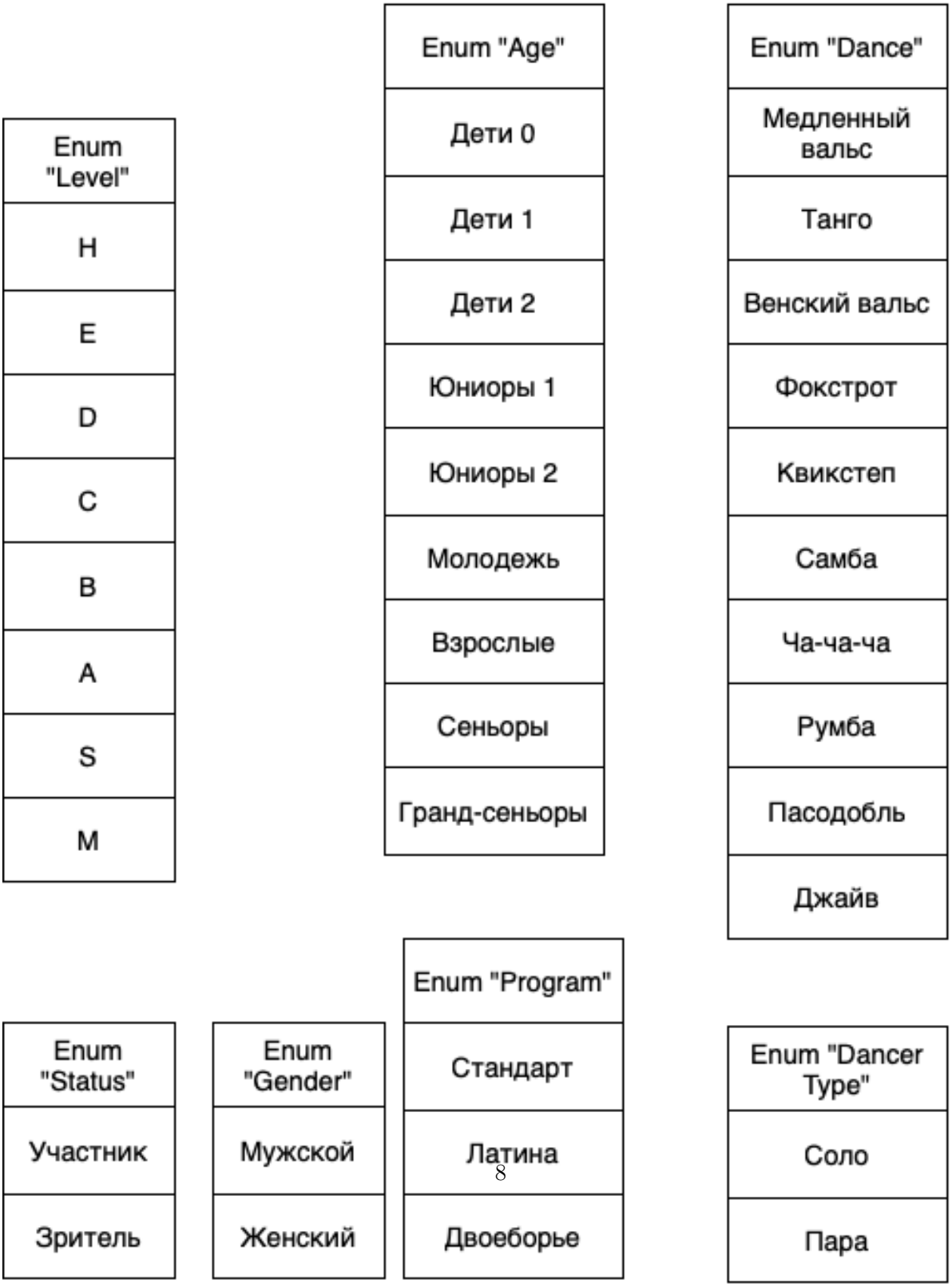
1. Получить полное расписание соревнований на каждый день.
2. Зарегистрироваться на конкретную категорию.
3. Купить входной билет на отделение.
4. Узнать свой номер захода для заданной категории, тура, танца.
5. Узнать прошел ли он в следующий тур в заданной категории.
6. Посмотреть протоколы по своим категориям.

Зритель должен иметь возможность с помощью системы:

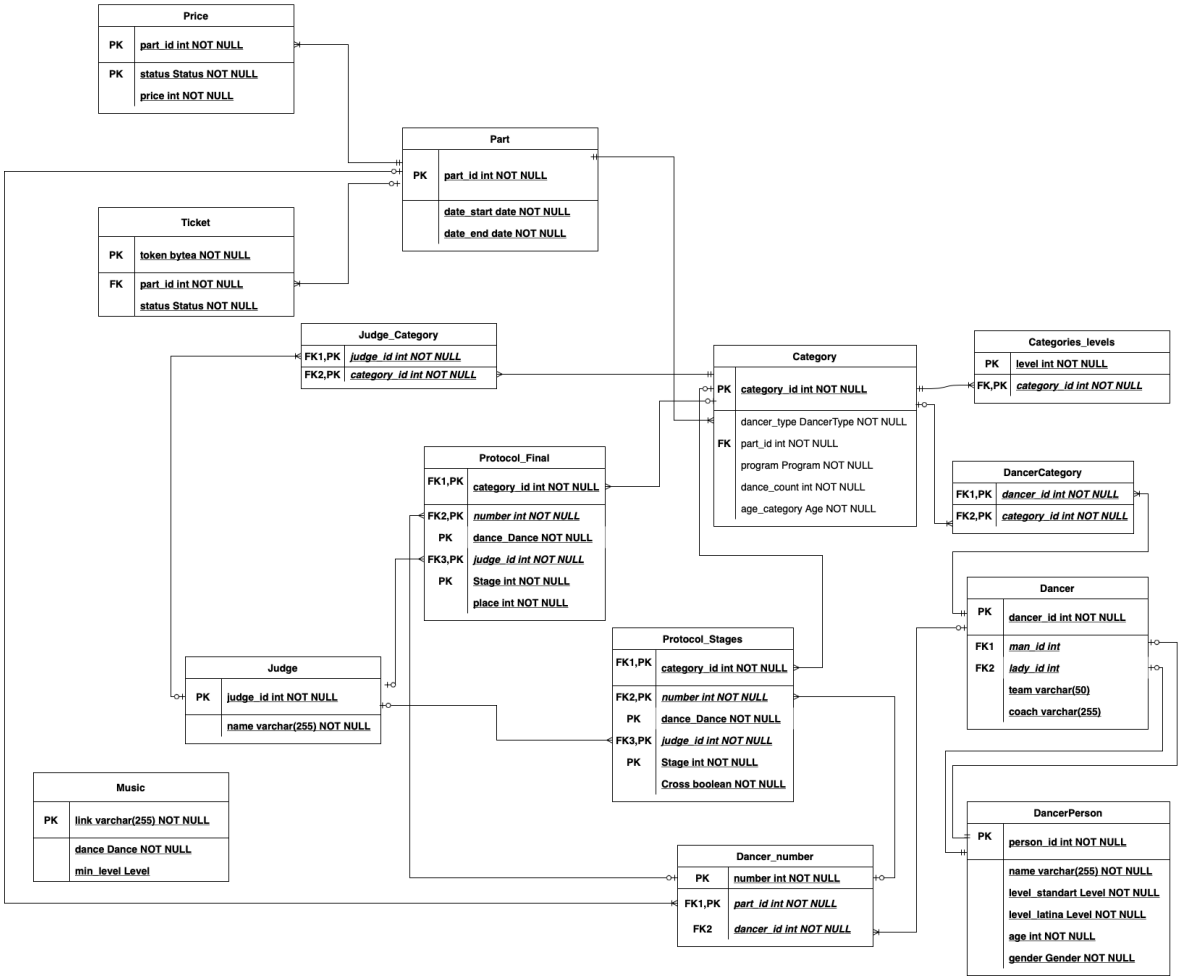
1. Получить полное расписание соревнований на каждый день.
2. Купить входной билет на отделение.

2.2 ER диаграмма

Для удобства введены следущие перечисления (Enum):

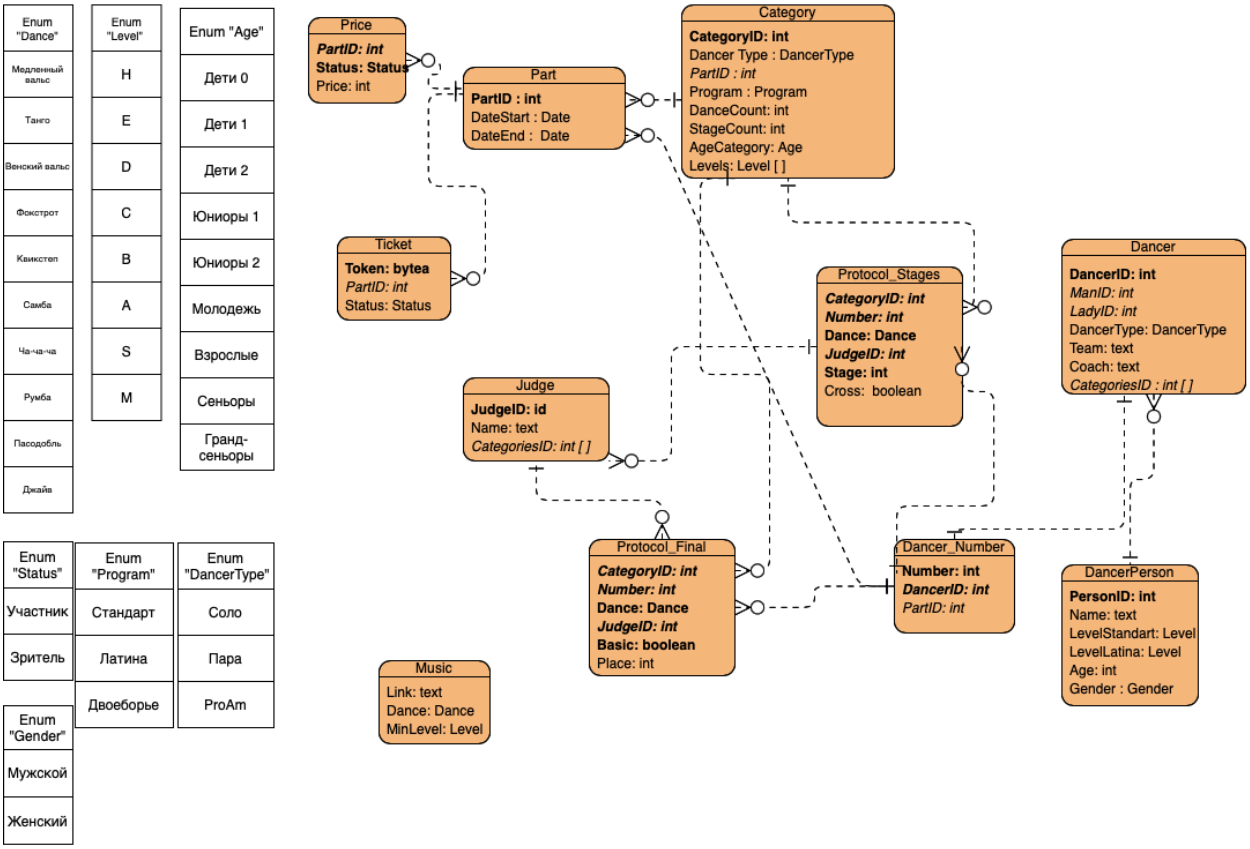


На основе пункта выше создана схема базы данных с описанием атрибутов и связей:



2.3 Нормализация

Диаграмма выше уже приведена к 3нф. На первом этапе таблица выглядела так:



2.3.1 Приведение к 1нф

Видно, что повторяющихся строк нет, а также все значения скалярные. Но не все атрибуты простые. Так атрибут Levels типа Level [ ] в таблице Category, атрибут CategoriesID типа int [ ] в таблице Dancer, атрибут CategoriesID типа int [ ] в таблице Judge. Чтобы это исправить выносим в отдельную таблицу Id из предыдущей таблицы и тот тип, который был массивом, и оба их делаем pk.

2.3.2 Приведение к 2нф

Условия для 2нф: У таблицы должен быть первичный ключ, все атрибуты должны описывать его полностью, а не только часть.
Видим, что во всех таблицах, кроме Music есть первичный ключ, а в музыке его нет. Сделаем атрибут Link pk.

Также во всех таблциах кроме Dancer_Number, атрибуты описывают первичный ключ полностью. Атрибут PartID зависит только от танцора, а номер при этом значения не имеет. Чтобы решить это- сделаем этот атрибут также первичным ключом.

2.3.3 Приведение к 3нф

Условие для 3нф: Не должно быть зависимостей одних ключевых атрибутов от других. Все атрибуты зависят только от первичного ключа.

Во всех таблицах кроме Dancer это условие выполняется. В этой же таблице атрибут DancerType напрямую зависит от значений атрибутов ManID и LadyID. Для решения этой проблемы решено убрать этот атрибут(сделать его вычисляемым).

Этап 2. Начало работы

Задача: Развернуть СУБД, создать таблиц и заполнить данными.

3.1 Развертывание СУБД PostgreSQL

Для того, чтобы развернуть СУБД PostgreSQL надо прописать в файл docker-compose.yaml следующие строки:

```
version: "3.9"

name: "balroom-service"

services:
  postgres:
    container_name: postgresDB
    image: postgres:14.5
    restart: unless-stopped
    env_file:
      - .env
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
```

И надо создать файл .env, через который подаются переменные среды:

```
DB_NAME=postgres
DB_USER=postgres
DB_PASSWORD=postgres
```

Чтобы база заработала-надо прописать в консоль

```
docker-compose up
```

3.2 Создание таблиц

Таблицы будем создавать, используя механизм миграций. Так, через файл.env передадим директорию с файлами миграции. Чтобы была возможность мигрировать до конкретной версии добавим переменную MIG_VERSION в .env , в также bash скрипт init.sh, который будет выбирать и запускать нужные миграции.(Но сначала создаст роль, от которой эти миграции будут запускаться) Чтобы это сделать пропишем в volumes в docker-compose.yaml:

```
    volumes:
      - ./migrations:/docker-entrypoint-initdb.d/migrations
      - ./.env:/docker-entrypoint-initdb.d/env
      - ./init.sh:/docker-entrypoint-initdb.d/init.sh
```

В .env:

```
    DB_NAME=postgres
    DB_USER=postgres
    DB_PASSWORD=postgres
    MIG_VERSION=0.0.0
```

В init.sh:

```
    #!/bin/bash
db_name=$(grep -E 'DB_NAME' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS =
    "=" } ; {print $2}')
db_user=$(grep -E 'DB_USER' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS =
    "=" } ; {print $2}')
psql -U $db_user -d $db_name -c "CREATE ROLE admin WITH LOGIN CREATEDB;"

last_version="0.0.0"
mig_path="/docker-entrypoint-initdb.d/migrations"
num=$(grep -E 'MIG_VERSION' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS =
    "=" } ; {print $2}')
if [[ "$num" == "" ]]
then
    psql -U admin -d $db_name -f "$mig_path/$last_version.sql"
else
    array=$(ls $mig_path | sort )
    for file in "$array"
    do
    psql -U admin -d $db_name -f "$mig_path/$file"
    if [[ "$file" == "$num.sql" ]]
    then
```

```
        break
    fi

    done;
fi
```

Теперь в файлах директории migrations можно написать первый файл миграции 0.0.0.sql . Вот его часть:

```
DROP TYPE IF EXISTS ParticipantStatus;
CREATE TYPE ParticipantStatus AS ENUM ('participant', 'viewer');

CREATE TABLE IF NOT EXISTS Part (
    partID      integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    date_start   timestamptz NOT NULL,
    date_finish  timestamptz NOT NULL
);
```

3.3 Генерация данных

По схожей выше схеме реализуем генерацию тестовых данных. Через переменную в .env передадим требуемое количество кортежей в основных отношениях:

```
   COUNT=1000000
```

Создадим директорию generator и файл для генерации 0.0.0.sql:

```
...
INSERT INTO Category(partId,dancer_type, competition_program,dance_count,
    age_category)
SELECT
    FLOOR(RANDOM() * FLOOR(:'count'/3)+1 )::INTEGER,
  (enum_range(NULL::DancerType))[random_choice(array[1,2])],
  (enum_range(NULL::CompetitionsProgram))[random_choice(array[1,2,3])],
    random_choice(array[2, 3, 4, 5, 6,8,10]),
    (enum_range(NULL::AgeCategory))[random_choice(array[1,2,3,4,5,6,7,8,9])]
FROM GENERATE_SERIES(1, FLOOR(:'count'/2)::INTEGER);
...
```

Допишем в docker-compose.yaml добавленную директорию:

```
    volumes:
      - ./generator:/docker-entrypoint-initdb.d/generator
```

Запуском файлов генерации будет заниматься написанный ранее init.sh скрипт. Допишем в него следующие строки:

```
gen_path="/docker-entrypoint-initdb.d/generator"
count=$(grep -E 'COUNT' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS = "=" } ; {print $2}')

if [[ "$num" == "" ]]
then
    psql  -U admin -d $db_name -v count=$count -f "$gen_path/$last_version.sql"
else
    array=$(ls $gen_path | sort )
    for file in "$array"
    do
    psql  -U admin -d $db_name -v count=$count -f "$gen_path/$file"
    if [[ "$file" == "$num.sql" ]]
    then
        break
    fi

    done;
fi
```

3.4 Добавление ролей

По условию требуется это сделать в bash скрипте. Создадим скрипт roles.sh и добавим его в в docker-compose.yaml:

```
    volumes:
    - ./roles.sh:/docker-entrypoint-initdb.d/roles.sh
```

В .env добавим переменную-массив, через которую будут передаваться имена ролей для групповой роли:

```
   NAMES=Liza,Morgenshtern,Stich
```

В самом скрипте напишем создание ролей:

```
    #!/bin/bash
db_name=$(grep -E 'DB_NAME' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS = "=" } ; {print $2}')
db_user=$(grep -E 'DB_USER' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS = "=" } ; {print $2}')
```

```
4
5  psql -U $db_user -d $db_name -c "CREATE ROLE reader WITH LOGIN; GRANT SELECT ON
       ALL TABLES IN SCHEMA public TO reader;"
6  psql -U $db_user -d $db_name -c "CREATE ROLE writer WITH LOGIN; GRANT SELECT,
       UPDATE, INSERT ON ALL TABLES IN SCHEMA public TO writer;"
7  psql -U $db_user -d $db_name -c "CREATE ROLE analytic WITH LOGIN; GRANT SELECT ON
       ProtocolFinal TO analytic;"
8
9
10 psql -U $db_user -d $db_name -c "CREATE ROLE group_role; GRANT CONNECT ON DATABASE
        $db_name TO group_role;"
11
12 names_str=$(grep -E 'NAMES' /docker-entrypoint-initdb.d/env | awk 'BEGIN { FS =
       "=" } ; {print $2}')
13 names=($(echo $names_str | tr ',' '\n'))
14
15 for i in "${names[@]}"
16 do
17 psql -U $db_user -d $db_name -c "CREATE ROLE $i; GRANT group_role TO $i;"
18 done
```

Этап 3. Администрирование и оптимизация

Задача: Повысить производительность и отказоустойчивость СУБД.

4.1 Запросы

1. Вывести опредленного танцора все категории в которых он участвовал, до куда он дошел
2. Вывести топ отделений по количеству купленных билетов
3. Вывести все категории и их победителей
4. Вывести всех судей и соревнования в которых они участвовали
5. Вывести пару которая набрала больше всего крестов за танец вальс (переменная)

4.1.1 1 запрос

```
EXPLAIN ANALYZE SELECT dc.categoryid,MIN(ps.stage),sum(pf.place_mark) as final_poits_sum from dancerperson  as dp
                                JOIN dancer as d on d.manid = dp.personid OR d.ladyid = dp.personid
                                JOIN dancernumber as dn on dn.dancerid = d.dancerid
                                JOIN dancercategory as dc on dc.dancerid = d.dancerid
                                JOIN category as c on c.categoryid = dc.categoryid  AND c.partid = dn.partid
                                LEFT JOIN protocolstages as ps on ps.dancer_number = dn.dancer_number AND ps.categoryid = dc.categoryid
                                LEFT JOIN protocolfinal as pf on pf.dancer_number = dn.dancer_number AND pf.categoryid = dc.categoryid
WHERE dp.personid=1
GROUP BY dc.categoryid
```

4.1.2 2 запрос

```
EXPLAIN ANALYZE SELECT part.partid,count(DISTINCT ticket.ticket_token) from ticket
    RIGHT JOIN part on part.partid = ticket.partid
GROUP BY part.partid
ORDER BY count(DISTINCT ticket.ticket_token) desc
limit 3
```

4.1.3 3 запрос

```
EXPLAIN ANALYZE WITH t1 AS (
    SELECT  c.categoryid,pf.dancer_number,min(pf.place_mark) as min from category
    as c JOIN protocolfinal as pf on pf.categoryid = c.categoryid
    GROUP BY c.categoryid,pf.dancer_number
   ), t2 AS (
    SELECT  c.categoryid,pf.dancer_number,SUM(pf.place_mark) as total from
    category as c JOIN protocolfinal as pf on pf.categoryid = c.categoryid
    GROUP BY c.categoryid,pf.dancer_number
   )
SELECT  t2.categoryid,t2.dancer_number from t2
    WHERE t2.total = (SELECT t1.min from t1  where t1.categoryid = t2.categoryid
    LIMIT 1)
    GROUP BY t2.categoryid,t2.dancer_number
```

4.1.4 4 запрос

```
EXPLAIN ANALYZE select j.judgeid,jc.categoryid from judge as j JOIN judgecategory
    as jc on jc.judgeid = j.judgeid
```

4.1.5 5 запрос

```
EXPLAIN ANALYZE select dancer_number, COUNT(cross_mark) from protocolstages
WHERE dance = :'dance' AND cross_mark
GROUP BY dancer_number
```

4.2 Анализ текущих запросов

С помощью Explain Analyze для каждого запроса измерим Execution Time. Для этого напишем сервис, который полученные результаты будет форматировать и записывать в отдельный файл: лучший, средний, худший случай для каждого запроса. Кол-во попыток к каждому запросу определяется в env):

```
#!/bin/bash
db_name=$(grep -E 'DB_NAME' ./.env | awk 'BEGIN { FS = "=" } ; {print $2}')
db_password=$(grep -E 'DB_PASSWORD' ./.env | awk 'BEGIN { FS = "=" } ; {print $2}'
    )
db_password=$(echo $db_password | awk '{print $NF}')
db_user=$(grep -E 'DB_USER' ./.env| awk 'BEGIN { FS = "=" } ; {print $2}')
n=$(grep -E 'req_n' ./.env2| awk 'BEGIN { FS = "=" } ; {print $2}')
gen_path="request"
dance=$(grep -E 'DANCE' ./.env2 | awk 'BEGIN { FS = "=" } ; {print $2}')

for file in `find $gen_path -type f -name "*.sql"`
do
    min=10000000000000000
    max=0
    sum=0
    for (( i=1; i <= n; i++))
    do
    > "tmp.txt"
    PGPASSWORD="$db_password" psql -h localhost -p 5000  -U $db_user -d $db_name
    -v dance="$dance" -f "$file" -o  "tmp.txt"
    execution=$(grep -E 'Execution Time' tmp.txt | tr ":" " " | awk '{print $3}' )
    sum="$(bc<<<"scale=5;$sum+$execution")"

    if (( $(echo "$min > $execution" |bc -l) ));
    then
     min="$execution"
    fi

    if (( $(echo "$execution > $max" |bc -l) ));
    then
      max="$execution"
    fi
```

```
31
32      #planning=$(grep -E 'Planning Time' tmp.txt | tr ":" " " | awk '{print $3}' )
33      #echo $planning
34      done
35      avg="$(awk -v v1=$sum -v v2=$n "BEGIN {print v1/v2}")"
36      echo "file: $file    n: $n" >> "results/$file_num"
37      echo "                       : $min"  >> "results/$file_num"
38      echo "                         : $avg"  >> "results/$file_num"
39      echo "                       : $max"  >> "results/$file_num"
40      echo " " >> "results/$file_num"
41  done
```

И надо создать файл .env2, через который подаются переменные среды:

```
1  DANCE=
2  req_n=10
```

4.3 Добавление индексов

Создадим индексы для тех столбцов в таблицах по которым происходит фильтр или join:
И надо создать файл .env, через который подаются переменные среды:

```
1  CREATE INDEX IF NOT EXISTS   ix_dancerID_DancerNumber ON DancerNumber (dancerID);
2  CREATE INDEX IF NOT EXISTS   ix_dancer_number ON DancerNumber (dancer_number);
3  CREATE INDEX IF NOT EXISTS  ix_manId ON Dancer  (manId);
4  CREATE INDEX IF NOT EXISTS  ix_ladyId ON Dancer  (ladyId);
5  CREATE INDEX IF NOT EXISTS   ix_dancerID_DancerCategory ON DancerCategory  (
       dancerID);
6  CREATE INDEX IF NOT EXISTS   ix_categoryID_DancerCategory ON DancerCategory  (
       categoryID);
7  CREATE INDEX IF NOT EXISTS   ix_partId ON Category  (partId);
8  CREATE INDEX IF NOT EXISTS  ix_dancer_number_stages ON ProtocolStages  (
       dancer_number);
9  CREATE INDEX IF NOT EXISTS  ix_categoryID_stages ON ProtocolStages  (categoryID);
10 CREATE INDEX IF NOT EXISTS   ix_dancer_number_final ON ProtocolFinal  (
       dancer_number);
11 CREATE INDEX IF NOT EXISTS  ix_categoryID_final ON ProtocolFinal  (categoryID);
12 CREATE INDEX IF NOT EXISTS  ix_partID_Ticket ON Ticket  (partID);
13 CREATE INDEX IF NOT EXISTS ix_judgeID ON JudgeCategory  (judgeID);
14 CREATE INDEX IF NOT EXISTS   ix_dance ON ProtocolStages  (dance);
```

```
15 CREATE INDEX IF NOT EXISTS  ix_cross_mark ON ProtocolStages  (cross_mark);
```

Напишем также скрипт drop_idx.sql, который будет эти индексы удалять.
Добавим в наш сервис перед выполнением первой части запуск скрипта drop_idx.sql, а после нее выполним скрипт на создание:

```
1  PGPASSWORD="$db_password" psql -h localhost -p 5000  -U $db_user -d $db_name  -v
       dance="$dance" -f "migrations/0.0.1.sql"
2
3  echo "C                    " >> "results/$file_num"
4  for file in ‘find $gen_path -type f -name "*.sql"‘
5  do
6      min=10000000000000000
7      max=0
8      sum=0
9      for (( i=1; i <= "$n"; i++ ))
10     do
11     > "tmp.txt"
12     PGPASSWORD="$db_password" psql -h localhost -p 5000  -U $db_user -d $db_name
       -v dance="$dance" -f "$file" -o  "tmp.txt"
13     execution=$(grep -E 'Execution Time' tmp.txt | tr ":" " " | awk '{print $3}' )
14     sum="$(bc<<<"scale=5;$sum+$execution")"
15     if (( $(echo "$min > $execution" |bc -l) ));
16     then
17      min="$execution"
18     fi
19     if (( $(echo "$execution > $max" |bc -l) ));
20     then
21       max="$execution"
22     fi
23     done
24     avg="$(awk -v v1=$sum -v v2=$n "BEGIN {print v1/v2}")"
25     echo "file: $file    n: $n" >> "results/$file_num"
26     echo "                       : $min"  >> "results/$file_num"
27     echo "                         : $avg"  >> "results/$file_num"
28     echo "                       : $max"  >> "results/$file_num"
29     echo " " >> "results/$file_num"
30 done
```

4.4 Добавление партиций

Таблицу DancerPerson было решено разбить на две части, mans и ladyies. С добавлением constraint, который проверяет пол человека:

```
create table IF NOT EXISTS mans ( like DancerPerson including all );
alter table mans inherit DancerPerson;
alter table mans  add constraint partition_check check (gender = '              '
    );

create table IF NOT EXISTS ladyies ( like DancerPerson including all );
alter table ladyies inherit DancerPerson;
alter table ladyies add constraint partition_check check (gender = '
    ');
```

Аналогично предыдущему пункту напишем скрипт, удаляющий эти таблицы. Добавим его вызов перед 1 этапом работы нашего сервиса. После окончания работы второго этапа - запустим создание этих партиций:

```
    PGPASSWORD="$db_password" psql -h localhost -p 5000  -U $db_user -d $db_name
    -v dance="$dance" -f "migrations/0.0.2.sql"
echo "C                      " >> "results/$file_num"

for file in `find $gen_path -type f -name "*.sql"`
do
    min=10000000000000000
    max=0
    sum=0
    for (( i=1; i <= "$n"; i++ ))
    do
    > "tmp.txt"
    PGPASSWORD="$db_password" psql -h localhost -p 5000  -U $db_user -d $db_name
    -v dance="$dance" -f "$file" -o  "tmp.txt"
    execution=$(grep -E 'Execution Time' tmp.txt | tr ":" " " | awk '{print $3}' )
    sum="$(bc<<<"scale=5;$sum+$execution")"
    if (( $(echo "$min > $execution" |bc -l) ));
    then
     min="$execution"
    fi
    if (( $(echo "$execution > $max" |bc -l) ));
```

```
    then
      max="$execution"
    fi
    done
    avg="$(awk -v v1=$sum -v v2=$n "BEGIN {print v1/v2}")"
    echo "file: $file    n: $n" >> "results/$file_num"
    echo "                        : $min"  >> "results/$file_num"
    echo "                          : $avg"  >> "results/$file_num"
    echo "                        : $max"  >> "results/$file_num"
    echo " " >> "results/$file_num"
done
```

4.5 Итоги

Чтобы подвести итоги-создадим для каждого этапа в сервисе свой массив. Тогда код для подведения результатов будет следующий:

```
   echo "          : " >> "results/$file_num"
echo $'\t\t\t                          \t\t                          \t\t
                          '  >> "results/$file_num"
i=0
for file in `find $gen_path -type f -name "*.sql"`
do
echo "$file" >> "results/$file_num"
echo -e $"                    :  \t\t ${min_array[$i]} \t\t  ${avg_array[$i]} \t\
    t  ${max_array[$i]}"  >> "results/$file_num"
echo -e "                     :  \t\t  ${min_array_2[$i]} \t\t  ${avg_array_2[$i]}
     \t\t  ${max_array_2[$i]}"  >> "results/$file_num"
min_proc="$(awk -v v1=${min_array_2[$i]} -v v2=${min_array[$i]} "BEGIN {print 100
    - v1/v2*100}")"
avg_proc="$(awk -v v1=${avg_array_2[$i]} -v v2=${avg_array[$i]} "BEGIN {print 100
    - v1/v2*100}")"
max_proc="$(awk -v v1=${max_array_2[$i]} -v v2=${max_array[$i]} "BEGIN {print 100
    - v1/v2*100}")"
echo -e $"                  2        1: \t  $min_proc % \t\t  $avg_proc % \t\t
    $max_proc %"  >> "results/$file_num"
echo -e $"                        : \t\t  ${min_array_3[$i]} \t\t  ${avg_array_3[$i
    ]} \t\t  ${max_array_3[$i]}"  >> "results/$file_num"
```

```
min_proc="$(awk -v v1=${min_array_3[$i]} -v v2=${min_array_2[$i]} "BEGIN {print 100 - v1/v2*100}")"
avg_proc="$(awk -v v1=${avg_array_3[$i]} -v v2=${avg_array_2[$i]} "BEGIN {print 100 - v1/v2*100}")"
max_proc="$(awk -v v1=${max_array_3[$i]} -v v2=${max_array_2[$i]} "BEGIN {print 100 - v1/v2*100}")"
echo -e $"                    3        2: \t  $min_proc % \t\t  $avg_proc % \t\t $max_proc %" >> "results/$file_num"
min_proc="$(awk -v v1=${min_array_3[$i]} -v v2=${min_array[$i]} "BEGIN {print 100 - v1/v2*100}")"
avg_proc="$(awk -v v1=${avg_array_3[$i]} -v v2=${avg_array[$i]} "BEGIN {print 100 - v1/v2*100}")"
max_proc="$(awk -v v1=${max_array_3[$i]} -v v2=${max_array[$i]} "BEGIN {print 100 - v1/v2*100}")"
echo -e $"                    3        1: \t  $min_proc % \t\t  $avg_proc % \t\t $max_proc %"  >> "results/$file_num"
echo "----------------------------- " >> "results/$file_num"
let "i=$i+1"
done
```

Полученный результат при запуске примерно следующий:

```
Without indexes
file: request/4.sql    n: 10
Best time: 0.111
Avg time: 0,134
Worst time: 0.255

file: request/5.sql    n: 10
Best time: 0.081
Avg time: 0,1142
Worst time: 0.215

file: request/2.sql    n: 10
Best time: 0.184
Avg time: 0,2508
Worst time: 0.396

file: request/3.sql    n: 10
```

```
Best time: 1.607
Avg time: 2,0666
Worst time: 2.841

file: request/1.sql    n: 10
Best time: 0.285
Avg time: 0,4269
Worst time: 1.142

With indexes
file: request/4.sql    n: 10
Best time: 0.106
Avg time: 0,1318
Worst time: 0.179

file: request/5.sql    n: 10
Best time: 0.077
Avg time: 0,0996
Worst time: 0.168

file: request/2.sql    n: 10
Best time: 0.181
Avg time: 0,307
Worst time: 0.671

file: request/3.sql    n: 10
Best time: 1.616
Avg time: 2,2774
Worst time: 4.250

file: request/1.sql    n: 10
Best time: 0.200
Avg time: 0,2972
Worst time: 0.668

With partitions
file: request/4.sql    n: 10
Best time: 0.107
```

```
Avg time: 0,1228
Worst time: 0.150

file: request/5.sql    n: 10
Best time: 0.077
Avg time: 0,1373
Worst time: 0.229

file: request/2.sql    n: 10
Best time: 0.180
Avg time: 0,2363
Worst time: 0.503

file: request/3.sql    n: 10
Best time: 1.579
Avg time: 2,0519
Worst time: 3.442

file: request/1.sql    n: 10
Best time: 0.294
Avg time: 0,3804
Worst time: 0.708

------------------------------
In result :
        Best time       Avg time       Worst time
request/4.sql
Without anyth:        0.111       0,134        0.255
With indexes:        0.106        0,1318       0.179
Update 2 of 1:    4,5045 %       1,64179 %        29,8039 %
With partitions:       0.107        0,1228       0.150
Update 3 of 2:    -0,943396 %        6,82853 %        16,2011 %
Update 3 of 1:    3,6036 %       8,35821 %        41,1765 %
------------------------------
request/5.sql
Without anyth:        0.081       0,1142       0.215
With indexes:        0.077        0,0996       0.168
Update 2 of 1:    4,93827 %        12,7846 %        21,8605 %
```

```
With partitions:       0.077        0,1373       0.229
Update 3 of 2:     0 %        -37,8514 %       -36,3095 %
Update 3 of 1:     4,93827 %        -20,2277 %       -6,51163 %
-----------------------------
request/2.sql
Without anyth:        0.184       0,2508       0.396
With indexes:        0.181        0,307        0.671
Update 2 of 1:     1,63043 %        -22,4083 %       -69,4444 %
With partitions:       0.180        0,2363       0.503
Update 3 of 2:     0,552486 %       23,0293 %        25,0373 %
Update 3 of 1:     2,17391 %        5,7815 %       -27,0202 %
-----------------------------
request/3.sql
Without anyth:        1.607       2,0666       2.841
With indexes:        1.616        2,2774       4.250
Update 2 of 1:     -0,56005 %       -10,2003 %       -49,5952 %
With partitions:       1.579        2,0519       3.442
Update 3 of 2:     2,2896 %       9,90164 %        19,0118 %
Update 3 of 1:     1,74238 %        0,711313 %       -21,1545 %
-----------------------------
request/1.sql
Without anyth:        0.285       0,4269       1.142
With indexes:        0.200        0,2972       0.668
Update 2 of 1:     29,8246 %        30,3818 %        41,5061 %
With partitions:       0.294        0,3804       0.708
Update 3 of 2:     -47 %        -27,9946 %       -5,98802 %
Update 3 of 1:     -3,15789 %       10,8925 %        38,0035 %
-----------------------------
```

4.6 Бэкапы

Напишем скрипт, который каждые n часов будет делать m бэкапов базы.(параметры передаются через env2)

```
   #!/bin/bash
db_name=$(grep -E 'DB_NAME' ./.env | awk 'BEGIN { FS = "=" } ; {print $2}')
db_password=$(grep -E 'DB_PASSWORD' ./.env | awk 'BEGIN { FS = "=" } ; {print $2}'
    )
db_password=$(echo $db_password | awk '{print $NF}')
```

```
db_user=$(grep -E 'DB_USER' ./.env| awk 'BEGIN { FS = "=" } ; {print $2}')
n=$(grep -E 'backup_n' ./.env2| awk 'BEGIN { FS = "=" } ; {print $2}')
m=$(grep -E 'backup_m' ./.env2| awk 'BEGIN { FS = "=" } ; {print $2}')
gen_path="backup"

while [[ true ]]
do
rm $gen_path/*.dump

idx=1
while  [[ -f "$gen_path/$idx.dump" ]]
do
let "idx=$idx+1"
done

for (( i=1; i <= "$m"; i++ ))
do
    PGPASSWORD="$db_password" pg_dump -U $db_user  -h localhost -p 5000  $db_name
   > "$gen_path/$idx.dump"
    let "idx=$idx+1"
done
let "sec=$n * 3600"
sleep $sec
done
```

4.7 Patroni

Для отказоустойчивости кластера СУБД развернем 2 реплики Postgres. Для этого скопируем docker-compose.yaml из официального репозитория . (до этого надо его склонирровать и сбилдить, чтобы появились нудные images) В нем нам нужно убрать одну из реплик patroni3. Прокинуть volums, которые были в предыдущем docker-compose.yml:

```
  version: "2"

networks:
    demo:
services:
    etcd1: &etcd
        image: ${PATRONI_TEST_IMAGE:-patroni}
```

```
        networks: [ demo ]
        environment:
            ETCD_LISTEN_PEER_URLS: http://0.0.0.0:2380
            ETCD_LISTEN_CLIENT_URLS: http://0.0.0.0:2379
            ETCD_INITIAL_CLUSTER: etcd1=http://etcd1:2380,etcd2=http://etcd2:2380,
    etcd3=http://etcd3:2380
            ETCD_INITIAL_CLUSTER_STATE: new
            ETCD_INITIAL_CLUSTER_TOKEN: tutorial
            ETCD_UNSUPPORTED_ARCH: arm64
        container_name: demo-etcd1
        hostname: etcd1
        command: etcd --name etcd1 --initial-advertise-peer-urls http://etcd1:2380

    etcd2:
        <<: *etcd
        container_name: demo-etcd2
        hostname: etcd2
        command: etcd --name etcd2 --initial-advertise-peer-urls http://etcd2:2380

    etcd3:
        <<: *etcd
        container_name: demo-etcd3
        hostname: etcd3
        command: etcd --name etcd3 --initial-advertise-peer-urls http://etcd3:2380

    haproxy:
        image: ${PATRONI_TEST_IMAGE:-patroni}
        networks: [ demo ]
        env_file: patroni.env
        hostname: haproxy
        container_name: demo-haproxy
        volumes:
        - ./migrations:/docker-entrypoint-initdb.d/migrations
        - ./generator:/docker-entrypoint-initdb.d/generator
        - ./.env:/docker-entrypoint-initdb.d/env
        - ./init.sh:/docker-entrypoint-initdb.d/init.sh
        - ./roles.sh:/docker-entrypoint-initdb.d/roles.sh
        ports:
```

```
            - "5000:5000"
            - "5001:5001"
        command: haproxy
        environment: &haproxy_env
            ETCDCTL_ENDPOINTS: http://etcd1:2379,http://etcd2:2379,http://etcd3
    :2379
            PATRONI_ETCD3_HOSTS: "'etcd1:2379','etcd2:2379','etcd3:2379'"
            PATRONI_SCOPE: demo

    patroni1:
        image: ${PATRONI_TEST_IMAGE:-patroni}
        networks: [ demo ]
        env_file: patroni.env
        hostname: patroni1
        container_name: demo-patroni1
        environment:
            <<: *haproxy_env
            PATRONI_NAME: patroni1

    patroni2:
        image: ${PATRONI_TEST_IMAGE:-patroni}
        networks: [ demo ]
        env_file: patroni.env
        hostname: patroni2
        container_name: demo-patroni2
        environment:
            <<: *haproxy_env
            PATRONI_NAME: patroni2
```

Запуская его, поднимится сервер haproxy, через который мы и будем получать доступ к базе по порту 5000 (поэтому везде в коде выше в скриптах используется именно этот порт).

Этап 4. Мониторинг СУБД PostgreSQL

Задача: Установить инструменты мониторинга PostgreSQL и вывести данные в Grafana.

5.1 Установка prometheus и postgres-exporter

Данные сервисы будем поднимать в docker. Для этого создадим папку в нашем проекте со своим docker-compose, конфигурационным файлом prometheus.yml и файлом запросов queries.yaml.Стоит также отметить, что была создана отдельная роль только для чтения для входа в базу exporter

Сожержимое docker-compose:

```
version: '3.9'

volumes:
    prometheus_data: {}

configs:

  postgres_exporter_config:
    file: ./queries.yaml

services:

  prometheus:
    image: prom/prometheus:latest
    volumes:
      - ./:/etc/prometheus2
      - ./:/etc/prometheus/
      - prometheus_data:/prometheus
    container_name: prometheus
    hostname: prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
```

```
    ports:
      - 9090:9090
    restart: unless-stopped
    environment:
      TZ: "Europe/Moscow"
    networks:
      - default
  postgres-exporter:
    image: wrouesnel/postgres_exporter:latest
    configs:
      - source: postgres_exporter_config
        target: /etc/postgres_exporter/queries.yaml
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    container_name: exporter
    hostname: exporter
    restart: unless-stopped
    environment:
      - DATA_SOURCE_NAME=user=exporter password=exporter host=host.docker.internal
     port=5000 dbname=postgres sslmode=disable
      - PG_EXPORTER_EXTEND_QUERY_PATH=/etc/postgres_exporter/queries.yaml
    ports:
      - "9187:9187"
    networks:
      - default
networks:
  default:
    ipam:
      driver: default
      config:
        - subnet: 172.28.0.0/16
```

Содержимое prometheus.yml:

```
scrape_configs:
  - job_name: postgres-exporter
    scrape_interval: 5s
    static_configs:
```

```
5       - targets: ['docker.for.mac.localhost:9187']
```

На порту 9090 поднялся prometheus. Важно проверить, что в таргетах видно наш экспортер.

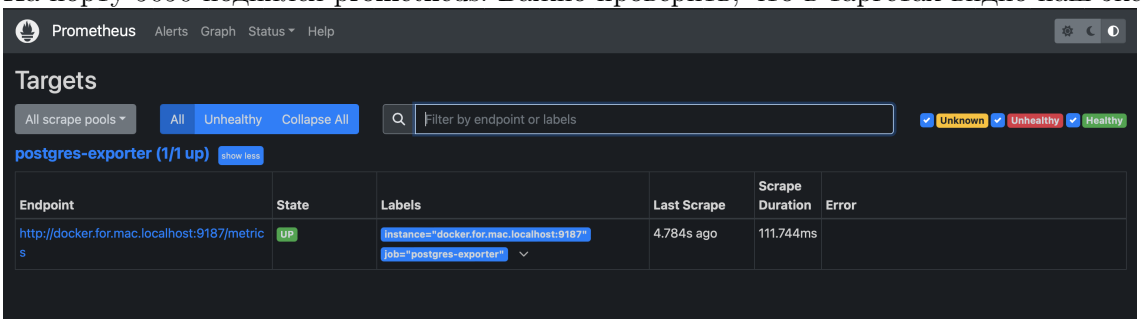


Рис. 5.1: Таргеты в prometheus

После этого на порту 9187 поднимется экспертер и можно увидеть метрики, которые он собирает.

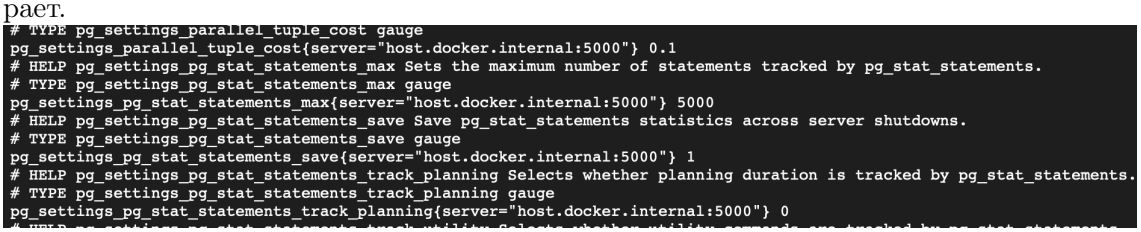


Рис. 5.2: Пример метрик postgres_exporter

5.2 Установка grafana

Для grafana создадим отдельную папку, ее также будем поднимать в docker. Для этого создадим docker-compose со следующим содержанием:

```
1  version: '3.9'
2
3  volumes:
4      grafana-data: {}
5
6  services:
7    grafana:
8      image: grafana/grafana
9      container_name: grafana
10     user: root
11     ports:
12       - 3000:3000
13     volumes:
14       - ./grafana:/var/lib/grafana/
```

```
      - grafana-data:/var/lib/grafana/
      - ./grafana/provisioning/:/etc/grafana/provisioning/
    hostname: grafana
    restart: unless-stopped
    environment:
      TZ: "Europe/Moscow"
```

Теперь на порту 3000 можем открыть Grafana. В нем нужно настроить связь с поднятым ранее prometheus. Для этого нажимаем Add new data source ищем там prometheus и пишем следущее:

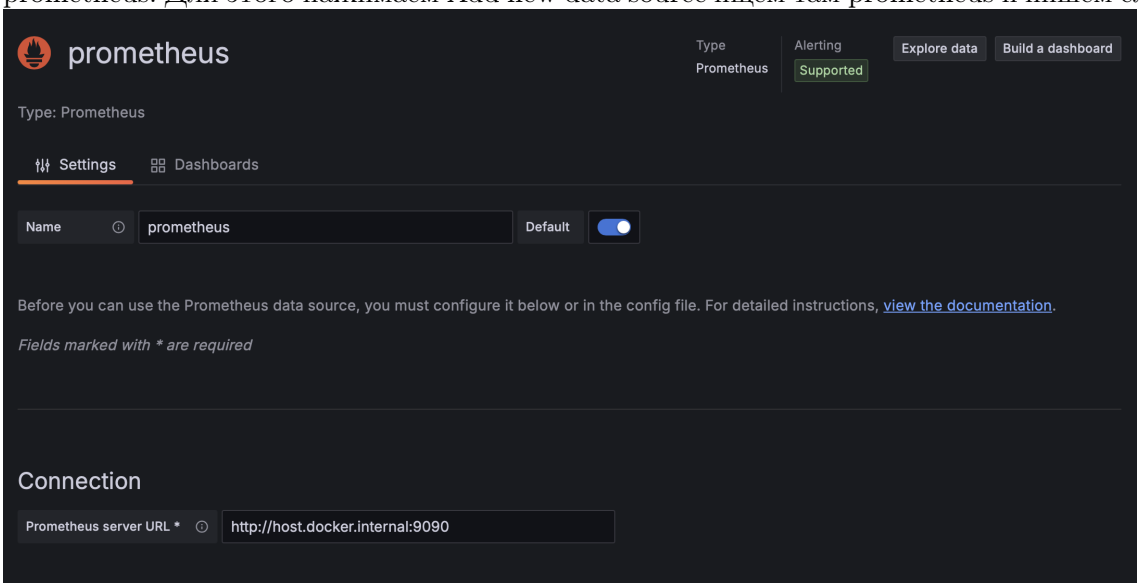


Рис. 5.3: Настройка prometheus в Grafana

Дальше жмем save & test.

5.3 Настройка метрик

Метрики будем настраивать с помощью ранее созданного файла queries.yaml. Благодаря ему помимо метрик по умолчанию вытянутся еще и те, что мы сами зададим. Пропишем в нем следующее

```
pg_table:
  query: "SELECT c.oid,nspname AS table_schema, relname AS TABLE_NAME, c.reltuples
    AS row_estimate , pg_total_relation_size(c.oid) AS total_bytes ,
   pg_indexes_size(c.oid) AS index_bytes , pg_total_relation_size(reltoastrelid)
   AS toast_bytes FROM pg_class c LEFT JOIN pg_namespace n ON n.oid = c.
   relnamespace WHERE relkind = 'r'"
  metrics:
    - oid:
```

```
        usage: "LABEL"
        description: "pg_class oid"
    - table_schema:
        usage: "LABEL"
        description: "table schema"
    - table_name:
        usage: "LABEL"
        description: "table name"
    - row_estimate:
        usage: "GAUGE"
        description: "row count estimate"
    - total_bytes:
        usage: "GAUGE"
        description: "total table bytes"
    - index_bytes:
        usage: "GAUGE"
        description: "table index bytes"
    - toast_bytes:
        usage: "GAUGE"
        description: "table toast bytes"

pg_replication:
  query: "SELECT EXTRACT(EPOCH FROM (now() - pg_last_xact_replay_timestamp())) as
    lag"
  metrics:
    - lag:
        usage: "GAUGE"
        description: "Replication last transaction replay behind master in seconds
    "


pg_stat_replication:
  query: "SELECT pid, application_name, client_addr, state FROM
    pg_stat_replication"
  metrics:
    - application_name:
        usage: "LABEL"
        description: "Replica recovery.conf application_name"
```

```
    - client_addr:
        usage: "LABEL"
        description: "Replica client database IP address"
    - state:
        usage: "LABEL"
        description: "Replica client streaming state"
    - pid:
        usage: "GAUGE"
        description: "Replication pid"

pg_postmaster:
  query: "SELECT pg_postmaster_start_time as start_time_seconds from
    pg_postmaster_start_time()"
  metrics:
    - start_time_seconds:
        usage: "GAUGE"
        description: "Time at which postmaster started"

pg_stat_user_tables:
  query: "SELECT schemaname, relname, seq_scan, seq_tup_read, idx_scan,
    idx_tup_fetch, n_tup_ins, n_tup_upd, n_tup_del, n_tup_hot_upd, n_live_tup,
    n_dead_tup, n_mod_since_analyze, last_vacuum, last_autovacuum, last_analyze,
    last_autoanalyze, vacuum_count, autovacuum_count, analyze_count,
    autoanalyze_count FROM pg_stat_user_tables"
  metrics:
    - schemaname:
        usage: "LABEL"
        description: "Name of the schema that this table is in"
    - relname:
        usage: "LABEL"
        description: "Name of this table"
    - seq_scan:
        usage: "COUNTER"
        description: "Number of sequential scans initiated on this table"
    - seq_tup_read:
        usage: "COUNTER"
        description: "Number of live rows fetched by sequential scans"
    - idx_scan:
```

```
      usage: "COUNTER"
      description: "Number of index scans initiated on this table"
  - idx_tup_fetch:
      usage: "COUNTER"
      description: "Number of live rows fetched by index scans"
  - n_tup_ins:
      usage: "COUNTER"
      description: "Number of rows inserted"
  - n_tup_upd:
      usage: "COUNTER"
      description: "Number of rows updated"
  - n_tup_del:
      usage: "COUNTER"
      description: "Number of rows deleted"
  - n_tup_hot_upd:
      usage: "COUNTER"
      description: "Number of rows HOT updated (i.e., with no separate index
    update required)"
  - n_live_tup:
      usage: "GAUGE"
      description: "Estimated number of live rows"
  - n_dead_tup:
      usage: "GAUGE"
      description: "Estimated number of dead rows"
  - n_mod_since_analyze:
      usage: "GAUGE"
      description: "Estimated number of rows changed since last analyze"
  - last_vacuum:
      usage: "GAUGE"
      description: "Last time at which this table was manually vacuumed (not
    counting VACUUM FULL)"
  - last_autovacuum:
      usage: "GAUGE"
      description: "Last time at which this table was vacuumed by the autovacuum
     daemon"
  - last_analyze:
      usage: "GAUGE"
      description: "Last time at which this table was manually analyzed"
```

```
    - last_autoanalyze:
        usage: "GAUGE"
        description: "Last time at which this table was analyzed by the autovacuum daemon"
    - vacuum_count:
        usage: "COUNTER"
        description: "Number of times this table has been manually vacuumed (not counting VACUUM FULL)"
    - autovacuum_count:
        usage: "COUNTER"
        description: "Number of times this table has been vacuumed by the autovacuum daemon"
    - analyze_count:
        usage: "COUNTER"
        description: "Number of times this table has been manually analyzed"
    - autoanalyze_count:
        usage: "COUNTER"
        description: "Number of times this table has been analyzed by the autovacuum daemon"

pg_statio_user_tables:
  query: "SELECT schemaname, relname, heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit, toast_blks_read, toast_blks_hit, tidx_blks_read, tidx_blks_hit FROM pg_statio_user_tables"
  metrics:
    - schemaname:
        usage: "LABEL"
        description: "Name of the schema that this table is in"
    - relname:
        usage: "LABEL"
        description: "Name of this table"
    - heap_blks_read:
        usage: "COUNTER"
        description: "Number of disk blocks read from this table"
    - heap_blks_hit:
        usage: "COUNTER"
        description: "Number of buffer hits in this table"
    - idx_blks_read:
```

```
        usage: "COUNTER"
        description: "Number of disk blocks read from all indexes on this table"
    - idx_blks_hit:
        usage: "COUNTER"
        description: "Number of buffer hits in all indexes on this table"
    - toast_blks_read:
        usage: "COUNTER"
        description: "Number of disk blocks read from this table's TOAST table (if any)"
    - toast_blks_hit:
        usage: "COUNTER"
        description: "Number of buffer hits in this table's TOAST table (if any)"
    - tidx_blks_read:
        usage: "COUNTER"
        description: "Number of disk blocks read from this table's TOAST table indexes (if any)"
    - tidx_blks_hit:
        usage: "COUNTER"
        description: "Number of buffer hits in this table's TOAST table indexes (if any)"

pg_database:
  query: " SELECT pg_database.datname, pg_database_size(pg_database.datname) as size FROM pg_database"
  metrics:
    - datname:
        usage: "LABEL"
        description: "Name of the database"
    - size:
        usage: "GAUGE"
        description: "Disk space used by the database"

ccp_transaction_wraparound:
  query: "WITH max_age AS ( SELECT 2000000000 as max_old_xid, setting AS autovacuum_freeze_max_age FROM pg_catalog.pg_settings WHERE name = 'autovacuum_freeze_max_age'), per_database_stats AS ( SELECT datname , m.max_old_xid::int , m.autovacuum_freeze_max_age::int , age(d.datfrozenxid) AS oldest_current_xid FROM pg_catalog.pg_database d JOIN max_age m ON (true) WHERE
```

```
    d.datallowconn) SELECT max(oldest_current_xid) AS oldest_current_xid , max(
    ROUND(100*(oldest_current_xid/max_old_xid::float))) AS
    percent_towards_wraparound , max(ROUND(100*(oldest_current_xid/
    autovacuum_freeze_max_age::float))) AS percent_towards_emergency_autovac FROM
    per_database_stats"
  metrics:
    - oldest_current_xid:
        usage: "GAUGE"
        description: "Oldest current transaction ID in cluster"
    - percent_towards_wraparound:
        usage: "GAUGE"
        description: "Percentage towards transaction ID wraparound"
    - percent_towards_emergency_autovac:
        usage: "GAUGE"
        description: "Percentage towards emergency autovacuum process starting"

pg_stat_activity_autovacuum:
  query: |
    SELECT COUNT(*) AS count FROM pg_stat_activity WHERE query LIKE 'autovacuum:%'
    ;
  metrics:
    - count:
        usage: GAUGE
        description: "Number of active autovacuum processes"

# # WARNING: This set of metrics can be very expensive on a busy server as every
    unique query executed will create an additional time series
pg_stat_statements:
  query: "SELECT t2.rolname, t3.datname, queryid, calls, total_exec_time / 1000 as
    total_time_seconds, min_exec_time / 1000 as min_time_seconds, max_exec_time /
    1000 as max_time_seconds, mean_exec_time / 1000 as mean_time_seconds,
    stddev_exec_time / 1000 as stddev_time_seconds, rows, shared_blks_hit,
    shared_blks_read, shared_blks_dirtied, shared_blks_written, local_blks_hit,
    local_blks_read, local_blks_dirtied, local_blks_written, temp_blks_read,
    temp_blks_written, blk_read_time / 1000 as blk_read_time_seconds,
    blk_write_time / 1000 as blk_write_time_seconds FROM pg_stat_statements t1 JOIN
    pg_roles t2 ON (t1.userid=t2.oid) JOIN pg_database t3 ON (t1.dbid=t3.oid)
    WHERE t2.rolname != 'rdsadmin'"
```

```yaml
master: true
metrics:
  - rolname:
      usage: "LABEL"
      description: "Name of user"
  - datname:
      usage: "LABEL"
      description: "Name of database"
  - queryid:
      usage: "LABEL"
      description: "Query ID"
  - calls:
      usage: "COUNTER"
      description: "Number of times executed"
  - total_time_seconds:
      usage: "COUNTER"
      description: "Total time spent in the statement, in milliseconds"
  - min_time_seconds:
      usage: "GAUGE"
      description: "Minimum time spent in the statement, in milliseconds"
  - max_time_seconds:
      usage: "GAUGE"
      description: "Maximum time spent in the statement, in milliseconds"
  - mean_time_seconds:
      usage: "GAUGE"
      description: "Mean time spent in the statement, in milliseconds"
  - stddev_time_seconds:
      usage: "GAUGE"
      description: "Population standard deviation of time spent in the statement
    , in milliseconds"
  - rows:
      usage: "COUNTER"
      description: "Total number of rows retrieved or affected by the statement"
  - shared_blks_hit:
      usage: "COUNTER"
      description: "Total number of shared block cache hits by the statement"
  - shared_blks_read:
      usage: "COUNTER"
```

```
        description: "Total number of shared blocks read by the statement"
    - shared_blks_dirtied:
        usage: "COUNTER"
        description: "Total number of shared blocks dirtied by the statement"
    - shared_blks_written:
        usage: "COUNTER"
        description: "Total number of shared blocks written by the statement"
    - local_blks_hit:
        usage: "COUNTER"
        description: "Total number of local block cache hits by the statement"
    - local_blks_read:
        usage: "COUNTER"
        description: "Total number of local blocks read by the statement"
    - local_blks_dirtied:
        usage: "COUNTER"
        description: "Total number of local blocks dirtied by the statement"
    - local_blks_written:
        usage: "COUNTER"
        description: "Total number of local blocks written by the statement"
    - temp_blks_read:
        usage: "COUNTER"
        description: "Total number of temp blocks read by the statement"
    - temp_blks_written:
        usage: "COUNTER"
        description: "Total number of temp blocks written by the statement"
    - blk_read_time_seconds:
        usage: "COUNTER"
        description: "Total time the statement spent reading blocks, in
    milliseconds (if track_io_timing is enabled, otherwise zero)"
    - blk_write_time_seconds:
        usage: "COUNTER"
        description: "Total time the statement spent writing blocks, in
    milliseconds (if track_io_timing is enabled, otherwise zero)"


pg_stat_activity_transaction_time:
  query: "
    SELECT
```

```
265       datname ,
266       usename ,
267       application_name ,
268       xact_start ,
269       now()-xact_start*1000 AS age_norm ,
270       GREATEST(EXTRACT(EPOCH FROM statement_timestamp() - MIN(xact_start*1000)),
    0) AS age
271     FROM pg_stat_activity
272     WHERE state != 'idle' AND usename IS NOT NULL AND datname IS NOT NULL
273     GROUP BY datname, usename, application_name,xact_start;"
274
275   metrics:
276     - usename:
277         usage: "LABEL"
278         description: "Name of the user logged into this backend"
279     - datname:
280         usage: "LABEL"
281         description: "Name of the database this backend is connected to"
282     - application_name:
283         usage: "LABEL"
284         description: "Name of the application that is connected to this backend"
285     - xact_start:
286         usage: "GAUGE"
287         description: "Age in seconds of oldest transaction"
288     - age:
289         usage: "GAUGE"
290         description: "Age in seconds of oldest transaction"
```

После этого может возникнуть проблема с отсутствием таблицы pg_stat_statements. Для ее решения надо прописать в конфиг postgresql

```
1 shared_preload_libraries=pg_stat_statements
2 pg_stat_statements.track=all.
```

Так как у нас настроен патрони, то там это можно сделать, запустив в терминале его контейнера команды:

```
1 python3 ./patronictl.py -c /home/postgres/postgres0.yml edit-config -p
    shared_preload_libraries=pg_stat_statements
2 python3 ./patronictl.py -c /home/postgres/postgres0.yml edit-config -p
    pg_stat_statements.track=all
```

После этого создать extention командой:

```
psql -h localhost -p 5000 -U $db_user -d $db_name -c "CREATE EXTENSION IF NOT
    EXISTS pg_stat_statements;"
```

5.4 Создание дашборда

Для создания дашборда в grafana надо нажать на + и на create new dashboard. После этого в качестве data source выбираем ранее настроенный prometheus.
Чтобы создать визуализацию для какой либо метрики, нажимаем add, Visualization. После потребуется написать PromQL запрос (по факту просто обратится к метрике, с возможностью произвести над некоторые операции)
Приведу пример. Запрос для параметра "Количество запросов в секунду". Функция rate берет данные за последнюю минуту, дальше они суммируются и делятся 60.

```
sum(rate(pg_stat_statements_calls[1m]))/60
```

Можно выбрать форму визуализации, так, Time series это график от времени, а stat число. Для этого параметра я выбрала форму stat, выглядит это так:

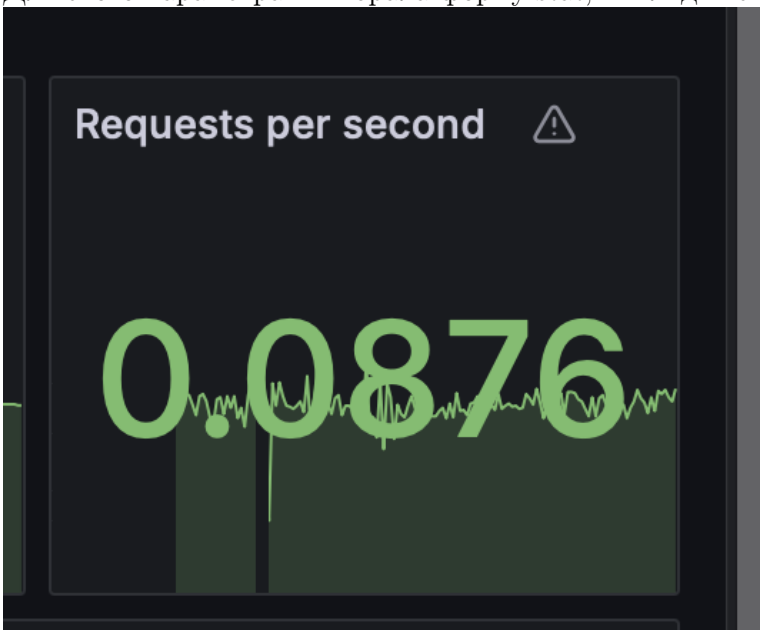


Рис. 5.4: Пример визуализации метрики

5.5 Немного о принципах выбора нужных метрик

И если говорить про мониторинг базы, то что нужно мониторить? В первую очередь нужно мониторить доступность, потому что база – это сервис, который предоставляет доступ к данным клиентам. Доступность в моем понимании – это способность базы обслуживать подключения, т. е. база поднята, она, как сервис, принимает подключения от клиентов. И эту доступность

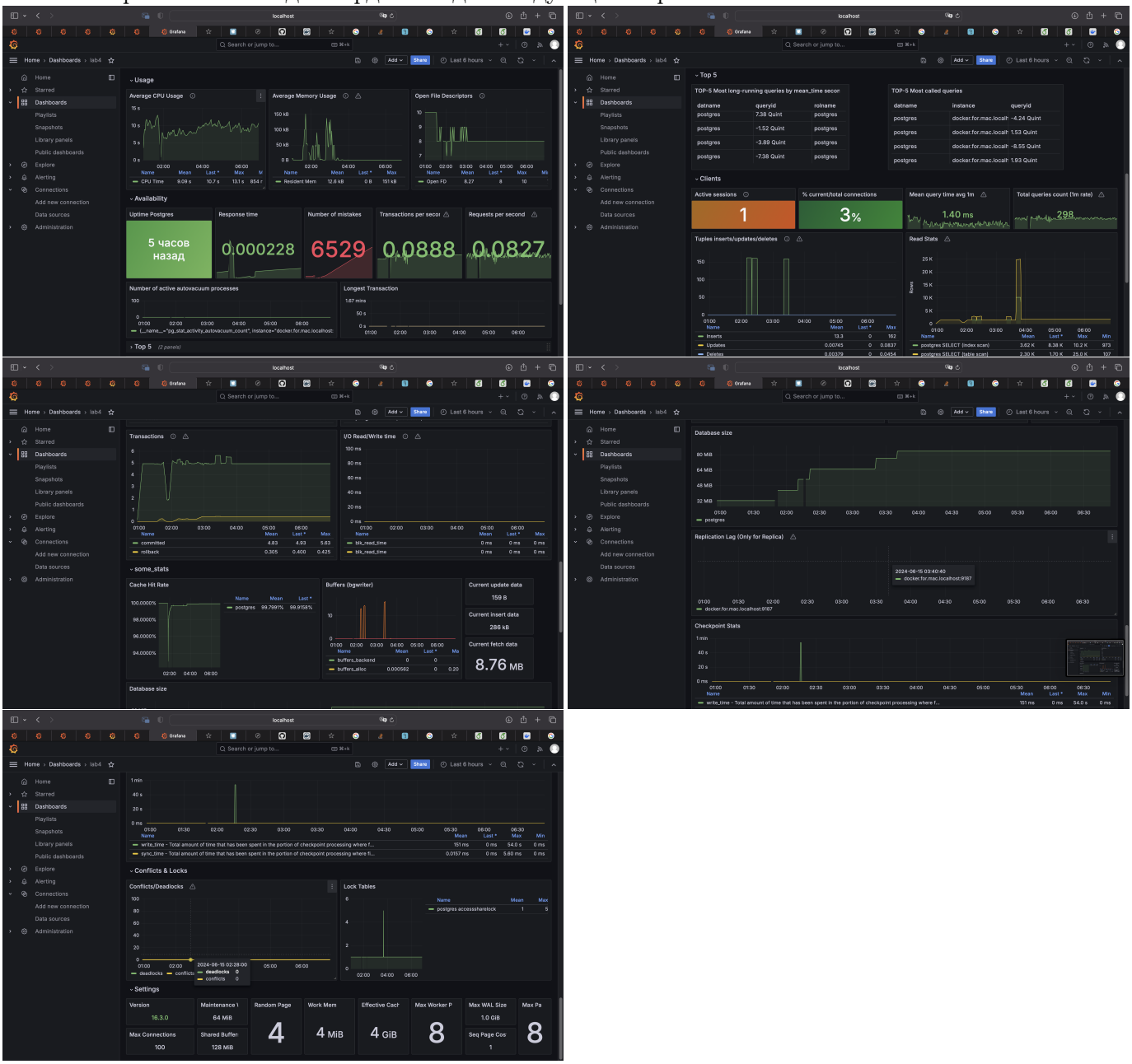
можно оценивать некоторыми характеристиками например, **время отклика**, **количество ошибок**. Также можно добавить **количество транзакций в секунду** и**количество запросов в секунду**. Мы сможем использовать эти цифры как текущую производительность базы данных и наблюдать есть ли пики запросов, пики транзакций или, наоборот, база недогружена, потому что какой-то backend отвалился. Эту цифру важно всегда смотреть и помнить, что для нашего проекта вот такая производительность является нормальной, а значения выше и ниже уже какие-то проблемные и непонятные, а значит, нужно смотреть, почему такие цифры. Также к этой теме можно отнести такие параметры как **uptime**, чтобы отслеживать dowtime, **самую долгую транзакцию**, ведь она поможет очень сильно навредить производительности и текущее количество воркеров вакуума (если их много, то это повод заглянуть в настройки) Также нужно мониторить клиентов, которые подключаются к нашей базе, потому что они могут быть как и нормальными клиентами, так и вредными клиентами, которые могут наносить вред базе данных. Их тоже нужно мониторить и отслеживать их деятельность.Нам нужно мониторить и то, как клиенты работают с данными: с какими таблицами, в меньшей степени с какими индексами. Т. е. нам нужно оценить ворклоад (workload), который создается нашими клиентами. Сюда относится **количество коннекшенов**, **средняя длительность их запросов**,**статистика по разным видам запросов**, ,**наиболее часто повторяющиеся запросы**, ,**наиболее долгие запросы**, это нужно, чтобы оценивать, какие запросы у нас в базе данных, отслеживать их адекватность, что они не являются криво написанными, что какие-то опции нужно переписать и сделать так, чтобы они работали быстрее и с более лучшей производительностью.

Фоновые процессы позволяют поддерживать производительность базы данных на хорошем уровне, поэтому для их работы они требуют некое количество ресурсов для себя. И в то же время они могут пересекаться с ресурсами клиентских запросов, поэтому жадная работа фоновых процессов может непосредственно влиять на производительность клиентских запросов. Поэтому их тоже нужно мониторить и отслеживать, что нет никаких перекосов в плане фоновых процессов. Например такие как **количество случающихся чекпоинтов**, **лаг репликации**, **текущий размер таблицы**, **скорость попадания в кэш**

С системными метриками относятся **утилизация процессоров, памяти**

5.6 Итог

Итого построенный мной дашборд выглядит следующим образом:



Чтобы его сохранить в качестве json надо нажать share, export, save to file

Выводы

В рамках работы над этим проектом была спроектированна база данных для соревнований поо бальным танцам. Были изучены функциональные требования, построена er-диаграмма, было проведена нормализация данных. Помимо этого удалось развернуть эту базу на СУБД PostgreSQL. Был проведен анализ базы и для оптимизации типовых запросов были дообавлены индекси и партиции. Чтобы повысить отказоустойчивость блы написан скрипт, делающий автоматические бэкапы, а также развернуты дополнительнуе кластеры с patroni. Наконец для поддержания базы были добавлены средства мониторинга на основе prometheus и grafana. Модно сказать, что начальная цель достигнута и все задачи выполнены.