**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное

образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

ОТЧЕТ по лабораторной работе

«Этап 3. Администрирование и оптимизация»

по дисциплине «**Проектирование баз данных**»

|  |  |
| --- | --- |
| Автор: | Константинова Елизавета Анатольевна |
| Факультет: | ФИТиП |
| Группа: | M3203 |
| Преподаватели: | Шевчик Софья Владимировна |
|  |  |

ИТМО

Санкт-Петербург 2024

Запросы к бд:  
Вывести всех пользователей и их роли на определенном сервере которые сейчас онлайн

SELECT u.username, ur.role\_id  
FROM servers AS s  
LEFT JOIN users u ON u.user\_id = s.owner\_id  
LEFT JOIN user\_roles AS ur ON s.server\_id = ur.server\_id and u.user\_id = ur.user\_id  
WHERE u.online\_status = TRUE and s.server\_id = 97664

Вывести пары людей которые сидят как минимум на двух одинаковых серверах

SELECT DISTINCT  
 u1.username AS username1,  
 u2.username AS username2  
FROM  
 user\_roles ur1  
 JOIN user\_roles ur2 ON ur1.server\_id = ur2.server\_id AND ur1.user\_id <> ur2.user\_id  
 JOIN users u1 ON ur1.user\_id = u1.user\_id  
 JOIN users u2 ON ur2.user\_id = u2.user\_id  
GROUP BY  
 u1.username, u2.username  
HAVING  
 COUNT(DISTINCT ur1.server\_id) >= 2;

Вывести все сервера и их обладателей у которых есть хотя бы один текстовый канал

SELECT  
 s.server\_name,  
 u.username AS owner\_username  
FROM  
 servers s  
 JOIN channels c ON s.server\_id = c.server\_id  
 JOIN users u ON s.owner\_id = u.user\_id  
WHERE  
 c.channel\_type = 'text\_channel'  
GROUP BY  
 s.server\_name, u.username;

Сервис, который вызывает Explain Analyze для каждого запроса и измеряет

Cost:

Dockerfile:

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build-env  
WORKDIR /discord  
  
COPY . ./   
  
  
RUN dotnet restore  
  
RUN dotnet publish -c Release -o out  
  
  
FROM mcr.microsoft.com/dotnet/runtime:8.0  
WORKDIR /discord  
COPY --from=build-env /discord/out .  
EXPOSE 8080  
ENTRYPOINT ["sh", "-c", "dotnet discord.dll"]

docker-compose:

analyze\_app**:** restart**:** no  
 container\_name**:** analyze\_app  
 environment**:** ATTEMPT\_COUNT**:** ${ATTEMPT\_COUNT}  
 build**:** context**:** ./discord  
 dockerfile**:** Dockerfile  
 volumes**:  
 -** ./itogi:/docker-entrypoint-initdb.d/itogi  
 depends\_on**:** db**:** condition**:** service\_healthy  
 env\_file**:  
 -** .env  
 ports**:  
 -** "8080:8080"

Analyze.cs:

using System.Text.RegularExpressions;  
using Npgsql;  
  
namespace discord;  
  
class Analyze  
{  
 static void Main(string[] args)  
 {  
 var timestamp = DateTime.Now.ToString("yyyyMMddHHmmss");  
 var filePath = $"/docker-entrypoint-initdb.d/itogi/query\_itogi\_{timestamp}.txt";  
 var connectionString = "Host=db;Username=postgres;Password=mypassword;Database=discord-db;Port=5432;";  
 Console.WriteLine("ANALYZE SERVICE STARTED");  
  
 var attemptCount = int.Parse(Environment.GetEnvironmentVariable("ATTEMPT\_COUNT") ?? "4");  
  
 string[] queries =  
 {  
 """  
 SELECT u.username, ur.role\_id  
 FROM servers AS s  
 LEFT JOIN users u ON u.user\_id = s.owner\_id  
 LEFT JOIN user\_roles AS ur ON s.server\_id = ur.server\_id and u.user\_id = ur.user\_id  
 WHERE u.online\_status = TRUE and s.server\_id = 97664  
 """,  
 """  
 SELECT DISTINCT  
 u1.username AS username1,  
 u2.username AS username2  
 FROM  
 user\_roles ur1  
 JOIN user\_roles ur2 ON ur1.server\_id = ur2.server\_id AND ur1.user\_id <> ur2.user\_id  
 JOIN users u1 ON ur1.user\_id = u1.user\_id  
 JOIN users u2 ON ur2.user\_id = u2.user\_id  
 GROUP BY  
 u1.username, u2.username  
 HAVING  
 COUNT(DISTINCT ur1.server\_id) >= 2;  
 """,  
 """  
 SELECT  
 s.server\_name,  
 u.username AS owner\_username  
 FROM  
 servers s  
 JOIN channels c ON s.server\_id = c.server\_id  
 JOIN users u ON s.owner\_id = u.user\_id  
 WHERE  
 c.channel\_type = 'text\_channel'  
 GROUP BY  
 s.server\_name, u.username;  
 """  
 };  
  
 var costs = new double[queries.Length][];  
 for (var i = 0; i < costs.Length; i++)  
 {  
 costs[i] = new double[attemptCount];  
 }  
  
 try  
 {  
 using (var writer = new StreamWriter(filePath))  
 {  
 using (var connection = new NpgsqlConnection(connectionString))  
 {  
 connection.Open();  
   
 for (var q = 0; q < queries.Length; q++)  
 {  
 var query = queries[q];  
 Console.WriteLine($"Executing query {q + 1}/{queries.Length}: {query}");  
 for (var a = 0; a < attemptCount; a++)  
 {  
 try  
 {  
 using (var command = new NpgsqlCommand($"EXPLAIN ANALYZE {query}", connection))  
 {  
 using (var reader = command.ExecuteReader())  
 {  
 var res = new List<double>();  
   
 while (reader.Read())  
 {  
 var result = reader.GetString(0);  
 var match = Regex.Match(result, @"cost=(\d+\.\d+)..(\d+\.\d+)");  
 if (match.Success)  
 {  
 var startCost = double.Parse(match.Groups[1].Value);  
 var endCost = double.Parse(match.Groups[2].Value);  
 res.Add(endCost);  
 }  
 }  
   
 if (a == attemptCount - 1)  
 {  
 var minC = res.Min();  
 var maxC = res.Max();  
 var avgC = res.Average();  
 writer.WriteLine($"query: {query}\n");  
 writer.WriteLine($"best case time: {minC}");  
 writer.WriteLine($"worst case time: {maxC}");  
 writer.WriteLine($"average case time: {avgC}");  
 writer.WriteLine('\n');  
   
 }  
 }  
 }  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"ERROR: {ex.Message}");  
 }  
 }  
 }  
 }  
 }  
  
 Console.WriteLine($"RESULTS SAVED: {filePath}");  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"error writing to file: {ex.Message}");  
 }  
 }  
}

Добавленные индексы:

CREATE INDEX idx\_servers\_owner\_id ON servers (owner\_id);  
CREATE INDEX idx\_user\_roles\_server\_user\_id ON user\_roles (server\_id, user\_id);  
CREATE INDEX idx\_user\_roles\_user\_id\_server\_id ON user\_roles (user\_id, server\_id);  
CREATE INDEX idx\_channels\_channel\_type ON channels (channel\_type);

Сравнительная таблица:

|  |  |  |  |
| --- | --- | --- | --- |
| изначально | 1 запрос | 2 запрос | 3 запрос |
| Best | 8.3 | 1454.06 | 0.38 |
| Worst | 1814.93 | 25352528.95 | 3869.82 |
| Avg | 728.044 | 5058657.189999999 | 2428.444285714286 |
| После индексов | | | |
| Best | 0.88 | 1404.62 | 0.37 |
| Worst | 17.51 | 16744308.71 | 3682.29 |
| Avg | 10.324 | 4240299.414285715 | 1909.295 |
| Лучше на | 7180% | 19,2% | 27,2% |

Создание партиции таблицы ролей пользователей:

ALTER TABLE user\_roles RENAME TO user\_roles\_old;  
  
CREATE TABLE user\_roles  
(  
 user\_role\_id SERIAL PRIMARY KEY,  
 user\_id **INTEGER** NOT NULL REFERENCES users (user\_id),  
 role\_id **INTEGER** NOT NULL REFERENCES roles (role\_id),  
 server\_id **INTEGER** NOT NULL REFERENCES servers (server\_id)  
) PARTITION BY RANGE (user\_role\_id);  
  
ALTER TABLE user\_roles\_old  
 ADD CONSTRAINT user\_roles\_old  
 CHECK (user\_role\_id >= 1 AND user\_role\_id <= 100000);  
  
CREATE TABLE user\_roles\_1 PARTITION OF user\_roles  
 FOR VALUES FROM  
(  
 1  
) TO  
(  
 50000  
);  
  
CREATE TABLE user\_roles\_2 PARTITION OF user\_roles  
 FOR VALUES FROM  
(  
 50000  
) TO  
(  
 100001  
);  
  
WITH *moved\_rows* AS (  
DELETE  
FROM user\_roles\_old *a*WHERE user\_role\_id >= 1  
 AND user\_role\_id < 50000 RETURNING a.\*  
)  
INSERT  
INTO user\_roles\_1  
SELECT \*  
FROM moved\_rows;  
  
WITH *moved\_rows* AS (  
DELETE  
FROM user\_roles\_old *a*WHERE user\_role\_id >= 50000  
 AND user\_role\_id < 100001 RETURNING a.\*  
)  
INSERT  
INTO user\_roles\_2  
SELECT \*  
FROM moved\_rows;  
  
ALTER TABLE user\_roles\_old DROP CONSTRAINT user\_roles\_old;

|  |  |  |  |
| --- | --- | --- | --- |
| изначально | 1 запрос | 2 запрос | 3 запрос |
| Best | 0.88 | 1404.62 | 0.37 |
| Worst | 17.51 | 16744308.71 | 3682.29 |
| Avg | 10.324 | 4240299.414285715 | 1909.295 |
| После партиции | | | |
| Best | 8.3 | 565.91 | 0.37 |
| Worst | 1820.78 | 16829892.86 | 3682.29 |
| Avg | 778.18 | 3313748.391 | 1909.295 |
| Лучше на | -98% | 30% | 0% |

Скрипт, который создает бэкапы базы данных каждые n-часов, последние m-бекапов хранятся

backup.sh:

**#!/bin/bash**DB\_USER=${DB\_SUPERUSER}  
DB\_NAME=${POSTGRES\_DB}  
DB\_PASSWORD=${POSTGRES\_PASSWORD}  
  
BACKUP\_DIR="/backup"  
  
BACKUP\_COUNT=${M}  
  
TIMESTAMP=$(date +'%Y%m%d%H%M%S')  
BACKUP\_FILE="$BACKUP\_DIR/backup\_$TIMESTAMP.sql"  
sleep\_time=${N}  
  
export PGPASSWORD="$DB\_PASSWORD"  
  
mkdir -p "$BACKUP\_DIR"  
  
sleep "$sleep\_time"  
  
pg\_dump -h db -p 5432 -U "$DB\_USER" -d "$DB\_NAME" > "$BACKUP\_FILE"  
  
BACKUP\_FILES\_COUNT=$(ls -1 "$BACKUP\_DIR"/\*.sql | wc -l)  
if [ "$BACKUP\_FILES\_COUNT" -gt "$BACKUP\_COUNT" ]; then  
 ls -1t "$BACKUP\_DIR"/\*.sql | tail -n +$(($BACKUP\_COUNT + 1)) | xargs rm -f  
fi  
  
unset PGPASSWORD  
  
echo "backup"

Dockerfile:

FROM alpine:latest  
RUN apk add --no-cache postgresql-client  
COPY backup.sh ./backup.sh  
RUN chmod +x /backup.sh  
EXPOSE 8081  
  
CMD ["sh", "/backup.sh"]

docker-compose:

backup**:** restart**:** always  
 container\_name**:** backup  
 environment**:** POSTGRES\_DB**:** ${POSTGRES\_DB}  
 POSTGRES\_PASSWORD**:** ${POSTGRES\_PASSWORD}  
 DB\_USERNAME**:** ${DB\_USERNAME}  
 DB\_SUPERUSER**:** ${DB\_SUPERUSER}  
 N**:** ${N}  
 M**:** ${M}  
 build**:** context**:** ./backup  
 dockerfile**:** Dockerfile  
 volumes**:  
 -** ./backup:/backup  
 **-** ./backup/backup.sh:/docker-entrypoint-initdb.d/backup.sh  
 depends\_on**:** db**:** condition**:** service\_healthy  
 env\_file**:  
 -** .env  
 ports**:  
 -** "8081:8081"

.env:

POSTGRES\_DB="discord-db"  
POSTGRES\_PASSWORD="mypassword"  
DB\_USERNAME="postgres"  
DB\_SUPERUSER="postgres"  
FILLING\_AMOUNT=100000  
ATTEMPT\_COUNT=10  
N=10m  
M=48

2 реплики Postgres с использованием Patroni:

docker-compose:  
patroni**:** image**:** cybertec/pgwatch2  
 container\_name**:** patroni  
 restart**:** unless-stopped  
 depends\_on**:  
 -** db  
 ports**:  
 -** "2392:2392"  
 environment**:** PATRONI\_RESTAPI\_CONNECT\_ADDRESS**:** "0.0.0.0:2392"  
  
 replica1**:** image**:** postgres  
 container\_name**:** replica1  
 restart**:** unless-stopped  
 environment**:** POSTGRES\_PASSWORD**:** ${POSTGRES\_PASSWORD}  
 POSTGRES\_USER**:** ${DB\_SUPERUSER}  
 POSTGRES\_DB**:** ${POSTGRES\_DB}  
 PGDATA**:** /var/lib/postgresql/data/pgdata  
 volumes**:  
 -** replica1\_data:/var/lib/postgresql/data  
 depends\_on**:  
 -** db  
  
 replica2**:** image**:** postgres  
 container\_name**:** replica2  
 restart**:** unless-stopped  
 environment**:** POSTGRES\_PASSWORD**:** ${POSTGRES\_PASSWORD}  
 POSTGRES\_USER**:** ${DB\_SUPERUSER}  
 POSTGRES\_DB**:** ${POSTGRES\_DB}  
 PGDATA**:** /var/lib/postgresql/data/pgdata  
 volumes**:  
 -** replica2\_data:/var/lib/postgresql/data  
 depends\_on**:  
 -** db  
  
volumes**:** replica2\_data**:** replica1\_data**:**

Итоговая структура проекта:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated