

Bazy Danych

Proxy Cache'ujące
19.06.2019

Grupa: L05

Wykonał:
Pazowski Marcin 158841
II EF-DI

1 Wprowadzenie

W ramach zadania należało napisać backend, który wykorzystuje Redis'a jako serwer cache'ujący. W momencie, w którym użytkownik formułuje zapytanie np. *SELECT*FROMtableY*; oraz czas przedawnienia klucza wykonywany jest następujący algorytm:

1. Dla zapytania wejściowego Z biblioteka ustala klucz cache'owania k oraz dopuszczalny termin przedawnienia t
2. Biblioteka idzie z $\langle k, t \rangle$ do Redisa
 - (a) Jeśli Redis posiada dane odpowiadające kluczowi k spełniające t to następuje zwrot danych
 - (b) Jeśli Redis nie posiada danych:
 - i. Biblioteka wykonuje zapytanie Z na bazie danych
 - ii. Biblioteka umieszcza wynikowe w Redis'ie pod kluczem k
 - iii. Biblioteka zwraca użytkownikowi dane

Do wykonania zadania należy wykorzystać dowolny ORM, Redis'a, dowolną bazę danych oraz dowolny język programowania.

2 Wykorzystanych narzędzi

2.1 Konteneryzacja - Docker

Jest to oprogramowanie służące do konteneryzacji. Dzięki niemu w łatwy sposób można było uruchomić niezbędne narzędzia: PostgreSQL, Redis.

Adres strony internetowej narzędzia: <https://www.docker.com/>

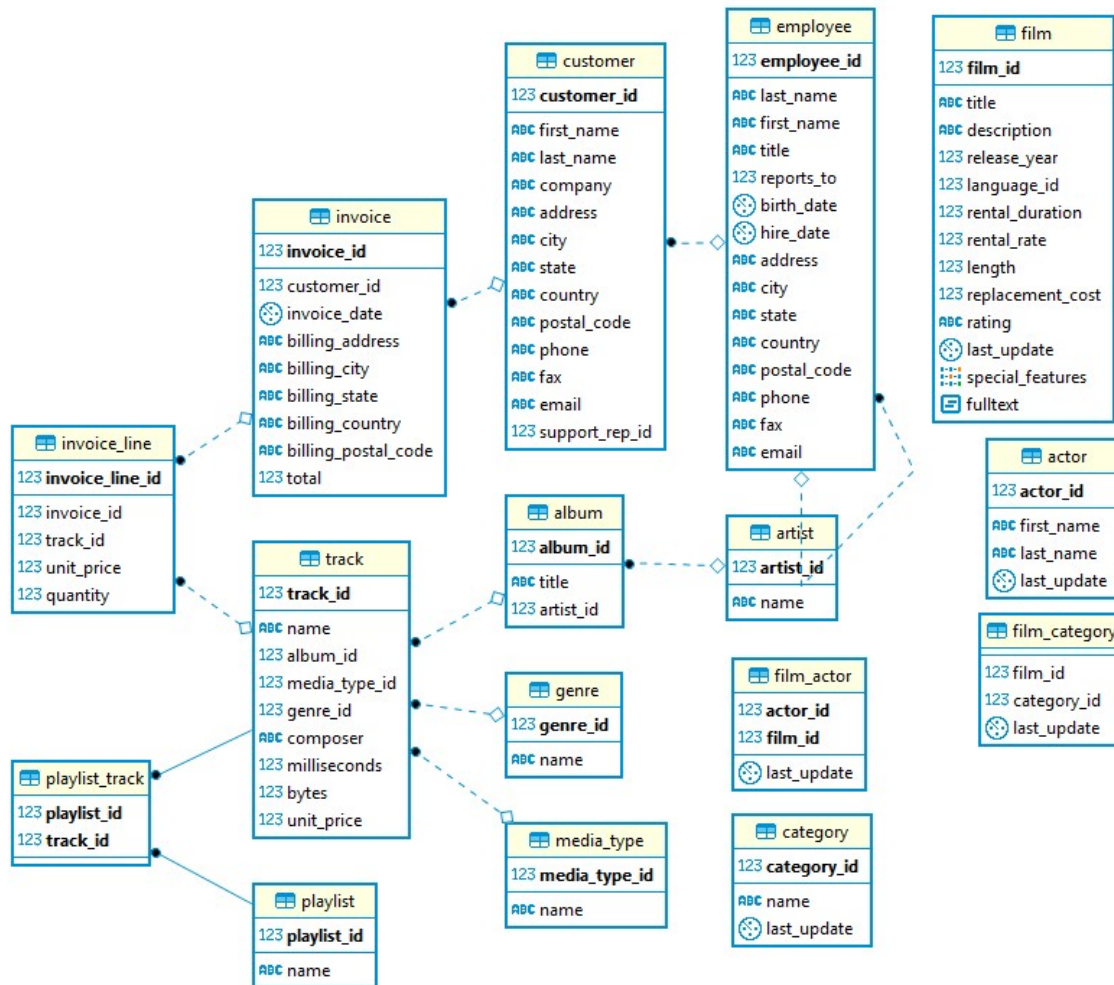
2.1.1 Baza danych - PostgreSQL

Jako baza danych został wykorzystany kontener Postgres (https://hub.docker.com/_/postgres). Uruchamiany komendą:

```
docker run --rm --name pg-docker -e POSTGRES_PASSWORD=docker
-d -p 5432:5432
-v $HOME/docker/volumes/postgres:/var/lib/postgresql/data postgres
```

Do zaprezentowania wykonania zadania wykorzystana znana, przykładową bazę danych chinook

(<https://github.com/lerocha/chinook-database>) z odpowiednimi modyfikacjami dla PostgreSQL (<https://github.com/xivSolutions/ChinookDb-Pg-Modified>)



Rysunek 1: Schemat bazy danych chinook wygenerowany w DBeaver

2.1.2 Baza cache'ująca - Redis

Redis jest magazynem danych NoSQL dostępnym na licencji open-source, który działa na zasadzie klucz-wartość.

Adres strony internetowej narzędzia: <https://redis.io/>

Redis również uruchomiony jest jako kontener (https://hub.docker.com/_/redis):

```
docker run -d -p 6379:6379 --name redis1 redis
```

2.1.3 ORM - Sequelize

Jako ORM wykorzystano Sequelize. Jest to narzędzie współpracujące z Node.js.

Adres: <https://sequelize.readthedocs.io/en/v3/>

2.2 Język programowania - JavaScript

2.2.1 Środowisko uruchomieniowe - Node.js

Node.js – wieloplatformowe środowisko uruchomieniowe o otwartym kodzie do tworzenia aplikacji typu server-side napisanych w języku JavaScript.

2.2.2 Web framework - Express.js

Strona internetowa: <https://expressjs.com/>

3 Opis programu

Stworzony został punkt wejścia dla użytkownika (linijka 1). Użytkownik w przeglądarce wpisuje

```
http://localhost:3000/queryRedis?  
qu=ZAPYTANIE SQL;&time=CZAS ZYCIA W SEKUNDACH
```

Następnie za pomocą funkcji queryToKey() zapytanie sql jest odpowiednio parsowane. Na początku usuwane są białe znaki. Następnie wszystkie litery są zmieniane na ich wielkie odpowiedniki. Na końcu ciąg znaków parsowany jest do postaci Base64.

```
1 var queryToKey = (query) => {  
2   console.log('Given query: ${query}');  
3   var r = query.replace(/\s+/g, ''); //remove white spacing  
4   r = r.toUpperCase()  
5   console.log('Parsed query: ${r}');  
6   r = Buffer.from(r).toString('Base64') //string to Base64  
7   console.log('Query to Base64 ${r}');  
8   return r  
9 }
```

Program w 5 linijce, próbuje pobrać wartość dla zapytania z bazy Redis'a. Jeśli nie ma tam wartości dla danego klucza przechodzi do wykonania zapytania za pomocą ORM'a. Gdy program dostanie odpowiedź z bazy danych (linijka 16) zapisuje wartości do Redisa (18) z odpowiednim kluczem oraz czasem życia i zwraca dane użytkownikowi (20).

```
1 app.get('/queryRedis', (req, res) => {  
2   let query = req.query.qu  
3   let time = req.query.time  
4   let key = cache.queryToKey(query) //parse sql query to key  
5   cache.client.get(key, (error, result) => {  
6     if(error) {  
7       console.log(error)  
8       throw error  
9     }  
10    if (result !== null) { //if key exist return from redis  
11      console.log('Results from REDIS')  
12      res.json(JSON.parse(result))  
13    } else {  
14      orm.sequelize.query(query, {  
15        type: orm.Sequelize.QueryTypes.SELECT  
16      }).then(answer => {  
17        console.log('Results from Postgres')  
18        cache.client.set(key, JSON.stringify(answer),  
19          'EX', time, cache.redis.print) //add key to Redis  
20        res.json(answer)  
21      }).catch(err => {  
22        console.log(err)  
23        res.send(err)  
24      })  
25    }  
26  })  
27 })
```

```

25     }
26   })
27 })

```

4 Przypadki testowe

4.1

Dla zapytania SQL:

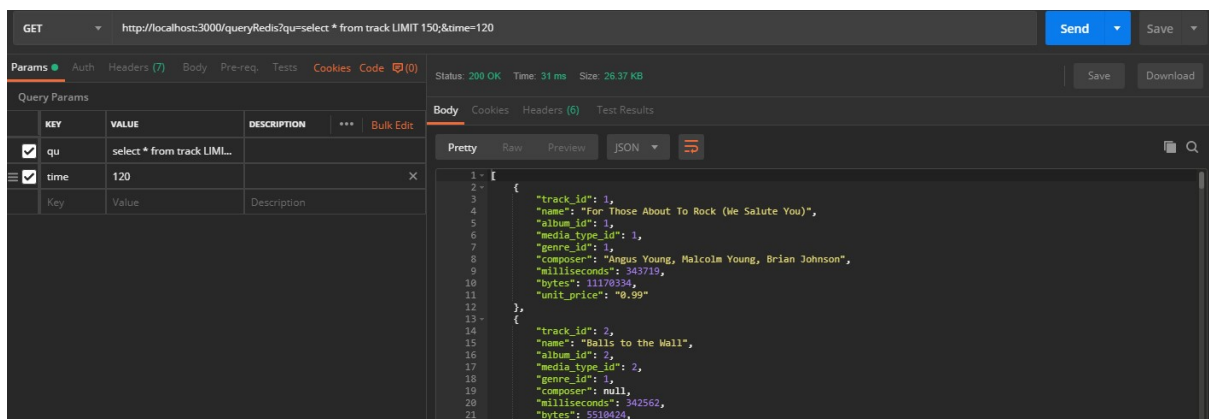
```
select * from track LIMIT 150;&time=120
```

Należy je odpowiednio wprowadzić adresem:

```
http://localhost:3000/queryRedis?
qu=select * from track LIMIT 150;&time=120
```

gdzie *qu* to odpowiednie zapytanie (query), a *time* to czas ważności klucza gdyby zapytanie nie znajdowało się jeszcze w bazie.

Niżej przedstawiona jest odpowiedź na zapytanie z programu Postman:



Tutaj są logi serwera

```

Given query: select * from track LIMIT 150;
Parsed query: SELECT*FROMTRACKLIMIT150;
Query to Base64 U0VMRUNUKkZST01UUKFDS0xJTUIUMTUwOw==
Executing (default): select * from track LIMIT 150;
Results from Postgres
Reply: OK

```

Gdy wyślemy zapytanie raz jeszcze zostanie ona wyświetlone z Redisa:

```

Given query: select * from track LIMIT 150;
Parsed query: SELECT*FROMTRACKLIMIT150;
Query to Base64 U0VMRUNUKkZST01UUKFDS0xJTUIUMTUwOw==
Results from REDIS

```

4.2

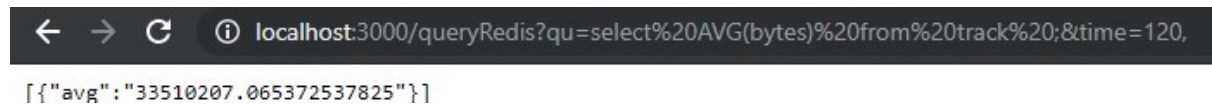
Dla zapytania SQL:

```
select AVG(bytes) from track;
```

Należy je odpowiednio wprowadzić adresem:

```
http://localhost:3000/queryRedis?  
qu=select AVG(bytes) from track;&time=120
```

Niżej przedstawiona jest odpowiedź na zapytanie z przeglądarki:



The screenshot shows a web browser's developer tools or REST client interface. The top bar displays the URL: `localhost:3000/queryRedis?qu=select%20AVG(bytes)%20from%20track%20;&time=120,`. Below the bar, the response body is shown as a JSON array: `[{"avg": "33510207.065372537825"}]`.

5 Wnioski

W przypadku zapytań testowych nie widać poprawy szybkości w zapytaniach. Spowodowane jest to tym, że Redis jak i sama baza danych PostgreSQL działają na tym samym serwerze wirtualnym. Jeśli Redis byłby po stronie użytkownika lub też w sieci lokalnej, a baza danych w odległym miejscu prędkość odczytu zapisanych danych do cache byłaby diametralnie szybsza.

Podczas wykonywania ćwiczenia największymi trudnościami było poznanie nowego języka programowania (JavaScript). Z tego też powodu funkcja realizująca zadanie nie jest odpowiednio podzielona. Problem wynikał z nikłej znajomości zwracania danych z funkcji synchronicznych i asynchronicznych.

Zadanie zostało wykonane w całości.