Sprawozdanie: Bazy danych

Mateusz Król 226400

June 15, 2018

1 Wstęp

Poniższe sprawozdanie podsomowuje pracę w pierwszej części laboratorium z Baz Danych. W części pierwszej sprawozdania zrelacjonowano proste komendy z języka MySQL. W kolejnej części pokazano zapytania z użyciem tranzakcji z różnymi poziomami izolacji. Wreszcie w ostatniej częsci pokazano przykładową aplikację opartą na Dokerze. Przeprowadzono tam badania mające na celu testy wydajnościowe bazy danych.

2 Język SQL, relacje, łączenia

w tej części laboratoriów zainstalowano pakiet MySQL i importowano istniejącą bazę danych na lokalny serwer o tytule: "sakila-data.sql". Wraz z bazą uruchomiono w MySQL Qorkbench jej schemat z pliku "sakila-schema.sql". tutaj mogliśmy zaobserwować jakie relacje łączą poszczególne tabele w bazie dotyczącej filografii. Teraz przystąpiono do eksploracji bazy danych. Wyprubowano następujące komendy:

show databases; - pokazuje dostepne bazy danych
use sakila - otwiera, załadowuje nam bazę danych o nazwie sakila
show tables - pokazuje nam zawartość bazy danych, tj. jej tabele
SELECT * FROM actor LIMIT 1; - wybiera i pokazuje nam wszystkie informacje z tabeli 'actor' i ogranicza strumień wyjściowy do jednego rekordu

W kolejnej części badano relacje wewnątrz bazy sakila:
1) Wyświetl aktora o ID=10:
SELECT * FROM actor WHERE actor id=10;
2) Wyświetl film, w którym gra aktor o ID 10:
SELECT * FROM film actor where actor id=10;
3) Wyświetl tytuł filmów z tabeli filmy używając składni z IN()
SELECT film id,title FROM film where film id IN (10,15)

W kolejnej części podjęto się łączenia różnych tabel i tak:
1) wyświetl rekordy z tabel o relacji 1:n z tabel filmy i język
SELECT film.film id, film.title, language.name FROM film INNER JOIN language ON (film.language id=language.language id) LIMIT 10;
2) wyswietl rekordy z tabel o relacjach n:n z tabel film i kateoria:
SELECT film.title, category.name FROM film INNER JOIN film category ON (film.film id =film category.film id) INNER JOIN category ON (film category.category id=category.category id) LIMIT 10;

W ostatniej części zajęć postawiono przed nami zadanie napisania kwerend, które: 1)Write a query that will select names of 10 top starring actors (those who cast in the highest number of movies)
select film actor.actor id, COUNT() from film actor group by actor id order by COUNT() DESC limit 10;
2) Write a query that will select movie categories having more than 10 films in the DB
select category.name, COUNT() from category inner join film category on (category.category id=film category.category id) group by name order by COUNT() DESC LIMIT 10

3) Write a query that will tell us which months have the most rentals (use payment table)
select MONTH(payment date) COUNT() from payment group by MONTH(payment date) order by COUNT() DESC
4) Write a query that will tell us what this the average price of rental
select SUM(amount)/COUNT(payment id) from payment
5) Check which movies are the most rented ones (top 10)
select film.film id, film.title, inventory.film id, inventory.inventory id, COUNT() from inventory inner join rental on(inventory.inventory id=rental.inventory id) inner join film on(inventory.film id=film.film id) group by rental.inventory id order by COUNT() DESC LIMIT 10
6) Determine the prices of the most frequently rented movies
select pyment.invenotry.id,rental.rental id, film.film id, film.title, inventory.film id, inventory.inventory id, COUNT() from inventory inner join rental on(inventory.inventory id=rental.inventory id) inner join payment on (payment.rental id=inventory.inventory id) inner join film on(inventory.film id=film.film id) group by rental.inventory id order by COUNT() DESC LIMIT 10

3 Tranzakcje

W tej części laboratoriów testowaliśmy transakcję z różnymi poziomami izolacji i tak:// W pierwszym shellu wstawialiśmy rekordy do tabeli poleceniem:
START TRANSACTION;
INSERT INTO test table values (0, NOW(), NOW());
INSERT INTO test table values (0, '2018-03-01', '2018-09-01');

W tym czasie uruchomiono drugi shell i wprowadzoo tam selecta:
SELECT * FROM test table;
W wyniku tego zapytania nie zwrócono żadnych rekordów, ponieważ tranzakcja z pierwszego shella nie zsotała jeszcza zacommitowana// Wracamy do pierwszego shella i dopisujemy na końcu "COMMIT;"
Teraz w drugim shellu ponawiamy zapytanie // SELECT * FROM test table;
Tym razem zauważymy wprowadzone wcześniej rekordy z pierwszego shella.

id	date of rental	date of return
1	2018-03 13	2018-03-13
2	2018-03-01	2018-09-01

W kolejnym kroku badano wpływ poziomów izolacji na tranzakcje
Wprowadzono w pierwszym shellu tranzakcję:
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
UPDATE test table SET date of return=DATE ADD(date of return, INTERVAL 1 DAY) WHERE id=2;

W drugim shellu wprowadzono"
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT * from test table where id=2;

Puszczamy pierwszą tranzakcję i próbujemy puśić drugą, w efekcie otrzymujemy:
Error Code 1568 - Transaction characteristic cannot be changed while a tranzaction is in progress

Dalej przeprowadzono podobne badania sprawdzając różne poziomy izolacji:
a) READ UNCOMMITED
b) REPEATABLE READ
c) SERIALIZABLE
Rozważania na temat różnych poziomów izolacji zostały zawarte we wnioskach

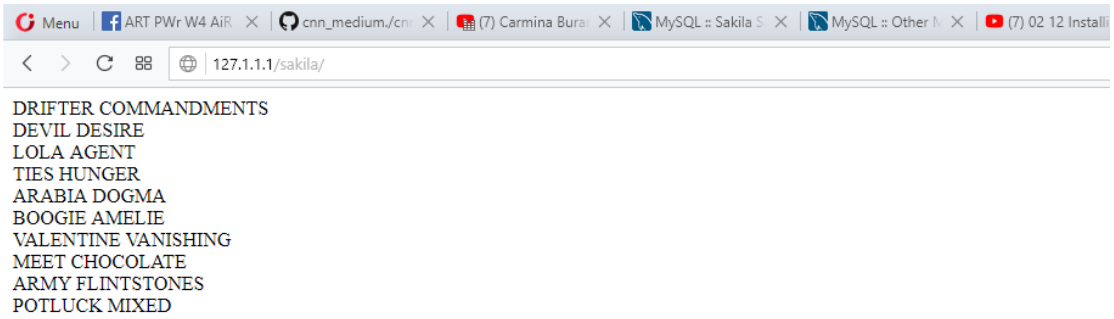

Figure 1: Załadowanie zapytania nr. 5 na serwerze lokalnym

4 Badania wydajnościowe MySQL

W tej części laboratoriów postawiono przed nami zadanie ustawienia serwera oraz aplikacji z front-endem i back-endem w celu testów bazy danych MySQL. Kolejne kroki:
1) Instalacja Dockera zgodnie z zaleceniami z:
https://docs.docker.com/install/linux/docker-ce/ubuntu/ 2) Wyłączanie apacha z portu :8000 komendą: "sudo /etc/init.d/apache2 stop" 3) Uruchomienie webservera w celu hostingu naszej aplikacji webowej
docker-compose build
docker-compose up -d
Teraz nasza strona jest widoczna pod lokalnym adresem:
http://localhost/app
4) Wprowadzenie skryptów z zapytaniami do bazy z poprzednich zajęć do aplikacji webowej
Do serwera wysłano zapytanie:
"select film.film id, film.title, inventory.film id, inventory.inventory id, COUNT(*) from inventory inner join rental on(inventory.inventory id=rental.inventory id) inner join film on(inventory.film id=film.film id) group by rental.inventory id order by COUNT(*) DESC LIMIT 10"
Co możemy tłumaczyć na:
"Check which movies are the most rented ones (top 10)"
Do wyświetlenia posłużono się pętlą for o liczbie iteracji równej ilości zwróconych rekordów, czyli 10.
W efekcie w przeglądarce pojawiła się lista 10 tytułów filmów:

W Kolejnej czesci przeprowadzono badanie wydajnosciowe polegajace na wysylaniu do bazy zapytania jak wyżej i pomiarze czasu jego realizacji. Obciążamy bazę z jednego źródła. Wyniki zestawiono w tabeli:

ilosc zapytań	czas oczekiwania [ms]
5	289
10	533
50	2446
100	4805
500	9384
1000	19657
5000	N/A

Table 1: Zestawienie czasów odczytu z DB w zależności od liczby zapytań.h

KOMENTARZ: Na wykresie niebieskie punkty odpowiadaja rzeczywistym pomiarom, zaś punkty

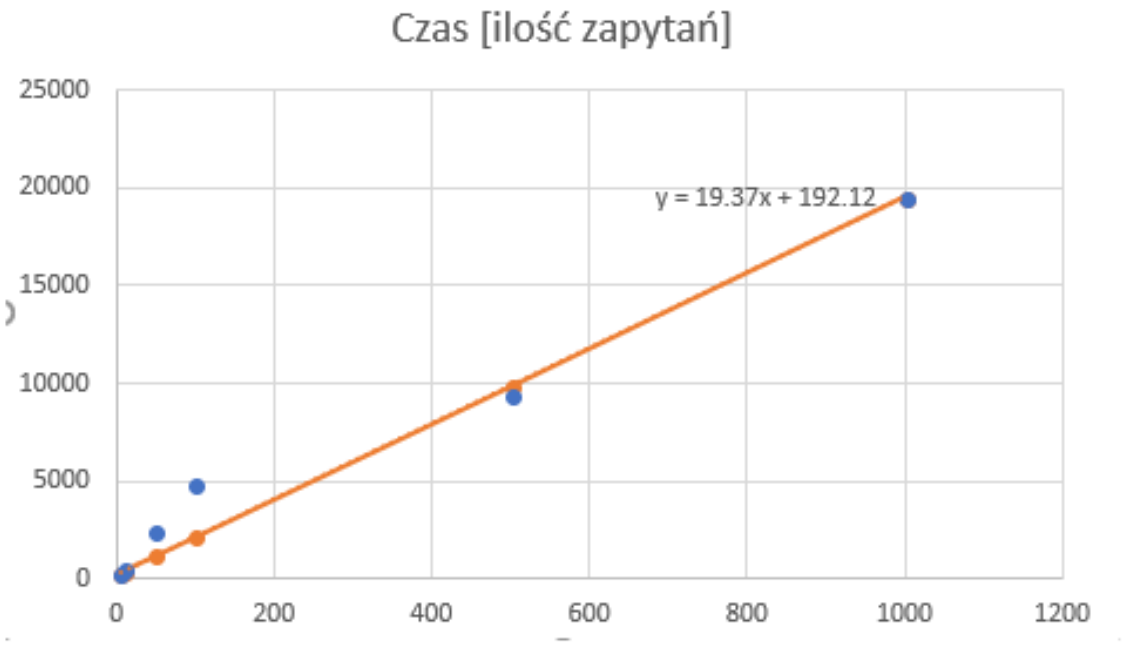

Figure 2: Załadowanie zapytania nr. 5 na serwerze lokalnym

pomarańczowe to aproksymacja pomiarów prostą o podanym równaniu

W ostatnim badaniu przetestowano bazę przez wysyłanie wielu zapytań z wielu źródeł. W ten sposób chciano zasymulować sytuację wielu użytkowników ciągle wysyła zapytania do bazy.
Metodyka badania:
Każdy użytkownik u generuje jaką ilość zapytań n. Po 0.5 sekundy włącza się następny użytkownik u+1 z idenntycznymi zapytaniami. Badamy czas po jakim otrzymają rezultaty poszczególni użytkownicy. Wyniki zestawiają tabele"

User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10
4851	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5131	5012	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5310	6541	6715	N/A	N/A	N/A	N/A	N/A	N/A	N/A
5474	8374	8225	8330	N/A	N/A	N/A	N/A	N/A	N/A
5584	9330	9543	9405	9343	N/A	N/A	N/A	N/A	N/A
5571	11784	11957	12011	11914	11789	N/A	N/A	N/A	N/A
5582	13882	14097	13945	13742	13895	13844	N/A	N/A	N/A
5584	14580	15468	14314	14628	14375	14898	14578	N/A	N/A
5548	14795	14852	14405	14343	14978	14657	14859	14978	N/A
5595	15214	15124	14999	15183	15021	15098	15134	15278	15151

Table 2: Zestawienie czasów odczytu z DB przy 100 zapytaniach dla różnej liczby Userów

User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10
19851	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
23981	28988	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28457	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28849	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28843	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28951	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28456	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28975	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28699	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
28421	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: Zestawienie czasów odczytu z DB przy 1000 zapytaniach dla różnej liczby Userów

KOMENTARZ: Wartość N/A w różnych komórkach pokazuja niemożność wykonania programu. Powodowane to jest przekroczeniem domyślnych 30s na połączenie przeglądarki z serwerem.

5 Spostrzeżenia i wnioski

Część I:
-Język MySQL pozwala nam w intuicyjny sposób przeszukiwać dane zawarte w tabelach wewnatrz DB. Często do poszukiwanych danych można się dostawać na kilka różnych sposobów, w związku z tym zachodzi potrzeba optymalizacji czasu wyszukiwania. Najczęstrzym kryterium optymalizacyjnym przy wyszukiwaniu danych w DB jest czas. Do badań złożoności obliczeniowej mogą nam posłużyć gotowe narzędzia. Jednak pomocna może okazać się też intuicja; wiemy, że czas wyszukiwania wydłuży się, gdy nie wprowadzimy limitu informacji zwrotnych. Również na naszą niekorzyść będzie miał wpływ łączenie wielu tabel w jedno, jeżeli chcemy pozyskać jaką pojedyńczą danę warto skonstruować podzapytania zamiast łączyć wiele tabel.
Część II:
-W tej części badaliśmy tranzakcję, jednak istotne jest to, że możliwość obsługi tranzakcji udostępniona jest jedynie w przypadku jednego silnika (InnoDB), szczęśliwie jest on implementowany domyślnie, jednak zawsze możemy sami zadeklarować wybór silnika przy tworzeniu tabeli:
CREATE TABLE actor (
...
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
Tranzakcję w silniku InnoDB pozwalają nam na spójność danych zgodnie z zasadą ACID (atomicity, consistency, isolation, and durability).
Tranzakcje w skrócie można opisać jako zestaw operacji SQL, które mogą być wykonane jedynie wszystkie albo żadna.

Poziom izolacji transakcji wpływa bezpośrednio na zachowanie się transakcji. Zmiana poziomu izolacji może prowadzić do zupełnie różnych wyników poleceń SQL.

Poziom izolacji transakcji oznacza jak "szczelnie" jest zaizolowana tranasakcja i jakiego rodzaju izolacja jest skojarzona z zapytaniami wewnątrz transakcji. Można wybrać jeden z czterech poziomów izolacji (wymienionych poniżej w kolejności rosnącej szczelności izolacji).

READ UNCOMMITTED // Ustawienie takiego poziomu transakcji powoduje dopuszczenie tzw. "dirty reads", tzn. że niepotwierdzone poleceniem COMMIT efekty poleceń z jednej transakcji są widoczne z poziomu drugiej transakcji.
READ COMMITTED
Potwierdzone poleceniem COMMIT zmiany danych w tablicach są widoczne z poziomu innych transakcji. Oznacza to, że identyczne polecenia w obrębie tej samej transakcji mogą zwrócić zupełnie inne wyniki.
REPEATABLE READ
Jest to domyślny sposób izolacji transakcji dla tablic typu InnoDB. W obrębie transakcji wszystkie zapytania są spójne.

SERIALIZABLE
Jeśli w obrębie jednej transakcji wykonujemy właśnie polecenie SELECT wówczas z poziomu dowolnej innej transakcji nie możemy wykonać zmiany danych, które są właśnie wybierane poleceniem SELECT. Inaczej mówiąc zapytania w obrębie tranakcji są wykonywane tak, jakby automatycznie była do nich dołączana klauzula

Część III:
W tej części przeprowadzono dość pobieżne badania wydajnościowe bazy danych MySQL z poziomu aplikacji webowej postawionej na loclahost, i tak: -w przypadku badań czasu wykonania wielu zapytań select wnioskujemy, że czas rośnie zależnością O(n(logn)), znajduje to swoje potwierdzenie w literaturze, ponieważ dane w bazie MySQL są organizowane w B-drzewo, które jest zbalansowane
.
- Bardzo ciekawe spostrzeżenia pojawiły się w przypadku drugiej części badań. Z danych w tabeli 2 wynika, że serwer zawsze wykonywał zapytania pierwszego klienta z tym samym czasem, pozostale różniły się znacząco od pierwszego, ale były do siebie podobne. Wraz ze wzrostem liczby użytkowników notowaliśmy wzrost czasu wyszukiwania. W badaniu dla wielu użytkowników i zapytaniu wielkości 1000 Select zauważamy, że serwer nie wyrabia, ale zawsze odpowiada pierwszemu klientowi, zaś pozostali kończą z błędem o przekroczeniu czasu połączenia.

Komentarz: Badania nie muszą odzwiercidlać rzeczywistości, ponieważ nie potrafimy wskazać, która część opragromowania nie radziła sobie z nadmiarowym ruchem. Mógł byc to serwer, ale równie dobrze mógłbyć to problem z silnikiem DB.
Czas wyszukiwania można zmniejszyć, jeśli wybierzemy jeden z innych silników dostęppnych z MySQL (MyIsam, Memory, Isam) lecz stracimy spójność danych oraz możliwość wprowadzania tranzakcji

6 Literatura

https://dev.mysql.com/doc/ - oficjalna dokumentacja MySQL
http://www.mif.pg.gda.pl/homepages/mate/bazydanych/transakcje.html - opis tranzakcji w DB
http://roman.ptak.staff.iiar.pwr.wroc.pl/BDwykladnr3ver5.pdf - wykład dotyczący MySQL Dr. Ptaka
https://web.stanford.edu/class/cs245/homeworks/b+tree/MySQLInnoDB.B+Tree.pdf - opis struktur danych stosowanych w bazach danych