

Języki i paradygmaty programowania
Optymalizacja zapytań za pomocą języka
Prolog

Samir Al-Azazi

w66045



WYŻSZA SZKOŁA
INFORMATYKI I ZARZĄDZANIA
z siedzibą w Rzeszowie

**Wyższa Szkoła Informatyki i Zarządzania
Kolegium Informatyki**

Rzeszów 16 stycznia 2024

Spis treści

1	Wstęp	2
2	Implementacja	2
2.1	Aplikacja serwerowa	2
2.2	Implementacja Prolog	3
3	Pomiary	6
3.1	Natywne zapytanie SQL	6
3.2	Zapytania wspomagane przez Prolog	6
4	Wnioski	7

1 Wstęp

Język Prolog¹ pozwala w prosty sposób rozwiązywać problemy związane z logiką. Pozwala również na wydajne działania na dużych zbiorach danych. W tym projekcie, wykorzystamy język Prolog do optymalizacji zapytań do bazy danych.

W tym celu, wykorzystamy SQLite² i użyjemy bazy wiedzy Prolog do zapisania danych w postaci cache³. Pozwoli to na uniknięcie wykonywania dodatkowych zapytań do bazy danych, co jest częstym problemem w aplikacjach webowych z mechanizmami mapowania obiektowo-relacyjnego (ORM⁴).

2 Implementacja

2.1 Aplikacja serwerowa

Aplikacja serwerowa została napisana w języku Golang⁵, w oparciu o narzędzia biblioteki standardowej.

Za pomocą interfejsu HTTP, aplikacja wystawia punkty końcowe (ang. endpoints⁶) do komunikacji REST⁷.

:

- GET /api/users
- POST /api/users?name={name}
- POST /tags?user_id={user_id}&name={name}
- DELETE /tags?user_id={user_id}

Za ich pomocą użytkownik może pobrać użytkowników, wraz z przypisanymi do niego tagami, dodać nowego użytkownika, dodać tag do użytkownika oraz usunąć wszystkie tagi użytkownika.

¹[https://pl.wikipedia.org/wiki/Prolog-\(j%C4%99zyk_programowania\)](https://pl.wikipedia.org/wiki/Prolog-(j%C4%99zyk_programowania))

²<https://pl.wikipedia.org/wiki/SQLite>

³https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_podr%C4%99czna

⁴https://pl.wikipedia.org/wiki/Mapowanie_obiektowo-relacyjne

⁵<https://go.dev/>

⁶https://en.wikipedia.org/wiki/Endpoint_interface

⁷https://pl.wikipedia.org/wiki/Representational_state_transfer

Zapytania do bazy danych zostały sztucznie opóźnione, by zasymulować opóźnienia w komunikacji sieciowej, używając wzorca projektowego **Decorator**⁸.

2.2 Implementacja Prolog

Został przygotowany program w języku Prolog, w pliku **curator.pl**.

```
1 :- set_prolog_flag(verbose, silent).
2 :- initialization main.
3 head([H|_], H).
4 main :-
5     current_prolog_flag(argv, Argv),
6     head(Argv, Who),
7     atom_number(Who, WhoNum),
8     related(WhoNum, X),
9     format('~w', X),
10    halt.
11 main :-
12     format('0'),
13    halt.
```

⁸<https://refactoring.guru/design-patterns/decorator>

Wyłączamy wypisywanie dodatkowych informacji po interpretacji programu.

```
1      :- set_prolog_flag(verbose, silent).
```

Oznaczamy predykat **main** jako punkt wejścia programu.

```
2      :- initialization main.
```

Definiujemy predykat **head**, który zwraca pierwszy element listy.

```
3      head([H|_], H).
```

Definiujemy predykat **main**, który jest wywoływany przy uruchomieniu programu.

```
4      main :-
```

Pobieramy listę argumentów wywołania programu.

```
4      current_prolog_flag(argv, Argv),
```

Pobieramy pierwszy argument, używając utworzonego wcześniej predykatu **head**.

```
4      head(Argv, Who),
```

Konwertujemy argument na atom liczbowy.

```
4      atom_number(Who, WhoNum),
```

Zapisujemy wynik zapytania **related** do zmiennej **X**.

```
4      related(WhoNum, X),
```

Wypisujemy wynik na wyjście standardowe.

```
4      format('~w', X),
```

I kończymy działanie programu.

```
4      halt.
```

Jeśli którykolwiek z powyższych predykatów nie został spełniony, program przechodzi do tego etapu.

```
4      main :-
```

Wypisujemy 0 na wyjście standardowe.

```
4     format('0'),
```

I kończymy działanie programu.

```
4     halt.
```

W momencie gdy dodamy nowy TAG do użytkownika, do bazy danych zostanie dodana nowa informacja.

```
related(1,1)
```

Informuje o tym, że użytkownik o ID 1 jest w relacji do tagów i aplikacja powinna odpytać o nie bazę danych.

Możemy sprawdzić działanie za pomocą terminala, wykonując polecenie

```
swipl curator.pl 1
```

W ten sam sposób, program jest wywołany z aplikacji serwerowej, gdzie 1 to ID użytkownika.

3 Pomiary

3.1 Natywne zapytanie SQL

W bazie danych znajduje się 4 użytkowników.

```
[
  {
    "ID": 1,
    "Name": "John",
    "Tags": null
  },
  {
    "ID": 2,
    "Name": "John",
    "Tags": null
  },
  {
    "ID": 3,
    "Name": "John",
    "Tags": null
  },
  {
    "ID": 4,
    "Name": "John",
    "Tags": null
  }
]
```

Czas pobrania ich za pomocą zapytania **GET /api/users** wynosi 6 sekund.

3.2 Zapytania wspomagane przez Prolog

Po dodaniu dekoracji Prolog do zapytania, czas pobrania 4 użytkowników wynosi 3 sekundy.

4 Wnioski

Zastosowanie języka Prolog do optymalizacji zapytań do bazy danych pozwala na znaczne przyspieszenie działania aplikacji przy dużych nakładach czasowych na mapowanie obiektowo-relacyjne.

Wadą takiego rozwiązania jest konieczność utrzymywania dodatkowej bazy wiedzy, która musi być aktualizowana w momencie zmian w bazie danych i może powodować problemy z synchronizacją z powodu dwóch źródeł prawdy.

Można minimalizować te problemy poprzez zastosowanie mechanizmów automatycznego czyszczenia bazy wiedzy co dany interwał czasowy, lub w momencie wykrycia zmian w bazie danych.

Zbudowanie generalnego mechanizmu, który można zintegrować z różnymi narzędziami może przynieść znaczne korzyści, gdzie występuje problem $N+1$ ⁹.

⁹<https://stackoverflow.com/questions/97197/what-is-the-n1-selects-problem-in-orm-object-relational-mapping>