Języki i środowiska przetwarzania danych rozproszonych

Wprowadzenie do przetwarzania kolekcji w językach programowania

Wykładowca: **Tomasz Kowalski**

Wykład przygotowany
na podstawie materiałów
prof. Kazimierza Subiety

Generalne założenia podejścia stosowego

stack-based approach, SBA

- **Podejście stosowe zakłada, że języki zapytań są szczególnym przypadkiem języków programowania.**
- Stąd teorie języków programowania są bardziej adekwatne niż podejścia takie jak algebra relacyjna, rachunek relacyjny lub logika matematyczna.
- W podejściu stosowym kluczową rolę odgrywa **stos środowisk** (*environment stack*), który jest podstawowym mechanizmem praktycznie wszystkich popularnych języków programowania.
- Jego rolą jest określenie zakresów nazw (*scoping*), wiązanie nazw (*binding*) oraz wprowadzenie dyscypliny w zakresie alokowania dynamicznych bytów programistycznych, w szczególności lokalnych danych (obiektów) i parametrów procedur.
- **EFEKTy:** Dwóch zdolnych studentów znających podejście stosowe potrafi zaprojektować i zaimplementować w ciągu roku lepszy i mocniejszy język zapytań niż duże konsorcja specjalistów z renomowanych firm zachodnich. Ten eksperyment został już przynajmniej 5-krotnie powtórzony w PJWSTK.

SBQL

- Języka zapytań SBQL (*Stack Based Query Language*) jest oparty na podejściu stosowym.
- SBQL jest pewną idealizacją opartą na abstrakcyjnej składni, pokazującą istotę semantyki operatorów wprowadzonych w większości języków zapytań.
- SBQL pełni on tę samą rolę, którą w relacyjnych językach zapytań pełni algebra relacji i rachunek relacyjny.
- Zasadniczą nowością wprowadzoną w SBQL są operatory *nie-algebraiczne*, takie jak operatory selekcji, projekcji/nawigacji, złączenia, kwantyfikatory, operator tranzytywnego domknięcia i operator porządkowania.
- Semantyka tych operatorów nie może być wyrażona w terminach algebry podobnej do algebry relacji. Jak się okazało, ich semantykę można w prosty sposób wyrazić poprzez operacje na stosie środowiskowym, przy czym pewnym zaskoczeniem jest to, że semantyka tych wydawałoby się bardzo różnych operatorów jest oparta o ten sam prosty mechanizm.

Zapytania = wyrażenia

- W ***naszym*** podejściu zapytania będą pełnić rolę *wyrażeń* znanych z języków programowania.
- Przyjęta przez nas zasada **ortogonalnej trwałości** nie różnicuje sposobów dostępu do trwałych i nietrwałych danych.
- Wobec tego nasze zapytania będą również stosowane do nietrwałych danych.
- Inaczej mówiąc przyjęliśmy, że w naszym hipotetycznym języku programowania nie będzie innych wyrażeń niż zapytania.
- Zapytaniami będą więc także klasyczne wyrażenia takie jak 2+2, *sin*(*x*), *A*[*i*+1], itd.
- To założenie jest pewną rewolucją w odniesieniu do języków zapytań.
- Tradycyjnie, np. w języku SQL zapytania (zagnieżdżone w język programowania) mogły odwoływać się do zmiennych języka programowania poprzez specjalną składnię.

SBQL vs SQL proste przykłady (1/2)

- Zwróć wszystkie informacje o pracownikach

SQL: **select** * **from** *Emp*

SBQL: *Emp*

- Zwróć nazwiska wszystkich pracowników

SQL: **select** name **from** *Emp*

SBQL: *Emp.name*

SBQL vs SQL proste przykłady (2/2)

- Zwróć wszystkie informacje o pracownikach o nazwisku „Doe”

SQL: **select** * **from** *Emp* **where** *name* = „Doe”

SBQL: *Emp* **where** *name* = „Doe”

- Zwróć nazwiska i zarobki pracowników o nazwisku „Doe”

SQL: **select** name, sal **from** *Emp* **where** *name* = „Doe”

SBQL: (*Emp* **where** *name* = „Doe”).(name, sal)

SBQL vs PYTHON przykład

- Zwróć unikatowe nazwiska pracowników (tzn. występujące tylko raz).

PYTHON:

```
di = {}
for w in names:
        di[w] = di.get(w, 0) + 1
uniqueWords = [k for k,v in di.items() if v == 1]
```

SBQL:

```
(Emp as e1 where
  count(Emp where name=e1.name)=1).e1.name
```

SQL:

```
select e1.name from Emp e1 where
  (select count(*) from Emp
        where name = e1.name)=1;
```

SBQL vs grupowanie w SQL przykład (1/2)

- Zwróć średnią liczbę pracowników w działach.

SQL: **select** avg(*aux.c*) **from**
(**select** count(*) **as** c **from**
Emp **group by** *dept_id*) **as** *aux*;

SBQL: avg(*Dept*.count(*Emp* **where** *id_dept* = *dept_id*))

SBQL: avg((**unique**(*Emp.dept_id*) **as** *id_dept*).
count(Emp **where** *id_dept* = *dept_id*))

SBQL (model danych objektowy):
avg(*Dept*.count(*employs*))

SBQL vs grupowanie w SQL przykład (2/2)

- Dla każdego departamentu, w którym pracuje przynajmniej 50 pracowników, pobierz nazwę działu i średnią pensje.

SQL:

select *d.name*, **avg**(*e.sal*) **from** *Emp e*, *Dept d*
where *e.dept_id* = *d.id_dept*
group by *e.dept_id*, *d.name*
having count(*e.**) > 50

SBQL:

((*Dept* **as** *d*) **join** ((*Emp* **where** *dept_id* = *d.id_dept*)
groupas *e*) **where count**(*e*) > 50).
(*d.name*, **avg**(*e.sal*))

SBQL (model danych objektowy):

(*Dept* **where count**(*employs*) > 50).
(*name*, **avg**(*employs*.*Emp*.*sal*))

SBQL4J - SBQL i Java

- Znajdź produkty, które są na magazynie i kosztują więcej niż 3 za sztukę.

```
List<Product> products = getProductList();
List<Product> expensiveInStockProducts = #{
   products
   where unitsInStock > 0 and unitPrice > 3.00
};
```

- Czy to prawda, że każdy departament zatrudnia pracownika zarabiającego tyle co jego szef?

```
List<Department> dept = data.getDepts();
Boolean res = #{
      all (dept as d)
            any ((d.employs minus d.boss) as e)
                  (e.salary == d.boss.salary)
};
```

Języki zapytań jako wyrażenia programistyczne

- To podejście zakłada nowy rodzaj języka programowania, w którym występują specyficzne wyrażenia (podobne do klasycznych wyrażeń języka oprogramowania) zwane „zapytaniami”.
- Istotą tych nowych wyrażeń jest obsługa kolekcji.
- W tej roli języki zapytań są wyższym szczeblem abstrakcji nad konstrukcjami organizującymi pętle (*while*, *repeat*, *goto*, *for*, *loop*, itp.), iteratorami, kursorami i innymi tego rodzaju udogodnieniami.
- Zapytania koncepcyjnie „hermetyzują” pętle iteracyjne w języku programowania przy pomocy operatorów takich jak selekcja (*where*), projekcja, złączenie, unia, kwantyfikatory, grupowanie, sortowanie, itp.
- Słowo „koncepcyjnie” jest tu istotne, gdyż chodzi o taką hermetyzację, która jest naturalna, zrozumiała i czytelna dla programisty; wspomagająca procesy modelowania pojęciowego przy tworzeniu aplikacji.
- W tej koncepcji języki zapytań są tworami całkowicie ortogonalnymi w stosunku do cechy trwałości danych (czyli bazy danych).