Query Optimization For Semistructured Data

Jason McHugh, Jennifer Widom

Stanford University

Autor: Mateusz Łupiński

Trochę o artykule

- Pełna implementacja systemu DBMS Lore i języka zapytań Lorel
- Rok powstania: 1999/2000
- Sun Ultra 2, 256mb RAM
- Bazy danych rozmiaru 5mb

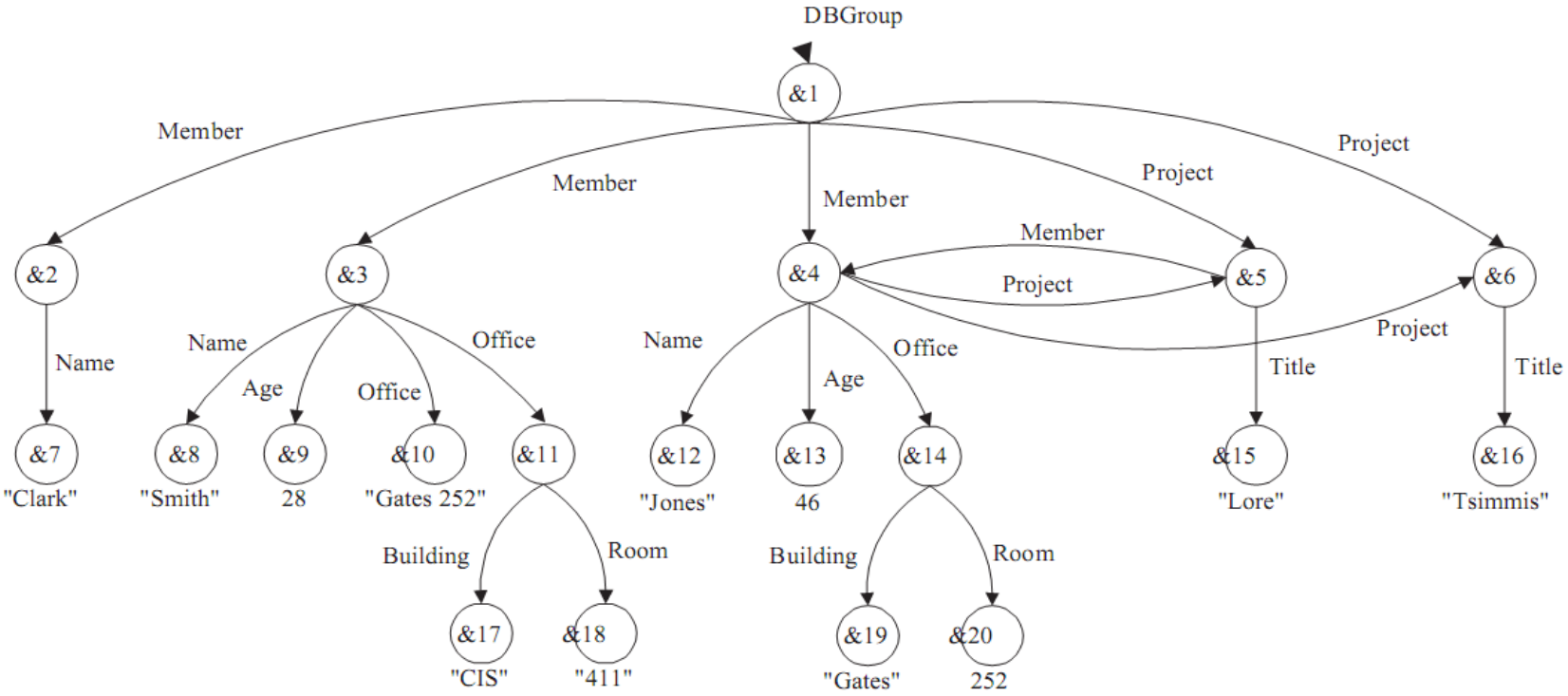
Semistructured Data

„Dane mogące być nieregularne i niekompletne oraz takie, których struktura może się łatwo i nieprzywidywalnie zmieniać”

Model oryginalny: Object Exchange Model

XML

OEM




```
<Member Project="&5 &6">
   <Name> Jones</Name>
   <Age>46</Age>
   <Office>
      <Building>Gates</Building><Room> 252</Room>
   </Office>
</Member>
```

OEM

- Wierzchołki – obiekty (złożone i atomowe)
- Simple Path Expression
  - x.l y ← x nazwa lub zmienna, l – etykieta, y - wartości
- Path Expression
  - np. DBGroup.Member x, x.Age y

Lorel

Przykład:

```
QUERY 1:
  Select x
  From   DBGroup.Member x
  Where  exists y in x.Age: y<30
```

```
RESULT: <Member>
          <Name>Smith</Name>
          <Age>28</Age>
          <Office>Gates 252</Office>
          <Office>
             <Building>CIS</Building>
             <Room>411</Room>
          </Office>
        </Member>
```

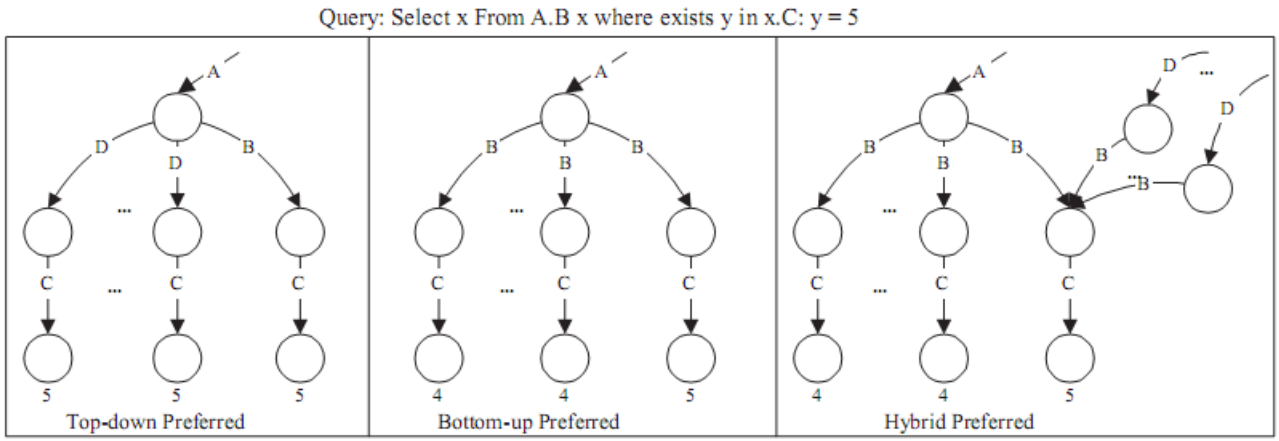
Po parsowaniu i preprocessingu generacja wysokopoziomowego, logicznego planu wykonania zapytania.

Indeksy w Lore

- Standardowe są nieprzydatne
- Vindex – in: etykieta, predykat; out: obiekty atomowe
- Lindex – in: obiekt, etykieta; out: rodzic
- Bindex – in: etykieta; out: pary rodzic – dziecko
- Pindex – in: path expression; out obiekty

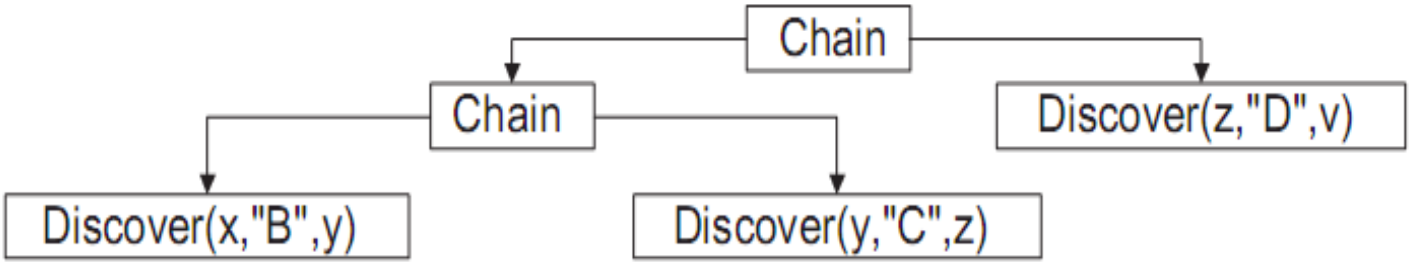
Motywacja

- Różne strategie wykonywania zapytań
  - Top-down („pointer-chasing”)
  - Bottom-up („reverse pointer-chasing”)
  - Strategie hybrydowe
- Dla jakich przypadków są one dobre?



Logiczny Plan Zapytania

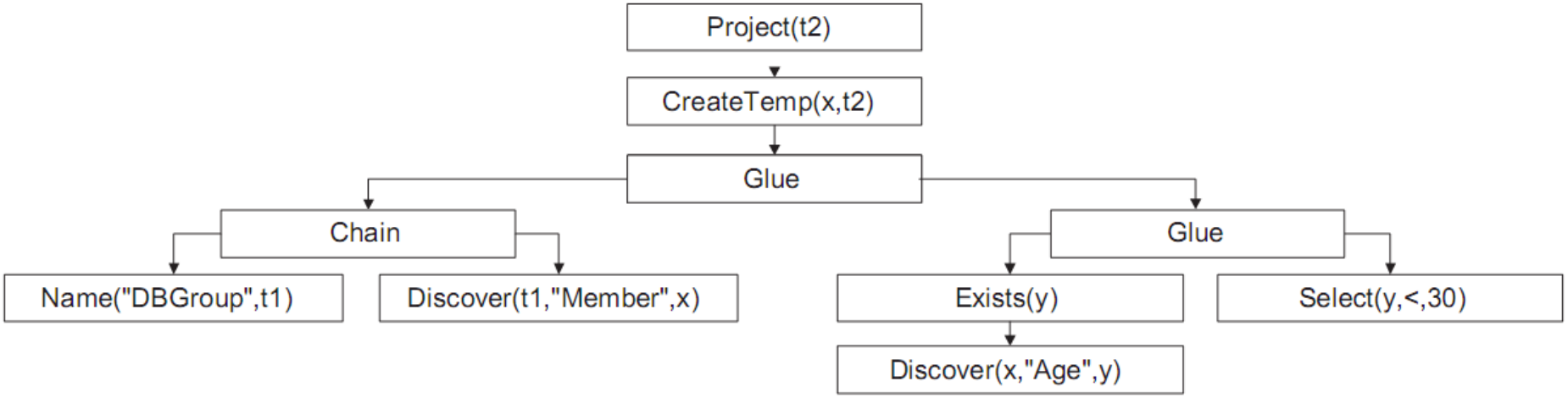
- Path Expression → Discover & Chain



Powyższy dla wyrażenia x.B y, y.C z, z.D v

Logiczny Plan Zapytania

```
QUERY 1:
  Select x
  From   DBGroup.Member x
  Where  exists y in x.Age: y<30
```



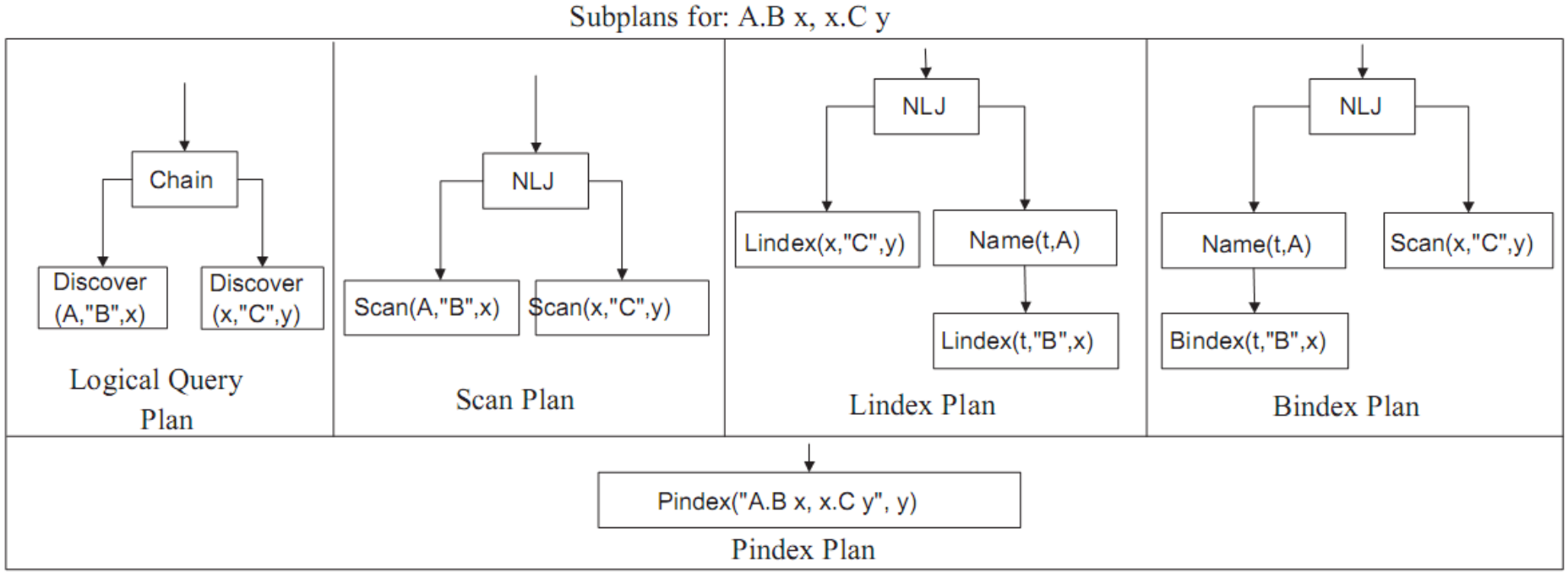
Z powyższego powstanie fizyczny plan zapytania.

Fizyczny Plan Zapytania

6 operatorów do pobierania danych z bazy

- Scan(x,I, y) – pointer-chasing, uzupełnia y
- Lindex(x, I, y) – uzupełnia x
- Pindex(PathExpression, x)
- Bindex(x, I, y) – uzupełnia x, y
- Name(x, n) – sprawdza nazwę
- Vindex(Op, Value, I, x) – uzupełnia x

Przykładowe fizczyne plany zapytań



NLJ – Nested Loops Join –
złączenia nie są każdy z każdym –
lewe wiązanie przekazuje prawemu
już związane zmienne.

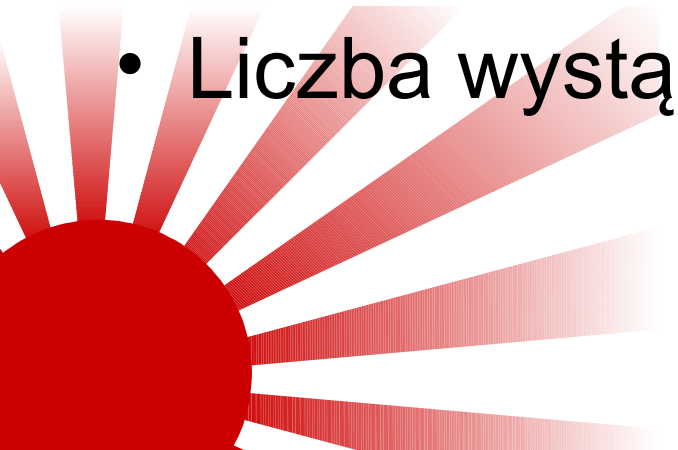
Koszt wykonania zapytania

- Podstawowe kryterium: Liczba operacji I/O
- Załadowanie każdego obiektu to operacja I/O
- Nie zakładamy ich gromadzenia się – pesymistycznie każdy jest na innej stronie w pamięci.
- Wyniki czasowe potwierdzają, że to dobre kryterium

Statystyki

k – wartość parametryzująca – zbieramy statystyki dla ścieżek p długości <= k

- Dla każdego typu atomowego, sumaryczna liczba atomowych obiektów tego typu osiągalnych z p oraz ich wartość minimalna i maksymalna
- Liczba wystąpień ścieżki p w bazie



Statystyki cd.

- Liczba różnych obiektów osiągalnych z p
- Dla etykiety l, liczba podobiektów osiągalnych etykietą l z obiektów osiągalnych z p
- Dla etykiety l, liczba obiektów z których można osiągnąć etykietą l obiekty osiągalne z p

Koszt wykonania zapytania

- Dzięki statystykom mamy oszacowania liczby operacji I/O dla każdego operatora
- Wartość drugorzędna: CPU time – wynika wprost z każdego operatora
- Dla każdego planu wyliczamy jego koszt (w operacjach I/O oraz CPU time)
- Niespodzianka: wybieramy najtańszy plan! :)

Generowanie planów

- Path Expression długości n to jak n joinów
  - Joiny to: (reverse) pointer-chasing, NLJ, hash - joins
- Niech będzie m sposobów na wykonanie każdego joina oraz k metod dostępu do każdej relacji
- Mamy $n!m^{(n-1)}k^n$ sposobów na wykonanie joina

Generowanie planów startegia zachłanna

- Zaczynamy od planu logicznego
- W każdym jego węźle podejmujemy lokalnie optymalną decyzję co do zastosowania konkretnego operatora bazując na wartościowaniu przekazanym od rodzica węzła i na informacjach uzyskanych od dzieci węzła dla proponowanych im przez nas różnych planów
- To wciąż jest wykładniczo dużo stanów...

Generowanie planów startegia zachłanna - heurystyki

- Nie rozważa się joinowania ścieżek jeżeli nie mają wspólnej zmiennej
- Pindex stosuje się tylko jeżeli zmienna jest jego ostatnim elementem
- Optymizator nie próbuje zmieniać kolejności niezależnych path expressions
- Klauzulę Select zawsze wykonuje się na końcu

Generowanie planów jak to działa?

- Dla każdego wierzchołka rozważamy najlepsze sposoby jego „implementacji” - tj. najlepszego operatora
- W każdym wierzchołku „glue” generujemy plany z-lewej-do-prawej i z-prawej-do-lewej

Generowanie planów czy to działa?

- Na pewno ogranicza przestrzeń poszukiwań:

n: number of simple path expressions	n=3	n=5	n=7
All possible plans/Lore's search space	1458 / 48	2,361,960 / 228	8,035,387,920 / 948

- Jeżeli nie znajduje rozwiązania optymalnego, to zawsze coś nieodległego od niego – potwierdzone empirycznie
- Zdecydowanie broni się wynikami

Implementacja

- 31000 linii kodu w C++
- Dla 5mb bazy, ok 8.1mb statystyk
- Graf bazy złożony z 62k węzłów i 130k etykiet
- Maszyna Sun Ultra 2, 256mb RAM
- Czasy optymizacji: średnio ok 0.5s
- Wyniki potwierdzają założenia o skuteczności

State of the Art

„ The Lore project was declared a success in the year 2000 and is now pretty much out of business. These pages represent a snapshot of the project at some time in the past. We hold no responsibility for the currency (or lack thereof) of their content. „

Ale działa już w pełni na XML.

Native XML Database System

About Sedna

Sedna is a free native XML database which provides a full range of core database services - persistent storage, ACID transactions, security, indices, hot backup. Flexible XML processing facilities include W3C XQuery implementation, tight integration of XQuery with full-text search facilities and a node-level update language.

» **Download Sedna 3.3**
Available for Windows, Linux, MacOS X (x86/PPC), FreeBSD, Solaris (x86)

» **Try Sedna in WikiXMLDB Demo**
Query English Wikipedia in XQuery

Basic Features

- Available for free in open source form under Apache License 2.0
- Native XML database system implemented in C/C++
- Support for W3C XQuery language validated by W3C XQuery Test Suite
- Full-text search integrated with XQuery (based on dtSearch)
- Support for a declarative node-level update language »
- Support for ACID transactions
- Support for fine-grained XML triggers »
- Incremental hot backup »
- Indexes (based on B-tree) »
- Support for Unicode (utf8)
- SQL connection from XQuery »
- XQuery external functions implemented in C »
- Database security (users, roles and privileges) »

Application Programming

The following drivers are available from the download page:

- API for Java/C/PHP/Python/Ruby/Perl/Delphi/C# and more
- XQJ and XML:DB drivers for Java
- Module for Integration with Apache HTTP server

Documentation

Getting Started »
Frequently Asked Questions (FAQ) »
Programmer's Guide »
Administration Guide »
» More

Latest News

07-04-2010
Perl Driver and Catalyst Adapter for Sedna by Daniel Ruoso Released »

11-03-2010
Sedna 3.3 Released »

17-09-2009
Sedna XQJ Beta 1 Released »

24-04-2009
William Greenly has contributed ColdFusion wrapper »

10-03-2009
SednaAdmin for Web Released »

16-12-2008
Alexander Kardailsky has conributed Delphi API »

14-12-2008
Rolf Timmermans has contributed Ruby API »

06-25-2008
SednaAdmin 0.1.4 Released »

» More

Site Search

Dziękuję