

XQuery Processing

Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Petr LUKÁŠ, 2013
petr.lukas@nativa.cz

Paměťově orientovaný procesor

XQuery

- Deklarativní jazyk pro dotazování nad XML
- XQuery pro XML ~ SQL pro relační databáze
- Standardizace organizací W3C
- Rozšíření jazyka XPath
- Poslední verze standardu 3.0 [4], poslední platné doporučení 1.0
- Základem jsou FLWOR výrazy
- Na XQuery nelze nahlížet jen jako na „filtr“ XML dokumentu

```
for $b in /library//book
let $a := $b/author
where $a = “John Smith”
order by $b/isbn
return $b
```

```
<summary>
  { for $item in //item
    return count($item/*) }
</summary>
```

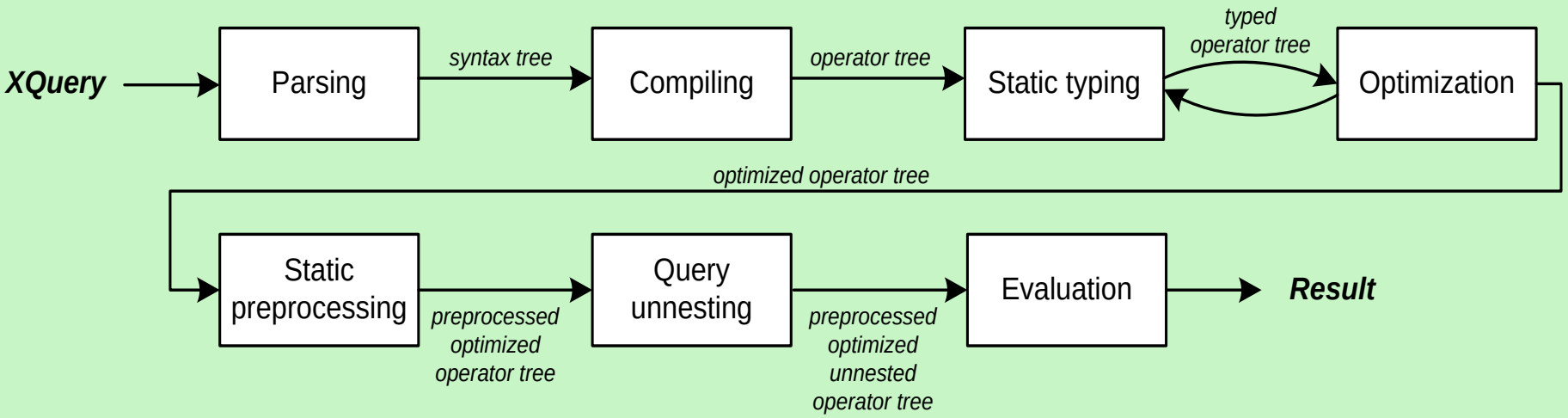
XQuery procesor

Práce se zabývá návrhem a implementací algebraického procesoru jazyka XQuery

---

- Původně procesor operující nad paměťovou DOM reprezentací XML dokumentu
- Podpora velkého množství rysů podle standardu:
  - FLWOR výrazy
  - XPath výrazy
  - Aritmetické výrazy a booleovská algebra
  - Kvantifikátory (some, every)
  - Větvení
  - Vestavěné funkce
  - Konstruktory XML
  - *a další …*
- Paměťově orientovaný procesor srovnatelný např. s implementací Saxon [5].

Blokové schéma



- **Plán dotazu = Strom operátorů** – aktuálně má každý operátor jeden výkonný algoritmus, do budoucna výběr z více algoritmů
- **XQuery Core** – podmnožina jazyka Xquery
- **Normalizace** – překlad z XQuery na XQuery Core
- Kompilační pravidla navržena tak, že si poradí s nenormalizovanou podobou dotazů.

Algebra

- **Datové položky a sekvence**
  (1, “text“, <xml_uzel />, atribut=“hodnota“)

- ***n*-tice a tabulky**

a	b
1	<item id=“1“>
3	(<item id=“2“>, 1, 2)

Algebra s úpravami převzata z [1]

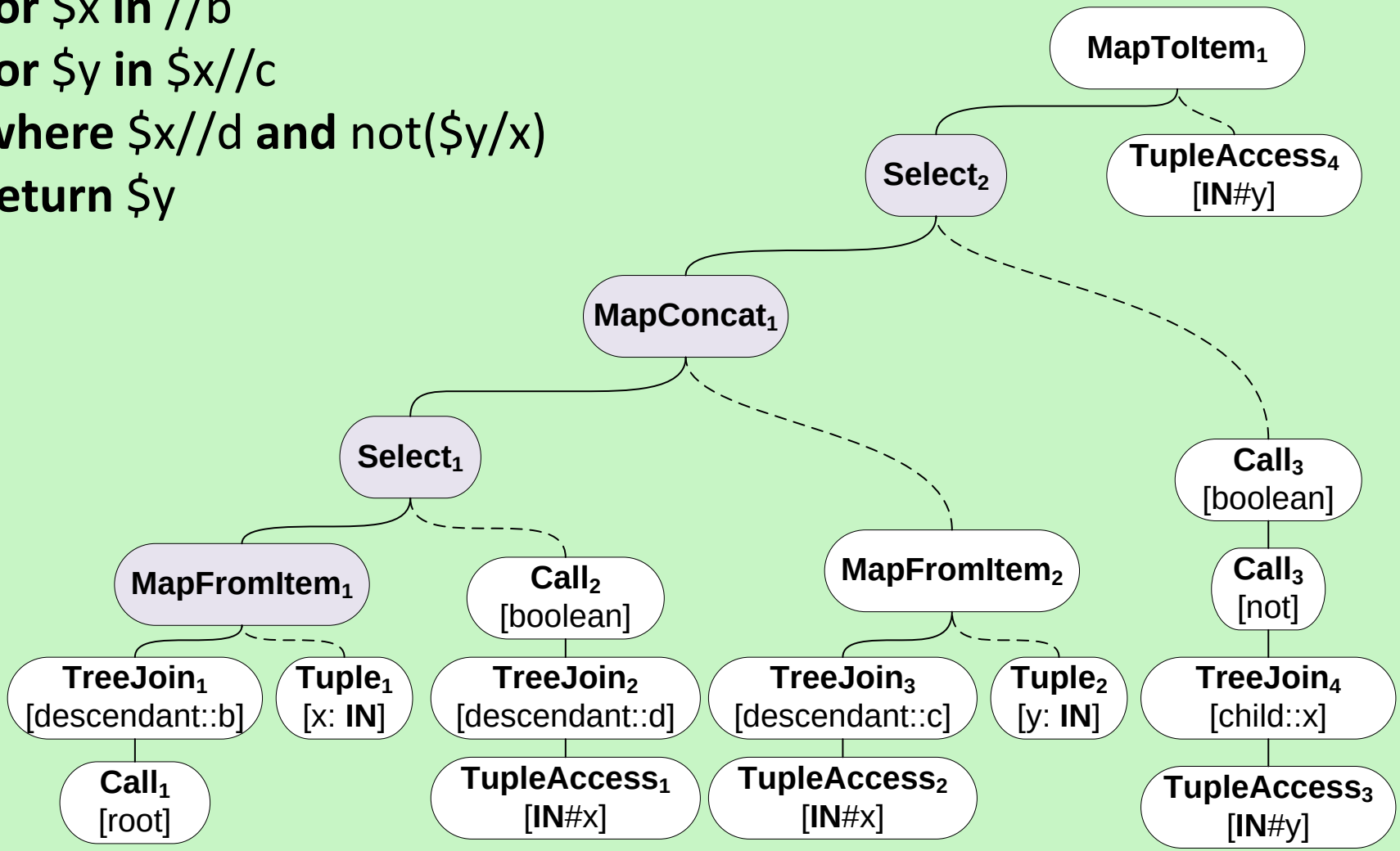
Operátory
Pracující nad sekvencemi
Pracující nad relacemi
Hybridní operátory

Stavba operátoru
Nezávislé vstupy
Závislé vstupy
Statické atributy

AND, *Attribute*, ***Call***, *Cast*, *Castable*, *Comment*, *Cond*, *Element*, *Empty*, ***EqualityJoin***, ***GroupBy***, ***IN***, ***Join***, ***LOuterEqualityJoin***, ***LOuterJoin***, ***MapConcat***, *MapEvery*, ***MapFromItem***, *MapIndex*, *MapIndexStep*, *MapSome*, ***MapToItem***, ***OMapConcat***, ***OR***, ***OrderBy***, ***Product***, *Scalar*, *Select*, *Sequence*, *Text*, ***TreeJoin***, *Tuple*, ***TupleAccess***, *TupleConcat*, ***TupleTreePattern***, *TypeMatches*

Plán dotazu

for $x **in** //b
for $y **in** $x//c
where $x//d **and** not($y/x)
return $y

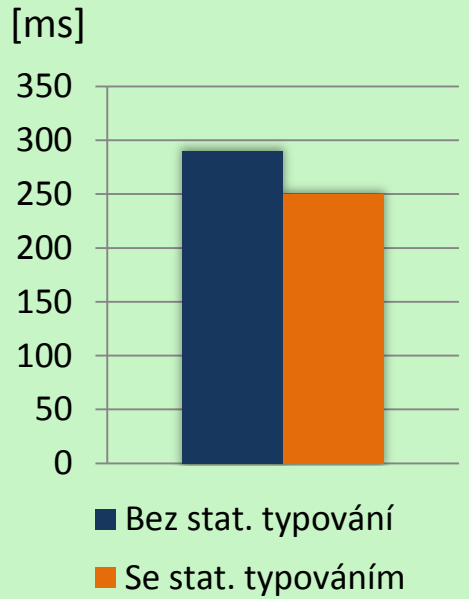


Statické typování

- Probíhá na úrovni plánu dotazu
- Každý operátor má definován výstupní typ, který může a nemusí záviset na podoperátorech
- U relací je vyhodnoceno schéma

//item[.//parlist//parlist]

- Nutno rozhodnout, jakého typu je výsledek predikátu
- Typ lze vypočítat ještě před spuštěním dotazu
- **Vede ke zjednodušení plánu dotazu a snazšímu nalezení TPQ** (viz dále)



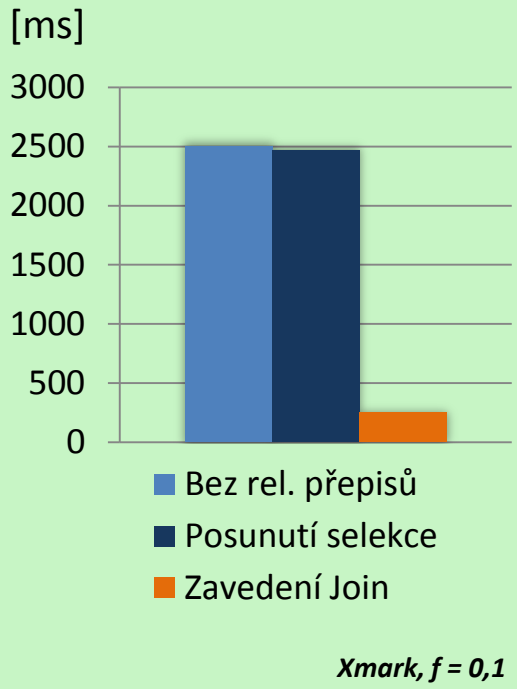
Xmark, f = 1

Relační přepisy

- Standardní přepisy z relační algebry
- Nebyly by uskutečnitelné bez znalosti schémat mezivýsledků a bez detekce přistupovaných proměnných

---

```
for $x in //closed_auctions/closed_auction
for $y in //people/person
where $x/buyer/@person = $y/@id
  and ($x/price/text() cast as integer) > 10
return <out> { $x/type/text() }, { $y/name/text() } </out>
```



Xmark, f = 0,1

- Rozklad bool. výrazu, posunutí selekce přímo za op. podstrom pro první **for**
- Příležitost použít operátor *Join*
- **Vede k výraznému urychlení zpracování dotazu a ke zjednodušení plánu**

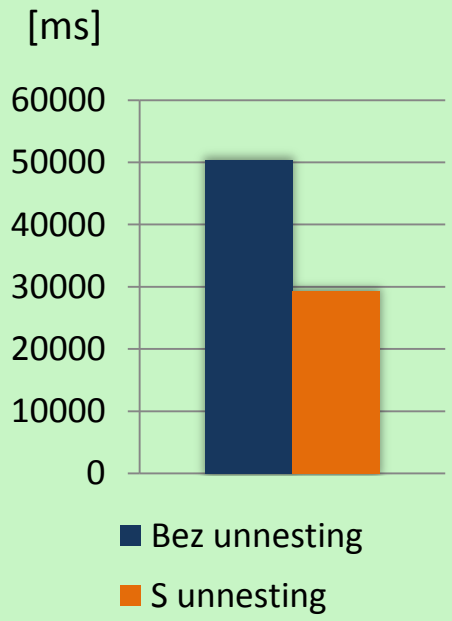
Query unnesting

- Obecně znamená eliminaci provádění zanořených smyček
- Může probíhat na úrovni optimalizace plánu dotazu (viz dále operátor *GroupBy*) nebo detekcí částí plánu, které budou mít pokaždé stejný mezivýsledek

---

```
for $location in distinct(//location/text())
let $count := count(//item[location/text() = $location])
order by $count
return <item> { $location }, { $count } </item>
```

- Např. část dotazu //item bude mít jistě pokaždé stejný výsledek a není ji tedy potřeba vyhodnocovat opakovaně
- **Vede k výraznému urychlení zpracování dotazu**

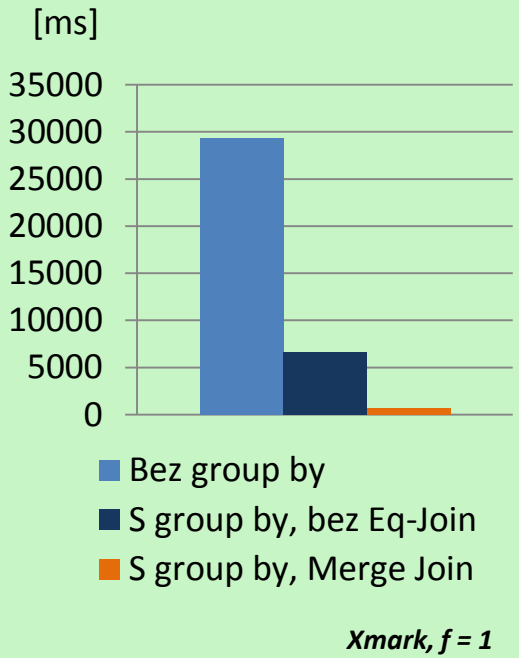


Xmark, f = 1

Operátor GroupBy

- Týká se použití klauzulí **let**
- Dotaz je vykonán podobně, jakoby místo **let** byla klauzule **for**, poté se provede seskupení a případný výpočet např. agregovaných hodnot.

```
for $location in distinct(//location/text())
let $count := count(//item[location/text() = $location])
order by $count
return <item> { $location }, { $count } </item>
```



- Zavedení *GroupBy* navíc umožní použití *LOuterJoin*
- **Vede k výraznému urychlení zpracování dotazu**

Integrace procesoru do frameworku

Integrace do frameworku

- Původní procesor operuje pouze nad paměťovou DOM reprezentací
- Problém v případě rozsáhlých XML, které nelze do paměti načíst najednou

---

1. **Využívání indexu pro elementární operace DOM**

Dokument se nenačítá do paměti, používá se persistentní úložiště QuickDb. Elementární metody navigace v DOM poskytuje třída **XMLDocumentStorage**.

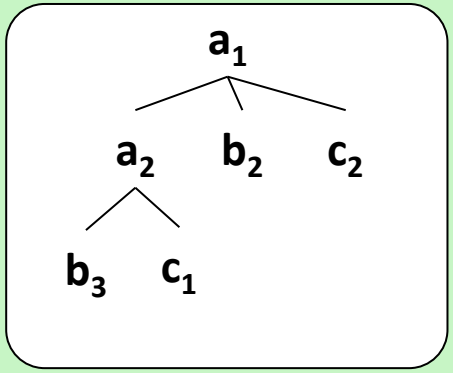
2. **Vyhledávání TPQ a aplikace algoritmu GTP Stack** [1]

Zavedení nových statických přepisů a speciálního operátoru

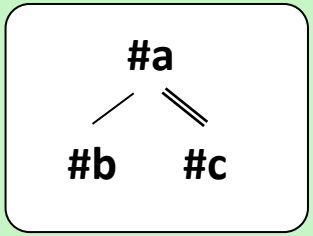
TPQ – Tree Pattern Query

- **Tree Pattern Query**, tzv. „větvený dotaz“
- Strom, jehož uzly tvoří názvy hledaných XML elementů a hrany vztahy, které mezi hledanými elementy musí platit.
- Výsledkem TPQ dotazu je množina *n*-tic, tzv. mapování.

XML



TPQ



výsledek

a	b	c
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_3	c_1

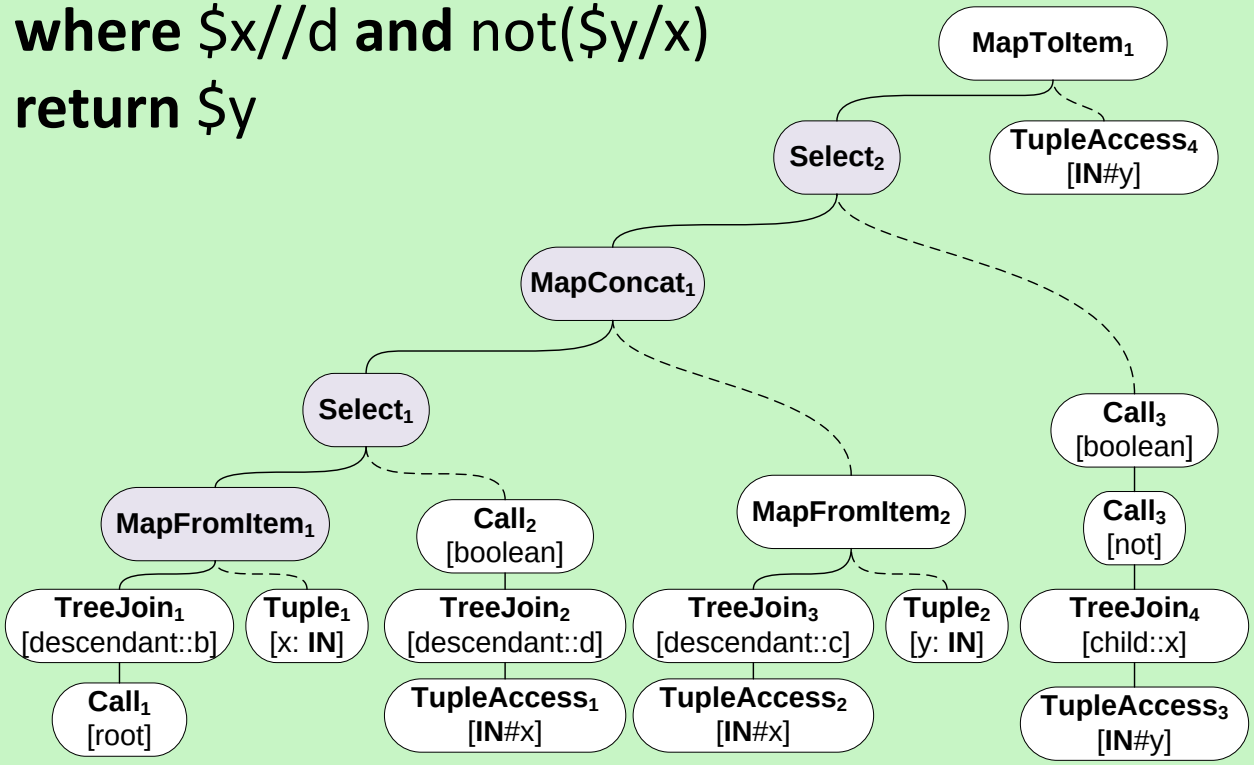
Operátor TupleTreePattern

- Vyhodnocování větvených dotazů jako celků je elementární úloha, kterou řeší pokročilé holistické algoritmy jako např. **GTP Stack** [1]
- Výhoda těchto algoritmů spočívá efektivním procházení streamů uzlů podle jejich názvu.
- Prozatím málo publikací (např. [3]) věnováno problému pokrývání XQuery dotazů pomocí TPQ
- Zavedení perátoru, jehož statickým argumentem je zobecněná varianta větveného dotazu – tzv. **GTP** (Generalized Tree Pattern)
- Operátor umožňuje vybraným uzlům větveného dotazu vnutit vypočtený proud XML uzlů – vyhodnocování složitějších predikátů.
- Cílem je pokrýt co největší část plánu dotazu jedním operátorem *TupleTreePattern*.

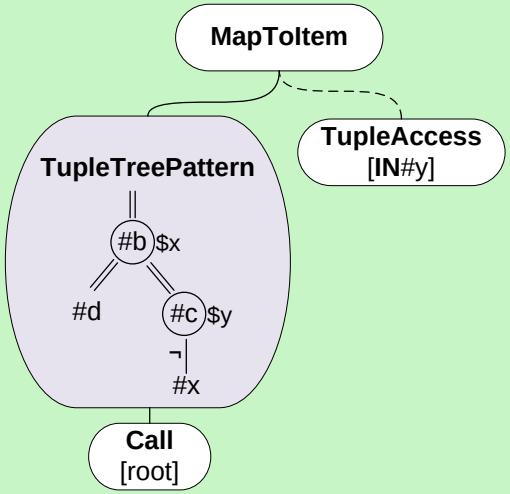
Ukázka plánů

for $x **in** //b, $y **in** $x//c
where $x//d **and** not($y/x)
return $y

Bez nalezení TPQ



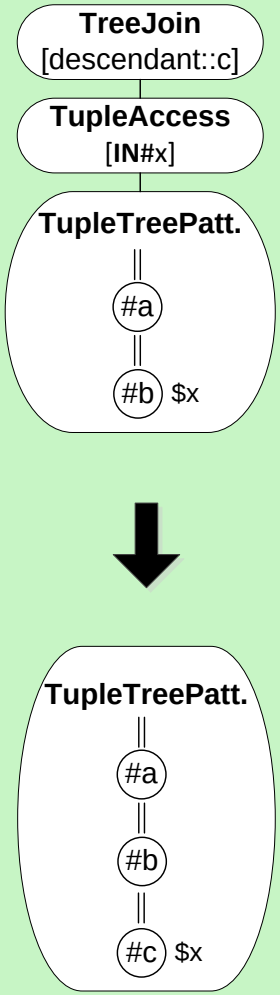
S nalezením TPQ



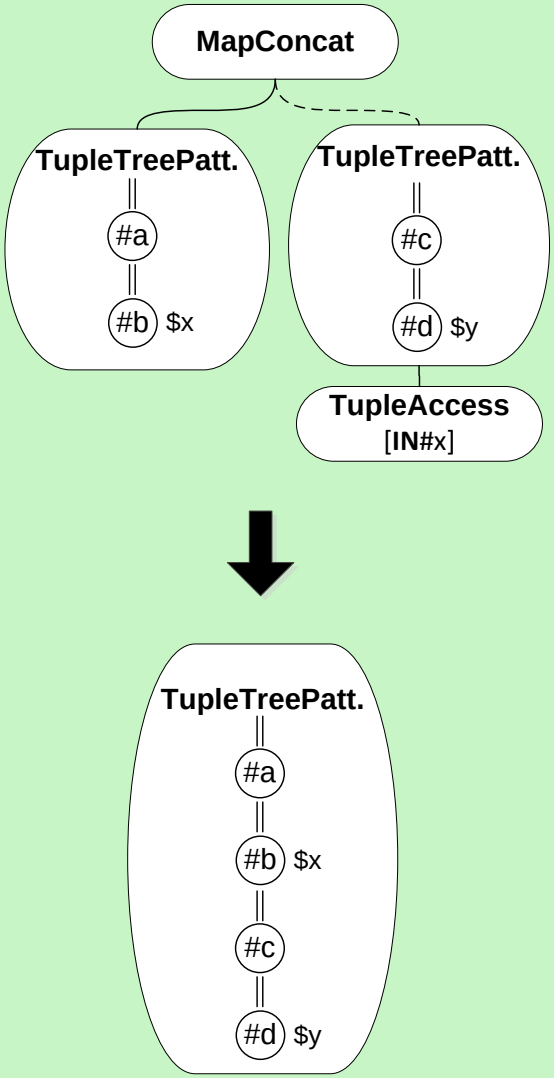
RemoveDescendantNodeTest, RemoveDescendantNodeTest, RemoveDescendantNodeTest, RemoveUnusedMappings, SelectThroughMapConcat, ***FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform,*** *RemoveUnusedMappings,* ***FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform, FindTreePatternTransform,*** *RemoveUnusedMappings,* ***FindTreePatternTransform***

Základní principy TPQ přepisů

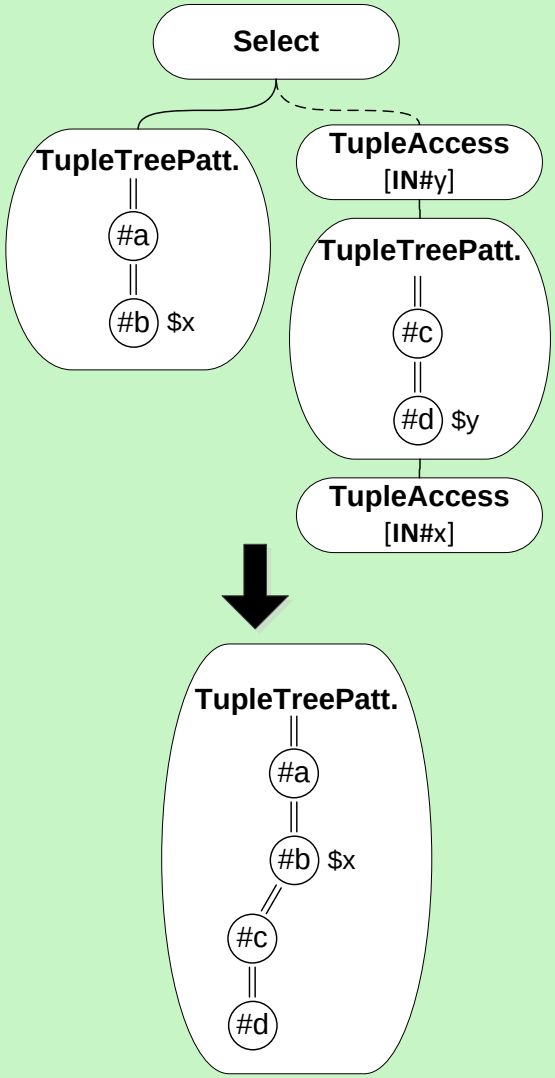
Spojení TreeJoins



Nahrazení MapConcat



Nahrazení selekce



Experimentální srovnání

Experimenty

- 5 porovnávaných procesorů
- 2 datové kolekce
- Pro každou kolekci 10 dotazů, rozdělení dotazů podle účelu do dvou skupin
  - **struktura** – neprobíhá dotazování textového obsahu elementů
  - **obsah** – probíhá dotazování textového obsahu elementů
  - Dotazy lze najít v technical report [6]
- Každý dotaz byl spuštěn 5x, časy uvedené dále představují průměry bez nejhorší a nejlepší naměřené hodnoty
- Testováno na **dbsys**.

Porovnávané procesory

- **Oracle database 11g** – bez indexu, s XML indexem
- **MS SQL Server 2012** – bez indexu, primární XML index, sekundární XML index
- **Saxon 9.4** (paměťový XQuery procesor)
- **Monet DB4** (nativní XML databáze)
- **BaseX 7.6** (nativní XML databáze)
- **QuickXDB (náš prototyp)** – bez indexu (původní paměťově orientovaný procesor), s indexem

Kolekce

1. **XMark**

- Kolekce vytvořená generátorem XMark s faktorem f = 10.
- Velikost **1,2 GB**
- Smyšlená databáze aukcí, obsahuje běžnou samopopisnou strukturu XML souboru a čitelný obsah

2. **TreeBank**

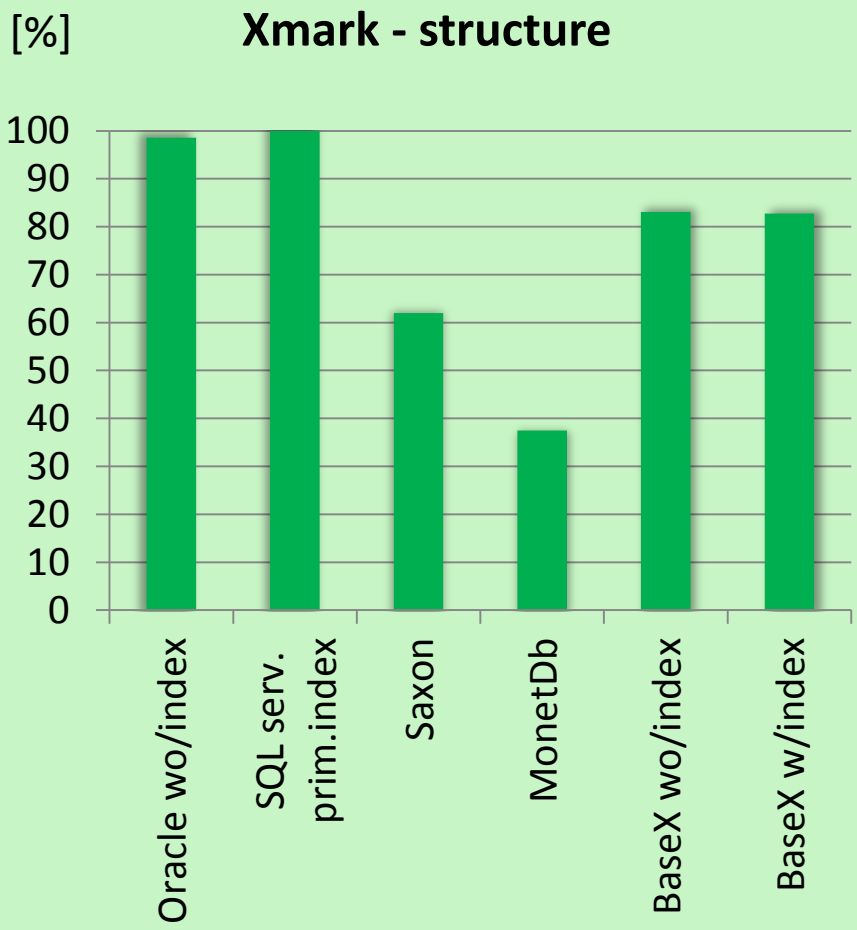
- Označkované anglické věty v zašifrované podobě. Vyznačuje se rekurzivní strukturou. Textový obsah nemá bez dešifrování smysluplný význam.
- Velikost **84 MB**

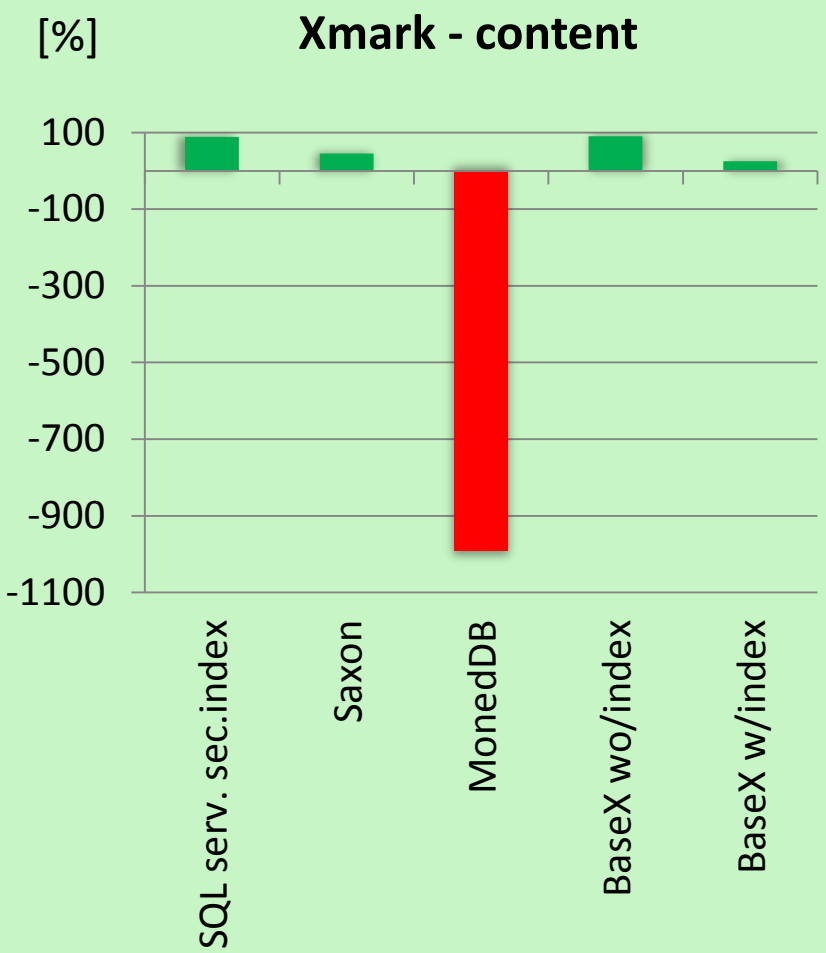
Výsledky – časy

		Oracle	Oracle	SQL Server	SQL Server	SQL Server	Saxon	MonetDB	BaseX	Base X	Quick XDB	Quick XDB
		wo/index	w/index	wo/index	prim.idx	sec.idx			wo/index	w/index	wo/index	w/index
XM - structure	XM1	7,510	DNF	DNF	201,576	DNF	0,374	0,171	1,129	1,023	-	**0,066**
	XM2	DNF	DNF	DNF	DNF	DNF	MEM	E	DNF	DNF	-	**0,076**
	XM3	DNF	DNF	DNF	DNF	DNF	MEM	E	DNF	DNF	-	**E**
	XM4	29,272	DNF	DNF	DNF	DNF	0,837	0,520	1,468	1,444	-	**0,470**
	XM5	24,067	DNF	DNF	DNF	DNF	0,853	0,676	1,843	1,945	-	**0,477**
XM - content	XM6	DNF	DNF	DNF	DNF	DNF	DNF	0,394	DNF	18,877	-	**10,051**
	XM7	E	E	E	E	E	2,694	E	DNF	DNF	-	**8,860**
	XM8	DNF	DNF	DNF	DNF	51,600	52,942	0,226	252,459	3,141	-	**6,271**
	XM9	E	E	DNF	DNF	DNF	9,048	2,079	9,044	9,409	-	**3,829**
	XM10	DNF	DNF	E	E	E	DNF	0,184	DNF	DNF	-	**E**
TB - structure	TB1	8,345	DNF	DNF	DNF	DNF	0,385	0,317	1,799	1,808	**0,930**	**0,043**
	TB2	8,233	DNF	DNF	DNF	DNF	0,494	0,313	2,543	2,655	**0,504**	**0,028**
	TB3	84,590	DNF	247,662	116,554	358,584	0,281	0,335	11,083	11,246	**1,103**	**0,045**
	TB4	11,561	DNF	DNF	DNF	148,258	0,499	0,448	2,448	2,239	**0,695**	**0,079**
	TB5	2,200	DNF	7,067	2,101	407,059	0,759	0,234	1,882	1,724	**0,660**	**0,380**
	TB6	19,815	DNF	E	E	E	0,250	0,330	7,385	7,242	**0,493**	**0,427**
	TB7	10,947	DNF	E	E	E	1,492	0,383	8,464	8,308	**0,846**	**0,740**
	TB8	DNF	E	E	E	E	0,374	1,336	3,058	3,228	**0,947**	**DNF**
TB - content	TB9	E	DNF	E	E	E	1,378	0,358	4,897	5,104	**0,480**	**1,746**
	TB10	1,190	DNF	4,233	2,303	2,189	0,260	0,332	1,236	1,264	**0,725**	**0,520**

- Časy uváděny v sekundách
- **DNF** – dotaz nedoběhl do 10-ti minut
- **E** – během vykonávání nastala chyba
- **MEM** – manuální ukončení z důvodu alokace více než 50 GB paměti

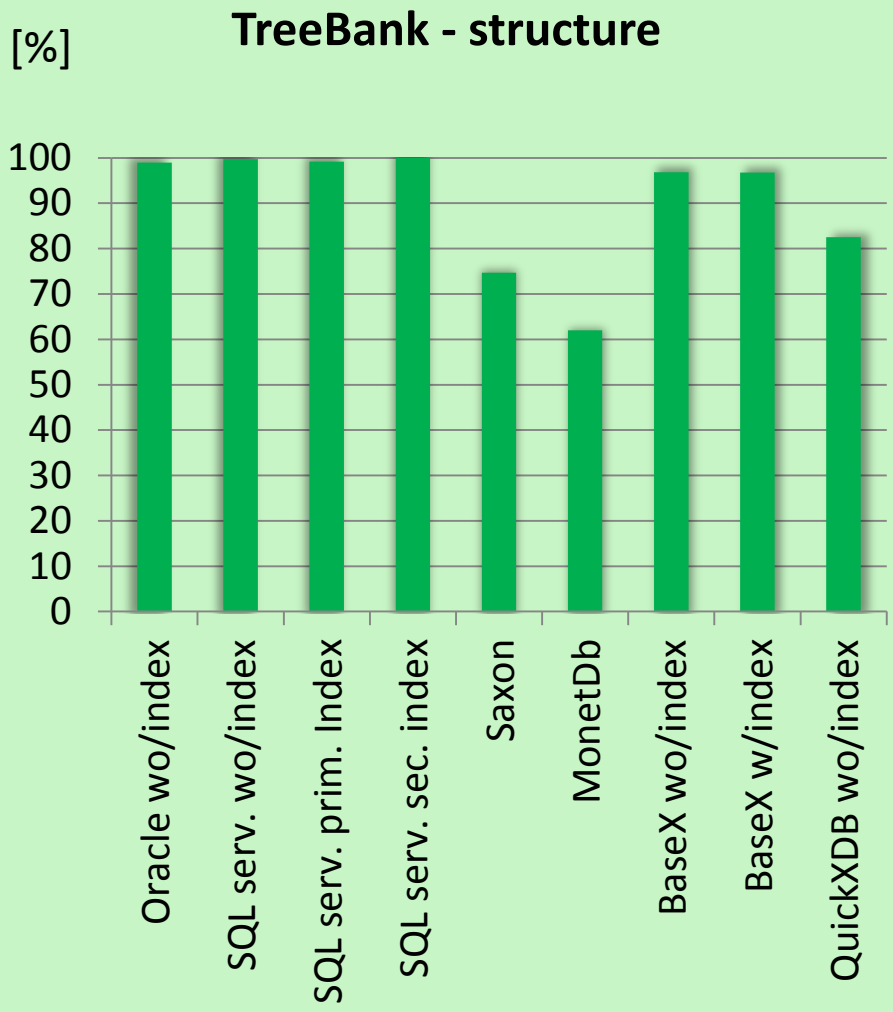
Relativní srovnání – XMark

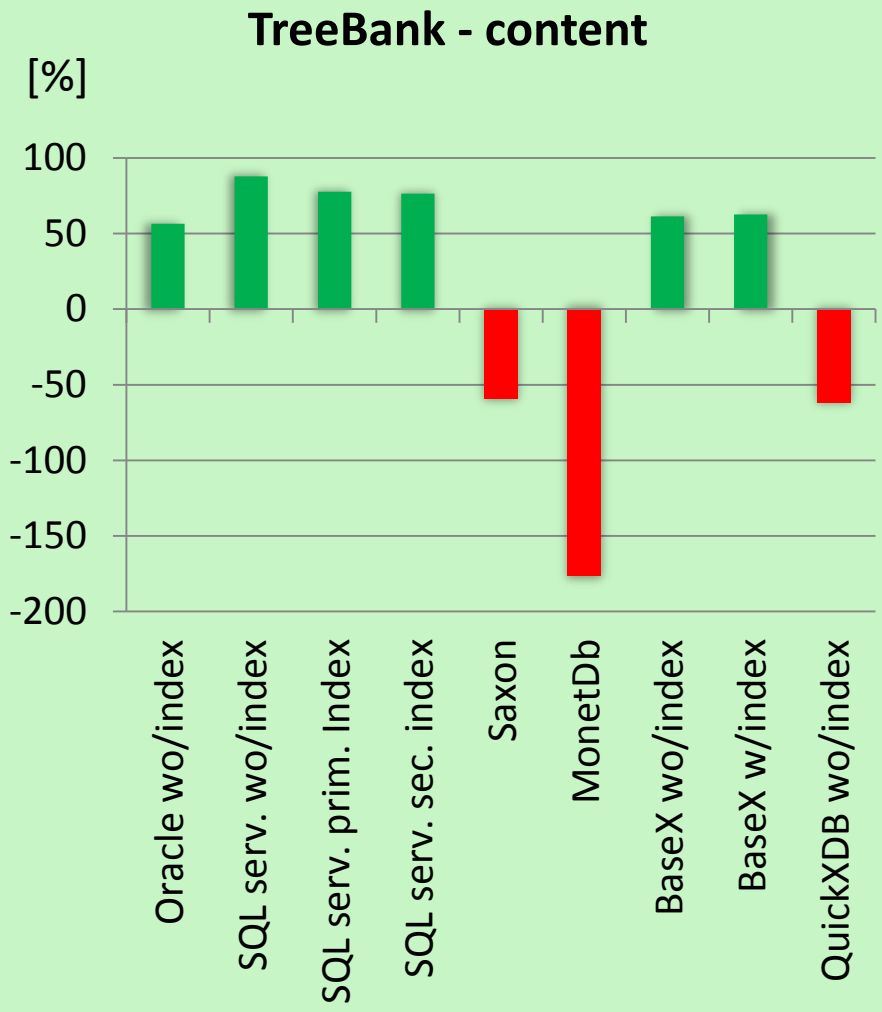




Hodnoty představují procentuální zrychlení. Např. 90% znamená, že náš procesor vykonával stejný typ dotazu 10% času porovnávaného procesoru.

Relativní srovnání – TreeBank





Směřování další práce

- **Využívání společného poolu instancí**
  Framework i algebra momentálně používají své oddělené systémy pro znovupoužívání alokované paměti.

- **Hlubší integrace holistických algoritmů do algebry**
  Momentálně zbytečně dublující se třídy s podobným významem.

- **Implementace obsahového indexu**
  Mělo by výrazně urychlit zpracování právě problematických dotazů na obsah XML elementů.

- **Zavedení Cost-Base optimalizací**
  V současné době přepisy fungují na jednoduchých pravidlech a neuvažují statistiky XML dokumentu. V určitých případech ale nemusí použití přepisu vést k urychlení vykonání dotazu.

Směřování další práce

- **Rozšíření databáze o XQuery Update facility**
  V současné době je možné z databáze pomocí XQuery pouze číst, tzn. neexistuje podpora DML operací.

- **Rozšíření jazyka pro správu QuickXDB**
  Aktuálně je např. přidávání kolekcí, indexování, nastavování vlastností řízeno kódem. Zavedením rozšiřujících příkazů by bylo možné dále naimplementovat např. aplikaci pro správu s GUI.

- **Vytvoření odladěné prezentovatelné verze**
  Aktuálně nedořešené implementační záležitosti, jsou případy, kdy výsledky nejsou zcela korektní např. vzhledem k uspořádání výsledku podle pořadí v dokumentu

Děkuji za pozornost

Reference

[1] R. Bača, M. Krátký, T. Link, and J. Lu. Optimal and efficient generalized twig patternt processing: a combination of preorder and postorder filterings. *The VLDB Journal*

[2] C. Re, J. Siméon, and M. Fernandez. A complete and efficient algegraic compiler for Xquery. In *Proceedings of the 22nd International Conference on Data Engineering,* 2006, *ICDE'06.*

[3] P. Michiels, G. Michaila, and J. Siméon. Put a Tree Pattern in Your Algebra. In *Proceedings of the 23nd International Conference on Data Engineering,* 2007, *ICDE'07.*

[4] W3 Consortium, XQuery 3.0: An XML XQuery Language, W3C Candidate Recommendation 8 January 2013, http://www.w3.org/TR/2013/CR-xquery-30-20130108/

[5] SAXON, The XSLT and XQuery Processor, http://saxon.sourceforge.net/

[6] P. Lukáš, R. Bača, and M. Krátký. QuickXDB: A Prototype of a Native XML DBMS. *Technical report No. CS/DBRG/2013-001*, 2013, http://db.cs.vsb.cz/TechnicalReports/CS-DBRG-2013-001.pdf