# Επεξεργασία Ερωτήσεων



## Εισαγωγή

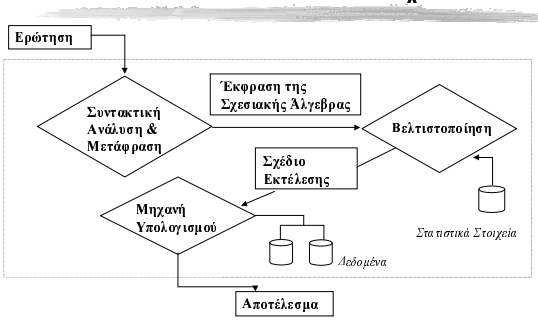
1. ΜΟΝΤΕΛΑ ΔΕΔΟΜΕΝΩΝ
   - Μοντέλα
   - Γλώσσες Ερωτήσεων
   - Επεξεργασία Ερωτήσεων
     - Επεξεργασία Ερωτήσεων σε Σχεσιακά ΣΔΒΔ
     - Επεξεργασία Ερωτήσεων σε Κατανεμημένα Σχεσιακά ΣΔΒΔ
     - → **Επεξεργασία Ερωτήσεων σε Ημιδομημένα Δεδομένα**
2. ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ
3. ΤΡΟΠΟΙ ΜΕΤΑΔΟΣΗΣ



# Ανακεφαλαίωση



## Επεξεργασία Ερωτήσεων σε Σχεσιακά ΣΔΒΔ






## Επεξεργασία Ερωτήσεων

Τα βασικά βήματα στην επεξεργασία μιας ερώτησης είναι

1. Συντακτική Ανάλυση & Μετάφραση
2. Βελτιστοποίηση
3. Υπολογισμός



## Επεξεργασία Ερωτήσεων

**1. Συντακτική Ανάλυση (Parsing) & Μετάφραση**

Η ερώτηση μεταφράζεται σε μια εσωτερική μορφή αφού γίνει ο απαραίτητος συντακτικός και σημασιολογικός έλεγχος (π.χ., τα ονόματα που αναφέρονται είναι ονόματα σχέσεων που υπάρχουν)

Αντικατάσταση των όψεων από τον ορισμό τους

Εσωτερική μορφή; Έκφραση της σχεσιακής άλγεβρας

**select** $A_1, A_2, \ldots, A_n$
**from** $R_1, R_2, \ldots, R_m$
**where** P

$\pi_{A1, A2, \ldots, An}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$

## 2. Βελτιστοποίηση

Μια SQL ερώτηση μπορεί να μεταφραστεί σε διαφορετικές (ισοδύναμες) εκφράσεις της σχεσιακής άλγεβρας

**select** balance
**from** account
**where** balance < 25000

- $\sigma_{balance < 2500}(\pi_{balance}(account))$
- $\pi_{balance}(\sigma_{balance < 2500}(account))$





## 2. Βελτιστοποίηση

Άρα δεν αρκεί ο προσδιορισμός της πράξης - πρέπει να προσδιορίζεται και ο αλγόριθμος που θα χρησιμοποιηθεί για την υλοποίηση της

π.χ., για την υλοποίηση της επιλογής μπορεί

είτε να σαρώσουμε (scan) όλο το αρχείο ελέγχοντας κάθε εγγραφή

είτε αν υπάρχει π.χ., ένα $B^+$ ευρετήριο στο γνώρισμα balance να χρησιμοποιήσουμε το ευρετήριο

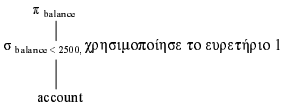




## 2. Βελτιστοποίηση

Κάθε πράξη της σχεσιακής άλγεβρας μπορεί να υλοποιηθεί με διαφορετικούς αλγορίθμους: βασικές (primitive) πράξεις (πράξη + αλγόριθμος)

**Σχέδιο εκτέλεσης** (execution plan): μια ακολουθία από βασικές πράξεις








## 2. Βελτιστοποίηση

- Τα διαφορετικά σχέδια εκτέλεσης έχουν και διαφορικό κόστος
- Βελτιστοποίηση: η διαδικασία επιλογής του σχεδίου εκτέλεσης που έχει το μικρότερο κόστος
- Εκτίμηση του κόστους (συνήθως χρήση στατιστικών στοιχείων)





## 3. Εκτέλεση

Μηχανή εκτέλεσης που εκτελεί τις βασικές πράξεις





**Δέντρο ερώτησης**

Φύλλα: σχέσεις εισόδου

Εσωτερικοί κόμβοι: πράξεις της σχεσιακής άλγεβρας

Εκτέλεση δέντρου ερώτησης

## Κατανεμημένα ΣΔΒΔ

- Data is stored *across several sites*
- Each site is (typically) managed by a DBMS that can run independently of the other sites



## Κατανεμημένα ΣΔΒΔ

**Transparency**

- **Distributed data independence**

  ask queries *without specifying where* the referenced relations, or copies of fragments of the relations are located

  queries that span multiple sites should be *optimized* systematically in a cost-based manner (taking into account communications costs and differences in local computation costs)

- **Distributed transaction processing**

  atomicity

*Not always efficiently achievable*



## Κατανεμημένα ΣΔΒΔ

**Είδη**

- **Ομοιογένεια**

- **Ετερογένεια**

  **gateway protocols**: API that exposes DBMS functionality to external applications (e.g., ODBC, JDBC)



## Κατανεμημένα ΣΔΒΔ

**Αρχιτεκτονικές**

- **Client/server**

clients (user-interface issues)

servers (manage data and execute transactions)

(client cache)



## Κατανεμημένα ΣΔΒΔ

**Αρχιτεκτονικές**

- **Collaborating server systems**

  *a single query spans multiple servers*

a collection of database servers, each capable of running transactions against local data which *cooperatively* execute transactions spanning multiple servers



## Κατανεμημένα ΣΔΒΔ

**Αρχιτεκτονικές**

- **Middleware systems**

just one special database server (layer of software) that coordinates the execution of queries and transactions across one or more independent database servers

## Κατανεμημένα ΣΔΒΔ

- Τμηματοποίηση (partitioning) σχέσεων

**Global relations:**

EMP(ENO,ENAME,TITLE)

ASG(ENO,PNO,RESP,DUR)

PRJ(PNO,PNAME,BUDGET)

SAL(TITLE,SALARY)

**Vertical Partitioning**:

$PRJ_1$: $\pi_{PNO,PNAME}$(PRJ)

$PRJ_2$: $\pi_{PNO,BUDGET}$(PRJ)

**Horizontal Partitioning:**

$EMP_1$: $\sigma_{ENO<=E3}$(EMP)

$EMP_2$: $\sigma_{E3<ENO<=E6}$(EMP)

$EMP_1$: $\sigma_{ENO>E6}$(EMP)

$ASG_1$: $\sigma_{ENO<=E3}$(ASG)

$ASG_2$: $\sigma_{ENO>E3}$(ASG)



## Query Optimization Objectives

Minimize a cost function

I/O cost + CPU cost + communication cost

These might have different weights in different distributed environments

Wide area networks

- communication cost will dominate
  - low bandwidth
  - low speed
  - high protocol overhead
- most algorithms ignore all other cost components

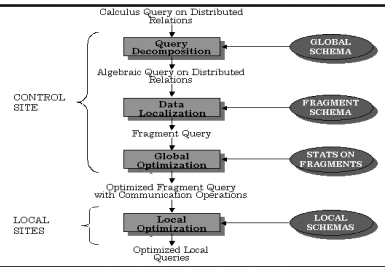
Local area networks

- communication cost not that dominant
- total cost function should be considered

Can also maximize throughput



## Distributed Query Processing Methodology






## Στόχοι Βελτιστοποίησης Ερωτήσεων σε Κατανεμημένα ΣΔΒΔ

- types of optimization (deterministic, randomized)
- optimization timing (dynamic, static)
- statistics
- decision sites
- exploitation of network topology
- exploitation of replicated fragments
- use of semijoins



## Step 2 – Data Localization

Input: Algebraic query on distributed relations

- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
  - optimize

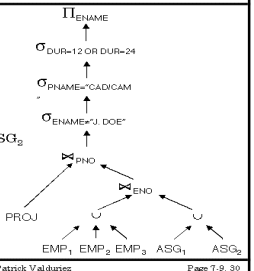


## Example

Assume

- EMP is fragmented into $EMP_1$, $EMP_2$, $EMP_3$ as follows:
  - $EMP_1=\sigma_{ENO\leq"E3"}(EMP)$
  - $EMP_2=\sigma_{"E3"<ENO\leq"E6"}(EMP)$
  - $EMP_3=\sigma_{ENO>"E6"}(EMP)$
- ASG fragmented into $ASG_1$ and $ASG_2$ as follows:
  - $ASG_1=\sigma_{ENO\leq"E3"}(ASG)$
  - $ASG_2=\sigma_{ENO>"E3"}(ASG)$

Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query



## Step 3 – Global Query Optimization

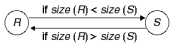
Input: Fragment query

- Find the *best* (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs ship-as-needed
  - Decide on the use of semijoins
    - Semijoin saves on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)



## Join Ordering

- Consider two relations only




- Multiple relations more difficult because too many alternatives.
  - Compute the cost of all alternatives and select the best one.
    - Necessary to compute the size of intermediate relations which is difficult.
  - Use heuristics



*Semijoin of R with S is the subset of tuples of R that participate in the join of R with S*

## Semijoin Algorithms

- Consider the join of two relations:
  - $R[A]$ (located at site 1)
  - $S[A]$ (located at site 2)
- Alternatives:
  1. Do the join $R \bowtie_A S$
  2. Perform one of the semijoin equivalents

$$\begin{aligned} R \bowtie_A S &\Leftrightarrow (R \ltimes_A S) \bowtie_A S \\ &\Leftrightarrow R \bowtie_A (S \ltimes_A R) \\ &\Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R) \end{aligned}$$



## Semijoin Algorithms

- Perform the join
  - send $R$ to Site 2
  - Site 2 computes $R \bowtie_A S$
- Consider semijoin $(R \ltimes_A S) \bowtie_A S$
  - $S' \leftarrow \Pi_A(S)$
  - $S' \rightarrow$ Site 1
  - Site 1 computes $R' = R \ltimes_A S'$
  - $R' \rightarrow$ Site 2
  - Site 2 computes $R' \bowtie_A S$

Semijoin is better if

$$size(\Pi_A(S)) + size(R \ltimes_A S)) < size(R)$$



## Step 4 – Local Optimization

Input: Best global execution schedule

- Select the best access path
- Use the centralized optimization techniques



# Επεξεργασία Ερωτήσεων για Ημι-δομημένα Δεδομένα

## Επεξεργασία Ερωτήσεων

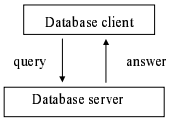
Τα ίδια βασικά στάδια

1. Μετάφραση - ένα σχέδιο εκτέλεσης
2. Βελτιστοποίηση
3. Μηχανή Εκτέλεσης

- Κατανομή/Αρχιτεκτονικές
- Έλλειψη σχήματος



## Αρχιτεκτονικές




Συνήθως από τον ίδιο vendor



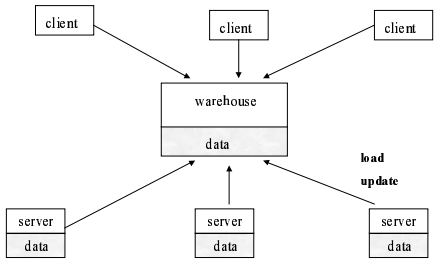
## Αρχιτεκτονικές

**Middleware systems**

just one special database server (layer of software) that coordinates the execution of queries and transactions across one or more independent database servers

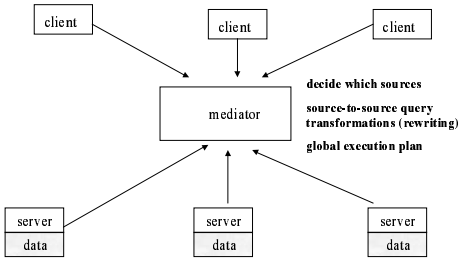


## Αρχιτεκτονικές






## Αρχιτεκτονικές






## Αρχιτεκτονικές

- Multitier system

## Αποθήκευση

- Storage back-end
- Type information



## Αποθήκευση

Τρόποι Αποθήκευσης

1. Κείμενο
2. Σχεσιακή Βάση Δεδομένων
3. Αντικειμενο-στραφή Βάση Δεδομένων
4. Αυτο-οργάνωση - Υβριδική αποθήκευση



## Αποθήκευση

**Text File**

```
<family id = "o1">
  <person id = "p1"> <name Joan </name>
      <age> 36 </age>
      <profession> database administrator </profession>
      <hobby> gardening </hobby>
      <cellular> 555-6234 </cellular>
  </person>
<person id = "p2"> <name John </name>
      <age> 38 </age>
      <profession> systems administrator </profession>
      <beeper> 555-3322 </beeper>
  </person>
```



## Αποθήκευση

**Text File**

- subobjects are naturally grouped together
- fixed granularity (an XML document)

  ok for parts of an XML document



## Αποθήκευση

**Σχεσιακή Βάση Δεδομένων**

- Σχεσιακά ΣΔΒΔ παρέχουν χρήσιμα χαρακτηριστικά (concurrency control, indexes, etc)
- Δύσκολο



## Αποθήκευση

**Σχεσιακή Βάση Δεδομένων**

Δύο σχέσεις ref(src, label, dst) Val(oid, value)

```
<family id = "o1">
  <person id = "p1"> <name Joan </name>
      <age> 36 </age>
      <profession> database administrator
</profession>
      <hobby> gardening </hobby>
      <cellular> 555-6234 </cellular>
  </person>
<person id = "p2"> <name John </name>
      <age> 38 </age>
      <profession> systems administrator </profession>
      <beeper> 555-3322 </beeper>
  </person>
```

| src | label | dst |
|---|---|---|
| "root" | "family" | "o1" |
| "o1" | "person" | "p1" |
| "o1" | "person" | "p2" |
| "p1" | "name" | "p11" |
| "p1" | "age" | "p12" |
| "p1" | "profession" | "p13" |
| "p1" | "hobby" | "p14" |
| "p1" | "cellular" | "p15" |
| "p2" | "name" | "p21" |
| "p2" | "age" | "p22" |
| "p2" | "profession" | "p23" |
| "p2" | "beeper" | "p24" |

| oid | value |
|---|---|
| "p11" | "Joan" |
| p11" | 36 |
| "p13" | "database admin" |
| "p14" | "gardening" |
| "p15" | "555-6234" |
| "p21" | "John" |
| "p22" | 38 |
| "o23" | "systems admin" |
| p24" | "555-3322" |

## Σχεσιακή Βάση Δεδομένων

```
<family id = "o1">
  <person id = "p1"> <name Joan </name>
       <age> 36 </age>
       <profession> database administrator </profession>
       <hobby> gardening </hobby>
       <cellular> 555-6234 </cellular>
  </person>
<person id = "p2"> <name John </name>
       <age> 38 </age>
       <profession> systems administrator </profession>
       <beeper> 555-3322 </beeper>
  </person>
```

```
select X
from family.person.hobby X
```





## Σχεσιακή Βάση Δεδομένων

Refs

| src | label | dst |
|---|---|---|
| "root" | "family" | "o1" |
| "o1" | "person" | "p1" |
| "o1" | "person" | "p2" |
| "p1" | "name" | "p11" |
| "p1" | "age" | "p12" |
| "p1" | "profession" | "p13" |
| "p1" | "hobby" | "p14" |
| "p1" | "cellular" | "p15" |
| "p2" | "name" | "p21" |
| "p2" | "age" | "p22" |
| "p2" | "profession" | "p23" |
| "p2" | "beeper" | "p24" |

Value

| oid | value |
|---|---|
| "p11" | "Joan" |
| "p11" | 36 |
| "p13" | "database admin" |
| "p14" | "gardening" |
| "p15" | "555-6234" |
| "p21" | "John" |
| "p22" | 38 |
| "o23" | "systems admin" |
| "p24" | "555-3322" |

```
select X
from family.person.hobby X

select v.value
from Refs r1, Refs r2, Refs r3, Value v
where  r1.src ="root" and
       r1.label = "family and
       r1.dst = r2.src and
       r2.label= person and
       r2.dst =r3.src and
       r3.label = hobby and
       r3.dst = v.oid
```

**4-way join! index?**





## Σχεσιακή Βάση Δεδομένων

- μεγάλο αριθμό Join
- οργάνωση στο δίσκο (clustering, indexing)



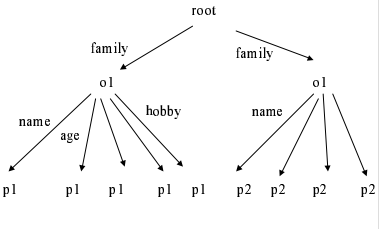


## Σχεσιακή Βάση Δεδομένων

Refs

| src | label | dst |
|---|---|---|
| "root" | "family" | "o1" |
| "o1" | "person" | "p1" |
| "o1" | "person" | "p2" |
| "p1" | "name" | "p11" |
| "p1" | "age" | "p12" |
| "p1" | "profession" | "p13" |
| "p1" | "hobby" | "p14" |
| "p1" | "cellular" | "p15" |
| "p2" | "name" | "p21" |
| "p2" | "age" | "p22" |
| "p2" | "profession" | "p23" |
| "p2" | "beeper" | "p24" |

Breadth-first search!





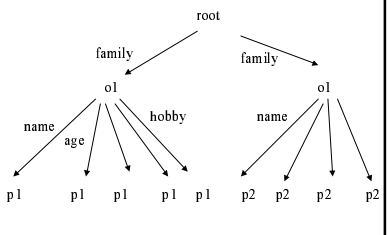



## Σχεσιακή Βάση Δεδομένων

Refs

| src | label | dst |
|---|---|---|
| "root" | "family" | "o1" |
| "o1" | "person" | "p1" |
| "p1" | "name" | "p11" |
| "p1" | "name" | "p11" |
| "p1" | "age" | "p12" |
| "p1" | "profession" | "p13" |
| "p1" | "hobby" | "p14" |
| "p1" | "cellular" | "p15" |
| "o1" | "person" | "p2" |
| "p2" | "name" | "p21" |
| "p2" | "age" | "p22" |
| "p2" | "profession" | "p23" |
| "p2" | "beeper" | "p24" |

Depth-first search!








## Σχεσιακή Βάση Δεδομένων

**Παρατήρηση**

incorrect: values of different types are mixed in the value column of Val

*Ένα διαφορετικό πίνακα για κάθε διαφορετική ατομική τιμή (ένα ακόμα join!!!)*

**Βελτιστοποίηση**

Inline the values of the Ref relation

## Αποθήκευση

**Σχεσιακή Βάση Δεδομένων**

| src | label | dst | flag | ValString | ValInt | ValData |
|---|---|---|---|---|---|---|
| 'root' | 'family' | 'o1' | 0 | null | null | null |
| 'o1' | 'person' | 'p1' | 0 | null | null | null |
| 'o1' | 'person' | 'p2' | 0 | null | null | null |
| 'p1' | 'name' | 'p11' | 1 | 'Joan' | null | null |
| 'p1' | 'age' | 'p12' | 2 | null | 36 | null |
| 'p1' | 'profession' | 'p13' | 1 | 'dba' | null | null |
| 'p1' | 'hobby' | 'p14' | 1 | 'gardening' | null | null |
| 'p1' | 'cellular' | 'p15' | 1 | '555-6234' | null | null |
| 'p2' | 'name' | 'p21' | 1 | 'John' | null | null |
| 'p2' | 'age' | 'p22' | 2 | null | 38 | null |
| 'p2' | 'profession' | 'p23' | 1 | 'sa' | null | null |
| 'p2' | 'beeper' | 'p24' | 1 | '555-3322' | null | null |



## Αποθήκευση

**Αντικειμενο-στραφή Βάση Δεδομένων**

**DTDs → ιεραρχία κλάσεων**

```
<!ELEMENT company (employee)*>
<!ELEMENT employee (name, address, project*, (phone|email))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (city, country)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
!ELEMENT project (name, deadline)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT deadline (#PCDATA)>
```

στοιχείο → κλάση
PCDATA → strings
regular expressions?



## Αποθήκευση

**Αντικειμενο-στραφή Βάση Δεδομένων**

**DTDs → ιεραρχία κλάσεων**

```
class Company  public type set(Employee)
class Employee public type tuple  (name: string,
                                   address: Address,
                                   projects: list(Project),
                                   pe: PE)
class Address public type tuple  (city: string, country:string)
class Project public type tuple (name:string, deadline:string),
class PE public type union (phone:int, email:string),
```

order



## Αποθήκευση

**Αντικειμενο-στραφή Βάση Δεδομένων**

- η ακριβής μετάφραση εξαρτάται από τον τύπο του αντικειμενο-στραφούς μοντέλου
- αλλαγή τύπων
- δεδομένα για τα οποία δε γνωρίζουμε τον τύπο τους
- αποθήκευση μικρών XML document ως text



## Αποθήκευση

**Αυτο-οργάνωση**

- στατιστικά στοιχεία (πιο συχνές ερωτήσεις, είδος αποθηκευμένων δεδομένων)

**Υβριδική αποθήκευση**

- διαχωρισμός της αποθήκευσης σε structured και unstructured



## Ευρετήρια

**Ευρετήρια για εκφράσεις μονοπατιών**

Σχήμα 8.8 (σελίδα 181)

Ερωτήσεις:

(R1) part.name

(R2) part.supplier.name

(R3) _*.supplier.name

(R4) part._*.subpart.name

**Ευρετήρια για εκφράσεις μονοπατιών**

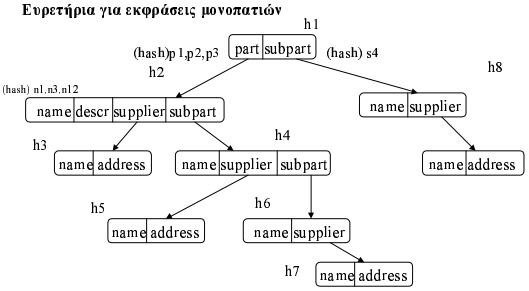
**Ευρετήριο**

- δέντρο
- ρίζα: όλα τα labels που εμφανίζονται στον root data node
- ένα κόμβο για κάθε ακολουθία labels που οδηγεί σε ένα εσωτερικό κόμβο του data tree
- κάθε κόμβος του δέντρου: hash table - κάθε θέση στο hash table δείχνει στους αντίστοιχους κόμβους του data tree





**Ευρετήρια για εκφράσεις μονοπατιών**








**Ευρετήρια για εκφράσεις μονοπατιών**

Ερωτήσεις:

(R1) part.name

*h1 → h2 → use hash at name*

(R2) part.supplier.name

*h1 → h2 → h3 → use hash at name*

(R3) _*.supplier.name

??

(R4) part._*.subpart.name

*search all nodes in the subtree rooted at h2*





**Ευρετήρια για εκφράσεις μονοπατιών**

κόμβοι που ακριβώς τα ίδια μονοπάτια από τη ρίζα (π.χ., κόμβοι n1, n3 και n12 - n2, n13 και n4 στο 8.8 --- p1 και p2 στο 8.11)

p1 and p2 **language-equivalent**: for any path expression query, either both p1 and p2 are in the answer or none is





**Ευρετήρια για εκφράσεις μονοπατιών**

Για κάθε κόμβο x

$L_x \equiv \{w \mid \exists$ path from the root to x labeled w$\}$

$L_x$

- μπορεί να είναι άπειρο αν ο γράφος έχει κύκλους
- γενικά περιγράφεται από μια κανονική έκφραση

x and y **language-equivalent**: $(x \equiv y)$ if $L_x = L_y$





**Ευρετήρια για εκφράσεις μονοπατιών**

Έστω [x] η κλάση ισοδυναμίας για το x

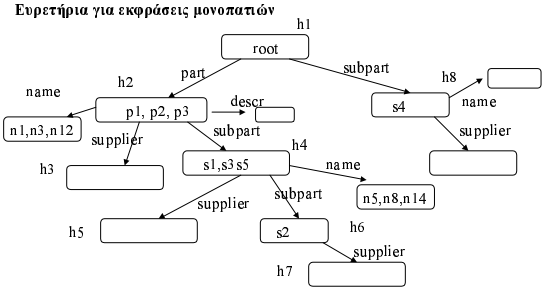
Κατασκευή του ευρετηρίου

ένα κόμβο για κάθε κλάση ισοδυναμίας

υπάρχει ακμή από το [x] στο [y] με label a αν υπάρχουν κόμβοι x' στο [x] και y' στο [y] που συνδέονται με κάποια ακμή με label a

## Ευρετήρια

**Ευρετήρια για εκφράσεις μονοπατιών**






## Ευρετήρια

**Ευρετήρια για εκφράσεις μονοπατιών**

Πως θα υπολογίσουμε αν $x \equiv y$ ;

Reverse graph

ισχύει

$\forall x,y,\ x \approx y \Rightarrow x \equiv y$



## Ευρετήρια

**Bisimulation:** δυαδική σχέση μεταξύ των κόμβων δύο γράφων t1 και t2 (έστω x, x' κόμβοι του t1 και y, y' κόμβοι του t2)

**1.** Αν x και y είναι ρίζες του t1 και t2 τότε x ~ y.

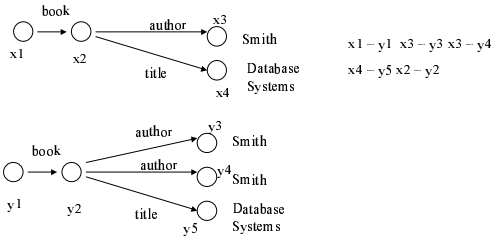
**2.** Αν x ~ y και ένας εκ των x και y είναι ρίζα, τότε και ο άλλος κόμβος είναι ρίζα

**3.** Αν x ~ y και (x, l, x') στο t1 τότε υπάρχει ακμή (y, l, y') στο t2 με την ίδια ετικέτα και x' ~ y'Αντίστοιχα, αν x ~ y και (y, l, y') στο t2 τότε υπάρχει ακμή (x, l, x') στο t1 με την ίδια ετικέτα και x'~ y'

**4.** Αν x ~ y και το x είναι φύλλο με τιμή v στο t1, τότε και το y είναι φύλλο με τιμή v στο t2. Αντίστοιχα, αν x ~ y και το y είναι φύλλο με τιμή v στο t2, τότε και το x είναι φύλλο με τιμή v στο t1.



## Ευρετήρια






## Ευρετήρια

**Bisimulation:** $x \approx y$

**1.** Αν $x \approx y$, x είναι ρίζα, τότε y είναι ρίζα, αντίστοιχα αν y είναι ρίζα, τότε το x είναι ρίζα

**2.** Αν $x \approx y$ και υπάρχει ακμή $x' \rightarrow^a x$, τότε υπάρχει ακμή $y' \rightarrow^a y$ και $x' \approx y'$, αντίστοιχα αν υπάρχει ακμή $y' \rightarrow^a y$, τότε υπάρχει ακμή $x' \rightarrow^a x$ και $x' \approx y'$



## Ευρετήρια

**Ευρετήρια για XML text**

- **region**: a (contiguous) segment of text in the file
- **region set**: set of regions such that any two regions are either disjoint or one is included in the other

*Στη δεντρική αναπαράσταση*:

κάθε κόμβος ορίζει μια region (π.χ., ο κόμβος p2 αντιστοιχεί στο κείμενο κάτω από τον κόμβο p2)

region set: σύνολο κόμβων

**Ευρετήρια για XML text**

*Ιδιαίτερο ενδιαφέρον - region sets που αντιστοιχούν σε XML tags*

$\pi_\chi$ , part ορίζει το region set {p1, p2, p3}

subpart ορίζει το region set {s1,s2,s3,s4,s5}





**Ευρετήρια για XML text**

**Region**

ζεύγος (x, y) x: start and end position of the region in the text file

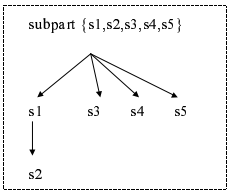
**Region Set**

ordered tree: κάθε κόμβος μια region

- r = (x, y) *ancestor* of r'(x', y') ( r' ⊆ r)

  if x ≤ x' ≤ y' ≤ y
- r to the *left* of r'

  if x ≤ y ≤ x' ≤ y'








**Ευρετήρια για XML text**

**Region Algebra (άλγεβρα για περιοχές)**

- πάνω σε σύνολα περιοχών
- τελεστές που δίνουν ως αποτέλεσμα σύνολα περιοχών: s1 op s2

*νέα σύνολα περιοχών -- όχι νέες περιοχές, αυτές θεωρούνται προκαθορισμένες (αντιστοιχούν στους κόμβους)*





**Ευρετήρια για XML text**

**Παραδείγματα τελεστών μιας region algebra**

s1 **intersect** s2 ≡ {r | r ∈ s1, r ∈ s2}

s1 **included** s2 ≡ {r | r ∈ s1, ∃ r ' ∈ s2, r ⊆ r'}

s1 **including** s2 ≡ {r | r ∈ s1, ∃ r ' ∈ s2, r ⊇ r'}

s1 **parent** s2 ≡ {r | r ∈ s1, ∃ r ' ∈ s2, r is a parent of r'}

s1 **child** s2 ≡ {r | r ∈ s1, ∃ r ' ∈ s2, r is a child of r'}





**Ευρετήρια για XML text**

Παραδείγματα

- subpart {s1,s2,s3,s4,s5} part {p1, p2, p3}

  subpart included part {s1, s2, s3, s5}

  part including subpart {p2, p3}
- name {n1, n2, …, n12} part {p1, p2, p3}

  name child part {n1, n3, n12}

  name included part {n1, n2, .., n9, n12}





**Ευρετήρια για XML text**

**Υπολογισμός Τελεστών**

s1 op s2

traverse the tree representations of s1 and s2 simultaneously similar to a merge join

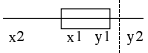
**Ευρετήρια για XML text**

s1 included s2

assuming that s1 and s2 are sets of disjoint regions

Αρχικά (x1, y1) το πρώτο στοιχείο του s1 και (x2, y2) το πρώτο στοιχείο του s2

Μέχρι το τέλος της λίστας s1 ή s2




Αν x1 < x2, advance s1

Αν y1 > y2, advance s2

Αλλιώς, πρόσθεσε το (x1, y1) στο αποτέλεσμα, advance s1





**Ευρετήρια για XML text**

Ερωτήσεις:

(R1) part.name

*name child (part child root)*

(R2) part.supplier.name

*name child (supplier child (part child root))*

(R3) _*.supplier.name

*name child supplier*

(R4) part._*.subpart.name

*name child (subpart included (part child root))*





**Ευρετήρια για XML text**

(R5)

```
select X
from _*.subpart: {name: X, _*.supplier.address: "Philadelphia"
```

*name child (subpart includes (supplier parent (address intersect "Philadelphia")))*





**Ευρετήρια για XML text**

• μόνο ένα περιορισμένο αριθμό από κανονικές εκφράσεις, συγκεκριμένα εκφράσεις

$R_1. R_2. \ldots R_n$

όπου $R_i$ label constant or the Kleene closure (_*)

• μόνο για ordered trees





**Ευρετήρια για κείμενο**

ένας σημαντικός τύπος ερώτησης: **keyword search**

• ο πιο συνηθισμένος τύπος ερώτησης στις μηχανές αναζήτης

•συχνά κρατείται και μια λίστα με συνώνυμα (π.χ, ερώτηση για car - επίσης, automobile)

•πιο περίπλοκες ερωτήσεις που περιλαμβάνουν and, or, not





**Ευρετήρια για κείμενο**

Δυο βασικοί τύποι ερωτήσεων:

- boolean
- ranked query

**Boolean** (conjuctive normal form)

$(t_{11} \vee t_{12} \vee \ldots t_{1i1}) \wedge \ldots \wedge (t_{j1} \vee t_{j2} \vee \ldots \vee t_{jij})$

όπου $t_{ij}$ είναι ανεξάρτητα query terms ή keywords

j conjucts (που αντιστοιχούν σε διαφορετικές έννοιες (concepts)) - καθεμία από πολλά disjuncts (που αντιστοιχούν σε διαφορετικούς όρους για την ίδια έννοια)

## Ευρετήρια

**Ranked query**

Αποτέλεσμα

σύνολο από documents που επιπρόσθετα είναι **ταξινομημένα με βάση τη σχετικότητα τους** (ranked by their relevance)

Information retrieval

Δύο κριτήρια

- **precision**: percentage of the retrieved documents that are relevant to the query
- **recall**: percentage of relevant documents in the database that are retrieved in response to a query



## Ευρετήρια

Ευρετήριο - ζεύγη <keyword, documentid>

με πιθανά επιπρόσθετα πεδία όπως πόσες φορές εμφανίζεται το keyword στο document

μια μηχανή αναζήτησης δημιουργεί ένα κεντρικό ευρετήριο για documents που είναι αποθηκευμένα σε διάφορα sites



## Ευρετήρια

**Inverted files**

Για κάθε όρο (term) μια ταξινομημένη λίστα (inverted list) από τα ids των documents που περιέχουν αυτόν το όρο

Επιπρόσθετα, όλοι οι πιθανοί όροι τοποθετούνται σε ένα δευτερεύον ευρετήριο (π χ, B+-δέντρο) -- ευρετήριο λεξιλογίου

Conjunction -- ξεκινώντας από τη συντομότερη λίστα



## Ευρετήρια

**Inverted files**

| Rid | Document | Signature |
|---|---|---|
| 1 | agent James Bond | 1100 |
| 2 | agent mobile computer | 1101 |
| 3 | James Madison movie | 1011 |
| 4 | James Bond movie | 1110 |

| Word | Inverted List | Hash |
|---|---|---|
| agent | <1,2> | 1000 |
| Bond | <1,4> | 0100 |
| computer | <2> | 0100 |
| James | <1,3,4> | 1000 |
| Madison | <3> | 0001 |
| mobile | <2> | 0001 |
| movie | <3,4> | 0010 |



## Ευρετήρια

**Signature files**

An index record (**signature**) for each document

Each signature has a fixed size of b bits - b is called the **signature width**

Which bits we set depends on the words that appear on the document

- apply a hash function at each word that appears in the document
- set the bits that appear in the result of the hash function

*the same bit may be set twice by different words*



## Ευρετήρια

**Signature files**

- Signature $S_1$ **matches** another signature $S_2$ if all the bits that are set in signature $S_2$ are also set in signature $S_1$
- If signature $S_1$ matches signature $S_2$, then signature $S_1$ has at least as many bits as signature $S_2$

**Signature files**

**Conjuction**

1. Generate the query signature by applying the hash function to each word in the query
2. Scan the signature file and retrieve all documents whose signatures match the query signature
3. Retrieve each potential match and check whether it actually contains the query terms

**false positive**





**Signature files**

**Disjuction**

1. Generate a list of query signature, one for each term in the query
2. Scan the signature file and retrieve all documents whose signatures match any signature in the list of the query signatures





**Signature files**

scan the *complete signature file*

Vertically partition the signature file into a set of bit slices - bit sliced signature file

For a query with b bits - retrieve only q bit slices





**Inverted files**

| Rid | Document | Signature |
|---|---|---|
| 1 | agent James Bond | 1100 |
| 2 | agent mobile computer | 1101 |
| 3 | James Madison movie | 1011 |
| 4 | James Bond movie | 1110 |

signature file width 4

Queries: “James”, “James” and “Bond”, “movie” and “Madison”

| Word | Inverted List | Hash |
|---|---|---|
| agent | <1,2> | 1000 |
| Bond | <1,4> | 0100 |
| computer | <2> | 0100 |
| James | <1,3,4> | 1000 |
| Madison | <3> | 0001 |
| mobile | <2> | 0001 |
| movie | <3,4> | 0010 |