Επεξεργασία Ερωτημάτων (επιπλέον υλικό)

Εαρινό εξάμηνο 2012-13

http://delab.csd.auth.gr/courses/c_dbimpl/index.html

Ευχαριστίες

- Υλικό βασισμένο στο βιβλίο “Database Systems: The Complete Book”
  - Κεφάλαια 15.1-15.6, 16.2,16.4-16.6

Φάσεις Επεξεργασίας

1. Δημιουργία πλάνου εκτέλεσης - Βελτιστοποίηση
2. Εκτέλεση ερωτήματος

- Σε κλασικά συστήματα, οι δύο φάσεις είναι ακολουθιακές. Όταν οι δύο φάσεις είναι ενεργές παράλληλα, τότε έχουμε **προσαρμόσιμη** επεξεργασία.

Εκτέλεση ερωτήματος

- **Μέρος Α΄: Αλγόριθμοι ενός περάσματος**
- Μέρος Β΄: Εκτέλεση συνδέσεων όταν καμία από τις δύο σχέσεις δεν χωράει εξολοκλήρου στην κύρια μνήμη.
  - Αλγόριθμοι δύο περασμάτων
- Μέρος Γ΄: Υλοποίηση με επαναλήπτες

Μέρος Α΄: Αλγόριθμοι ενός περάσματος

- Διαβάζουν μόνο 1 φορά την είσοδό τους.
- Διαχωρίζονται σε
  - Αλγορίθμους που επεξεργάζονται κάθε πλειάδα ξεχωριστά.
  - Σε μοναδιαίους τελεστές που απαιτούν να διαβάσουν όλη (σχεδόν) την είσοδο.
  - Σε δυαδικούς τελεστές.
- Οι δύο τελευταίες κατηγορίες εφαρμόζονται όταν τουλάχιστον μία από τις εισόδους τους χωράει ολόκληρη στην κύρια μνήμη.

Αλγόριθμοι που επεξεργάζονται κάθε πλειάδα ξεχωριστά

- Αναφέρονται και ως tuple-at-a-time
- Στην κατηγορία αυτή ανήκουν η επιλογή και η προβολή (χωρίς διαγραφή διπλοτύπων).
- Χρειάζεται να υπάρχει τουλάχιστον μία διαθέσιμη σελίδα στην κύρια μνήμη.

Άλλοι μοναδιαίοι αλγόριθμοι ενός περάσματος (1)

- Απαλοιφή διπλοτύπων *δ(R)*
  - Σε μία σελίδα διαβάζουμε πλειάδες της R και στις υπόλοιπες διαθέσιμες σελίδες της μνήμης κρατάμε αντίτυπα από τις προηγούμενες πλειάδες.
  - Για μείωση του υπολογιστικού κόστους απαιτείται κάποια δομή όπως hash table ή ισορροπημένο δυαδικό δένδρο.
  - Προσοχή: ένας τέτοιος αλγόριθμος ενός περάσματος πρέπει να χρησιμοποιείται μόνο όταν είμαστε σίγουροι ότι δεν θα προκύψουν προβλήματα μνήμης.
    - Για να αποφεύγεται το καταστροφικό για την απόδοση φαινόμενο του thrashing.

Άλλοι μοναδιαίοι αλγόριθμοι ενός περάσματος (2)

- Ομαδοποίηση $\gamma_L(R)$
  - Πρέπει να διαβαστεί όλη η είσοδος (R) πριν αρχίσουν να παράγονται αποτελέσματα.
  - Στη μνήμη αποθηκεύεται μία δομή, η οποία έχει μία εγγραφή για κάθε ομάδα.
  - Η εγγραφή περιέχει χαρακτηριστικά όπως min, max, count, sum, αλλά όχι average (υπολογίζεται έμμεσα από count και sum).

Δυαδικοί αλγόριθμοι ενός περάσματος (1)

R U S (set union)

- Έστω ότι S η μικρότερη σχέση που καταλαμβάνει M-1 από τις M συνολικά διαθέσιμες σελίδες της μνήμης.
- Διαβάζουμε κάθε πλειάδα της S και την εγγράφουμε κατευθείαν στην έξοδο.
- Ταυτόχρονα, δημιουργούμε μία δομή αναζήτησης στις M-1 σελίδες της μνήμης με κλειδί όλη την πλειάδα.
- Διαβάζουμε τις σελίδες της R μία προς μία χρησιμοποιώντας την ελεύθερη σελίδα στη μνήμη. Για κάθε πλειάδα της R, ελέγχουμε αν υπάρχει και στην S και μόνο αν δεν υπάρχει την εγγράφουμε στην έξοδο.

Δυαδικοί αλγόριθμοι ενός περάσματος (2)

R ∩ S (set intersection)

- Έστω ότι S η μικρότερη σχέση που καταλαμβάνει M-1 από τις M συνολικά διαθέσιμες σελίδες της μνήμης.
- Διαβάζουμε κάθε πλειάδα της S και δημιουργούμε μία δομή αναζήτησης στις M-1 σελίδες της μνήμης με κλειδί όλη την πλειάδα.
- Για κάθε πλειάδα t της R, ελέγχουμε αν υπάρχει και στην S και μόνο αν υπάρχει στην S, εγγράφουμε την t στην έξοδο.
- Αν θέλουμε να αγνοήσουμε τα διπλότυπα: κάθε διπλότυπο της S αποθηκεύεται στη μνήμη 1 φορά αλλά υπάρχει και ένας μετρητής για τις φορές εμφάνισης.
  - Για κάθε πλειάδα t της R που υπάρχει και στην S, μειώνουμε κατά ένα αυτόν τον μετρητή.
  - Αν δεν έχει μηδενιστεί ο μετρητής, παράγεται η t.

Δυαδικοί αλγόριθμοι ενός περάσματος (3)

Συνδέσεις $R \bowtie_C S$:

- Έστω ότι S η μικρότερη σχέση που καταλαμβάνει M-1 από τις M συνολικά διαθέσιμες σελίδες της μνήμης.
- Διαβάζουμε κάθε πλειάδα της S και δημιουργούμε μία δομή αναζήτησης στις M-1 σελίδες της μνήμης με κλειδί το χαρακτηριστικό σύνδεσης C.
- Για κάθε πλειάδα t της R, ελέγχουμε αν υπάρχει στην S πλειάδα με ίδια πεδία C. Αν ισχύει αυτή η συνθήκη, παράγονται τα αντίστοιχα ζεύγη.
- Με αυτή τη μέθοδο, ένα one pass hash join προκαλεί μόνο (b(R)+b(S)) προσπελάσεις.

Εκτέλεση ερωτήματος

- Μέρος Α΄: Αλγόριθμοι ενός περάσματος
- **Μέρος Β΄: Εκτέλεση συνδέσεων όταν καμία από τις δύο σχέσεις δεν χωράει εξολοκλήρου στην κύρια μνήμη.**
  - **Αλγόριθμοι δύο περασμάτων**
- Μέρος Γ΄: Υλοποίηση με επαναλήπτες

Αλγόριθμοι για την υλοποίηση συνδέσεων

- Εμφωλιασμένοι βρόχοι (nested loops)
- Ταξινόμηση – σύνδεση (sort-merge join)
- Σύνδεση με κατάλογο (join with index)
- Σύνδεση με κατακερματισμό (hash join)

Ψευδοκώδικας για nested loops

Έστω η σύνδεση $R(A,C) \bowtie S(C,D)$:

Για κάθε πλειάδα $r \in R$

  Για κάθε πλειάδα $s \in S$

    Αν r.C = s.C

      τότε επέστρεψε στην έξοδο το ζεύγος r,s

Ψευδοκώδικας για sort-merge join

1. ταξινόμησε τους R και S αν δεν είναι ήδη ταξινομημένοι
2. $i \leftarrow 1$; $j \leftarrow 1$;

   Όσο $(i \leq n(R)) \wedge (j \leq n(S))$

   Αν R{i}.C = S{j}.C τότε outputTuples(i,j)

   Αλλιώς αν R{i}.C > S{j}.C τότε $j \leftarrow j+1$

   Αλλιώς αν R{i}.C < S{j}.C τότε $i \leftarrow i+1$

```
outputTuples(i,j) {
  Όσο (R{i}.C = S{j}.C) ∧ (i ≤ n(R))
      {jj ← j;
      Όσο (R{i}.C = S{jj}.C) ∧ (jj ≤ n(S)) {
          επέστρεψε το ζεύγος R{i}, S{ jj }
          jj ← jj+1  }
       i ← i+1  }
}
```

Παράδειγμα

i	R{i}.C	S{j}.C	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

Ύπαρξη καταλόγου

- Έστω ότι υπάρχει κατάλογος για το S.C

Για κάθε r ∈ R

X ← index (S, C, r.C)

για κάθε s ∈ X

επέστρεψε το ζεύγος r,s

//το X περιλαμβάνει τις πλειάδες s του S για τις οποίες ισχύει s.C = r.C

Ψευδοκώδικας για hash join

- Έστω η συνάρτηση κατακερματισμού h→[0,k]
- Κάδοι για R: G0, G1, ... Gk
- Κάδοι για S: H0, H1, ... Hk

Αλγόριθμος

1. Τοποθέτησε τις πλειάδες του R στους G κάδους.
2. Τοποθέτησε τις πλειάδες του S στους H κάδους.
3. Για i = 0… k

   συνδύασε τις πλειάδες στους Gi και Hi κάδους

Παράδειγμα κόστους για nested loops

R(A,C) ⋈ S (C,D):

- Έστω ότι οι σχέσεις **ΔΕΝ** είναι σε συνεχόμενες θέσεις στο δίσκο
- Έστω ότι n(R) = 10000 πλειάδες, n(S) = 5000, bfr_T =10 (δηλ. 10 πλειάδες ανά σελίδα) και ότι η διαθέσιμη μνήνη MEM=101 σελίδες.

Κόστος:

- Για κάθε 1 πλειάδα του R διαβάζουμε όλες τις πλειάδες του S. Κάθε πλειάδα μπορεί να είναι σε διαφορετική σελίδα:
  - Σύνολο: 10000(1+5000) = 50010000 I/O

Τι μπορούμε να κάνουμε καλύτερα

Καλύτερη χρησιμοποίηση μνήμης (block nested loop):

1. Διαβάσουμε 100 σελίδες από την R.
2. Διαβάζουμε όλες τις πλειάδες μία-μία από την S και πραγματοποιούμε τη σύνδεση.
3. Επαναλαμβάνουμε μέχρι να διαβαστεί όλη η R.

Κόστος:

- 1000 I/O για να γεμίσουμε τις 100 σελίδες του (1)
- 5000 I/O για το (2)
- Επαναλαμβάνουμε 10 φορές
- Σύνολο 60000 I/O

Τι μπορούμε να κάνουμε ακόμη καλύτερα

- Να χρησιμοποιήσουμε ως εξωτερική τη μικρότερη σχέση.
- Ισχύει $R(A,C) \bowtie S(C,D) = S(C,D) \bowtie R(A,C)$

Κόστος:

- 1000 I/O για να γεμίσουμε τις 100 σελίδες του (1) αυτή τη φορά με πλειάδες του S
- 10000 I/O για το (2) που πλέον αναφέρεται στην R
- Επαναλαμβάνουμε 5 φορές
- Σύνολο 5(1000+10000) = 55000 I/O

Άλλη εκδοχή

- Οι σχέσεις είναι αποθηκευμένες σε συνεχόμενες θέσεις στο δίσκο.

Κόστος προηγούμενης περίπτωσης:

- 100 I/O για να γεμίσουμε τις 100 σελίδες με πλειάδες του S
- 1000 I/O για το R
- Επαναλαμβάνουμε 5 φορές
- Σύνολο 5(100+1000) = 5500 I/O

Χρήση merge-join

- Έστω ότι και οι δύο σχέσεις είναι ήδη ταξινομημένες και αποθηκευμένες σε συνεχόμενες θέσεις στο δίσκο.
- Χρειάζεται να διαβάσουμε μία φορά τις σχέσεις R, S.
- Κόστος: 1000 + 500 = 1500.

Ταξινόμηση

- Η μέθοδος sort-merge join είναι πολύ γρήγορη αλλά απαιτεί ταξινομημένες εισόδους.
- Αν οι είσοδοι δεν είναι ταξινομημένοι ήδη…

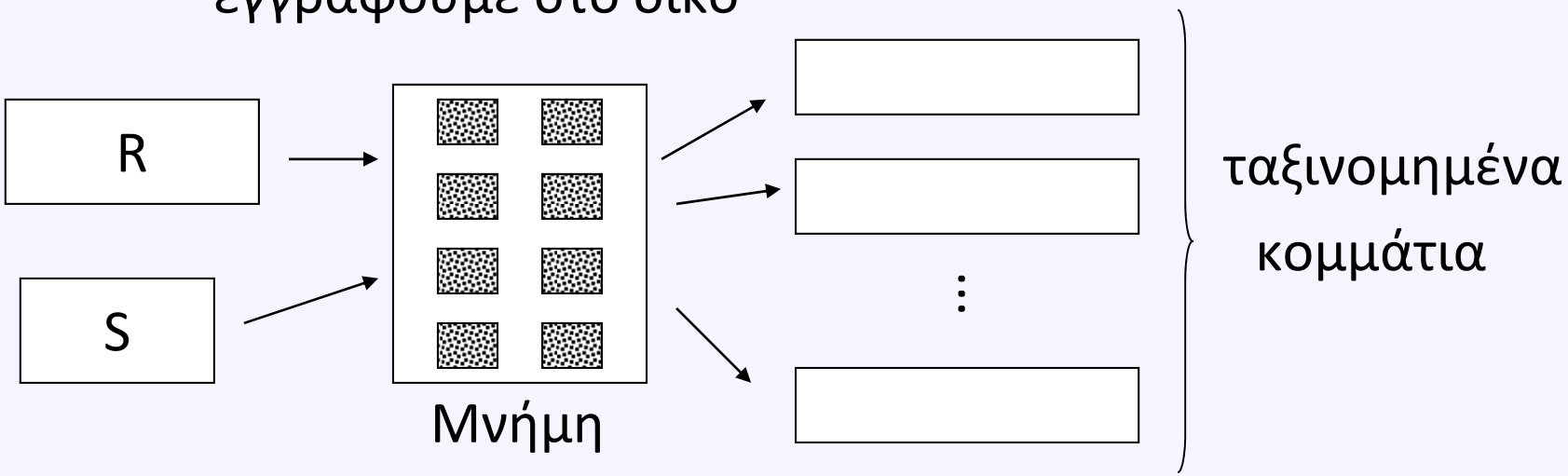
Χρήση Merge Sort

Φάση Α΄:

Χωρίζουμε την R και την S σε κομμάτια των MEM σελίδων

Για κάθε τέτοιο κομμάτι της R και της S:

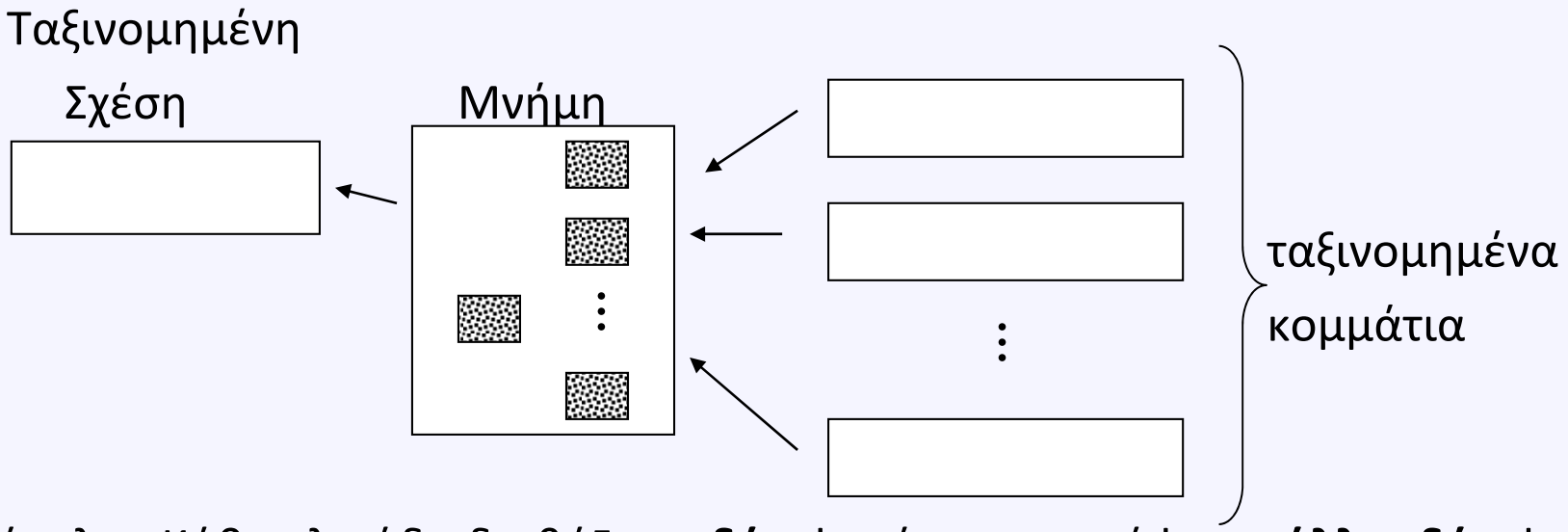
- το διαβάζουμε
- ταξινομούμε στη μνήμη
- εγγράφουμε στο δίκο



Χρήση Merge Sort

Φάση Β΄:

Διαβάζουμε τα ταξινομημένα κομμάτια ανά σελίδα + συγχωνεύουμε + εγγράφουμε στο δίκο



Σύνολο: Κάθε πλειάδα διαβάζεται **δύο** φορές και εγγράφεται **άλλες δύο** φορές.

Στο παράδειγμα το κόστος είναι: 4(1500) = 6000

Ποιος είναι πιο αποδοτικός αλγόριθμος

- Ο Block nested loop ή ο merge-join;;
- Αν δεν είναι ταξινομημένες οι πλειάδες έχει κόστος 5500, ενώ ο merge-join 6000 + 1500 = 7500 I/O.

Έστω ότι 10πλασιάζουμε το μέγεθος των σχέσεων:

- BNL: 50(100 + 10000) = 505000 I/O
- SMJ: 5(5000 + 10000) = 75000 I/O

Απαιτήσεις μνήμης για merge-sort

- Έστω ότι είναι διαθέσιμες MEM σελίδες στη μνήμη
- και ότι η σχέση έχει μέγεθος x σελίδες
- Πλήθος κομματιών (# chunks) = x/MEM
- Για τη Φάση Β΄ χρειάζονται τουλάχιστον (x/MEM) σελίδες στη μνήμη.
- Άρα (x/MEM) < MEM, δηλ. $MEM > \sqrt{x}$

Βελτίωση στον αλγόριθμο merge join

- Αντί να κάνουμε την πράξη σύνδεσης στους ταξινομημένους πίνακες, μπορούμε να την κάνουμε απευθείας στα ταξινομημένα κομμάτια.
- Γλιτώνουμε 2(b(R)+b(S)).
- Στο παράδειγμα στις προηγ. διαφάνειες το κόστος πέφτει στα 4500 I/O.
- Αυξημένες απαιτήσεις μνήμης: $MEM > \sqrt{b_R + b_S}$

Ύπαρξη καταλόγου

- Έστω ότι υπάρχει κατάλογος στο R.C που χωράει εξ ολοκλήρου στην κύρια μνήμη.
- Ο S είναι σε συνεχόμενες θέσεις στο δίσκο αλλά δεν είναι ταξινομημένος.

Κόστος:

- Για κάθε πλειάδα του S (500 I/O), ελέγχεται ο κατάλογος (μηδέν I/O).
- Αν υπάρχει αντίστοιχη πλειάδα του R: 1 I/O.
- Το κόστος σχετίζεται με το πλήθος των πλειάδων εξόδου

Χρήση hash join

- Κάθε σχέση διαβάζεται μία φορά, εφαρμόζεται η συνάρτηση κατακερματισμού και οι κάδοι που προκύπτουν αποθηκεύονται στο δίσκο.
  - Οι κάδοι πρέπει να είναι λιγότεροι από τις θέσεις μνήμης.
  - Κόστος 2b για κάθε σχέση.
- Κατόπιν οι αντίστοιχοι κάδοι μεταφέρονται στη μνήμη για την τελική σύνδεση.
- Συνολικό (προσεγγιστικό) κόστος στο παράδειγμα: 3(1000+500)= 4500 I/O.

Απαιτήσεις στη μνήμη

- Πρέπει για τουλάχιστον μία από τις σχέσεις, ο κάθε κάδος να χωράει στην κύρια μνήμη.
- Προκύπτουν απαιτήσεις παρόμοιες με την ταξινόμηση συγχώνευσης: $MEM > \sqrt{x}$
- Στην περίπτωση όμως του hash join, το μέγεθος x αρκεί να αναφέρεται στη μικρότερη σχέση, ενώ στην mergesort αναφέρεται στη μεγαλύτερη.

Γενικές Παρατηρήσεις

- Οι αλγόριθμοι τύπου nested loops είναι απλοί, αλλά εφαρμόζονται μόνο σε μικρές σχέσεις.
- Για συνδέσεις ισότητας, όπου οι σχέσεις δεν είναι ταξινομημένες ούτε υπάρχουν κατάλληλοι κατάλογοι, τότε ο αλγόριθμος hash join είναι συνήθως ο καλύτερος.
- Για συνδέσεις ανισότητας, ο καταλληλότερος συνήθως είναι ο sort-merge join (όπως και όταν οι είσοδοι της σύνδεσης είναι ταξινομημένοι).
- Αν υπάρχουν κατάλογοι, πρέπει να υπολογίζεται το μέγεθος της εξόδου πριν ληφθεί απόφαση χρησιμοποίησης.

Ερωτήματα

- Τι γίνεται όταν οι σχέσεις είναι τόσο μεγάλες που δεν ικανοποιούν τις συνθήκες για hash join, merge join;
- Γιατί το κόστος να καθορίζεται μόνο από τα I/O;
- Γιατί δεν προσμετράμε την αποθήκευση του αποτελέσματος;

Μέρος Γ΄: Υλοποίηση τελεστών

- Οι τελεστές μπορούν να υλοποιηθούν προγραμματιστικά ως **επαναλήπτες (iterators)**
  - Οι οποίοι αποτελούνται από 3 μεθόδους:
  1. void Open()
  2. Tuple getNext()
  3. void Close()

Διευκολύνεται η διοχέτευση (pipelining).

Open()

- Αρχικοποιεί την διαδικασία παραγωγής αποτελεσμάτων.
- Αρχικοποιεί ό,τι εσωτερικές δομές απαιτούνται για τον τελεστή.
- Καλεί την open() σε κάθε μία από τις εισόδους

getNext()

- Επιστρέφει την επόμενη πλειάδα.
  - Αυτό κατά βάση πραγματοποιείται με αναδρομικές κλήσεις της getNext() στα ορίσματα του τελεστή.
- Τροποποιεί τις εσωτερικές δομές ώστε να είναι έτοιμες για την επεξεργασία της επόμενης πλειάδας εισόδου.
- Στέλνει ειδικό μήνυμα με τη μορφή πλειάδας ότι δεν υπάρχουν άλλες πλειάδες εξόδου, όταν ολοκληρωθεί η επεξεργασία.

Close()

- Χειρίζεται την ολοκλήρωση της επεξεργασίας.
- Καλεί την close() στα ορίσματα.

Table Scan (R)

```
Open() {
    b ← first block of R
    t ← first tuple of b
}
```

```
Tuple GetNext(){
    IF (t is empty) {
        b ← next block
        IF (b is empty) RETURN Tuple(EOF)
        ELSE t ← first tuple of b
        }
    oldt ← t;      t← next tuple in b;      RETURN oldt;
}
```

```
Close () {}
```

BagUnion(TableScan(R),TableScan(S))

```
Open () {
      (TableScan(R)).Open();
      CurrentRelation ← R;
}
```

```
Tuple getNext() {
      IF (CurrentRelation = R) {
            t← (TableScan R)).getNext();
            IF (t<>EOF) RETURN t;
            ELSE { (TableScan(S)).Open(); CurrentRelation ← S; }
      RETURN (TableScan(S)).getNext();
}
```

```
Close(){
      (TableScan(R)).Close();          (TableScan(S)).Close;
}
```

Projection(Operator X, list of attributes L)

```
Open () {
    X.Open();
}
```

```
Tuple getNext() {
    t←X.getNext();
    IF (t<>EOF)  {
            t ← project L
    }
    RETURN t;
}
```

```
Close(){
    X.Close();
}
```

GROUP (Operator X)

```
Open () {
    X.Open();
    repeat {
        t←X.getNext();    create-updateGroup(t);
    } until (t==EOF)
    X.Close();
}
```

```
Tuple getNext() {
    g ← nextGroupTansformedtoTuple();
    IF (g is empty) RETURN EOF;
    ELSE RETURN g;
}
```

```
Close(){
    destroyGroups();
}
```

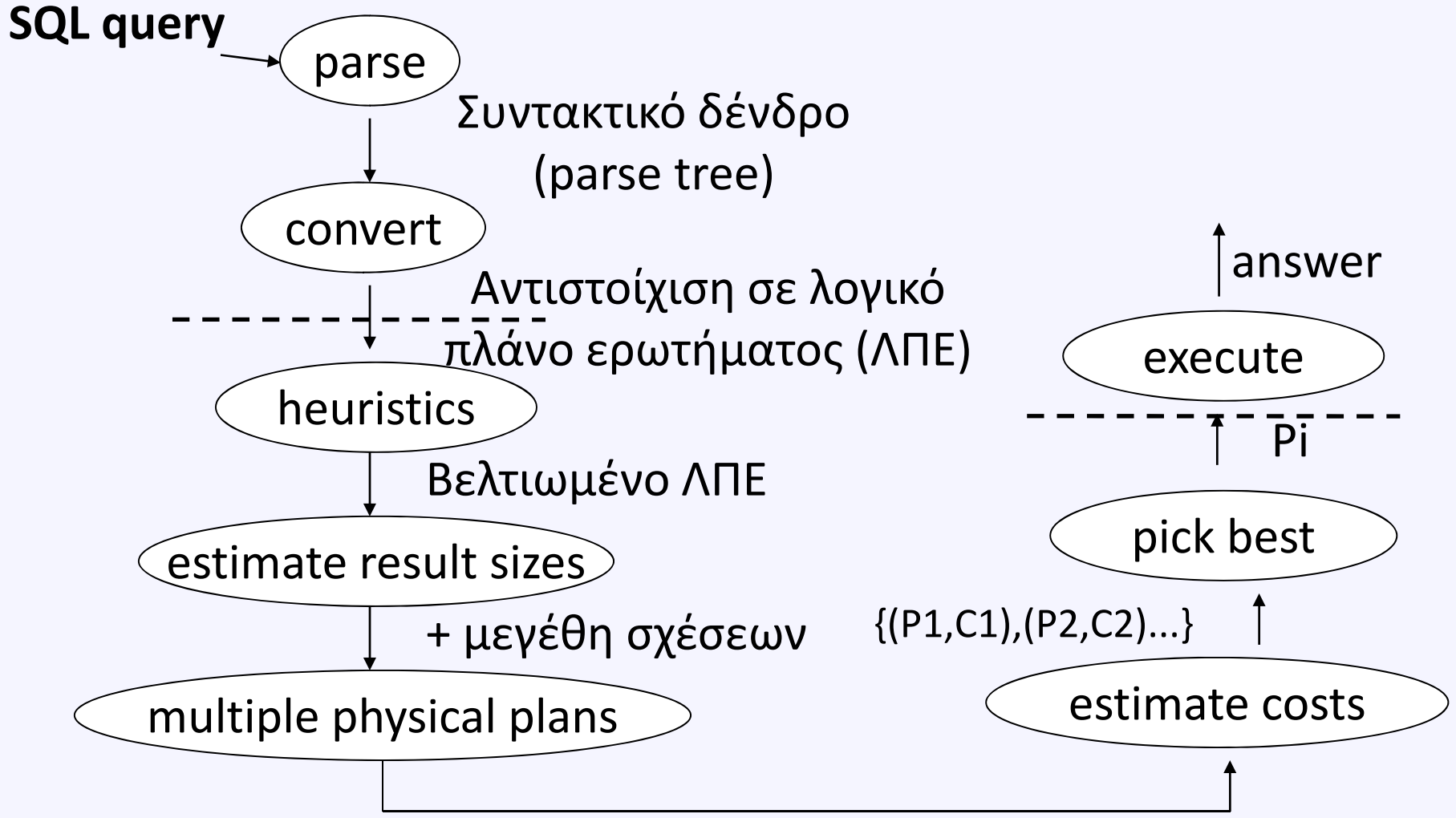
Επαναλήπτες

- Ομοιόμορφος και κομψός σχεδιασμός.
- Δεν απαιτεί ειδικά μηνύματα ελέγχου.
  - Το ΠΕ αυτοχρονοδρομολογείται.
- Τα ενδιάμεσα αποτελέσματα παράγονται όταν χρειάζονται.
  - Όχι πλημμυρισμός συστήματος.
  - Καλύτερη χρήση μνήμης.
- Υποστήριξη διοχέτευσης (pipelining)
- Έχουν μεγάλη ευελιξία.
  - Η χρήση τους δεν αποτρέπει την υποστήριξη blocking τελεστών ή σημείων προσωρινής αποθήκευσης ενδιάμεσων αποτελεσμάτων (για αποφυγή προβλημάτων έλλειψης μνήμης).
- Απαιτείται προσοχή στη διαχείριση μνήμης.
- Όχι καλή υποστήριξη ροών δεδομένων.

Φάσεις Επεξεργασίας

1. **Δημιουργία πλάνου εκτέλεσης - Βελτιστοποίηση**
2. *Εκτέλεση ερωτήματος*

Ανάλυση Βελτιστοποίησης



Τεχνικές Βελτιστοποίησης

Κύρια χαρακτηριστικά:

- Ισοδύναμα ΠΕ που εξετάζονται (search space).
- Υπολογισμός κόστους.
  - Πλήθος I/O.
  - Μέγεθος σχέσεων.
    - Για τις ενδιάμεσες σχέσεις, χρειάζεται υπολογισμός μεγέθους αποτελεσμάτων.
    - Βασίζεται σε στατιστική πληροφορία (μεταδεδομένα στα ΣΔΒΔ).
    - Είναι ανεξάρτητο της φυσικής υλοποίησης.
- Επιλογή «βέλτιστου» ΠΕ (search strategy).

Υπολογισμός μεγέθους αποτελεσμάτων

- Έστω μία σχέση Χ που επεξεργάζεται από ένα σχεσιακό τελεστή *o,* και παράγεται η σχέση Υ.
- Η επιλεξιμότητα (selectivity) του *o* είναι

  $sel_o = n_y / n_x$
- Για συνδέσεις ισχύει:

  $sel_{(R \bowtie S)} = n_{(R \bowtie S)} / n_R \cdot n_S$
- Όταν δεν υπάρχει ακριβής πληροφορία για την επιλεξιμότητα, το μέγεθος αποτελεσμάτων υπολογίζεται με τη βοήθεια απλών κανόνων.
  - Οι κανόνες που παρουσιάζονται στη συνέχεια δεν είναι οι μοναδικοί!

Στατιστικά για το μέγεθος αποτελεσμάτων

- N(R) : # πλειάδων της R
- S(R) : # bytes (μέγεθος) της R
- B(R): # σελίδων της R
- d(R, A) : # μοναδικές διαφορετικές τιμές του χαρακτηριστικού A στη σχέση R

Παράδειγμα

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

A: 20 byte string

B: 4 byte integer

C: 8 byte date

D: 5 byte string

N(R) = 5

d(R,A) = 3

d(R,B) = 1

S(R) = 37

d(R,C) = 5

d(R,D) = 4

Καρτεσιανό γινόμενο R1 x R2

$W = R1 \times R2$

$N(W) = N(R1) \cdot N(R2)$

$S(W) = S(R1) + S(R2)$

Τελεστής επιλογής (1)

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

d(R,A)=3

d(R,B)=1

d(R,C)=5

d(R,D)=4

$W = \sigma_{z=val}(R)$

N(W)=N(R) / d(R,z)

- Έμμεση παραδοχή ομοιόμορφης κατανομής
- Εναλλακτικά, αντί για d(R,z) μπορούμε να έχουμε το μέγεθος του πεδίου ορισμού του z.

Τελεστής επιλογής (2)

$W = \sigma_{z \geq val}(R)$

$N(W) = N(R)/2$

ή

$N(W) = N(R)/3$

ή

$N(W) = f \cdot N(R)$,

$f = (maxval - val + 1)/(maxval - minval + 1)$

Το f μπορεί να είναι και το ποσοστό των μοναδικών τιμών πάνω από val.

Σύνδεση $R \bowtie_C S$ – C: κύριο/ξένο κλειδί

Έστω ότι S.C το ξένο κλειδί.

$N(R \bowtie_C S) = N(S)$

Σύνδεση $R \bowtie_C S$ - γενικά

Παραδοχές/συμβολισμοί:

- $d(R,C) \leq d(S,C) \Rightarrow$ Κάθε τιμή του C στην R υπάρχει και στην S.
- $d(S,C) \leq d(R,C) \Rightarrow$ Κάθε τιμή του C στην S υπάρχει και στην R.

Έστω ότι $d(R,C) \leq d(S,C)$:

- Κάθε πλειάδα της R συνδέεται με $N(S)/d(S,C)$ πλειάδες της S, δηλ. $N(R \bowtie_C S) = N(R) \cdot N(S) / d(S,C)$

Γενικά:

- $N(R \bowtie_C S) = N(R) \cdot N(S) / \max\{d(S,C), d(R,C)\}$
- Πάντα ισχύει: $S(R \bowtie_C S) = S(R) + S(S) - S(C)$

Υπολογισμός μοναδικών τιμών (1)

- Π.χ., U = $\sigma_{A=a}$ (R) και έστω ότι η R έχει χαρακτηριστικά A,B,C,D

  d(U, A) = 1

  d(U, B) = ?

  d(U, C) = ?

  d(U, D) = ?

- Απλή υπόθεση:

  d(U,x) = *min{d(R,x), N(U)}*

Υπολογισμός μοναδικών τιμών (2)

Αντίστοιχα για συνδέσεις:

U = R(A,C) $\bowtie_C$ S (B,C):

- d(U,C) = min{d(R,C), d(S,C)}
- d(U,A) = d(R,A)
- d(U,B) = d(S,B)

- Αυτή η μέθοδος, κάνει την παραδοχή της διατήρησης των συνόλων τιμών.

Παράδειγμα

$Z = R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D)$

$U = R1(A,B) \bowtie R2(B,C) \rightarrow Z = U \bowtie R3(C,D)$

N(R1) = 1000	d(R1,A)=50	d(R1,B)=100
N(R2) = 2000	d(R2,B)=200	d(R2,C)=300
N(R3) = 3000	d(R3,C)=90	d(R3,D)=500

N(U) = 1000 X 2000 / max{100,200} = 10000

d (U,A) = 50	d(U,B) = 100	d(U,C) = 300

Παράδειγμα (συνέχεια)

$Z = R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D)$

$U = R1(A,B) \bowtie R2(B,C) \rightarrow Z = U \bowtie R3(C,D)$

$N(U) = 10000 \quad d(U,A)=50 \quad d(U,B)=100 \quad d(U,C)=300$

$N(R3) = 3000 \quad d(R3,C)=90 \quad d(R3,D)=500$

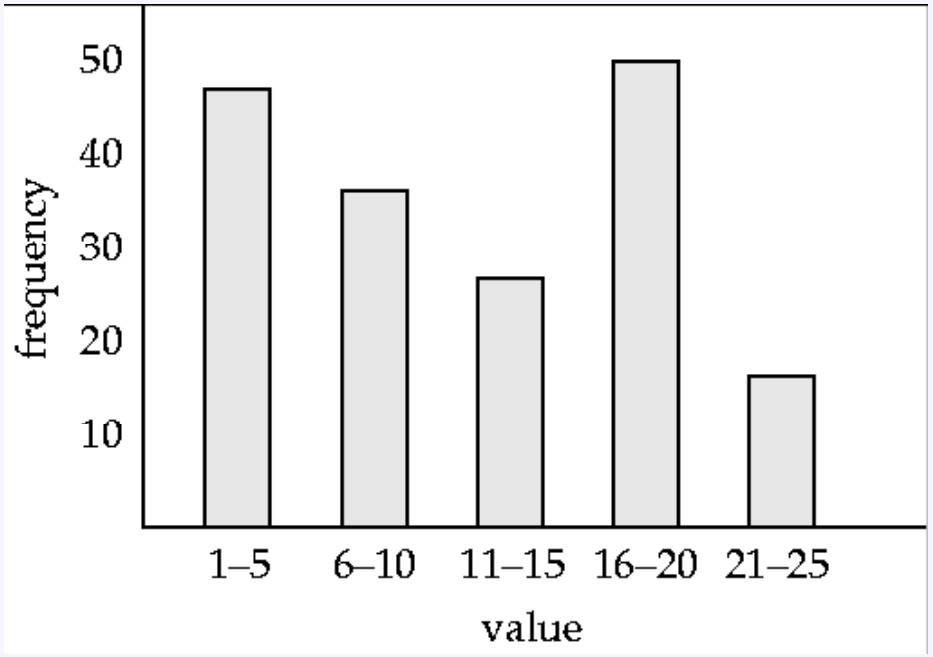
$N(Z) = 10000 \text{ X } 3000 / \max\{300,90\} = 100000$

$d(Z,A) = 50 \qquad d(Z,B) = 100 \qquad d(Z,C) = 90$

$d(Z,D) = 500$

Ιστογράμματα

- Περιέχουν πιο αναλυτική στατιστική πληροφορία:
- Π.χ. στην εικόνα φαίνεται ένα παράδειγμα ισο-ευρούς ιστογράμματος.



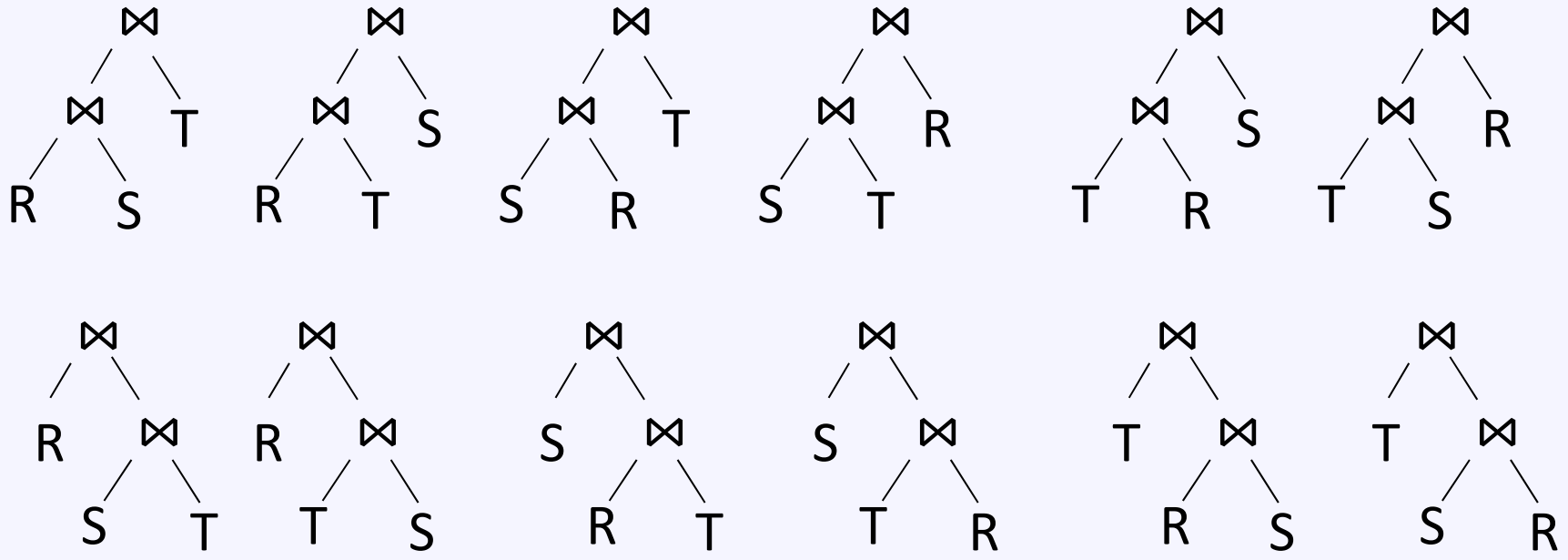
Υπολογισμός κόστους ΠΕ

- Έστω ότι μετράται σε πλήθος προσπελάσεων (I/O).
- Έχουμε ήδη εξετάσει
  - πως να υπολογίζουμε το πλήθος I/O για κάθε ένα απο τους συνηθισμένους τελεστές.
  - Και πως να υπολογίζουμε το μέγεθος των ενδιάμεσων σχέσεων.

- Άλλοι παράγοντες που επηρεάζουν το κόστος:
  - *Τρόπος μετάδοσης των ενδιάμεσων αποτελεσμάτων μεταξύ των τελεστών (π.χ., pipelining).*
  - **Σειρά των τελεστών.**

Πλήθος ΠΕ

- Συνήθως, το πιο σημαντικό είναι να καθοριστεί η διάταξη των συνδέσεων (join ordering).
- Έστω ένα ερώτημα: R ⋈ S ⋈ T
- Τα διαφορετικά λογικά πλάνα εκτέλεσης (ΛΠΕ) είναι



Απαρίθμηση πλάνων (1)

- Έστω *n* σχέσεις που συνδέονται μεταξύ τους.
- Από αυτές, οι *i (1≤i ≤n-1)* μπορούν να τοποθετηθούν στο αριστερό υποδένδρο και οι υπόλοιπες στο δεξί.
- Συνολικά, το πλήθος των σχημάτων των δένδρων δίνεται από τον αναδρομικό τύπο:

$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i)$$

- Επιπλέον, οι σχέσεις τοποθετούνται στα φύλλα με οποιαδήποτε διάταξη:
  - Άρα #ΛΠΕ = *T(n)· n! = (2(n-1))!/(n-1)!*
  - Έχει σημασία ποια είσοδος είναι αριστερά και ποια δεξιά στις περισσότερες συνδέσεις.

Απαρίθμηση πλάνων (2)

- Το πλήθος των ΛΠΕ αυξάνει πολύ γρήγορα.
- Ενδεικτικά:

#σχέσεις	#ΛΠΕ
2	2
3	12
4	120
5	1680
6	30240
7	665280
8	17297280

- Το πλήθος των φυσικών πλάνων εκτέλεσης (ΦΠΕ) είναι ακόμη πιο μεγάλο κατά ένα παράγοντα k^n όπου k το πλήθος των φυσικών υλοποιήσεων της σύνδεσης.

Δυναμικός προγραμματισμός (Δ.Π.)

- Έστω ότι το *n* είναι σχετικά μικρό.
- Εφαρμόζονται *n* περάσματα.
- Στο *k-οστό* πέρασμα, υπολογίζονται τα βέλτιστα πλάνα για σύνδεση *k* σχέσεων.

Παράδειγμα Δ.Π. για 4 σχέσεις R1, R2, R3, R4

- 1ο πέρασμα:

  Υπολογίζεται ο βέλτιστος τρόπος προσπέλασης της κάθε σχέσης

- 2ο πέρασμα:

  Για κάθε ζεύγος (Ri,Rj) υπολογίζεται ποιο (υπο)πλάνο από τα (Ri ⋈ Rj), (Rj ⋈ Ri) είναι το καλύτερο → 12 (υπο)πλάνα.

- 3ο πέρασμα:
  - Για κάθε μία από τις 4 τριπλέτες (Ri,Rj,Rk) υπολογίζεται ποιο από τα υποπλάνα (Ri ⋈(Rj ⋈ Rk)), ((Rj ⋈ Rk) ⋈ Ri), (Rj ⋈(Ri ⋈ Rk)), ((Ri ⋈ Rk) ⋈ Rj) κ.ο.κ. είναι το καλύτερο <u>επαναχρησιμοποιώντας</u> τα αποτελέσματα του 2ου περάσματος (για τα Rj ⋈ Rk, Ri ⋈ Rk, Ri ⋈ Rj) → 24 νέα (υπο)πλάνα.

Παράδειγμα (συνέχεια)

- 4ο πέρασμα
  - Ελέγχονται όλα τα πλάνα της μορφής

    (X ⋈ (Y ⋈ Z ⋈ W)) και (Y ⋈ Z ⋈ W) ⋈ X) με τη βοήθεια των αποτελεσμάτων του 3ου περάσματος (για τα βέλτιστα υποπλάνα Y ⋈ Z ⋈ W) →8 νέα υπο(πλάνα).
  - Ελέγχονται όλα τα πλάνα της μορφής

    (X ⋈ Y) ⋈ (Z ⋈ W) και (Z ⋈ W) ⋈ (X ⋈ Y) με τη βοήθεια των αποτελεσμάτων του 2ου περάσματος (για τα βέλτιστα υποπλάνα X ⋈ Y και Z ⋈ W) → 6 νέα υπο(πλάνα).
- Συνολικά εξετάζονται 50 αντί για 120 πλάνα.
- Έχει όμως πάλι μεγάλη πολυπλοκότητα (*$O(3^n)$*)
  - Επίσης μεγάλη χωρική πολυπλοκότητα (*$O(2^n)$*)

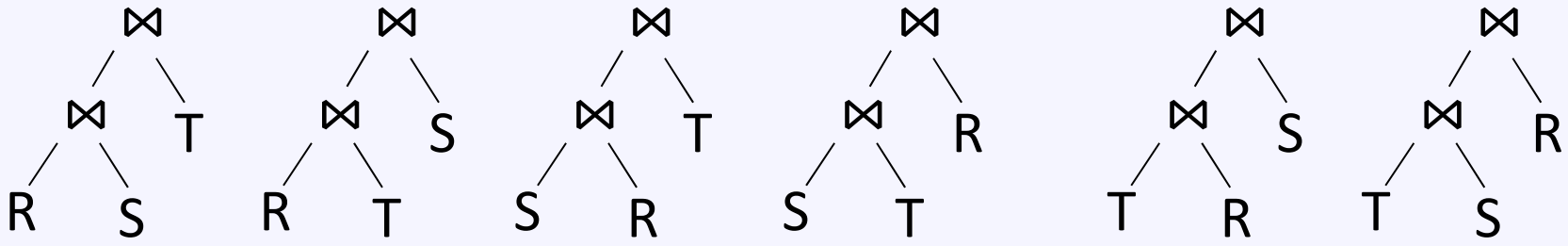
Βελτιστοποίηση τύπου Selinger

- Βελτιώνει την τεχνική δυναμικού προγραμματισμού.
- Κρατάει και το κόστος μη βέλτιστων υποπλάνων που παράγουν όμως ταξινομημένα αποτελέσματα
  - Σύμφωνα με απαιτήσεις του ερωτήματος (πχ. το ερώτημα περιέχει σχετική έκφραση order by)
  - Ή όταν η ταξινόμηση ενδέχεται να διευκολύνει την εκτέλεση μεταγενέστερων τελεστών σύνδεσης ή ομαδοποίησης.

- Γενικά υπάρχουν πολλές στρατηγικές βελτιστοποίησης. Πρακτικά, για μεγάλο πλήθος σχέσεων εφαρμόζονται ευριστικές ή άπληστες.

Γρηγορότερες βελτιστοποιήσεις

- Αποφυγή καρτεσιανών γινομένων.
- Έλεγχος το πολύ *n!* ΛΠΕ, έτσι ώστε οι συνδέσεις να είναι είτε η ρίζα είτε το αριστερό παιδί μιας άλλης σύνδεσης (left-deep trees).
  - Ο ΔΠ εξετάζει πολύ λιγότερες περιπτώσεις!



- Όταν οι συνδέσεις υλοποιούνται ως αλγόριθμοι ενός περάσματος, πχ. one-pass hash join, τότε οι απαιτήσεις μνήμης είναι συνήθως μικρότερες.
- Όταν οι συνδέσεις υλοποιούνται ως αλγόριθμοι εμφωλιασμένων βρόχων, τότε δεν χρειάζεται να επανα-υπολογίζονται ενδιάμεσες σχέσεις.