

Anna Östlin Pagh and Rasmus Pagh  
IT University of Copenhagen

# Advanced Database Technology

March 7, 2005

# QUERY COMPILATION II

Lecture based on [GUW, 16.4-16.7] and [AGMS '99, sec. 1,2,4,5]

Slides based on  
Notes 06-07: Query execution Part I-II  
for Stanford CS 245, fall 2002  
by Hector Garcia-Molina

# Overview: Physical query planning

- We assume that we have an **algebraic expression** (tree), and consider:
  - Statistical estimates of the **size** of relations given by subexpressions.
  - Choosing an **order** for operations, using various optimization techniques.
  - Completing the **physical query plan**.

## Estimating sizes of relations

The **sizes** of intermediate results are important for the choices made when planning query execution.

- Time for operations grow (at least) linearly with size of (largest) argument.
- The total size can even be used as a crude estimate on the running time.

## Classical approach: Heuristics

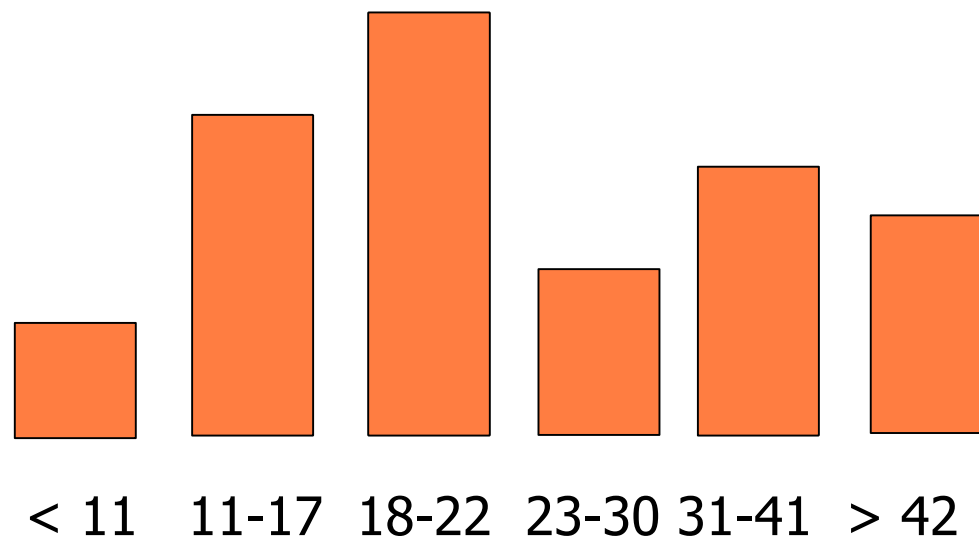
- In GUW section 16.4 a number of **heuristics** for estimating sizes of intermediate results are presented.
- This classical approach works well in some cases, but is unreliable in general.
- The modern approach is based on maintaining suitable **statistics** summarizing the data. (Focus of lecture.)

## Some possible types of statistics

- Random sample of, say 1% of the tuples. (NB. Should fit main memory.)
- The 1000 most frequent values of some attribute, with tuple counts.
- Histogram with number of values in different ranges.
- The "*skew*" of data values.

# Histogram

- Number of values/tuples in each of a number of intervals.



## On-line vs off-line statistics

- **Off-line:** Statistics only computed periodically, often operator-controlled (e.g. Oracle). Typically involves sorting data according to all attributes.
- **On-line:** Statistics maintained automatically at all times by the DBMS. Focus of this lecture.

# Maintaining a random sample

- To get a sample of expected size 1% of full relation:
  - Add a new tuple to the sample with probability 1%.
  - If a sampled tuple is deleted or updated, remember to remove from or update in sample.

# Estimating selects

- To estimate the size of a select statement  $\sigma_C(R)$ :
  - Compute  $|\sigma_C(R')|$ , where  $R'$  is the random sample of  $R$ .
  - If the sample is 1% of  $R$ , the estimate is  $100 |\sigma_C(R')|$ , etc.
  - The estimate is reliable if  $|\sigma_C(R')|$  is not too small (the bigger, the better).

## Problem session

- Suppose you want to estimate the size of a join statement  $R_1 \bowtie R_2$ .
- You have random samples of 1% of each relation.
- How do you do the estimation?
- Warning: Answer on next slide!

# Estimating join sizes

- To estimate the size of a join statement  $R_1 \bowtie R_2$ :
  - Compute  $|R'_1 \bowtie R'_2|$ , where  $R'_1$  and  $R'_2$  are samples of  $R_1$  and  $R_2$ .
  - If samples are 1% of the relations, estimate is  $100^2 |R'_1 \bowtie R'_2|$ . (Why?)
  - Shortly, you will see a considerably more reliable method, especially when we can store only a very small fraction of relations internally.

# Keeping a sample of bounded size

Reservoir sampling (Vitter '85):

- Initial sample consists of  $s$  tuples.
- A tuple inserted in  $R$  is stored in sample with probability  $s/(|R|+1)$ .
- When storing a new tuple, it *replaces* a randomly chosen tuple in existing sample (unless sample has size  $< s$  due to a deletion).

## Next: Statistics for join estimation (AGMS '99)

### Goal:

- Maintain information that "summarizes" or "sketches" each relation.
- Should be able to compute (reliable) join size estimates from sketches of **two** relations. (General case will not be covered.)

## Special case: Joining R with itself

- Assume at most  $t$  distinct join values.
- Observation:
  - If all values are equally frequent, self-join size  $SJ(R)$  is smallest.
  - If data is *skewed*  $SJ(R)$  is bigger. Worst case is where there is only 1 join value.
- Intuitively, we need to measure skew.

# Tug-of-war



- Each join value  $\mathbf{v}$  is assigned a random team: All occurrences of  $\mathbf{v}$  are members of that team.
- Compute  $Z$ =difference between number of members of the two teams.
- Estimate for  $SJ(R)$  is  $Z^2$ . (Outside of scope of this course: Provably good.)

## Computing Z on-line

- Pick hash function  $h$  "assigns teams", i.e., maps join value  $i$  to  $\varepsilon_i = h(i) \in \{-1, 1\}$ .
- When new tuple arrives with join value  $i$ , add  $h(i)$  to current value of  $Z$ .
- When tuple with join value  $i$  is deleted, subtract  $h(i)$  from  $Z$ .

# Estimating size of a join

- Have: Sketches  $Z_R$  and  $Z_S$  of relations  $R$  and  $S$ .
- Estimator for  $|R \bowtie S|$  is  $Z_R \cdot Z_S$ . (Outside of scope of this course: Provably good.)
- Standard deviation is  $\sqrt{SJ(R) \cdot SJ(S)}$  - roughly "how much we expect the estimate to deviate from the true value".
- AGMS uses **Variance**, which is the square of the standard deviation.

# Boosting reliability

- The reliability of estimates can be improved by storing **k** sketches  $Z_1, \dots, Z_k$ , and taking the mean of the resulting k estimates.
- Decreases standard deviation by a factor of  $\sqrt{k}$ .

## To further improve estimates

- "Intelligently split" the domain of the join attribute in several parts, and estimate join sizes for each part. (Dobra et al, 2002)
- "Skim" the most frequent values (by sampling) and use tug-of-war only for less frequent elements. (Ganguly et al, 2004)

## Next: Physical query planning

- Have:
  - Relational algebra expression
  - Size estimates
- Need to determine:
  - In which order to compute subexpressions.
  - What algorithm to use for each operation.



## Choosing a physical plan

### **Option 2:** "Dynamic programming".

- Find best plans for subexpressions in bottom-up order.
- Might find several best plans:
  - The best plan that produces a sorted relation wrt. a later join or grouping attribute.
  - The best plan in general.

# Choosing a physical plan

## **Options 3,4,...:**

Other (heuristic) techniques from combinatorial optimization.

- Greedy plan selection.
- Hill climbing.
- ...

## Order for grouped operations

- Recall that we **grouped** commutative and associative operators, e.g.  
 $R1 \mid \rangle \langle \mid R2 \mid \rangle \langle \mid R3 \mid \rangle \langle \mid \dots \mid \rangle \langle \mid Rk.$
- For such expressions we must choose an evaluation order (a parenthesized expression), e.g.  
 $(R1 \mid \rangle \langle \mid R4) \mid \rangle \langle \mid (R3 \mid \rangle \langle \mid \dots) \dots ( \dots \mid \rangle \langle \mid Rk).$

## Order for grouped operations

- Book recommends considering just **left balanced** expressions  
(...((R4 |><| R2) |><| R7) |><| ...) |><| Rk.
- This gives **k!** possible expressions.
- Considering all possible expressions gives around **2<sup>k</sup>k!** possibilities - not so many more.

## Choosing final algorithms

- Usually best to use existing indexes.
- Sometimes building indexes or sorting on the fly is advantageous.
- Sorting based algorithms may beat hashing based algorithms if one of the relations is already sorted.
- Just do the calculation and see!

## Pipelining and materialization

- Some algorithms (e.g.  $\sigma$  implemented as a scan) require little internal memory.
- **Idea:** Don't write result to disk, but feed it to the next algorithm immediately.
- Such **pipelining** may make many algorithms run "at the same time".
- Sometimes even possible with algorithms using more memory, such as sorting.

# Influencing the query plan

- One of the great things about DBMSs is that the user does **not** need to know about query compilation/optimization.
- ...unless things turn out to run too slowly - then manual tuning may be needed.
- Tuning can use statistics and query plans to suggest the creation of certain indexes, for example.

## Summary

- **Size estimation** (using statistics) is an important part of query optimization.
- Given size estimates and a relational algebra expression, **query optimization** essentially consists of computing the (estimated) cost of all possible query plans.
- Other issues are **pipelining** and memory usage during execution.