# Lecture 23: Query Execution

Friday, November 17, 2000

## Outline

- An algebra for queries (6.1)
- Introduction to physical-query-plan operators (6.2)
- One-pass algorithms (6.3)
- Nested-loop joins (6.4)
- Two-pass algorithms based on sorting (6.5)

## An Algebra for Queries

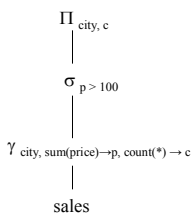
- Logical operators
  - *what* they do
- Physical operators
  - *how* they do it

## Logical Operators in the Algebra

- Union, intersection, difference
- Selection $\sigma$
- Projection $\Pi$
- Join $\bowtie$
- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$

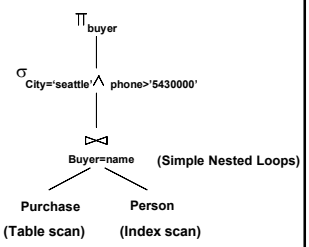
## Example

Select city, count(*)
From sales
Group by city
Having sum(price) > 100




## Physical Operators

```
SELECT S.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
      Q.city='seattle' AND
      Q.phone > '5430000'
```




Query Plan:
• logical tree
• implementation choice at every node
• scheduling of operations.

Some operators are from relational algebra, and others (e.g., scan, group) are not.

## Scanning Tables

- The table is *clustered* (I.e. blocks consists only of records from this table):
  - Table-scan: if we know where the blocks are
  - Index scan: if we have a sparse index to find the blocks
- The table is unclustered (e.g. its records are placed on blocks with other tables)
  - May need one read for each record

## Sorting While Scanning

- Sometimes it is useful to have the output sorted
- Three ways to scan it sorted:
  - If there is a primary or secondary index on it, use it during scan
  - If it fits in memory, sort there
  - If not, use multiway merging

## Estimating the Cost of Operators

- Very important for the optimizer (next week)
- Parameters for a relation R
  - B(R) = number of blocks holding R
    - Meaningful if R is clustered
  - T(R) = number of tuples in R
    - E.g. may need when R is unclustered
  - V(R,a) = number of distinct values of the attribute a

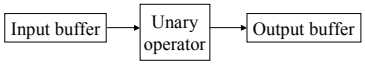
## Cost of the Scan Operator

- Clustered relation:
  - Table scan: B(R); to sort: 3B(R)
  - Index scan: B(R); to sort: B(R) or 3B(R)
- Unclustered relation
  - T(R); to sort: T(R) + 2B(R)

## One-Pass Algorithms

Selection $\sigma(R)$, projection $\Pi(R)$

- Both are *tuple-at-a-Time* algorithms
- Cost: B(R)




## One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep tuples in memory
- When new tuple arrives, need to compare it with previously seen tuples
- Balanced search tree, or hash table
- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

## One-pass Algorithms

Grouping: $\gamma_{city,\ sum(price)}$ (R)

- Need to store all cities in memory
- Also store the sum(price) for each city
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: number of cities fits in memory

## One-pass Algorithms

Binary operations: R ∩ S, R U S, R – S

- Assumption: min(B(R), B(S)) <= M
- Scan one table first, then the next, eliminate duplicates
- Cost: B(R)+B(S)

## Nested Loop Joins

- Tuple-based nested loop R ⋈ S

For each tuple r in R do
  For each tuple s in S do
    if r and s join then output (r,s)

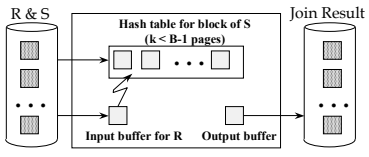
- Cost: T(R) T(S), sometimes T(R) B(S)

## Nested Loop Joins

- Block-based Nested Loop Join

For each (M-1) blocks bs of S do
  for each block br of R do
    for each tuple s in bs
      for each tuple r in br do
        if r and s join then output(r,s)

## Nested Loop Joins




## Nested Loop Joins

- Block-based Nested Loop Join
- Cost:
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-1) times, and each time need to read R: costs B(S)B(R)/(M-1)
  - Total cost: B(S) + B(S)B(R)/(M-1)
- Notice: it is better to iterate over the smaller relation first
- R ⋈ S: R=outer relation, S=inner relation

## Two-Pass Algorithms Based on Sorting

- Recall: multi-way merge sort needs only two passes !
- Assumption: $B(R) <= M^2$
- Cost for sorting: 3B(R)

## Two-Pass Algorithms Based on Sorting

Duplicate elimination $\delta(R)$

- Trivial idea: sort first, then eliminate duplicates
- Step 1: sort chunks of size M, write
  - cost 2B(R)
- Step 2: merge M-1 runs, but include each tuple only once
  - cost B(R)
- Total cost: 3B(R), Assumption: $B(R) <= M^2$

## Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{city, sum(price)}(R)$

- Same as before: sort, then compute the sum(price) for each group
- As before: compute sum(price) during the merge phase.
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

## Two-Pass Algorithms Based on Sorting

Binary operations: $R \cap S$, $R \cup S$, $R - S$

- Idea: sort R, sort S, then do the right thing
- A closer look:
  - Step 1: split R into runs of size M, then split S into runs of size M. Cost: 2B(R) + 2B(S)
  - Step 2: merge M/2 runs from R; merge M/2 runs from S; ouput a tuple on a case by cases basis
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R)+B(S) <= M^2$

## Two-Pass Algorithms Based on Sorting

Join $R \bowtie S$

- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Difficulty: many tuples in R may match many in S
  - If at least one set of tuples fits in M, we are OK
  - Otherwise need nested loop
- Total cost: 5B(R)+5B(S)
- Assumption: $B(R) <= M^2$, $B(S) <= M^2$

## Two-Pass Algorithms Based on Sorting

Join $R \bowtie S$

- If the number of tuples in R matching those in S is small (or vice versa) we can compute the join during the merge phase
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R) + B(S) <= M^2$