Lecture 20:
Query Execution

Monday, November 18, 2002

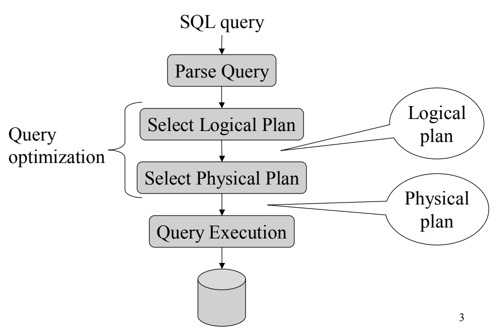
1

Outline

- Query execution: 15.1 – 15.5

2

Architecture of a Database Engine



3

An Algebra for Queries

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

4

Logical Operators in the Algebra

- Union, intersection, difference
- Selection σ
- Projection Π
- Join $\bowtie$
- Duplicate elimination δ
- Grouping γ
- Sorting τ

(Union through Join: Relational Algebra)

5

Example

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

$\Pi_{city, c}$

|

$\sigma_{p > 100}$

|

$\gamma_{city, sum(price) \rightarrow p, count(*) \rightarrow c}$

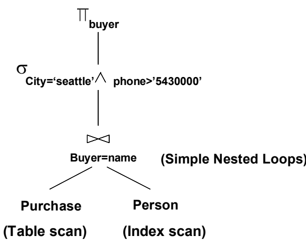
|

sales

6

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.

7

Question in Class

Logical operator:

Product(pname, cname) ⋈ Company(cname, city)

Propose three physical operators for the join, assuming the tables are in main memory:

1.
2.
3.

8

Question in Class

Product(pname, cname) ⋈ Company(cname, city)

- 1000000 products
- 1000 companies

How much time do the following physical operators take if the data is **in main memory** ?

- Nested loop join — time =
- Sort and merge = merge-join — time =
- Hash join — time =

9

Cost Parameters

In database systems the data is on *disks*, not in main memory

The *cost* of an operation = total number of I/Os

Cost parameters:

- B(R) = number of blocks for relation R
- T(R) = number of tuples in relation R
- V(R, a) = number of distinct values of attribute a

10

Cost Parameters

- *Clustered* table R:
  - Blocks consists only of records from this table
  - B(R) ≈ T(R) / blockSize
- *Unclustered* table R:
  - Its records are placed on blocks with other tables
  - When R is *unclustered*: B(R) ≈ T(R)
- When a is a key, V(R,a) = T(R)
- When a is not a key, V(R,a)

11

Cost

Cost of an operation =
number of disk I/Os needed to:
- read the operands
- compute the result

Cost of writing the result to disk is *not included* on the following slides

Question: the cost of sorting a table with B blocks ?

Answer:

12

Scanning Tables

- The table is *clustered*:
  - 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*
  - May need one read for each record

13

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 multi-way merge sort

14

Cost of the Scan Operator

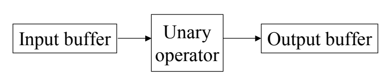
- Clustered relation:
  - Table scan:
    - Unsorted: B(R)
    - Sorted: 3B(R)
  - Index scan
    - Unsorted: B(R)
    - Sorted: B(R) or 3B(R)
- Unclustered relation
  - Unsorted: T(R)
  - Sorted: T(R) + 2B(R)

15

One-Pass Algorithms

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

- Both are <u>*tuple-at-a-time*</u> algorithms
- Cost: B(R)



16

One-pass Algorithms

Hash join: $R \bowtie S$

- Scan S, build buckets in main memory
- Then scan R and join

- Cost: B(R) + B(S)
- Assumption: B(S) <= M

17

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: $B(\delta(R)) <= M$

18

Question in Class

Grouping:
Product(name, department, quantity)
$\gamma_{department,\ sum(quantity)}$ (Product) → Answer(department, sum)

Question: how do you compute it in main memory ?
Answer:

19

One-pass Algorithms

Grouping: $\gamma_{a,\ sum(b)}$ (R)
- Need to store all a's in memory
- Also store the sum(b) for each a
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: number of cities fits in memory

20

One-pass Algorithms

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

21

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)

22

Nested Loop Joins

- We can be much more clever
- *Question*: how would you compute the join in the following cases ? What is the cost ?
  - B(R) = 1000, B(S) = 2, M = 4
  - B(R) = 1000, B(S) = 4, M = 4
  - B(R) = 1000, B(S) = 6, M = 4

23

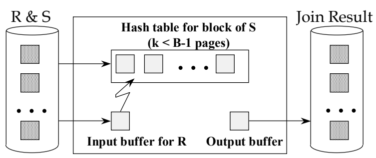
Nested Loop Joins

- 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)
```

24

Nested Loop Joins



25

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
- $R \bowtie S$: R=outer relation, S=inner relation

26

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)

27

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: $B(R) <= M^2$

28

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{a,\ sum(b)}$ (R)

- Same as before: sort, then compute the sum(b) for each group of a's
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

29

Two-Pass Algorithms Based on Sorting

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

- 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: $B(R)+B(S)<= M^2$

30

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, higher cost
- Total cost: 5B(R)+5B(S)
- Assumption: B(R) <= M^2, B(S) <= M^2

31

Two-Pass Algorithms Based on Sorting

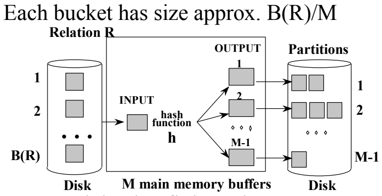
Join $R \bowtie S$

- 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

32

Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. B(R) <= M^2

33

Hash Based Algorithms for δ

- Recall: δ(R) = duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption:B(R) <= M^2

34

Hash Based Algorithms for γ

- Recall: γ(R) = grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply γ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption:B(R) <= M^2

35