Introduction to Database Systems CSE 444

Lecture 21: Query Execution

November 19-21, 2007

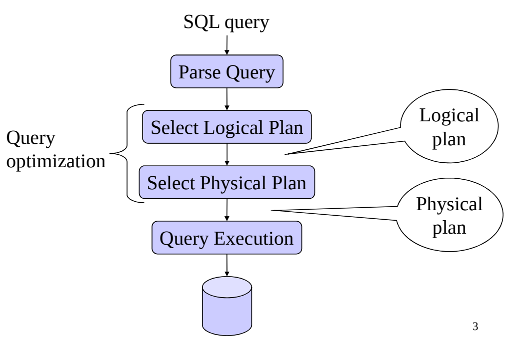
1

Outline

- Hash-tables (13.4)
- Query execution: 15.1 – 15.5

2

Architecture of a Database Engine



3

Logical Algebra Operators

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

4

Physical Operators

Will learn today and the following lectures:

- Join:
  - Main-memory hash based join
  - Block-based nested-loop join
  - Partitioned hash-based join
  - Merge-join
  - Index-join
- Group-by / Duplicate-elimination:
  - . . . .

5

Question in Class

Logical operator:

Product(pname, cname) $\bowtie$ **Company(cname, city)**

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

1.
2.
3.

6

Question in Class

Product(pname, cname) $\bowtie$ Company(cname, city)

- 1000000 products
- 1000 companies

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

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

7

Cost Parameters

The *cost* of an operation = total number of I/Os
- result assumed to be delivered in main memory

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
- M = size of main memory buffer pool, in blocks

8

Cost Parameters

- *Clustered* table R:
  - Blocks consists only of records from this table
  - B(R) << T(R)
- *Unclustered* table R:
  - Its records are placed on blocks with other tables
  - B(R) ≈ T(R)

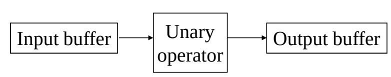
- When a is a key, V(R,a) = T(R)
- When a is not a key, V(R,a)

9

Selection and Projection

Selection σ(R), projection Π(R)

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



10

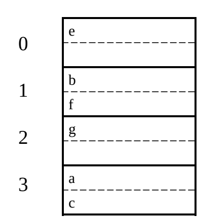
Hash Tables

- Key data structure used in many operators
- May also be used for indexes, as alternative to B+trees
- Recall basics:
  - There are n *<u>buckets</u>*
  - A hash function f(k) maps a key k to {0, 1, …, n-1}
  - Store in bucket f(k) a pointer to record with key k
- Secondary storage: bucket = block, use overflow blocks when needed

11

Hash Table Example

- Assume 1 bucket (block) stores 2 keys + pointers
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3

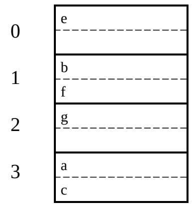


Here: h(x) = x mod 4

12

Searching in a Hash Table

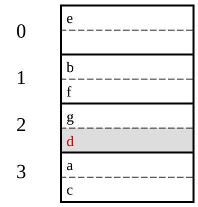
- Search for a:
- Compute h(a)=3
- Read bucket 3
- 1 disk access



13

Insertion in Hash Table

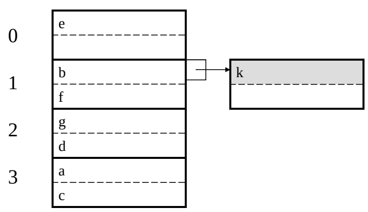
- Place in right bucket, if space
- E.g. h(d)=2



14

Insertion in Hash Table

- Create overflow block, if no space
- E.g. h(k)=1



- More over-flow blocks may be needed

15

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (I.e. many overflow blocks).

16

Main Memory Hash Join

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

Duplicate Elimination

Duplicate elimination $\delta(R)$

- Hash table in main memory

- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

18

Grouping

Grouping:

Product(name, department, quantity)

$\gamma_{\text{department, sum(quantity)}}$ (Product) → Answer(department, sum)

Main memory hash table

Question: How ?

19

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) B(S) when S is clustered
- Cost: T(R) T(S) when S is unclustered

20

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) = 3, M = 4
  - B(R) = 1000, B(S) = 6, M = 4

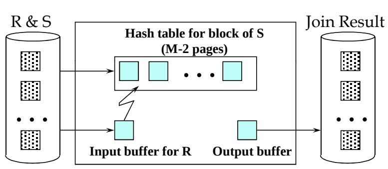
21

Block-Based Nested-loop Join

```
for each (M-2) 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)
```

22

Block-Based Nested-loop Join



23

Block-Based Nested-loop Join

- Cost:
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-2) times, and each time need to read R: costs B(S)B(R)/(M-2)
  - Total cost: B(S) + B(S)B(R)/(M-2)
- Notice: it is better to iterate over the smaller relation first
- R⋈S: R=outer relation, S=inner relation

24

Index Based Join

- $R \bowtie S$
- Assume S has an index on the join attribute

```
for each tuple r in R do
  lookup the tuple(s) s in S using the index
  output (r,s)
```

25

Index Based Join

Cost (Assuming R is clustered):

- If index is clustered: B(R) + T(R)B(S)/V(S,a)
- If index is unclustered: B(R) + T(R)T(S)/V(S,a)

26

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)
  - We have seen that this is like a join

27

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan (assuming R is clustered):
  - B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R)/V(R,a) = 100 I/Os
  - If index is unclustered: T(R)/V(R,a) = 5,000 I/Os
- Lesson: don't build unclustered indexes when V(R,a) is small !

28

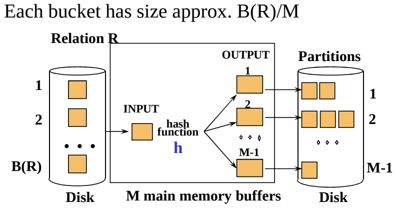
Operations on Very Large Tables

- Partitioned hash algorithms
- Merge-sort algorithms

29

Partitioned Hash Algorithms

- 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

30

Duplicate Elimination

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

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

31

Grouping

- Recall: $\gamma(R)$ = 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:$B(R) <= M^2$

32

Partitioned Hash Join

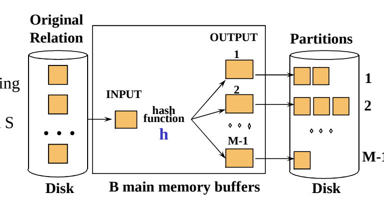
$R \bowtie S$

- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

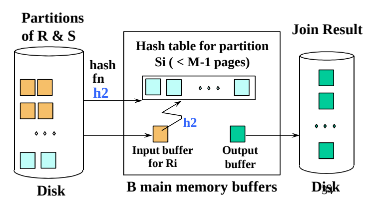
33

Hash-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.



- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.



34

Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption: $\min(B(R), B(S)) <= M^2$

35

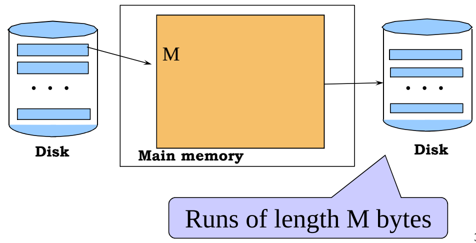
External Sorting

- Problem:
- Sort a file of size B with memory M
- Where we need this:
  - ORDER BY in SQL queries
  - Several physical operators
  - Bulk loading of B+-tree indexes.
- Will discuss only 2-pass sorting, for when $B < M^2$

36

External Merge-Sort: Step 1

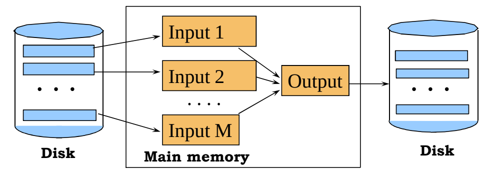
- Phase one: load M bytes in memory, sort



37

External Merge-Sort: Step 2

- Merge M – 1 runs into a new run
- Result: runs of length M (M – 1)≈ M^2



If B <= M^2 then we are done

38

Cost of External Merge Sort

- Read+write+read = 3B(R)
- Assumption: B(R) <= M^2

39

Duplicate Elimination

Duplicate elimination δ(R)

- Idea: do a two step merge sort, but change one of the steps
- Question in class: which step needs to be changed and how ?
- Cost = 3B(R)
- Assumption: B(δ(R)) <= M^2

40

Grouping

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

41

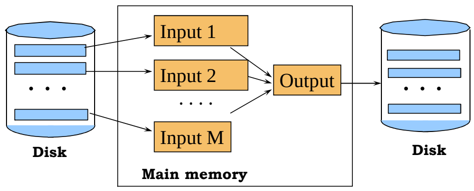
Merge-Join

Join R ⋈ S

- Step 1a: initial runs for R
- Step 1b: initial runs for S
- Step 2: merge and join

42

Merge-Join



M_1 = B(R)/M runs for R
M_2 = B(S)/M runs for S
If B <= M^2 then we are done

43

Two-Pass Algorithms Based on Sorting

Join R ⋈ 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

44

Summary of External Join Algorithms

- Block Nested Loop: B(S) + B(R)*B(S)/M
- Index Join: B(R) + T(R)B(S)/V(S,a)
- Partitioned Hash: 3B(R)+3B(S);
  - min(B(R),B(S)) <= M^2
- Merge Join: 3B(R)+3B(S
  - B(R)+B(S) <= M^2

45