CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture #14: Implementation of Relational Operations

(R&G ch. 12 and 14)

CMU SCS

Outline

- introduction
- selection
- projection
- join
- set & aggregate operations

CMU SCS

Introduction

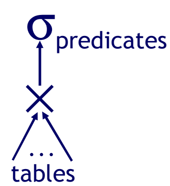
- Today's topic: QUERY PROCESSING
- Some database operations are EXPENSIVE
- Can greatly improve performance by being "smart"
  - e.g., can speed up 1,000,000x over naïve approach

Introduction (cnt'd)

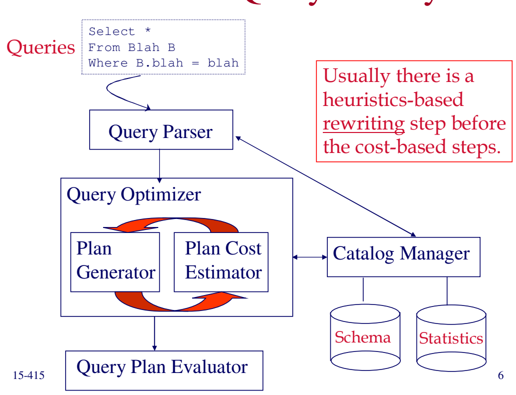
- Main weapons are:
  - clever implementation techniques for operators
  - exploiting "equivalencies" of relational operators
  - using statistics and cost models to choose among these.

A Really Bad Query Optimizer

- For each Select-From-Where query block
  - do cartesian products first
  - then do selections
  - etc, ie.:
    - GROUP BY; HAVING
    - projections
    - ORDER BY
- Incredibly inefficient
  - Huge intermediate results!



Cost-based Query Sub-System



CMU SCS

The Query Optimization Game

- "Optimizer" is a bit of a misnomer…
- Goal is to pick a "good" (i.e., low expected cost) plan.
  - Involves choosing access methods, physical operators, operator orders, …
  - Notion of cost is based on an abstract "cost model"

15-415 Faloutsos 7

CMU SCS

Relational Operations

- We will consider how to implement:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Join* ($\bowtie$) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
  - *Aggregation* (SUM, MIN, etc.) and GROUP BY
- Recall: ops can be *composed* !
- Later, we'll see how to *optimize* queries with many ops

15-415 Faloutsos 8

CMU SCS

Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- Similar to old schema; *rname* added for variations.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - N=500, p_S=80.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - M=1000, p_R=100.

15-415 Faloutsos 9

CMU SCS

Outline

- introduction
- selection
- projection
- join
- set & aggregate operations

CMU SCS

Simple Selections

- Of the form $\sigma_{R.attr\ op\ value}(R)$
- Question: how best to perform?

```
SELECT  *
FROM    Reserves R
WHERE   R.rname < 'C%'
```


CMU SCS

Simple Selections

- A: Depends on:
  - what indexes/access paths are available
  - what is the expected size of the result (in terms of number of tuples and/or number of pages)

CMU SCS

Simple Selections

- Size of result approximated as

  *size of R * reduction factor*

  – "reduction factor" is also called *selectivity*.

  – estimate of reduction factors is based on statistics – we will discuss shortly.

15-415 Faloutsos 13

CMU SCS

Alternatives for Simple Selections

- With no index, unsorted:
  – Must essentially scan the whole relation
  – cost is M (#pages in R). For "reserves" = 1000 I/Os.

15-415 Faloutsos 14

CMU SCS

Simple Selections (cnt'd)

- With no index, sorted:
  – cost of binary search + number of pages containing results.
  – For reserves = 10 I/Os + $\lceil$selectivity*#pages$\rceil$

15-415 Faloutsos 15

CMU SCS

Simple Selections (cnt'd)

- With an index on selection attribute:
  - – Use index to find qualifying data entries,
  - – then retrieve corresponding data records.
  - – (Hash index useful only for equality selections.)

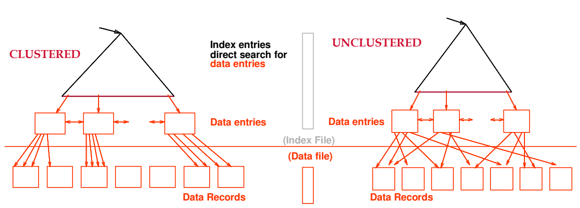
CMU SCS

Using an Index for Selections

- Cost depends on #qualifying tuples, and clustering.
  - – Cost:
    - finding qualifying data entries (typically small)
    - plus cost of retrieving records (could be large w/o clustering).

CMU SCS

Selections using Index (cnt'd)



CMU SCS

Selections using Index (cnt'd)

- In example "reserves" relation, if 10% of tuples qualify (100 pages, 10,000 tuples).
  - With a *clustered* index, cost is little more than 100 I/Os;
  - if *unclustered*, could be up to 10,000 I/Os! unless…

15-415 Faloutsos 19

CMU SCS

Selections using Index (cnt'd)

- *Important refinement for unclustered indexes*:
  1. Find qualifying data entries.
  2. Sort the rid's of the data records to be retrieved.
  3. Fetch rids in order. This ensures that each data page is looked at just once (though # of such pages likely to be higher than with clustering).

15-415 Faloutsos 20

CMU SCS

General Selection Conditions

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- Q: What would you do?

15-415 Faloutsos 21

CMU SCS

General Selection Conditions

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- Q: What would you do?
- A: try to find a selective (clustering) index. Specifically:

15-415 Faloutsos 22

CMU SCS

General Selection Conditions

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- Convert to *conjunctive normal form* (CNF):
  - *(day<8/9/94* OR *bid=5* OR *sid=3)* AND *(rname='Paul'* OR *bid=5* OR *sid=3)*
- We only discuss the case with no ORs (a conjunction of *terms* of the form *attr **op** value*).

15-415 Faloutsos 23

CMU SCS

General Selection Conditions

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- A B-tree index *matches* (a conjunction of) terms that involve only attributes in a *prefix* of the search key.
  - Index on *<a, b, c>* matches *a=5* AND *b= 3*, but not *b=3*.
- For Hash index, must have all attributes in search key

15-415 Faloutsos 24

CMU SCS

Two Approaches to General Selections

- First approach: Find the *cheapest access path*, retrieve tuples using it, and apply any remaining terms that don't match the index
- Second approach:get rids from first index; rids from second index; intersect and fetch.

15-415 Faloutsos 25

CMU SCS

Two Approaches to General Selections

- First approach: Find the *cheapest access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - *Cheapest access path:* An index or file scan with fewest I/Os.
  - Terms that match this index reduce the number of tuples *retrieved*; other terms help discard some retrieved tuples, but do not affect number of tuples/pages fetched.

15-415 Faloutsos 26

CMU SCS

Cheapest Access Path - Example

- Consider *day < 8/9/94 AND bid=5 AND sid=3*.
- A B+ tree index on *day* can be used;
  - then, *bid=5* and *sid=3* must be checked for each retrieved tuple.
- Similarly, a hash index on <*bid, sid*> could be used;
  - *Then, day<8/9/94* must be checked.

15-415 Faloutsos 27

CMU SCS

Cheapest Access Path - cnt'd

- Consider *day < 8/9/94 AND bid=5 AND sid=3*.
- *How about a B+tree on <rname,day>?*
- *How about a B+tree on <day, rname>?*
- *How about a Hash index on <day, rname>?*

15-415 Faloutsos 28

CMU SCS

Intersection of RIDs

- Second approach: if we have 2 or more matching indexes (w/Alternatives (2) or (3) for data entries):
  - Get sets of rids of data records using each matching index.
  - Then *intersect* these sets of rids.
  - Retrieve the records and apply any remaining terms.

15-415 Faloutsos 29

CMU SCS

Intersection of RIDs (cnt'd)

- EXAMPLE: Consider *day<8/9/94 AND bid=5 AND sid=3*.
- With a B+ tree index on *day* and an index on *sid*,
- we can retrieve rids of records satisfying *day<8/9/94* using the first,
- rids of recs satisfying *sid=3* using the second,
- intersect,
- retrieve records and check *bid=5.*

15-415 Faloutsos 30

CMU SCS

Outline

- introduction
- selection
- ➡ projection
- join
- set & aggregate operations

CMU SCS

The Projection Operation

```
SELECT DISTINCT
       R.sid, R.bid
FROM   Reserves R
```

- Issue is removing duplicates.
- Basic approach: sorting
  - 1. Scan R, extract only the needed attrs (why?)
  - 2. Sort the resulting set
  - 3. Remove adjacent duplicates

  Cost: Reserves with size ratio 0.25 = 250 pages. With 20 buffer pages can sort in 2 passes, so 1000 +250 + 2 * 2 * 250 + 250 = 2500 I/Os

CMU SCS

Projection

- Can improve by modifying external sort algorithm (see chapter 13):
  - Modify Pass 0 of external sort to eliminate unwanted fields.
  - Modify merging passes to eliminate duplicates.

  Cost: for above case: read 1000 pages, write out 250 in runs of 40 pages, merge runs = 1000 + 250 +250 = 1500.

CMU SCS

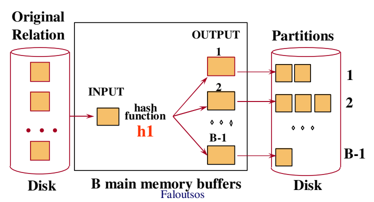
Projection with Hashing

- we saw it earlier:
- Phase1: partition
- Phase2: reHash (now: dup. elim.)

CMU SCS

Projection Based on Hashing

- *Partitioning phase*: Read R using one input buffer. For each tuple, discard unwanted fields, apply hash function *h1* to choose one of B-1 output buffers.



CMU SCS

DupElim with Hashing (cnt'd)

- *Duplicate elimination phase*: For each partition, read it and build an in-memory hash table, using hash fn *h2* (<> *h1*) on all fields, while discarding duplicates.
  - If partition does not fit in memory, can apply hash-based projection algorithm recursively to this partition.

CMU SCS

DupElim with Hashing (cnt'd)

- Cost:
  - assuming partitions fit in memory (i.e. #bufs >= square root of the #of pages of projected tuples)
  - read 1000 pages and write out partitions of projected tuples (250 pages)
  - Do dup elim on each partition (total 250 page reads)
  - Total : 1500 I/Os.

15-415 Faloutsos 37

CMU SCS

Discussion of Projection

- Sort-based approach: better handling of skew and result is sorted.
- If enough buffers, both have same I/O cost: M + 2T where M is #pgs in R, T is #pgs of R with unneeded attributes removed.
  - Although many systems don't use the specialized sort.

15-415 Faloutsos 38

CMU SCS

Discussion of Projection

- If an index on the relation contains all wanted attributes in its search key, can do *index-only* scan.
  - Apply projection techniques to data entries (much smaller!)

15-415 Faloutsos 39

Discussion of Projection

- If an ordered (i.e., tree) index contains all wanted attributes as *prefix* of search key, can do even better:
  - Retrieve data entries in order (index-only scan), discard unwanted fields, compare adjacent tuples to check for duplicates.

Outline

- introduction
- selection
- projection
- join
- set & aggregate operations

Joins

- Joins are very common.
- Joins can be very expensive (cross product in worst case).
- Many approaches to reduce join cost.

CMU SCS

Joins

- Join techniques we will cover:
  - Nested-loops join
  - Index-nested loops join
  - Sort-merge join
  - Hash join

CMU SCS

Equality Joins With One Join Column

```
SELECT  *
FROM    Reserves R1, Sailors S1
WHERE   R1.sid=S1.sid
```

- In algebra: $R \bowtie S$. Common! Must be carefully optimized. $R \times S$ is large; so, $R \times S$ followed by a selection is inefficient.
- Remember, join is associative and commutative.

CMU SCS

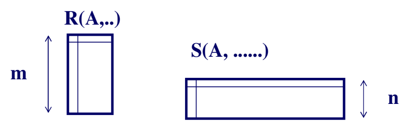
Equality Joins

- Assume:
  - M pages in R, p_R tuples per page, *m* tuples total
  - N pages in S, p_S tuples per page, *n* tuples total
  - In our examples, R is Reserves and S is Sailors.
- We will consider more complex join conditions later.
- *Cost metric*: # of I/Os. We will ignore output costs.

CMU SCS

Nested loops

- Algorithm #0: (naive) nested loop (**SLOW**!)



CMU SCS

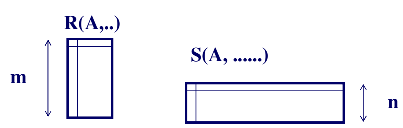
Nested loops

- Algorithm #0: (naive) nested loop (**SLOW**!)

  for each tuple r of R

  for each tuple s of S

  print, if they match



CMU SCS

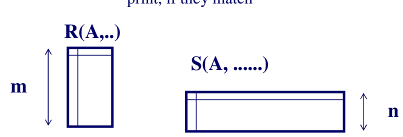
Nested loops

- Algorithm #0: (naive) nested loop (**SLOW**!)

  for each tuple r of R — outer relation

  for each tuple s of S — inner relation

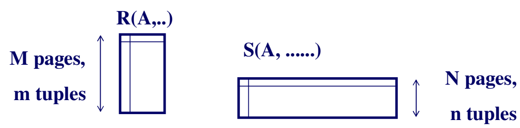
  print, if they match



CMU SCS

Nested loops

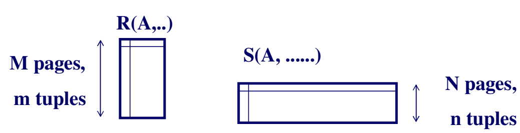
- Algorithm #0: why is it bad?
- how many disk accesses ('M' and 'N' are the number of blocks for 'R' and 'S')?



CMU SCS

Nested loops

- Algorithm #0: why is it bad?
- how many disk accesses ('M' and 'N' are the number of blocks for 'R' and 'S')? M+m*N



CMU SCS

Simple Nested Loops Join

- Actual number
- (p_R * M) * N + M = 100*1000*500 + 1000 I/Os.
  - At 10ms/IO, Total: ???
- What if smaller relation (S) was outer?
- What assumptions are being made here?

CMU SCS

Simple Nested Loops Join

- Actual number
- $(p_R * M) * N + M = 100*1000*500 + 1000$ I/Os.
  - At 10ms/IO, Total: ~6days (!)
- What if smaller relation (S) was outer?
  - slightly better
- What assumptions are being made here?
  - 1 buffer for each table (and 1 for output)

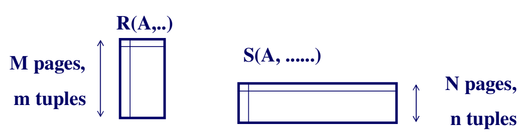
15-415 Faloutsos 52

CMU SCS

Nested loops

- Algorithm #1: Blocked nested-loop join
  - read in a block of R
    - read in a block of S
      - print matching tuples

COST?



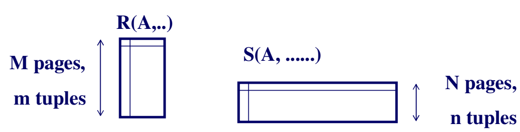
15-415 Faloutsos 53

CMU SCS

Nested loops

- Algorithm #1: Blocked nested-loop join
  - read in a block of R
    - read in a block of S
      - print matching tuples

COST= M+M*N

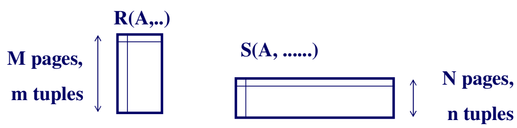


15-415 Faloutsos 54

Nested loops

- Which one should be the outer relation?

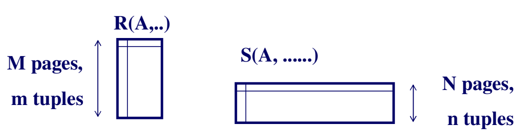
COST= M+M*N



Nested loops

- Which one should be the outer relation?
- A: the smallest (page-wise)

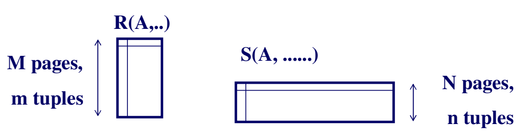
COST= M+M*N



Nested loops

- M=1000, N=500
- Cost = 1000 + 1000*500 = 501,000
- = 5010 sec ~ 1.4h

COST= M+M*N

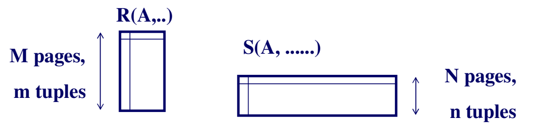


CMU SCS

Nested loops

- M=1000, N=500 - if smaller is outer:
- Cost = 500 + 1000*500 = 500,500
- = 5005 sec ~ 1.4h

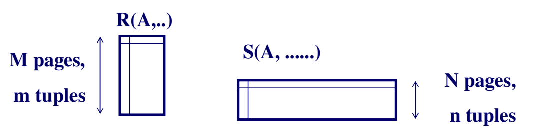
COST= N+M*N



CMU SCS

Nested loops

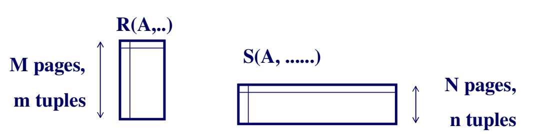
- What if we have B buffers available?



CMU SCS

Nested loops

- What if we have B buffers available?
- A: give *B-2* buffers to outer, 1 to inner, 1 for output

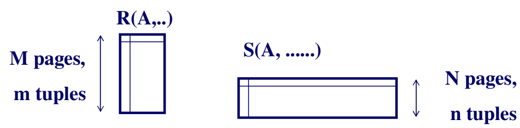


CMU SCS

Nested loops

- Algorithm #1: Blocked nested-loop join
  - read in *B-2* blocks of R
    - read in a block of S
      - print matching tuples

COST= ?



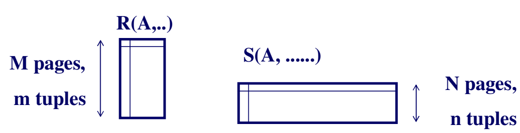
15-415 Faloutsos 61

CMU SCS

Nested loops

- Algorithm #1: Blocked nested-loop join
  - read in *B-2* blocks of R
    - read in a block of S
      - print matching tuples

COST= M+M/(B-2)*N



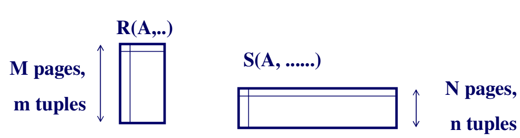
15-415 Faloutsos 62

CMU SCS

Nested loops

- and, actually:
- Cost = M + ceiling(M/(B-2)) * N

COST= M+M/(B-2)*N



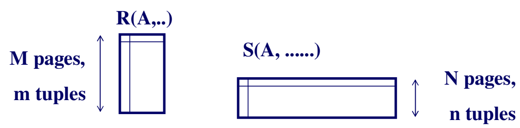
15-415 Faloutsos 63

CMU SCS

Nested loops

- If smallest (outer) fits in memory
- (ie., B= N+2),
- Cost =?

COST= N+N/(B-2)*M



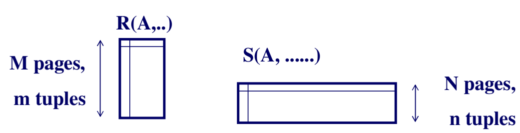
15-415 Faloutsos 64

CMU SCS

Nested loops

- If smallest (outer) fits in memory
- (ie., B= N+2),
- Cost =N+M (minimum!)

COST= N+N/(B-2)*M



15-415 Faloutsos 65

CMU SCS

Nested loops - guidelines

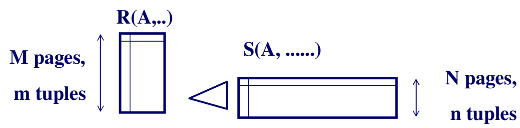
- pick as outer the smallest table (= fewest pages)
- fit as much of it in memory as possible
- loop over the inner

15-415 Faloutsos 66

CMU SCS

Index NL join

- use an existing **index**, or even build one on the fly
- cost: M + m * c (c: look-up cost)

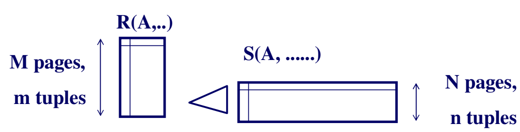


15-415 Faloutsos 67

CMU SCS

Index NL join

- cost: M + m * c (c: look-up cost)
- ‘c’ depends whether the index is clustered or not.



15-415 Faloutsos 68

CMU SCS

Joins

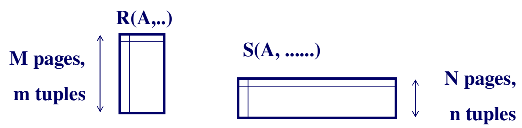
- Join techniques we will cover:
  - Nested-loops join
  - Index-nested loops join
  - Sort-merge join
  - Hash join

15-415 Faloutsos 69

CMU SCS

Sort-merge join

- sort both on joining attributed
- scan each and merge
- Cost, given B buffers?

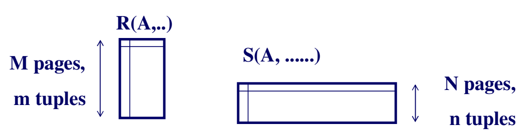


15-415 Faloutsos 70

CMU SCS

Sort-merge join

- Cost, given B buffers?
- ~ 2*M*logM/logB + 2*N* logN/logB + M + N



15-415 Faloutsos 71

CMU SCS

Sort-Merge Join

- Useful if

15-415 Faloutsos 72

CMU SCS

Sort-Merge Join

- Useful if
  - one or both inputs are already sorted on join attribute(s)
  - output is required to be sorted on join attributes(s)
- "Merge" phase can require some back tracking if duplicate values appear in join column

CMU SCS

Example of Sort-Merge Join

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

CMU SCS

Example of Sort-Merge Join

- With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500.
- (while *BNL cost: 2,500 to 15,000 I/Os*)

CMU SCS

Sort-merge join

- Worst case for merging phase?
- Cost?

CMU SCS

Refinements

- All the refinements of external sorting
- plus overlapping of the merging of sorting with the merging of joining.

CMU SCS

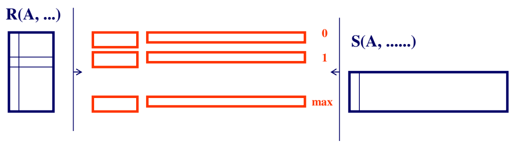
Joins

- Join techniques we will cover:
  - Nested-loops join
  - Index-nested loops join
  - Sort-merge join
  - Hash join

CMU SCS

Hash joins

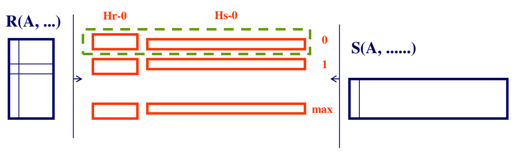
- hash join: use hashing function h()
  - hash ‘R’ into (0, 1, ..., ‘max’) buckets
  - hash ‘S’ into buckets (same hash function)
  - join each pair of matching buckets



CMU SCS

Hash join - details

- how to join each pair of partitions Hr-i, Hs-i ?
- A: build another hash table for Hs-i, and probe it with each tuple of Hr-i



CMU SCS

Hash join - details

- In more detail:
- Choose the (page-wise) smallest - if it fits in memory, do ~NL
  - and, actually, build a hash table (with h2() != h())
  - and probe it, with each tuple of the other

CMU SCS

Hash join details

- what if Hs-i is too large to fit in main-memory?
- A: recursive partitioning
- more details (overflows, hybrid hash joins): in book
- cost of hash join? (if we have enough buffers:)
  3(M + N) (why?)

15-415 Faloutsos 82

CMU SCS

Cost of Hash-Join

- In partitioning phase, read+write both relns; 2(M+N). In matching phase, read both relns; M+N I/Os.
- In our running example, this is a total of 4500 I/Os.

15-415 Faloutsos 83

CMU SCS

Hash join details

- [cost of hash join? (if we have enough buffers:)
  3(M + N)
  ]
- What is ‘enough’? sqrt(N), or sqrt(M)?

15-415 Faloutsos 84

CMU SCS

Hash join details

- [cost of hash join? (if we have enough buffers:)
  3(M + N)
  ]
- What is 'enough'? sqrt(N), or sqrt(M)?
- A: sqrt(smallest) (why?)

CMU SCS

Sort-Merge Join vs. Hash Join

- Given a minimum amount of memory (*what is this, for each?*) both have a cost of 3(M+N) I/Os.

CMU SCS

Sort-Merge vs Hash join

- Hash Join Pros:
  - ??
  - ??
  - ??
- Sort-Merge Join Pros:
  - ??

CMU SCS

Sort-Merge vs Hash join

- Hash Join Pros:
  - Superior if relation sizes differ greatly
  - Shown to be highly parallelizable *(beyond scope of class)*
- Sort-Merge Join Pros:
  - ??

15-415 Faloutsos 88

CMU SCS

Sort-Merge vs Hash join

- Hash Join Pros:
  - Superior if relation sizes differ greatly
  - Shown to be highly parallelizable *(beyond scope of class)*
- Sort-Merge Join Pros:
  - Less sensitive to data skew
  - Result is sorted (may help “upstream” operators)
  - goes faster if one or both inputs already sorted

15-415 Faloutsos 89

CMU SCS

General Join Conditions

- Equalities over several attributes (e.g., *R.sid=S.sid* AND *R.rname=S.sname*):
  - all previous methods apply, using the composite key

15-415 Faloutsos 90

CMU SCS

General Join Conditions

- Inequality conditions (e.g., *R.rname < S.sname*):
- which methods still apply?
  - NL
  - index NL
  - Sort merge
  - Hash join

15-415 Faloutsos 91

CMU SCS

General Join Conditions

- Inequality conditions (e.g., *R.rname < S.sname*):
- which methods still apply?
  - NL (probably, the best!)
  - index NL (only if clustered index)
  - Sort merge (does not apply!) (why?)
  - Hash join (does not apply!) (why?)

15-415 Faloutsos 92

CMU SCS

Outline

- introduction
- selection
- projection
- join
- set & aggregate operations

15-415 Faloutsos 93

CMU SCS

Set Operations

- Intersection and cross-product: special cases of join
- Union (Distinct) and Except: similar; we'll do union:
- Effectively: concatenate; use sorting or hashing
- Sorting based approach to union:
  - Sort both relations (on combination of all attributes).
  - Scan sorted relations and merge them.
  - *Alternative*: Merge runs from Pass 0 for *both* relations.

15-415 Faloutsos 94

CMU SCS

Set Operations, cont'd

- Hash based approach to union:
  - Partition R and S using hash function *h*.
  - For each S-partition, build in-memory hash table (using *h2*), scan corresponding R-partition and add tuples to table while discarding duplicates.

15-415 Faloutsos 95

CMU SCS

Aggregate Operations (AVG, MIN, etc.)

- Without grouping:
  - In general, requires scanning the relation.
  - Given index whose search key includes all attributes in the SELECT or WHERE clauses, can do index-only scan.

15-415 Faloutsos 96

CMU SCS

Aggregate Operations (AVG, MIN, etc.)

- With grouping:
  - Sort on group-by attributes, then scan relation and compute aggregate for each group. (Can improve upon this by combining sorting and aggregate computation.)
  - Hashing: similarly.
  - Given tree index whose search key includes all attributes in SELECT, WHERE and GROUP BY clauses, can do index-only scan; if group-by attributes form prefix of search key, can retrieve data entries/tuples in group-by order.

15-415 Faloutsos 97

CMU SCS

Impact of Buffering

- If several operations are executing concurrently, estimating the number of available buffer pages is guesswork.
- Repeated access patterns interact with buffer replacement policy.
  - e.g., Inner relation is scanned repeatedly in Simple Nested Loop Join. With enough buffer pages to hold inner, replacement policy does not matter. Otherwise, MRU is best, LRU is worst (*sequential flooding*).
  - Does replacement policy matter for Block Nested Loops?
  - What about Index Nested Loops?

15-415 Faloutsos 98

CMU SCS

Summary

- A virtue of relational DBMSs:
  - queries are composed of **a few basic operators**
  - The implementation of these operators can be **carefully tuned**
  - **Important** to do this!
- Many alternative implementation techniques for each operator
  - No universally superior technique for most operators.

"it depends" [Guy Lohman (IBM)]

15-415 Faloutsos 99

CMU SCS

Summary cont'd

- Must consider available alternatives for each operation in a query and choose best one based on system statistics, etc.
  - Part of the broader task of optimizing a query composed of several ops.