CMU SCS

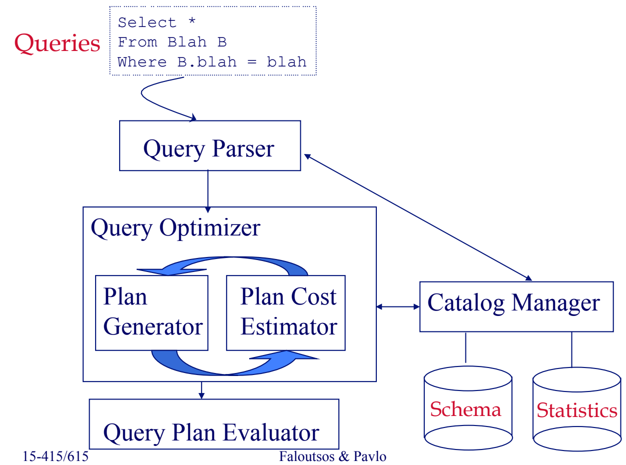
Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 – DB Applications

Lecture #13: Query Evaluation

(R&G ch. 12 and 14)

CMU SCS

Cost-based Query Sub-System



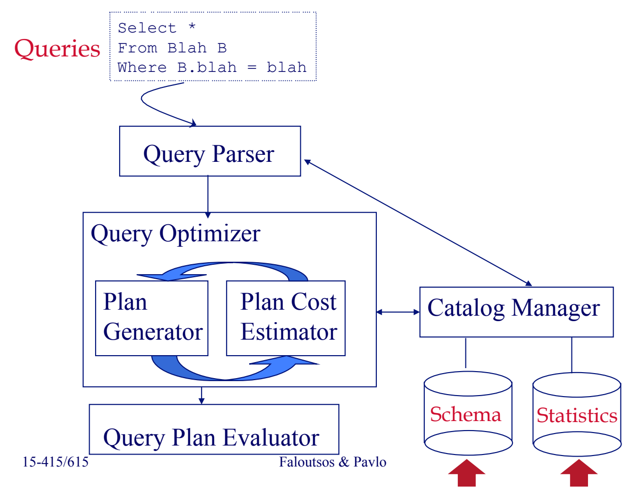
CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

CMU SCS

Cost-based Query Sub-System



CMU SCS

Schema

- What would you store?
- How?

CMU SCS

Schema

- What would you store?
- A: info about tables, attributes, indices, users
- How?
- A: in tables! eg.,
  - Attribute_Cat (attr_name: **string**, rel_name: **string**; type: **string**; position: **integer**)

CMU SCS

Statistics

- Why do we need them?
- What would you store?

CMU SCS

Statistics

- Why do we need them?
- A: To estimate cost of query plans
- What would you store?
  - NTuples(R): # records for table R
  - NPages(R): # pages for R
  - NKeys(I): # distinct key values for index I
  - INPages(I): # pages for index I
  - IHeight(I): # levels for I
  - ILow(I), IHigh(I): range of values for I
  - ...

CMU SCS

Outline

- (12.1) Catalog
- ➡ (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

15-415/615 Faloutsos & Pavlo 9

CMU SCS

Operator evaluation

3 methods we'll see often:

15-415/615 Faloutsos & Pavlo 10

CMU SCS

Operator evaluation

3 methods we'll see often:
- indexing
- iteration (= seq. scanning)
- partitioning (sorting and hashing)

15-415/615 Faloutsos & Pavlo 11

CMU SCS

``Access Path''

- Eg., index (tree, or hash), or scanning
- Selectivity of an access path:
  - % of pages we retrieve
- eg., selectivity of a hash index, on range query: 100% (no reduction!)

15-415/615 Faloutsos & Pavlo 12

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- ➡ (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

15-415/615 Faloutsos & Pavlo 13

CMU SCS

Algorithms

- selection:
- projection
- join
- group by
- order by

15-415/615 Faloutsos & Pavlo 14

CMU SCS

Algorithms

- selection: scan; index
- projection (dup. elim.):
- join
- group by
- order by

15-415/615 Faloutsos & Pavlo 15

CMU SCS

Algorithms

- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join
- group by
- order by

15-415/615 Faloutsos & Pavlo 16

CMU SCS

Algorithms

- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join: many ways (loops, sort-merge, etc)
- group by
- order by

CMU SCS

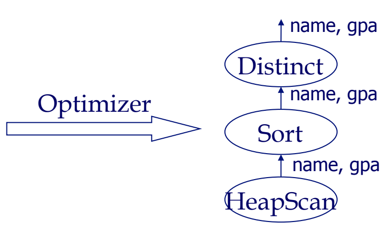
Algorithms

- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join: many ways (loops, sort-merge, etc)
- group by: hashing; sorting
- order by: sorting

CMU SCS

Iterator Interface

SELECT DISTINCT name, gpa
FROM Students



CMU SCS

iterator

Iterators

- Relational operators: subclasses of **iterator**:

```
class iterator {
     void init();
     tuple next();
     void close();
     iterator &inputs[];
   // additional state goes here
  }
```

- iterators can be cascaded

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- ➡ (12.6) Typical Q-optimizer
- (14.3.2) Hashing

CMU SCS

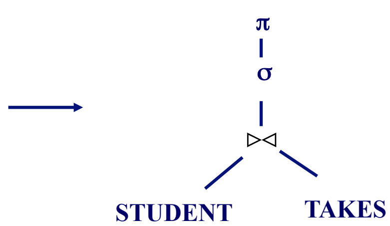
Q-opt steps

- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best

CMU SCS

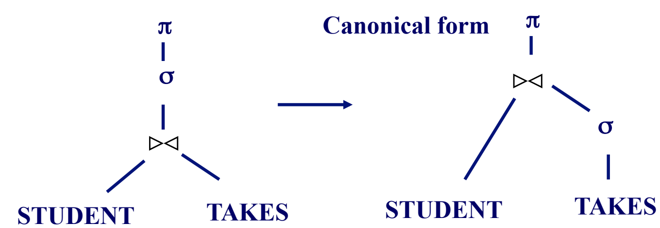
Q-opt - example

select name
from STUDENT, TAKES
where c-id='415' and
STUDENT.ssn=TAKES.ssn



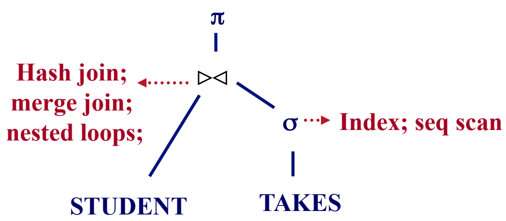
CMU SCS

Q-opt - example



CMU SCS

Q-opt - example



15-415/615 Faloutsos & Pavlo 25

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- ➡ (14.3.2) Hashing

15-415/615 Faloutsos & Pavlo 26

CMU SCS

Grouping; Duplicate Elimination

select distinct ssn
from TAKES

- (Q1: what does it do, in English?)
- Q2: how to execute it?

15-415/615 Faloutsos & Pavlo 27

CMU SCS

An Alternative to Sorting: Hashing!

- Idea:
  - maybe we don't need the *order* of the sorted data
  - e.g.: forming groups in GROUP BY
  - e.g.: removing duplicates in DISTINCT
- Hashing does this!
  - And may be cheaper than sorting! (why?)
  - But what if table doesn't fit in memory??

15-415/615 Faloutsos & Pavlo 28

CMU SCS

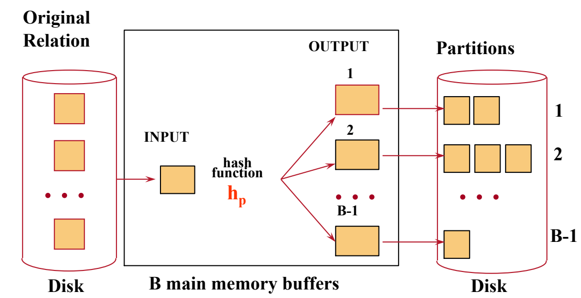
General Idea

- Two phases:
  - Phase1: Partition: use a hash function h_p to split tuples into partitions on disk.
    - We know that all matches live in the same partition.
    - Partitions are “spilled” to disk via output buffers

CMU SCS

Two Phases

- Partition:



CMU SCS

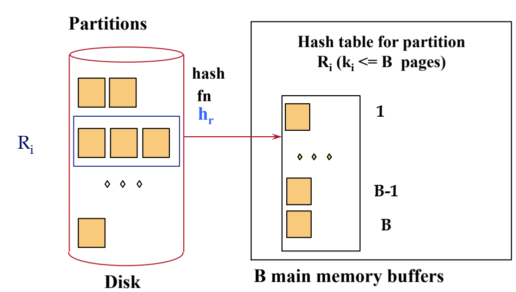
General Idea

- Two phases:
  - Phase 2: ReHash: for each partition on disk
    - (assuming it fits in memory)
    - read it into memory and build a main-memory hash table based on a hash function h_r
    - Then go through each bucket of this hash table to bring together matching tuples

CMU SCS

Two Phases

- Rehash:



CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 "spill partitions" in Phase 1
  - Each should be no more than B blocks big

CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 "spill partitions" in Phase 1
  - Each should be no more than B blocks big
  - Answer: B(B-1).
    - ie., a table of N blocks needs about sqrt(N) buffers
  - What assumption do we make?

CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 "spill partitions" in Phase 1
  - Each should be no more than B blocks big
  - Answer: B(B-1).
    - ie., a table of N blocks needs about sqrt(N) buffers
  - Note: assumes hash function distributes records evenly!
    - use a 'fudge factor' f>1 for that: we need B ~ sqrt(f * N)

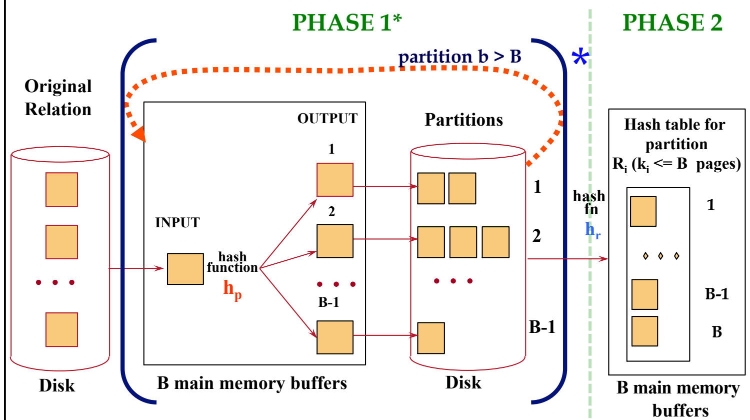
CMU SCS

Analysis

- Have a bigger table? Recursive partitioning!
  - In the ReHash phase, if a partition *b* is bigger than B, then recurse:
  - pretend that *b* is a table we need to hash, run the Partitioning phase on *b*, and then the ReHash phase on each of its (sub)partitions

CMU SCS

Recursive partitioning



CMU SCS

break

Real story

- Partition + Rehash
- Performance is very slow!
- What could have gone wrong?

CMU SCS

break

Real story

- Partition + Rehash
- Performance is very slow!
- What could have gone wrong?
- Hint: some buckets are empty; some others are way over-full.

CMU SCS

Hashing vs. Sorting

- Which one needs more buffers?

CMU SCS

Hashing vs. Sorting

- **<u>Recall</u>: can hash a table of size N blocks in sqrt(N) space**
- How big of a table can we sort in 2 passes?
  - Get N/B sorted runs after Pass 0
  - Can merge all runs in Pass 1 if $N/B \le B-1$
    - Thus, we (roughly) require: $N \le B^2$
    - We can sort a table of size N blocks in about space sqrt(N)
  - <u>Same as hashing!</u>

CMU SCS

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case …
  - Already discussed some optimizations for sorting:

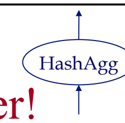
CMU SCS

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case …
  - Already discussed some optimizations for sorting:
    - (Heapsort in Pass 0 for longer runs)
    - Chunk I/O into large blocks to amortize seek+RD costs
    - Double-buffering to overlap CPU and I/O
  - Another optimization when using sorting for aggregation:
    - "Early aggregation" of records in sorted runs
  - We will discuss some optimizations for hashing next…

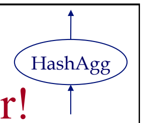
CMU SCS

Hashing: We Can Do Better!



- Combine the summarization into the hashing process - How?

CMU SCS



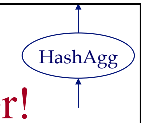
Hashing: We Can Do Better!

- Combine the summarization into the hashing process - How?
  - During the ReHash phase, don't store tuples, store pairs of the form <GroupVals, RunningVals>
  - When we want to insert a new tuple into the hash table
    - If we find a matching GroupVals, just update the RunningVals appropriately
    - Else insert a new <GroupVals, RunningVals> pair

select ssn, **sum**(credits)
from takes
group by ssn

(groupVal, runningVal)
(12345, 12)
(45678, 18)

CMU SCS



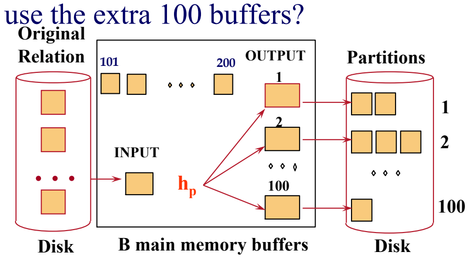
Hashing: We Can Do Better!

- Combine the summarization into the hashing process
- What's the benefit?
  - Q: How many entries will we have to handle?
  - A: Number of distinct values of GroupVals columns
    - **Not** the number of tuples!!
  - Also probably "narrower" than the tuples

CMU SCS

Even Better: Hybrid Hashing

- What if B > sqrt(N)?
- e.g., N=10,000, B=200
- B=100 (actually, 101) would be enough for 2 passes
- How could we use the extra 100 buffers?

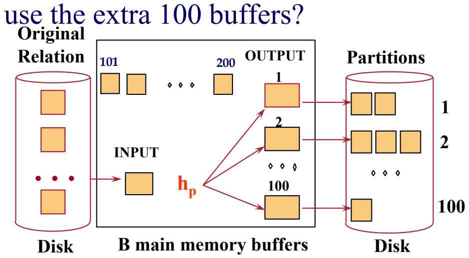


CMU SCS

Even Better: Hybrid Hashing

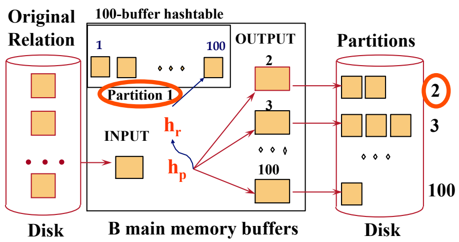
- What if B > sqrt(N)?
- e.g., N=10,000, B=200
- B=100 (actually, 101) would be enough for 2 passes
- How could we use the extra 100 buffers?

A: 1ph for first partition;
2 for all others



Even Better: Hybrid Hashing

- Idea: hybrid! … keep 1st partition in memory during phase 1!
  - Output its stuff at the end of Phase 1.



Even Better: Hybrid Hashing

- What if B=**300**? (and N=10,000, again)
- i.e., 200 extra buffers?

Even Better: Hybrid Hashing

- What if B=**300**? (and N=10,000, again)
- i.e., 200 extra buffers?
- A: keep the first **2** partitions in main memory

Even Better: Hybrid Hashing

- What if B=**150**? (and N=10,000, again)
- i.e., 50 extra buffers?

CMU SCS

Even Better: Hybrid Hashing

- What if B=**150**? (and N=10,000, again)
- i.e., 50 extra buffers?
- A: keep half of the first bucket in memory

15-415/615 Faloutsos & Pavlo 53

CMU SCS

Hybrid hashing

- can be used together with the summarization idea

15-415/615 Faloutsos & Pavlo 54

CMU SCS

So, hashing's better … right?

- Any caveats?

15-415/615 Faloutsos & Pavlo 55

CMU SCS

So, hashing's better … right?

- Any caveats?
- A1: sorting is better on non-uniform data
- A2: ... and when sorted output is required later.

Hashing vs. sorting:

- Commercial systems use either or both

15-415/615 Faloutsos & Pavlo 56

CMU SCS

Summary

- Query processing architecture:
  - Query optimizer translates SQL to a query plan = graph of iterators
  - Query executor "interprets" the plan
- Hashing is a useful alternative to sorting for dup. elim / group-by
  - Both are valuable techniques for a DBMS