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

C. Faloutsos – A. Pavlo

Lecture#13: Query Evaluation

Today's Class

- Intro to Operator Evaluation
- Typical Query Optimizer
- Projection/Aggregation: Sort vs. Hash

Example Database

STUDENT

sid	name	login	age	gpa
53666	Kayne	kayne@cs	39	4.0
53688	Bieber	jbieber@cs	22	3.9
53655	Tupac	shakur@cs	26	3.5

ENROLLED

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	B
53666	15-721	C

COURSE

cid	name
15-415	Database Applications
15-721	Database Systems
15-826	Data Mining
15-823	Advanced Topics in Databases

Query Plan Example

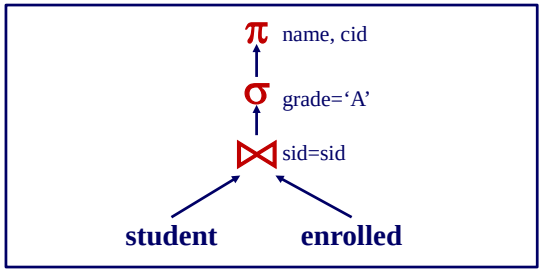
```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```

Relational Algebra:

$$\pi_{name,cid}\,(\sigma_{grade='A'}\,(student \bowtie enrolled))$$

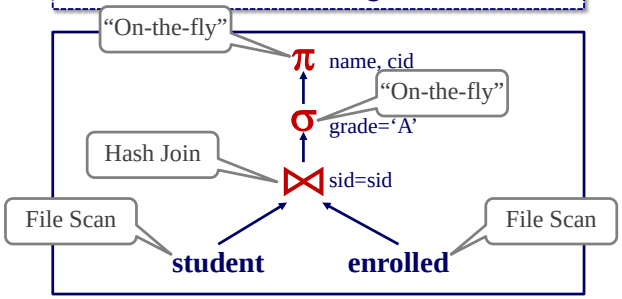
Query Plan Example

```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```



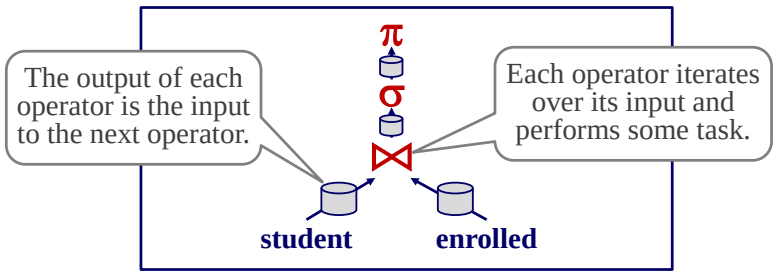
Query Plan Example

```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```



Query Plan Example

```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```



Operator Evaluation

- Several algorithms are available for different relational operators.
- Each has its own performance trade-offs.
- The goal of the query optimizer is to choose the one that has the lowest "cost".

After Midterm: *How the DBMS finds the best plan.*

Operator Execution Strategies

- Indexing
- Iteration (= seq. scanning)
- Partitioning (sorting and hashing)

Operator Algorithms

- **Selection:** file scan; index scan
- **Projection:** hashing; sorting
- **Join:** looping; hashing; sorting
- **Group By:** hashing; sorting
- **Order By:** sorting

Operator Algorithms

Next Lecture

Today

- **Selection:** file scan; index scan
- **Projection:** hashing; sorting

Next Lecture

Today

- **Join:** looping; hashing; sorting
- **Group By:** hashing; sorting

Today

- **Order By:** sorting

Today's Class

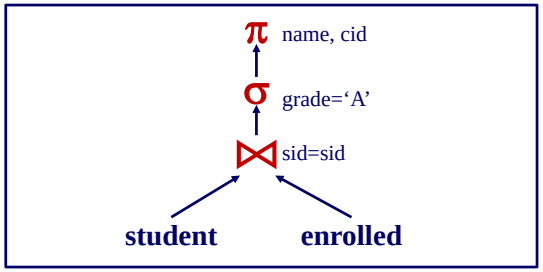
- Intro to Operator Evaluation
- Typical Query Optimizer
- Projection/Aggregation: Sort vs. Hash

Query Optimization

- Bring query in internal form (eg., parse tree)
- … into "canonical form" (syntactic q-opt)
- Generate alternative plans.
- Estimate cost for each plan.
- Pick the best one.

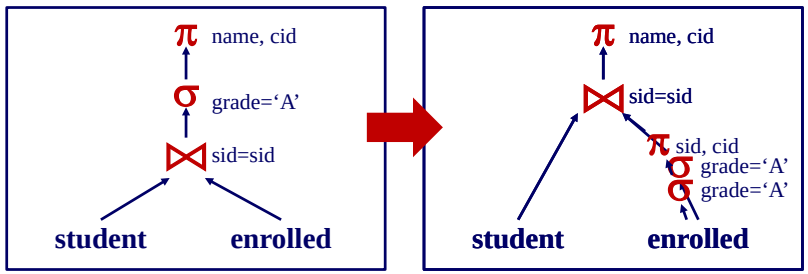
Query Plan Example

```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```



Query Plan Example

```
SELECT name, cid
  FROM student, enrolled
 WHERE student.sid =
       enrolled.sid
   AND enrolled.grade = 'A'
```



Today's Class

- Intro to Operator Evaluation
- Typical Query Optimizer
- Projection/Aggregation: Sort vs. Hash

Duplicate Elimination

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

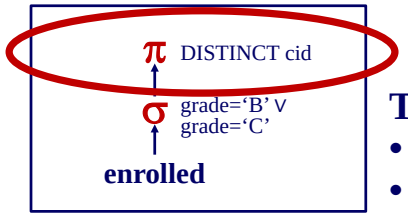
- What does it do, in English?
- How to execute it?

$\pi_{\textbf{DISTINCT}\ \text{cid}}\ (\sigma_{\text{grade='B'} \vee \text{grade='C'}}\ (\text{enrolled}))$

Not technically correct because RA doesn't have "DISTINCT"

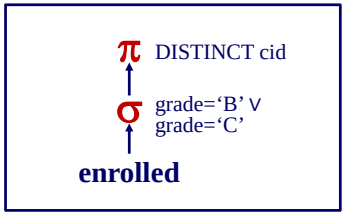
Duplicate Elimination

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

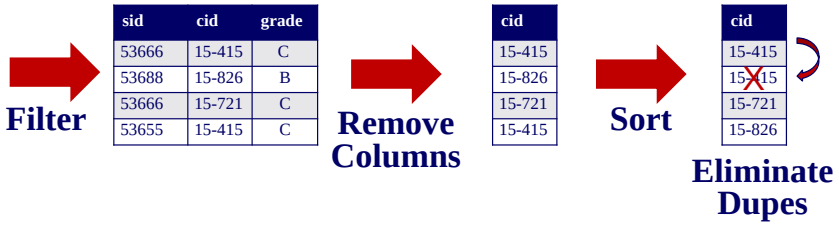


Two Choices:
- Sorting
- Hashing

Sorting Projection



sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C



Alternative to Sorting: Hashing!

- What if we don't need the *order* of the sorted data?
  - Forming groups in **GROUP BY**
  - Removing duplicates in **DISTINCT**
- Hashing does this!
  - And may be cheaper than sorting! (why?)
  - But what if table doesn't fit in memory?

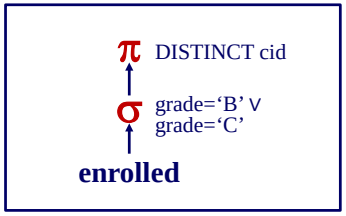
Hashing Projection

- Populate an ephemeral hash table as we iterate over a table.
- For each record, check whether there is already an entry in the hash table:
  - **`DISTINCT`**: Discard duplicate.
  - **`GROUP BY`**: Perform aggregate computation.
- Two phase approach.

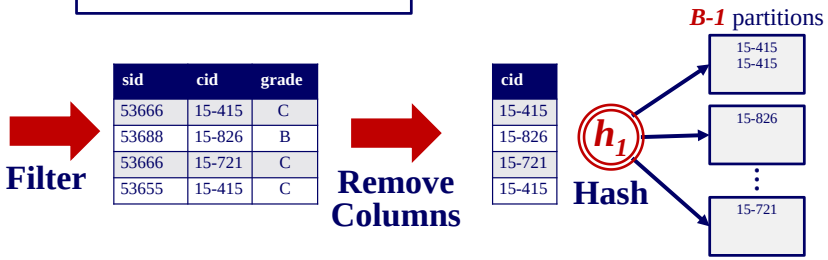
Phase 1: Partition

- Use a hash function h_1 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.
- Assume that we have ***B*** buffers.

Phase 1: Partition



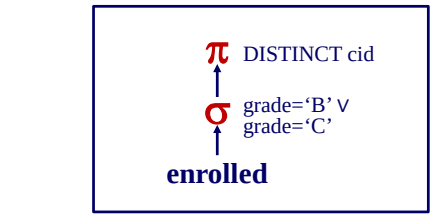
sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C



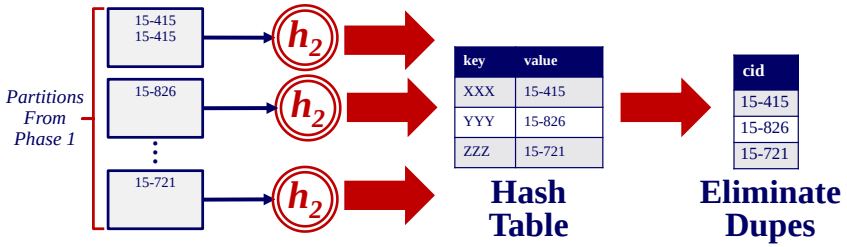
Phase 2: ReHash

- For each partition on disk:
  - Read it into memory and build an in-memory hash table based on a hash function h_2
  - Then go through each bucket of this hash table to bring together matching tuples
- This assumes that each partition fits in memory.

Phase 2: ReHash



sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C



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

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)***.
    - A table of ***N*** blocks needs about ***sqrt(N)*** buffers
  - What assumption do we make?

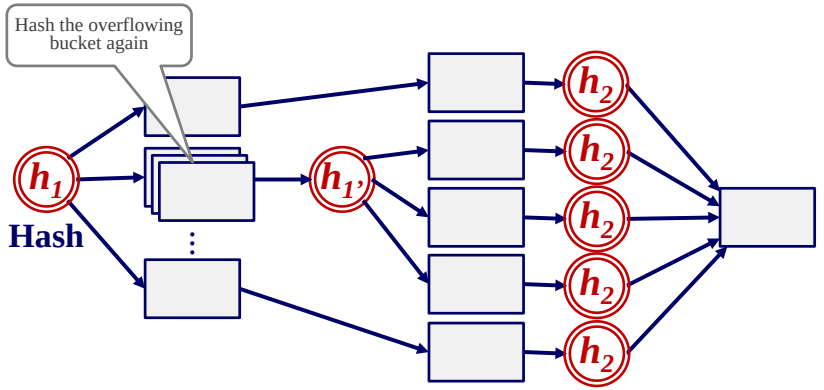
Analysis

- 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)***.
    - A table of ***N*** blocks needs about ***sqrt(N)*** buffers
  - Assumes hash distributes records evenly!
    - Use a "fudge factor" ***f >1*** for that: we need
    - ***B ~ sqrt(f ·N)***

Analysis

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

Recursive Partitioning



Hashing vs. Sorting

- Which one needs more buffers?

Hashing vs. Sorting

- **Recall: We 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 ≤ B-1***
    - Thus, we (roughly) require: ***N ≤ B2***
    - We can sort a table of size ***N*** blocks in about space ***sqrt(N)***
  - **Same as hashing!**

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case
- Already discussed optimizations for sorting:
  - Chunk I/O into large blocks to amortize seek+RD costs
  - Double-buffering to overlap CPU and I/O

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case
- Another optimization when using sorting for aggregation:
  - "Early aggregation" of records in sorted runs
- Let's look at some optimizations for hashing next…

Hashing: We Can Do Better!

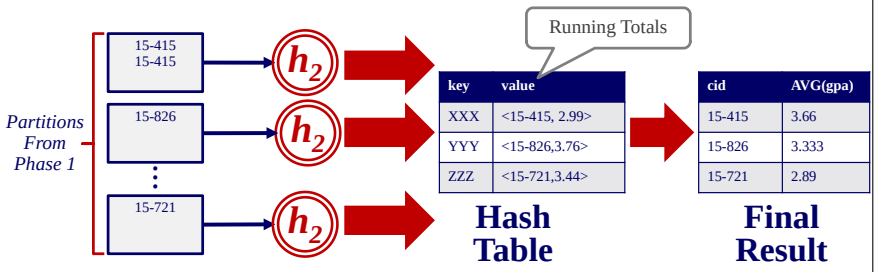
- Combine the summarization into the hashing process - How?

Hashing: We Can Do Better!

- During the ReHash phase, store pairs of the form **`<GroupKey, RunningVal>`**
- When we want to insert a new tuple into the hash table:
  - If we find a matching **`GroupKey`**, just update the **`RunningVal`** appropriately
  - Else insert a new **`<GroupKey, RunningVal>`**

Hashing Aggregation

```
SELECT cid, AVG(s.gpa)
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
 GROUP BY cid
```



Hashing Aggregation

- What's the benefit?
- How many entries will we have to handle?
  - Number of distinct values of GroupKeys columns
  - Not the number of tuples!
  - Also probably "narrower" than the tuples

So, Hashing is Better…right?

- Any caveats?
  - Sorting is better on non-uniform data.
  - Sorting is better when result needs to be sorted.
- Hashing vs. sorting:
  - Commercial systems use either or both

Summary

- Query processing architecture:
  - Query optimizer translates SQL to a query plan
  - Query executor "interprets" the plan
- Hashing is a useful alternative to sorting for duplicate elimination / group-by
  - Both are valuable techniques for a DBMS

Next Class

- How to actually use indexes.
- Join algorithms.
- More query optimization.