CMU SCS

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

*C. Faloutsos – A. Pavlo*

Lecture#13: Query Evaluation

CMU SCS

## Administrivia

- HW4 is due this Thursday.
- Mid-term on Tues March 3rd
  - Will cover everything up to last week.
  - Closed book, one sheet of notes (double-sided)
  - Please email Christos + Andy if you need special accommodations.
  - More Info: http://cmudb.io/s15-midterm



CMU SCS

## Extended Office Hours

- **Christos:**
  - Wednesday Feb 25th 1:00pm-2:00pm
  - Friday Feb 27th 1:00pm-2:00pm
- **Andy:**
  - Wednesday Feb 25th 3:30pm-4:30pm
  - Monday Mar 2nd 1:00pm-2:00pm



CMU SCS

## Last Class

- **Sorting:**
  - External Merge Sort
- **Projection:**
  - External Merge Sort
  - Two-Phase Hashing

**These are for when the data is larger than the amount of memory available.**

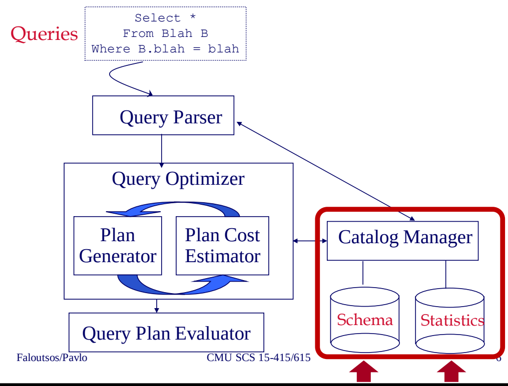
## Today's Class

- Catalog (12.1)
- Intro to Operator Evaluation (12.2-3)
- Typical Query Optimizer (12.6)
- Projection/Aggregation: Sort vs. Hash (14.3.2)





## Cost-based Query Sub-System








## Catalog: Schema

- What would you store?
- How?





## Catalog: Schema

- What would you store?
  – *Info about tables, attributes, indices, users*
- How?
  – *In tables!*
  Attribute_Cat (attr_name: **string**, rel_name: **string**; type: **string**; position: **integer**)

CMU SCS

## Accessing Table Schema

- You can query the DBMS's internal **INFORMATION_SCHEMA** catalog to get info about the database.
- ANSI standard set of read-only views that provide info about all of the tables, views, columns, and procedures in a database
- Every DBMS also have non-standard shortcuts to do this.



CMU SCS

## Accessing Table Schema

- List all of the tables in the current database:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
 WHERE table_catalog = '<db name>'
```

```
\d;
```
**Postgres**

```
SHOW TABLES;
```
**MySQL**

```
.tables;
```
**SQLite**



CMU SCS

## Accessing Table Schema

- List the column info for the student table:

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS
 WHERE table_name = 'student'
```

```
\d student;
```
**Postgres**

```
DESCRIBE student;
```
**MySQL**

```
.schema student;
```
**SQLite**



CMU SCS

## Catalog: Statistics

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

CMU SCS

## Catalog: Statistics

- Why do we need them?
  - *To estimate cost of query plans*
- What would you store?
  - **Tables**: # tuples, # pages
  - **Indexes:** # distinct values, # pages, min/max



CMU SCS

## Today's Class

- Catalog (12.1)
- Intro to Operator Evaluation (12.2-3)
- Typical Query Optimizer (12.6)
- Projection/Aggregation: Sort vs. Hash (14.3.2)



CMU SCS

## Query Plan Example

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```

**Relational Algebra:**

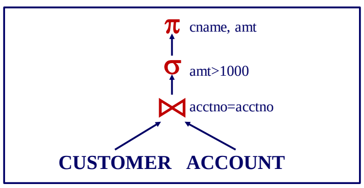
$$\pi_{\text{cname, amt}}(\sigma_{\text{amt>1000}}\ (\text{customer} \bowtie \text{account}))$$



CMU SCS

## Query Plan Example

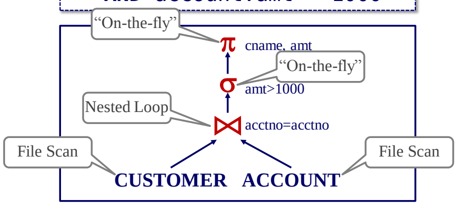
```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



CMU SCS

## Query Plan Example

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```

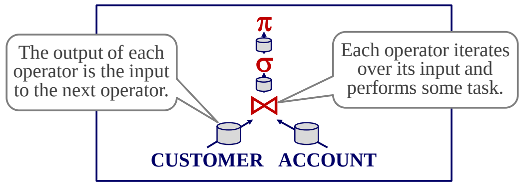



Faloutsos/Pavlo CMU SCS 15-415/615 17

CMU SCS

## Query Plan Example

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```




Faloutsos/Pavlo CMU SCS 15-415/615 18

CMU SCS

## 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".

***Next Week:*** *How the DBMS finds the best plan.*

Faloutsos/Pavlo CMU SCS 15-415/615 19

CMU SCS

## Operator Execution Strategies

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

Faloutsos/Pavlo CMU SCS 15-415/615 20

CMU SCS

## Operator Algorithms

- **Selection:**
- **Projection:**
- **Join:**
- **Group By:**
- **Order By:**



CMU SCS

## Operator Algorithms

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



CMU SCS

## Operator Algorithms

- **Selection:** file scan; index scan
- **Projection:** hashing; sorting
- **Join:** many ways (loops, sort-merge, etc)
- **Group By:**
- **Order By:**



CMU SCS

## Operator Algorithms

- **Selection:** file scan; index scan
- **Projection:** hashing; sorting
- **Join:** many ways (loops, sort-merge, etc)
- **Group By:** hashing; sorting
- **Order By:** sorting

CMU SCS

## Operator Algorithms

Next Lecture

Today

- **Selection:** file scan; index scan
- **Projection:** hashing; sorting
- **Join:** many ways (loops, sort-merge, etc)
- **Group By:** hashing; sorting
- **Order By:** sorting

Next Lecture

Today

Today



CMU SCS

## Today's Class

- Catalog (12.1)
- Intro to Operator Evaluation (12.2-3)
- Typical Query Optimizer (12.6)
- Projection/Aggregation: Sort vs. Hash (14.3.2)



CMU SCS

## Query Optimization

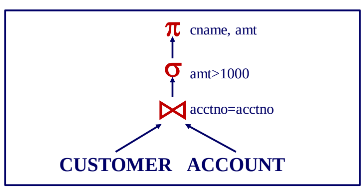
- Bring query in internal form (eg., parse tree)
- … into "canonical form" (syntactic q-opt)
- Generate alternative plans.
- Estimate cost for each plan.
- Pick the best one.



CMU SCS

## Query Plan Example

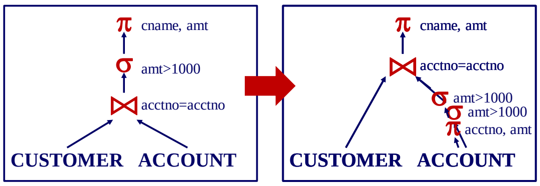
```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



CMU SCS

## Query Plan Example

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```






CMU SCS

## Today's Class

- Catalog (12.1)
- Intro to Operator Evaluation (12.2,3)
- Typical Query Optimizer (12.6)
- Projection/Aggregation: Sort vs. Hash (14.3.2)



CMU SCS

## Duplicate Elimination

```
SELECT DISTINCT bname
  FROM account
 WHERE amt > 1000
```

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

$$\pi_{\textbf{DISTINCT}\ \text{bname}}\ (\sigma_{\text{amt}>1000}\ (\text{account}))$$

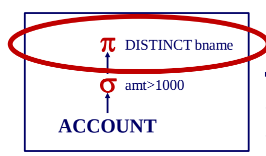
Not technically correct because RA doesn't have "DISTINCT"



CMU SCS

## Duplicate Elimination

```
SELECT DISTINCT bname
  FROM account
 WHERE amt > 1000
```




**Two Choices:**
- Sorting
- Hashing

CMU SCS

# Sorting Projection

$\pi$ DISTINCT bname

$\sigma$ amt>1000

ACCOUNT

| acctno | bname | amt |
|---|---|---|
| A-123 | Redwood | 1800 |
| A-789 | Downtown | 2000 |
| A-123 | Perry | 1500 |
| A-456 | Downtown | 1300 |

Filter

| acctno | bname | amt |
|---|---|---|
| A-123 | Redwood | 1800 |
| A-789 | Downtown | 2000 |
| A-123 | Perry | 1500 |
| A-456 | Downtown | 1300 |

Remove Columns

| bname |
|---|
| Redwood |
| Downtown |
| Perry |
| Downtown |

Sort

| bname |
|---|
| Downtown |
| ~~Downtown~~ |
| Perry |
| Redwood |

Eliminate Dupes



CMU SCS

# 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?



CMU SCS

# 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.



CMU SCS

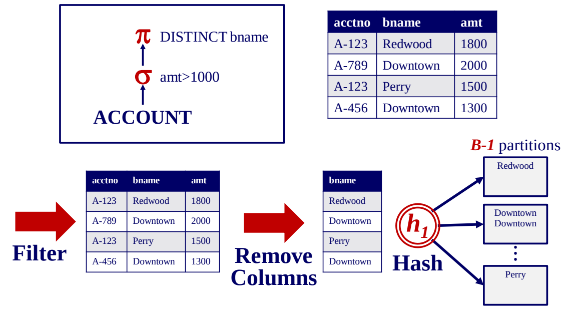
# Phase 1: Partition

- 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.

CMU SCS

## Phase 1: Partition

| acctno | bname | amt |
|---|---|---|
| A-123 | Redwood | 1800 |
| A-789 | Downtown | 2000 |
| A-123 | Perry | 1500 |
| A-456 | Downtown | 1300 |






CMU SCS

## Phase 2: ReHash

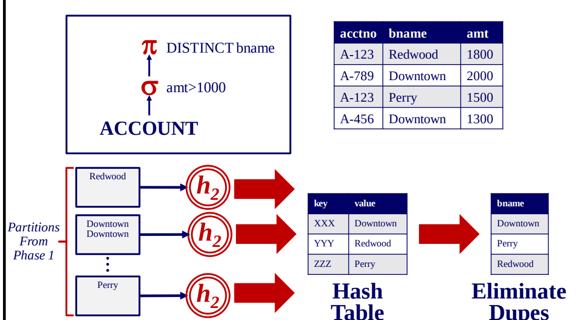
- For each partition on disk:
  - Read it into memory and build an in-memory hash table based on a hash function $h_2$
  - Then go through each bucket of this hash table to bring together matching tuples
- This assumes that each partition fits in memory.



CMU SCS

## Phase 2: ReHash

| acctno | bname | amt |
|---|---|---|
| A-123 | Redwood | 1800 |
| A-789 | Downtown | 2000 |
| A-123 | Perry | 1500 |
| A-456 | Downtown | 1300 |






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)***.
    - 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)***.
    - 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)***



CMU SCS

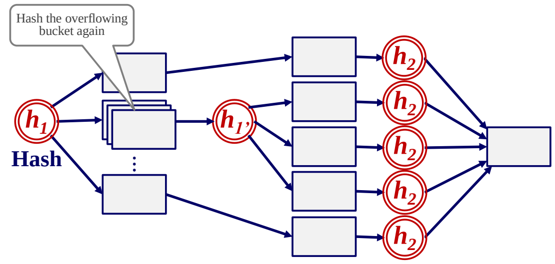
# Analysis

- 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



CMU SCS

# Recursive Partitioning



CMU SCS

## Hashing vs. Sorting

- Which one needs more buffers?



CMU SCS

## 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!**



CMU SCS

## Hashing vs. Sorting

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



CMU SCS

## 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…

CMU SCS

## Hashing: We Can Do Better!

- Combine the summarization into the hashing process - How?



CMU SCS

## Hashing: We Can Do Better!

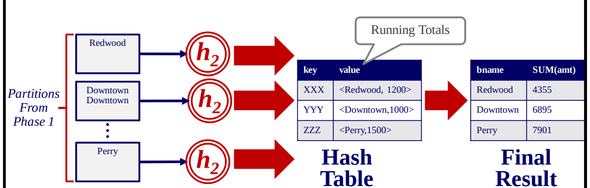
- During the ReHash phase, store pairs of the form **`<GroupKey, RunningVal>`**
- When we want to insert a new tuple into the hash table:
  - If we find a matching **`GroupKey`**, just update the **`RunningVal`** appropriately
  - Else insert a new **`<GroupKey, RunningVal>`**



CMU SCS

## Hashing Aggregation

```
SELECT bname, SUM(amt)
  FROM account
 GROUP BY bname
```






CMU SCS

## 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

CMU SCS

## So, hashing is better…right?

- Any caveats?



CMU SCS

## So, hashing is 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



CMU SCS

## 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



CMU SCS

## Next Class

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