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

C. Faloutsos – A. Pavlo  
Lecture#13: Query Evaluation

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.  

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

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?
  - Info about tables, attributes, indices, users
- How?
  - In tables!
    Attribute_Cat (attr_name: string, rel_name: string; type: string; position: integer)
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.

Accessing Table Schema

- List all of the tables in the current database:

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

- Postgres

```
SHOW TABLES;
```

- MySQL

```
.tables;
```

- SQLite

```
```

Catalog: Statistics

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

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)

Query Plan Example

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

Relational Algebra:

\[ \pi_{cname, \text{amt}}(\sigma_{\text{amt}>1000}(\text{customer} \bowtie \text{account})) \]
SELECT cname, amt
FROM customer, account
WHERE customer.acctno = account.acctno
AND account.amt > 1000

CUSTOMER ACCOUNT

The output of each operator is the input to the next operator.

Each operator iterates over its input and performs some task.

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.

Operator Execution Strategies

• Indexing
• Iteration (= seq. scanning)
• Partitioning (sorting and hashing)
Operator Algorithms

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

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

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)

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

```sql
SELECT cname, amt
FROM customer, account
WHERE customer.acctno = account.acctno
  AND account.amt > 1000
```
Query Plan Example

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

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)

Duplicate Elimination

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

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

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

ACCOUNT

DISTINCT bname

amt>1000

Filter

Remove Columns

Sort

Eliminate Dupes

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 $h_j$ 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

- DISTINCT bname
- amt>1000

ACCOUNT

acctno | bname  | amt
-------|--------|----
A-123  | Redwood| 1800
A-789  | Downtown| 2000
A-123  | Perry   | 1900
A-456  | Downtown| 1800

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.

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: \( B \cdot (B - 1) \).
    • A table of \( N \) blocks needs about \( \sqrt{N} \) buffers
    • What assumption do we make?

• A table of \( N \) blocks needs about \( \sqrt{N} \) buffers
  – What assumption do we make?
  – Use a “fudge factor” \( f > 1 \) for that; we need
    - \( B \sim \sqrt{f \cdot N} \)

Recursive Partitioning

• 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

Hash

\[ h_1 \]
\[ h_1' \]
\[ \vdots \]

Hash the overflowing bucket again
Hashing vs. Sorting

- Which one needs more buffers?

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 $N/B \leq B-1$
    - Thus, we (roughly) require: $N \leq B^2$
    - We can sort a table of size $N$ blocks in about space $\sqrt{N}$
    - Same as hashing!

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

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?

During the ReHash phase, store pairs of the form $<\text{GroupKey}, \text{RunningVal}>$

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 $<\text{GroupKey}, \text{RunningVal}>$

Hashing Aggregation

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

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?

- 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

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.