CMU SCS

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

C. Faloutsos – A. Pavlo

Lecture#14: Implementation of Relational Operations

CMU SCS

Last Class

- Catalog
- Intro to Operator Evaluation
- Typical Query Optimizer
- Projection/Aggregation

CMU SCS

Today's Class

- More on Indexes
- Explain
- Joins
- Mid-term Review (Christos)

CMU SCS

Access Paths

- How the DBMS retrieves tuples from a table for a query plan.
  - **File Scan** (aka Sequential Scan)
  - **Index Scan** (Tree, Hash, List, …)
- Selectivity of an access path:
  - % of pages we retrieve
  - e.g., Selectivity of a hash index, on range query: 100% (no reduction!)

CMU SCS

Selection Conditions

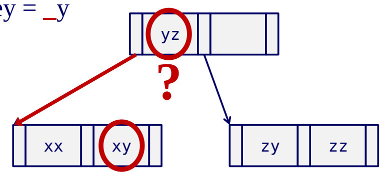
- A B-tree index matches (a conjunction of) terms that involve only attributes in a prefix of the search key.
  - Index on `<a,b,c>` matches `(a=5 AND b=3)`, but not `b=3`.
- For Hash index, we must have **all** attributes in search key.

CMU SCS

B+Tree Prefix Search

Key = xy

Key = _y



CMU SCS

Partial Indexes

- Create an index on a ***subset*** of the entire table. This potentially reduces its size and the amount of overhead to maintain it.

```
CREATE INDEX idx_foo
      ON foo (a, b)
   WHERE c = 'WuTang'
```

```
SELECT b FROM foo
 WHERE a = 123 AND c = 'WuTang'
```


CMU SCS

Covering Indexes

- If all of the fields needed to process the query are available in an index, then the DBMS does not need to retrieve the tuple.

```
CREATE INDEX idx_foo
          ON foo (a, b)
```

```
SELECT b FROM foo WHERE a = 123
```

CMU SCS

Index Include Columns

- Embed additional columns in indexes to support index-only queries.
- Not part of the search key.

```
CREATE INDEX idx_foo
          ON foo (a, b)
     INCLUDE (c)
```

```
SELECT b FROM foo
 WHERE a = 123 AND c = 'WuTang'
```


CMU SCS

Today's Class

- More on Indexes
- Explain
- Joins
- Mid-term Review (Christos)

CMU SCS

EXPLAIN

- When you precede a **SELECT** statement with the keyword **EXPLAIN**, the DBMS displays information from the optimizer about the statement execution plan.
- The system "explains" how it would process the query, including how tables are joined and in which order.

CMU SCS

EXPLAIN

```
SELECT bid, COUNT(*) AS cnt
  FROM Reserves
 GROUP BY bid
 ORDER BY cnt
```

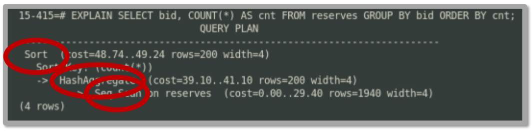
Pseudo Query Plan:

SORT
↑
COUNT
↑
GROUP BY
↑
π_{bid}
↑
RESERVES

CMU SCS

EXPLAIN

```
EXPLAIN SELECT bid, COUNT(*) AS cnt
  FROM Reserves
 GROUP BY bid
 ORDER BY cnt
```



Postgres v9.1

Faloutsos/Pavlo CMU SCS 15-415/615 13

CMU SCS

EXPLAIN

```
EXPLAIN SELECT bid, COUNT(*) AS cnt
  FROM Reserves
 GROUP BY bid
 ORDER BY cnt
```

MySQL v5.5

Faloutsos/Pavlo CMU SCS 15-415/615 14

CMU SCS

EXPLAIN ANALYZE

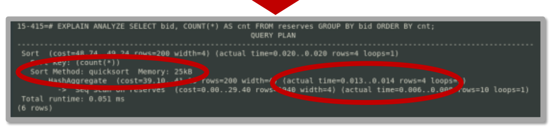
- **ANALYZE** option causes the statement to be actually executed.
- The actual runtime statistics are displayed.
- This is useful for seeing whether the planner's estimates are close to reality.
- Note that **ANALYZE** is a Postgres idiom.

Faloutsos/Pavlo CMU SCS 15-415/615 15

CMU SCS

EXPLAIN ANALYZE

```
EXPLAIN ANALYZE
SELECT bid, COUNT(*) AS cnt
  FROM Reserves
 GROUP BY bid
 ORDER BY cnt
```



Postgres v9.1

Faloutsos/Pavlo CMU SCS 15-415/615 16

CMU SCS

EXPLAIN ANALYZE

- Works on any type of query.
- Since **ANALYZE** actually executes a query, if you use it with a query that modifies the table, that modification <u>will</u> be made.

Faloutsos/Pavlo CMU SCS 15-415/615 17

CMU SCS

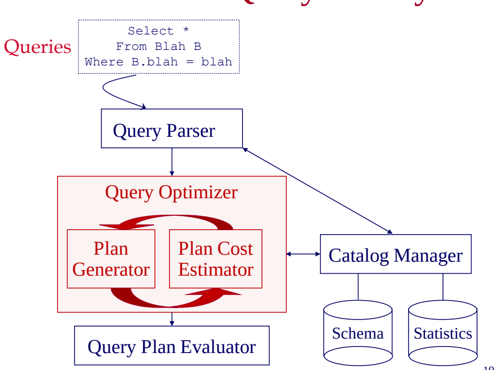
Today's Class

- More on Indexes
- Explain
- Joins
- Mid-term Review (Christos)

Faloutsos/Pavlo CMU SCS 15-415/615 18

CMU SCS

Cost-based Query Sub-System



19

CMU SCS

Sample Database

SAILORS

sid	sname	rating	age
1	Christos	999	45.0
3	Obama	50	52.0
2	Tupac	32	26.0
6	Bieber	10	19.0

RESERVES

sid	bid	day	rname
6	103	2014-02-01	matlock
1	102	2014-02-02	macgyver
2	101	2014-02-02	a-team
1	101	2014-02-01	dallas

Sailors(<u>*sid*: int</u>, *sname*: varchar, *rating*: int, *age*: real)
Reserves(<u>*sid*: int</u>, <u>*bid*: int</u>, <u>*day*: date</u>, *rname*: varchar)

Hoofer Sailing Club

Faloutsos/Pavlo CMU SCS 15-415/615 20

CMU SCS

Sample Database

SAILORS

sid	sname	rating	age
1	Christos	999	45.0
3	Obama	50	52.0
2	Tupac	32	26.0
6	Bieber	10	19.0

Each tuple is 50 bytes
80 tuples per page
500 pages total
N=500, p_S=80

RESERVES

sid	bid	day	rname
6	103	2014-02-01	matlock
1	102	2014-02-02	macgyver
2	101	2014-02-02	a-team
1	101	2014-02-01	dallas

Each tuple is 40 bytes
100 tuples per page
1000 pages total
M=1000, p_R=100

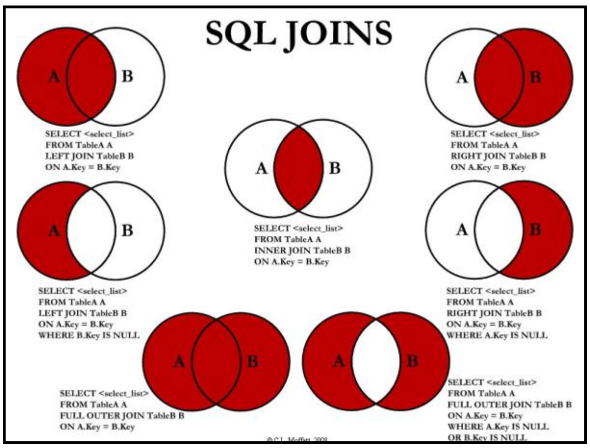
Faloutsos/Pavlo CMU SCS 15-415/615 21

CMU SCS

Joins

- **R⋈S** is very common and thus must be carefully optimized.
- **R×S** followed by a selection is inefficient because cross-product is large.
- There are many approaches to reduce join cost, but no one works best for all cases.
- Remember, join is associative and commutative.

Faloutsos/Pavlo CMU SCS 15-415/615 22



CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins

Faloutsos/Pavlo CMU SCS 15-415/615 24

CMU SCS

Joins

- Assume:
  - ***M*** pages in R, ***pR*** tuples per page, ***m*** tuples total
  - ***N*** pages in S, ***pS*** 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

First Example

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

- Assume that we don't know anything about the tables and we don't have any indexes.

CMU SCS

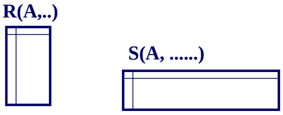
Simple Nested Loop Join

- **Algorithm #0:** Simple Nested Loop Join

```
foreach tuple r of R
  foreach tuple s of S
    output, if they match
```

R(A,..)

S(A, ...)



CMU SCS

Simple Nested Loop Join

- **Algorithm #0:** Simple Nested Loop Join

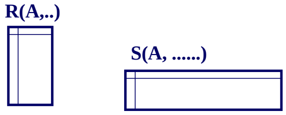
outer relation

```
foreach tuple r of R
  foreach tuple s of S
    output, if they match
```

inner relation

R(A,..)

S(A, ...)

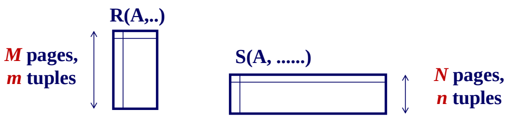


CMU SCS

Simple Nested Loop Join

SLOW

- **Algorithm #0:** Why is it bad?
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: M + (pR · M) · N***



CMU SCS

Simple Nested Loop Join

SLOW

- Actual number:
  - ***M + (pR · M) · N*** = 1000 + 100 · 1000 · 500 = 50,001,000 I/Os
  - At 10ms/IO, Total time ≈ **5.7 days**

CMU SCS

Simple Nested Loop Join

SLOW

- Actual number:
  - ***M + (pR · M) · N*** = 1000 + 1 = 50,001,0
  - At 10ms/IO, Total time ≈ **5.7 days**

SSD ≈ 1.3 hours at 0.1ms/IO

CMU SCS

Simple Nested Loop Join

SLOW

- Actual number:
  - ***M + (pR · M) · N*** = 1000 + 1 = 50,001,0
  - At 10ms/IO, Total time ≈ **5.7 days**
- What if smaller relation (S) was outer?
- What assumptions are being made here?

SSD ≈ 1.3 hours at 0.1ms/IO

CMU SCS

Simple Nested Loop Join

SLOW

- Actual number:
  - $M + (pR \cdot M) \cdot N$ = 1000 + 1
    = 50,001,0
  - At 10ms/IO, Total time ≈ **5.7 days**

SSD ≈ 1.3 hours
at 0.1ms/IO

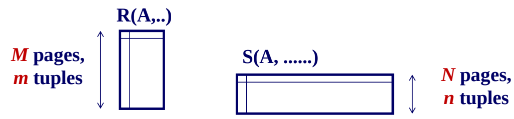
- What if smaller relation (S) was outer?
  - *Slightly better…*
- What assumptions are being made here?
  - *1 buffer for each table (and 1 for output)*

CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Block Nested Loop Join

```
read block from R
  read block from S
    output, if tuples match
```



CMU SCS

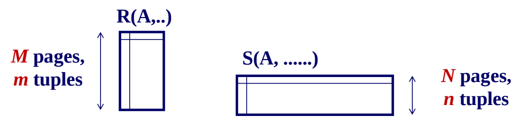
Block Nested Loop Join

- **Algorithm #1:** Things are better.
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: M + (M·N)***

CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Optimizations
- Which one should be the outer relation?
  - *The smallest (in terms of # of pages)*



Block Nested Loop Join

- Actual number:
  - $M + (M \cdot N)$ = 1000 + 1000 · 500 = 501,000 I/Os
  - At 10ms/IO, Total time ≈ **1.4 hours**

Block Nested Loop Join

- Actual number:
  - $M + (M \cdot N)$ = 1000 + 1000 ·
  - At 10ms/IO, Total time ≈ **1.4 hours**

SSD ≈ 50 seconds
at 0.1ms/IO

- What if we use the smaller one as the outer relation?

Block Nested Loop Join

- Actual number:
  - $N + (M \cdot N)$ = 500 + 1000 · 500 = 500,500 I/Os
  - At 10ms/IO, Total time ≈ **1.4 hours**

- What if we have ***B*** buffers available?

Block Nested Loop Join

- **Algorithm #1:** Using multiple buffers.

```
read B-2 blocks from R
  read block from S
    output, if tuples match
```

CMU SCS

Block Nested Loop Join

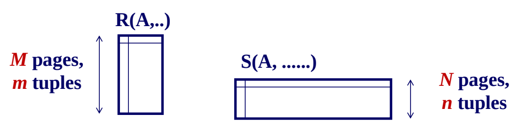
- **Algorithm #1:** Using multiple buffers.
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: M+ (⌈M/(B-2)⌉·N)***

CMU SCS

Block Nested Loop Join

- **Algorithm #1:** Using multiple buffers.
- But if the outer relation fit
  - ***Cost: M+N*** = 1000 + 500 =
  - At 10ms/IO, Total time ≈ **15 seconds**

SSD ≈ 0.15 seconds
at 0.1ms/IO



CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins

CMU SCS

Index Nested Loop

- Why do basic nested loop joins suck?
  - *For each tuple in the outer table, we have to do a sequential scan to check for a match in the inner table.*
- A better approach is to use an **index** to find inner table matches.
  - We could use an existing index, or even build one on the fly.

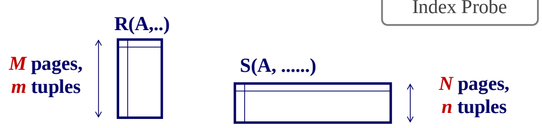
CMU SCS

Index Nested Loop Join

- **Algorithm #2:** Index Nested Loop Join

```
foreach tuple r of R
  foreach tuple s of S, where ri==sj
    output
```

Index Probe

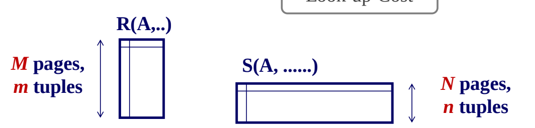


CMU SCS

Index Nested Loop

- **Algorithm #2:** Index Nested Loop Join
- How many disk accesses ('*M*' and '*N*' are the number of blocks for '*R*' and '*S*')?
  - *Cost:* $M + m \cdot C$

Look-up Cost



CMU SCS

Nested Loop Joins Guideline

- Pick the smallest table as the outer relation
  - *i.e., the one with the fewest pages*
- Put as much of it in memory as possible
- Loop over the inner

CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins

CMU SCS

Sort-Merge Join

- **Sort Phase:** First sort both tables on joining attribute.
- **Merge Phase:** Then step through each one in lock-step to find matches.

CMU SCS

Sort-Merge Join

- This algorithm is useful if:
  - One or both tables are already sorted on join attribute(s)
  - Output is required to be sorted on join attributes
- The "Merge" phase can require some back tracking if duplicate values appear in join column.

CMU SCS

Sort-Merge Join Example

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

sid	sname	rating	age
1	Christos	999	45.0
3	Obama	50	52.0
2	Tupac	32	26.0
6	Bieber	10	19.0

Sort!

sid	bid	day	rname
6	103	2014-02-01	matlock
1	102	2014-02-02	macgyver
2	101	2014-02-02	a-team
1	101	2014-02-01	dallas

Sort!

CMU SCS

Sort-Merge Join Example

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

sid	sname	rating	age
1	Christos	999	45.0
2	Tupac	32	26.0
3	Obama	50	52.0
6	Bieber	10	19.0

Merge!

sid	bid	day	rname
1	102	2014-02-02	macgyver
1	101	2014-02-01	dallas
2	101	2014-02-02	a-team
6	103	2014-02-01	matlock

Merge!

CMU SCS

Sort-Merge Join

- **Algorithm #3:** Sort-Merge Join
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: (2M · logM/logB) + (2N · logN/logB) + M + N***

R(A,..)

M **pages,** ***m*** **tuples**

S(A, ...)

N **pages,** ***n*** **tuples**

CMU SCS

Sort-Merge Join

- **Algorithm #3:** Sort-Merge Join
- How many disk accesses (‘***M***’ and ‘***N***’ are the number of blocks for ‘***R***’ and ‘***S***’)?
  - ***Cost: (2M · logM/logB) + (2N · logN/logB) + M + N***

Sort Cost

Sort Cost

Merge Cost

R(A,..)

M **pages,** ***m*** **tuples**

S(A, ...)

N **pages,** ***n*** **tuples**

CMU SCS

Sort-Merge Join Example

- With 100 buffer pages, both Reserves and Sailors can be sorted in 2 passes:
  - ***Cost: 7,500 I/Os***
  - At 10ms/IO, Total time ≈ **75 seconds**
- Block Nested Loop:
  - ***Cost: 2,500 to 15,000 I/Os***

CMU SCS

Sort-Merge Join Example

- With 100 buffer pages, both Reserves and Sailors can be sorted in 2 passes:
  - ***Cost: 7,500 I/Os***
  - At 10ms/IO, Total time ≈ **75 seconds**
- Block Nested Loop:
  - ***Cost: 2,500 to 15,000 I/Os***

SSD ≈ 0.75 seconds at 0.1ms/IO

CMU SCS

Sort-Merge Join

- Worst case for merging phase?
  - When all of the tuples in both relations contain the same value in the join attribute.
  - ***Cost: (M · N) + (sort cost)***
- Don't worry kids! This is unlikely!

CMU SCS

Sort-Merge Join Optimizations

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

CMU SCS

Joins

- Join techniques we will cover:
  - Nested Loop Joins
  - Index Nested Loop Joins
  - Sort-Merge Joins
  - Hash Joins

CMU SCS

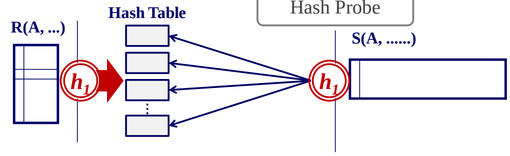
In-Memory Hash Join

- **Algorithm #4:** In-Memory

This assumes H fits in memory!

```
build hash table H for R
foreach tuple s of S
  output, if h(sj)∈ H
```

Hash Probe

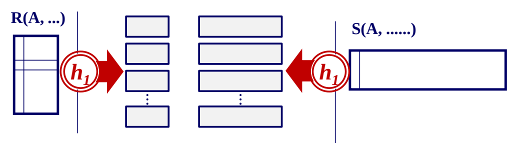


Grace Hash Join

- Hash join when tables don't fit in memory.
  - **Partition Phase:** Hash both tables on the join attribute into partitions.
  - **Probing Phase:** Compares tuples in corresponding partitions for each table.
- Named after the GRACE database machine.

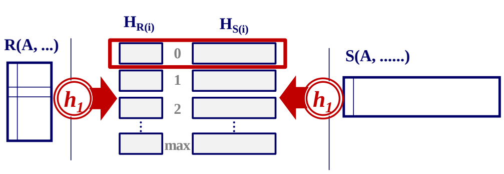
Grace Hash Join

- Hash R into (0, 1, ..., 'max') buckets
- Hash S into buckets (same hash function)



Grace Hash Join

- Join each pair of matching buckets:
  - Build another hash table for $H_{S(i)}$, and probe it with each tuple of $H_{R(i)}$



Grace Hash Join

- Choose the (page-wise) smallest - if it fits in memory, do a **nested loop join**
  - Build a hash table (with H_2 != H)
  - And then probe it for each tuple of the other

CMU SCS

Grace Hash Join

- What if $H_{S(i)}$ is too large to fit in memory?
  - Recursive Partitioning!
  - More details (overflows, hybrid hash joins) available in textbook (Ch 14.4.3)

Faloutsos/Pavlo CMU SCS 15-415/615 65

CMU SCS

Grace Hash Join

- Cost of hash join?
  - Assume that we have enough buffers.
  - ***Cost: 3(M + N)***
- **Partitioning Phase:** read+write both tables
  - ***2(M+N) I/Os***
- **Probing Phase:** read both tables
  - ***M+N I/Os***

Faloutsos/Pavlo CMU SCS 15-415/615 66

CMU SCS

Grace Hash Join

- Actual number:
  - ***3(M + N)*** = 3 · (1000 + 500)
  - At 10ms/IO, Total time ≈ **45 seconds**

SSD ≈ 0.45 seconds at 0.1ms/IO

Faloutsos/Pavlo CMU SCS 15-415/615 67

CMU SCS

Sort-Merge Join vs. Hash Join

- Given a minimum amount of memory both have a cost of ***3(M+N) I/Os***.
- When do we want to choose one over the other?

Faloutsos/Pavlo CMU SCS 15-415/615 68

CMU SCS

Sort-Merge Join vs. Hash Join

- **Sort-Merge:**
- **Hash:**

Faloutsos/Pavlo CMU SCS 15-415/615 69

CMU SCS

Sort-Merge Join vs. Hash Join

- **Sort-Merge:**
  - Less sensitive to data skew.
  - Result is sorted (may help upstream operators).
  - Goes faster if one or both inputs already sorted.
- **Hash:**

Faloutsos/Pavlo CMU SCS 15-415/615 70

CMU SCS

Sort-Merge Join vs. Hash Join

- **Sort-Merge:**
  - Less sensitive to data skew.
  - Result is sorted (may help upstream operators).
  - Goes faster if one or both inputs already sorted.
- **Hash:**
  - Superior if relation sizes differ greatly.
  - Shown to be highly parallelizable.

Faloutsos/Pavlo CMU SCS 15-415/615 71

CMU SCS

Summary

- There are multiple ways to do selections if you have different indexes.
- Joins are difficult to optimize.
  - Index Nested Loop when selectivity is small.
  - Sort-Merge/Hash when joining whole tables.

Faloutsos/Pavlo CMU SCS 15-415/615 72