RELATIONAL OPERATORS #2

CS 564- Spring 2018

ACKs: Jeff Naughton, Jignesh Patel, AnHai Doan
WHAT IS THIS LECTURE ABOUT?

Algorithms for relational operators:
• joins
• set operators
• aggregation
JOINS
JOIN OPERATOR

Algorithms for equijoin:

```sql
SELECT *
FROM R, S
WHERE R.a = S.a
```

Why can’t we compute it as cartesian product?
JOIN ALGORITHMS

Algorithms for equijoin:

- nested loop join
- block nested loop join
- index nested loop join
- block index nested loop join
- sort merge join
- hash join
NESTED LOOP JOIN (1)

• for each page $P_R$ in $R$
  • for each page $P_S$ in $S$
    • join the tuples on $P_R$ with the tuples in $P_S$

$I/O = M_R + M_S \cdot M_R$

• $M_R =$ number of pages in $R$
• $M_S =$ number of pages in $S$

Observe that we ignore the cost of writing the output to disk!
NESTED LOOP JOIN (2)

• Which relation should be the outer relation in the loop?
  – The smaller of the two relations

• How many buffer pages do we need?
  – only 3 pages suffice
BLOCK NESTED LOOP JOIN (1)

Assume $B$ buffer pages

• for each block of $B-2$ pages from $R$
  • for each page $P_S$ in $S$
    • join the tuples from the block with the tuples in $P_S$

$I/O = M_R + M_S \cdot \left\lceil \frac{M_R}{B-2} \right\rceil$
• To increase CPU efficiency, create an in-memory hash table for each block
  – what will be the key of the hash table?

• What happens if R fits in memory?
  – The I/O cost is only \( M_R + M_S \)!
NLJ VS BNLJ

Example:

- $M_R = 500$ pages
- $M_S = 1000$ pages
- 100 tuples / page
- $B = 12$

\[
\begin{align*}
NLJ \text{ I/O} &= 500 + 500 \times 1000 = 500,500 \\
BNLJ \text{ I/O} &= 500 + \frac{500 \times 1000}{12-2} = 50,500
\end{align*}
\]

The difference in I/O cost in an order of magnitude!
INDEX NESTED LOOP JOIN

S has an index on the join attribute

• for each page $P_R$ in $R$
  • for each tuple $r$ in $R$
    • probe the index of $S$ to retrieve any matching tuples

$I/O = M_R + |R| \cdot I^*$

• $I^*$ is the I/O cost of searching an index, and depends on the type of index and whether it is clustered or not
BLOCK INDEX NESTED LOOP JOIN

• for each block of $B$-2 pages in $R$
  • sort the tuples in the block
  • for each tuple $r$ in the block
    • probe the index of $S$ to retrieve any matching tuples

• Why do we need to sort here?
SORT MERGE JOIN
SORT MERGE JOIN: BASIC VERSION

The basic version:

• sort $R$ and $S$ on the join attribute (using external merge sort)
• read the sorted relations in the buffer and merge

If $R$, $S$ are already sorted on the join attribute we can skip the first step!
The two relations are already sorted on attribute A.
READ AND MERGE

R(A,B)  S(A,C)

(1,5)  (3,2)  (5,6)
(1,a)  (2,b)  (3,c)

B = 3 frames

buffer

read

(1,5)
(1,a)
(1,5,a)

write

output

read 1 page from each list
merge the tuples
READ AND MERGE

\[ R(A,B) \]

\[ S(A,C) \]

\[ \text{output} \]

\[ \text{disk} \]

\[ \text{buffer} \]

\[ B = 3 \text{ frames} \]

\[ (1,5) \quad (3,2) \quad (5,6) \]

\[ (1,a) \quad (2,b) \quad (3,c) \]

\[ (1,5,a) \]

\[ (3,2) \]

\[ (2,b) \]

sine 2<3, we have to read a new page from S
**READ AND MERGE**

- **disk**
  - \( R(A,B) \):
    - (1,5)
    - (3,2)
    - (5,6)
  - \( S(A,C) \):
    - (1,a)
    - (2,b)
    - (3,c)

- **buffer**
  - \( B = 3 \) frames
    - (3,2)
    - (3,c)
    - (3,2,c)

- **output**
  - (1,5,a)

- \( B = 3 \) frames

- Read:
  - (3,2)

- Write:
  - (3,2,c)

And so on...
SORTING WITH DUPLICATES

in the case of multiple duplicate values, merging may not take linear time!
SORTING WITH DUPLICATES

\[(1,5) \quad (1,2) \quad (1,6)\]

\[(1,a) \quad (1,b) \quad (1,c)\]

\[(1,5,a) \quad (1,5,b)\]

\[(1,5)\]

\[(1,b)\]

\[\text{buffer} \quad B = 3 \text{ frames}\]

which page should we read next? we need to \textbf{backup} to compute the full result
SMJ: I/O COST

- If there is no backup, the I/O cost of read + merge is only $M_R + M_S$
- If there is backup, in the worst case the I/O cost could be $M_R \times M_S$
  - this happens when there is a single join value

Total I/O cost $\sim \text{sort}(R) + \text{sort}(S) + M_R + M_S$
SORT MERGE JOIN: OPTIMIZED

- Generate sorted runs of size $\sim 2B$ for $R$ and $S$
- Merge the sorted runs for $R$ and $S$
  - while merging check for the join condition and output the join tuples

I/O cost $\sim 3(M_R + M_S)$

But how much memory do we need for this to happen?
SMJ: MEMORY ANALYSIS

- In the first phase, we create runs of length $\sim 2B$
- Hence, the number of runs is $\frac{M_R + M_S}{2B}$
- To perform a $k$-way merge, we need $k+1$ buffer pages, so:
  \[
  \frac{M_R + M_S}{2B} \leq B - 1 \text{ or } B^2 \geq \max\{M_S, M_R\}
  \]

If $B^2$ is larger than the maximum number of pages of the two relations, then SMJ has I/O cost $\sim 3(M_R + M_S)$
We will use a hash function $h$ to map values of the join attribute (A) into buckets $[1, B-1]$.

Tuple $t$ is then hashed to bucket $h(t.A)$.

A hash collision occurs when $x \neq y$ but $h(x) = h(y)$.

Note however that it will never happen that $x = y$ but $h(x) \neq h(y)$.
HASH JOIN: OVERVIEW

Start with a hash function $h$ on the join attribute

- **Partition phase:** partition $R$ and $S$ into $k$ partitions using $h$
- **Matching phase:** join each partition of $R$ with the corresponding (same hash value) partition of $S$ using BNLJ
PARTITION PHASE

R(A,B)

(1,a) (1,c) (5,a)
(2,b) (2,d) (3,e)

P1

P2

buffer

B = 3 frames

we will create k=2 partitions

we will create k=2 partitions

read

write
PARTITION PHASE

R(A,B)

(1,a) (1,c) (5,a)
(2,b) (2,d) (3,e)

P1

disk

read

buffer

B = 3 frames

(1,a) (1,a)
(2,b) (2,b)

P2

hash the tuples to the partition

reach one page at a time

reach (1)=1

h(2)=2

write
PARTITION PHASE

disk

\[ R(A, B) \]

(1, a)  (1, c)  (5, a)  (2, b)  (2, d)  (3, e)

buffer

\[ B = 3 \text{ frames} \]

\( h(1) = 1 \)

\( h(2) = 2 \)

next page in

P1

P2
PARTITION PHASE

R(A,B)

(1,a)  (1,c)  (5,a)
(2,b)  (2,d)  (3,e)

P1

(1,a)  (1,c)

P2

(2,b)  (2,d)

Buffer

B = 3 frames

(5,a)

h(5)=1

(5,a)

h(3)=2

(3,e)

R(A,B)

read

write

read next page
PARTITION PHASE

R(A,B)

(1,a)  (1,c)  (3,a)
(2,b)  (2,d)  (3,e)

P1

(1,a)  (5,a)
(1,c)   

P2

(2,b)  (3,e)
(2,d)   

disk

buffer  B = 3 frames

read

write

write to disk

P1 write to disk

P2 write to disk

B = 3 frames
BUCKET SIZE

• We can create up to \( k = B-1 \) partitions in one pass

• How big are the buckets we create?
  – Ideally, each bucket has \( \sim M/(B-1) \) pages
  – but hash collisions can occur!
  – or we may have many duplicate values on the join attribute (skew)

• In the matching phase, we join two buckets from \( R, S \) with the same hash value
  – We want to do this in linear time using BNLJ, so we must guarantee that each bucket from one of the two relations is at most \( B-1 \) pages
HJ: I/O COST

- Suppose $M_R \leq M_S$
- The partition phase gives buckets of size $\sim M_R/B$
- To make BNLJ run in one pass we need to make sure that:
  \[
  \frac{M_R}{B} \leq B - 2 \quad \text{or equivalently:} \quad B^2 \geq M_R
  \]

If $B^2$ is larger than the \textbf{minimum} number of pages of the two relations, then HJ has I/O cost $\sim 3(M_R + M_S)$
COMPARISON OF JOIN ALGORITHMS

Hash Join vs Block Nested Loop Join
• the same if smaller table fits into memory
• otherwise, hash join is much better
COMPARISON OF JOIN ALGORITHMS

Hash Join vs Sort Merge Join

- Suppose $M_R > M_S$
- To do a two-pass join, SMJ needs $B > \sqrt{M_R}$
  - the I/O cost is: $3(M_R + M_S)$
- To do a two-pass join, HJ needs $B > \sqrt{M_S}$
  - the I/O cost is: $3(M_R + M_S)$
GENERAL JOIN CONDITIONS

• Equalities over multiple attributes
  – e.g., $R.sid = S.sid$ and $R.rname = S.sname$
  – for Index Nested Loop
    • index on <sid, sname>
    • index on sid or sname
  – for SMJ and HJ, we can sort/hash on combination of join attributes
GENERAL JOIN CONDITIONS

- Inequality conditions
  - e.g., \( R.rname < S.sname \)
  - For BINL, we need (clustered) B+ tree index
  - SMJ and HJ not applicable
  - BNLJ likely to be the winner (why?)
SET OPERATIONS & AGGREGATION
SET OPERATIONS

• Intersection is a special case of a join
• Union and difference are similar
• Sorting:
  – sort both relations (on all attributes)
  – merge sorted relations eliminating duplicates
• Hashing:
  – partition R and S
  – build in-memory hash table for partition \( R_i \)
  – probe with tuples in \( S_i \), add to table if not a duplicate
AGGREGATION: SORTING

• sort on group by attributes (if any)
• scan sorted tuples, computing running aggregate
  – max/min: max/min
  – average: sum, count
• when the group by attribute changes, output aggregate result
• cost = sorting cost
AGGREGATION: HASHING

- Hash on group by attributes (if any)
  - Hash entry = group attributes + running aggregate
- Scan tuples, probe hash table, update hash entry
- Scan hash table, and output each hash entry
- cost = scan relation
- What happens if we have many groups?
AGGREGATION: INDEX

• Without grouping
  – Can use B+ tree on aggregate attribute(s)

• With grouping
  – B+ tree on all attributes in SELECT, WHERE and GROUP BY clauses
    • Index-only scan
    • If group-by attributes prefix of search key, the data entries/tuples are retrieved in group-by order