COMP 3311
DATABASE MANAGEMENT SYSTEMS

TUTORIAL 9
QUERY OPTIMIZATION

COST ESTIMATION INFORMATION

- The DBMS system catalog stores statistics for each relation r:

  n_r the number of tuples in r.

  B_r the number of pages containing tuples of r.

  l_r the size of a tuple of r in bytes.

  bf_r the blocking factor of r (i.e., the number of tuples of r that fit into one disk page).

  HT_i the number of levels in the index i (i.e., the height of i).

  - For a B$^+$-tree index on attribute A of relation r, $HT_i = \lceil \log (V(A, r)) \rceil$.
  - For a hash index, HT_i is 1 (or 1.2 if there are overflow buckets).

  $V(A, r)$ the number of distinct values that appear in r for attribute A.

  - Equivalent to $\pi_A(r)$.

  ☞ If tuples of r are stored together physically in a file, then
$$B_r = \lceil n_r / bf_r \rceil.$$

SELECTION CARDINALITY: REVIEW

$SC(\theta, r)$, the selection cardinality of predicate θ for relation r, is the *average number of tuples* that satisfy the predicate θ.

Selectivity$(\theta, r) = SC(\theta, r) / n_r$ (i.e., the fraction of tuples that satisfy θ)

Equality selection: $\sigma_{A=v}(r)$

$SC(A=v, r) = n_r / V(A, r) \implies$ number of r tuples / number of distinct A values

Selectivity$(A=v, r) = 1 / V(A, r) \implies$ 1 / number of distinct A values

Range selection: $\sigma_{A \leq v}(r)$

$SC(A \leq v, r) = 0$ if $v < \min(A, r)$ (since no tuples will qualify)

$SC(A \leq v, r) = n_r$ if $v \geq \max(A, r)$ (since all tuples will qualify)

$SC(A \leq v, r) = n_r * \dfrac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$ (assumes uniform distribution of tuples over the values)

SELECTION CARDINALITY: REVIEW

Conjunction: $\sigma_{\theta1 \wedge \theta2 \wedge \ldots \wedge \theta n}(r)$

$$SC(\sigma_{\theta1 \wedge \theta2 \wedge \ldots \wedge \theta n}, r) = n_r * \frac{s_1}{n_r} * \frac{s_2}{n_r} * \cdots * \frac{s_n}{n_r} = n_r * \frac{s_1 * s_2 * \cdots * s_n}{n_r^n}$$

Disjunction: $\sigma_{\theta1 \vee \theta2 \vee \ldots \vee \theta n}(r)$

$$SC(\sigma_{\theta1 \vee \theta2 \vee \ldots \vee \theta n}, r) = n_r * (1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r}))$$

Projection: $SC(\pi_A(r)) = V(A, r)$

Aggregation: $SC(\text{group-by } A) = V(A, r)$

Set Operations

$SC(r \cup s) = $ size of r + size of s

$SC(r \cap s) = $ minimum (size of r , size of s)

$SC(r - s) = r$

SELECTION CARDINALITY: REVIEW

Join: $SC(r \bowtie s)$

If $r \cap s = \varnothing$

> same as r x s

If $r \cap s = A$ is a key for r

> no greater than number of tuples in s

If $r \cap s = A$ is a (*not null*) foreign key *in* s referencing r

> same as the number of tuples in s

If $r \cap s = A$ is *not a key* for r or s

> $\frac{n_r * n_s}{V(A,s)}$ or $\frac{n_r * n_s}{V(A,r)}$ (select the lower estimate)

HISTOGRAM: REVIEW

- The previous estimates assume that each value of A has the same probability (i.e., that the values of A are uniformly distributed over the tuples).

  ☞ **The uniformity assumption rarely holds in practice.**

- A histogram assumes local uniformity within each partition (but not global uniformity).
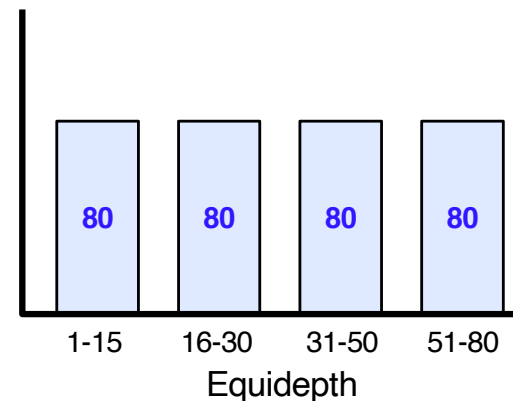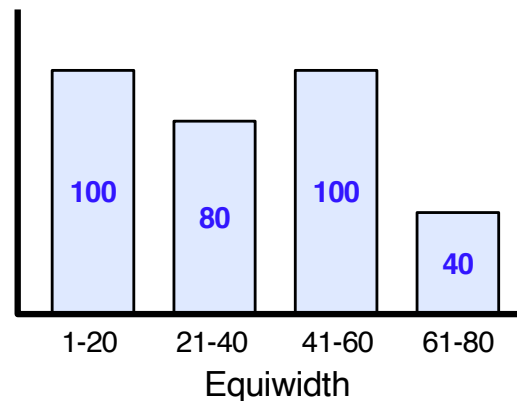
The numbers in the bars indicate the total number of tuples that fall into that range.
For equiwidth, the ranges are the same, but each partition contains a different number of tuples.
For equidepth, each partition contains exactly 80 tuples, but the ranges are different for each bucket.

EXERCISE 1

Sailor(sailorId, sName, rating, age)   6,000 tuples   10 tuples/page

Reserves(sailorId, boatId, rDate)   1,500 tuples   15 tuples/page

Boat(boatId, bName, color)   3,000 tuples   20 tuples/page

There is no index on any relation.   15% of boats are red.

a) Estimate the size of $\pi_{boatId}(\sigma_{color='red'}Boat)$.

b) Estimate the size of (Sailor Join Reserves Join Boat). Briefly explain the reason(s) for your answer.

c) Which of the following two join strategies is better? Briefly explain the reason(s) for your choice.

   i.   (Sailor Join Boat) Join Reserves

   ii. (Sailor Join Reserves) Join Boat

Sailor tuples: 6,000
Reserves tuples: 1,500
Boat tuples: 3,000
bf_{Sailor}:   10 tuples/page
$bf_{Reserves}$:   15 tuples/page
bf_{Boat}:   20 tuples/page
15% red boats

EXERCISE 1 (CONT'D)

a) Estimate the size of $\pi_{boatId}(\sigma_{color='red'}Boat)$.

Size of $\pi_{boatId}(\sigma_{color='red'}$BOATS): 15% * 3000 = <u>450</u> tuples

b) Estimate the size of (Sailor Join Reserves Join Boat) and briefly explain the reason(s) for your answer.

Since sailorId in Reserves is a foreign key referencing Sailor, one tuple in Reserves will join with exactly one tuple in Sailor. Therefore, Sailor Join Reserves has exactly 1500 tuples.

Since boatId in Reserves is a foreign key referencing Boat, each tuple in the preceding intermediate result will join with exactly one tuple in Boat. Therefore, the final join also contains exactly 1500 tuples.

Size of the final result: <u>1500</u> tuples

c) Which of the following two join strategies is better? Explain the reason(s) for your choice.

i. (Sailor JOIN Boat) JOIN Reserves

In this strategy, since there is no common attribute, Sailor JOIN Boat is equal to the cross-product of the two relations and the size of the join result will become as large as 6000 * 3000 = 18,000,000 tuples.

This intermediate result is very large and later, when joining this intermediate result with Reserves, the cost is also large to read all the tuples of the intermediate result. The final result has 1500 tuples.

ii. (Sailor JOIN Reserves) JOIN Boat

In this strategy, since the common attribute, sailorId, is a not null foreign key in Reserves that references Sailor, Sailor JOIN Reserves has only 1500 tuples.

When joining this intermediate result with Boat, the cost is also small and the final result has 1500 tuples.

Therefore, the second strategy is preferred.

EXERCISE 2

For the query $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$ assume that the projection routine uses external sorting and eliminates all unwanted attributes during the initial sort pass and also removes duplicate tuples on-the-fly during the merge passes. Furthermore, assume the following:

> Can eliminate unwanted attributes during join rather then during sort!

- R is 10 pages and each R tuple is 300 bytes.
- S is 100 pages and each S tuple is 500 bytes.
- The combined size of attributes A, B, C and D is 450 bytes.
- A and B are in R and have combined size 200 bytes; C and D are in S.
- A is a key for R.
- Each S tuple joins with exactly one R tuple.
- The page size is 1024 bytes.
- The main memory buffer size M is 3 pages.
- Only the (optimized) block nested-loop join method is implemented.

Number of R tuples: $\lfloor 1024/300 \rfloor$ * 10 = 30 tuples

Number S tuples: $\lfloor 1024/500 \rfloor$ * 100 = 200 tuples

Query: $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$

EXERCISE 2 (CONT'D)

n_R: 30 tuples
B_R: 10 pages
bf_R: 3 tuples/page
n_S: 200 tuples
B_S: 100 pages
bf_S: 2 tuples/page
page size: 1024 bytes
M: 3 pages

a) What is the cost of processing the query?

Use R (smaller relation as outer relation)

Join cost: $\lceil B_r / (M\text{-}2) \rceil * B_s + B_r = \lceil 10 / (3\text{-}2) \rceil * 100 + 10$
$$= \underline{1010} \text{ page I/Os}$$

Since each S tuple joins with exactly one R tuple, the join produces 200 tuples, each of size 300+500=800 bytes. Thus, $\lceil 1024 / 800 \rceil = 1$ tuple fits on a page and there are 200 / 1 = 200 page I/Os (write to disk).

Projection cost:

The initial sort pass eliminates unwanted attributes producing tuples containing only A, B, C, and D of size 450 bytes. Thus, $\lceil 1024 / 450 \rceil = 2$ tuples fit on a page and there are 200 / 2 = 100 pages.

Sort cost pass 0: 200 (reads) + 100 (writes) = 300 page I/Os

The initial sort pass produces 33 runs of 3 pages each and 1 run of 1 page (i.e., 100 pages). These runs are merged in 6 additional passes.

Merge cost: (100 * 2 * 6) - 100 = 1100 page I/Os

Total cost: 1010 + 200 + 300 + 1100 = 2610 page I/Os

The cost of 100, to write the final result, is subtracted from the total merge cost since we do not include the cost of writing the final result.

Query: $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$

n_R: 30 tuples
B_R: 10 pages
bf_R: 3 tuples/page
n_S: 200 tuples
B_S: 100 pages
bf_S: 2 tuples/page
page size: 1024 bytes
M: 3 pages

b) **What is the cost of processing the query if merge join is used instead of block-nested loop using a main memory buffer of 3 pages?**

Using merge join, the projection can be done on-the-fly as the relations are joined. So, the projection cost is 0 and the only cost is for the join. Moreover, when initially sorting R and S, unwanted attributes can be removed, which reduces the number of pages written after sorting.

Cost to sort R (A and B are in R and have size 200 bytes)

Sort pass: Read 3 pages of R into memory and eliminate unwanted attributes. Each R page contains 3 tuples and after eliminating unwanted attributes, these tuples occupy $\lceil 9 / \lfloor 1024 / 200 \rfloor \rceil = 2$ pages. Therefore, we can read one more page into memory and sort 4 pages at once creating a sorted run of 3 pages containing 12 tuples.

We do the same with the next 4 pages creating a second sorted run of 3 pages containing 12 tuples.

Finally, we read the final 2 pages and create a third sorted run of 2 pages containing 6 tuples. (Not all pages are full.)

Query: $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$

EXERCISE 2 (CONT'D)

n_R: 30 tuples
B_R: 10 pages
bf_R: 3 tuples/page
n_S: 200 tuples
B_S: 100 pages
bf_S: 2 tuples/page
page size: 1024 bytes
M: 3 pages

Sort pass cost: read 10 pages + write 8 pages = 18 page I/Os

Merge pass 1: Merge a sorted run of 3 pages (12 tuples) and one of 2 pages (6 tuples) creating a sorted run of $\lceil 18 / \lfloor 1024 / 200 \rfloor \rceil = 4$ pages.

Merge pass 2: Merge the run of 4 pages (18 tuples) with the remaining sorted run of 3 pages (12 tuples) creating a final sorted output of $\lceil 30 / \lfloor 1024 / 200 \rfloor \rceil = 6$ pages.

Merge pass cost: 3 + 2 + 4 + 3 + 4 + 6 = 22 page I/Os

<small>pass 1   pass 2   pass 1 pass 2</small>
<small>   read   write</small>

(We note that the cost to first merge the 3 pages and then the 2 pages would be: read 3 + 3 + 5 + 2; write 5 + 6 = 24 page I/Os.)

Cost to sort R: 18 + 22 = <u>40</u> page I/Os

Cost to sort S (C and D are in S and have size 450-200=250 bytes)

Sort pass: Read 3 pages of S into memory and eliminate unwanted attributes. Each S page contains 2 tuples and after eliminating unwanted attributes, these tuples occupy $\lceil 6 / \lfloor 1024 / 250 \rfloor \rceil = 2$ pages.

Therefore, read one more page into memory. After removing unwanted

attributes, the 8 tuples will still fit in $\lceil 8 / \lfloor 1024 / 250 \rfloor \rceil = 2$ pages.

Therefore, read one more page into memory. After removing unwanted attributes, the 10 tuples will fit in $\lceil 10 / \lfloor 1024 / 250 \rfloor \rceil = 3$ pages.

We sort these 10 tuples and create a sorted run of 3 pages.

We do the same for each of the remaining 95 pages creating 20 sorted runs of 3 pages each containing 10 tuples. (Not all pages are full.)

Sort pass cost: read 100 pages + write 60 pages = 160 page I/Os

Merge pass 1: Merge 20 sorted runs of 3 pages each creating 10 runs of $\lceil 20 / \lfloor 1024 / 250 \rfloor \rceil = 5$ pages containing 20 tuples each. Read 60, write 50 pages. (We note that the first merge pass creates more full pages.)

Merge pass 2: Merge 10 sorted runs of 5 pages each creating 5 runs of $\lceil 40 / \lfloor 1024 / 250 \rfloor \rceil = 10$ pages containing 40 tuples each. Read 50, write 50 pages.

Merge pass 3: Merge 4 sorted runs of 10 pages each creating 2 runs of $\lceil 80 / \lfloor 1024 / 250 \rfloor \rceil = 20$ pages containing 80 tuples each. Read 40, write 40 pages. (We leave the extra run of 10 pages for later merging.)

EXERCISE 2 (CONT'D)

n_R: 30 tuples
B_R: 10 pages
bf_R: 3 tuples/page
n_S: 200 tuples
B_S: 100 pages
bf_S: 2 tuples/page
page size: 1024 bytes
M: 3 pages

Merge pass 4: Merge 1 run of 20 pages and one run of 10 pages creating 1 run of 30 pages containing 120 tuples. Read 30, write 30 pages.

Merge pass 5: Merge one run of 20 pages and one run of 30 pages to create the final sorted run of $\lceil 200 / \lfloor 1024 / 250 \rfloor \rceil = 50$ pages containing 200 tuples. Read 50 pages, write 50 pages.

Merge pass cost: 60 + 50 + 40 + 30 + 50 +
read → pass 1 pass 2 pass 3 pass 4 pass 5

50 + 50 + 40 + 30 + 50 = 450 page I/Os
write → pass 1 pass 2 pass 3 pass 4 pass 5

Cost to sort S: 160 + 450 = <u>610</u> page I/Os

<u>Cost to join</u>

B_r + B_s = 6 + 50 = <u>56</u> page I/Os

Total cost: 40 + 610 + 56 = <u>706</u> page I/Os