LECTURE 6

EXECUTION PLAN

- Execution plan includes
  - Access methods available for each relation
  - Algorithms to be used in computing the relational operators represented in the tree
- For materialized evaluation, the result of operation is stored as a temporary relation
- For pipelined evaluation, as the resulting tuples of an operation are produced, they are forwarded directly to the next operation in the query sequence
- Advantage of pipelining – cost savings (No need to write intermediate results to disk)

COST ESTIMATES IN QUERY OPTIMIZATION (1/2)

- More suitable for **compiled queries** where the optimization is done at compile time and the resulting execution strategy code is stored and executed directly at runtime
- For interpreted queries, where entire process occurs at runtime, a full-scale optimization may slow down the response time
- This approach is called **cost-based query optimization**

COST COMPONENTS FOR QUERY EXECUTION (1/3)

1. **Access cost to secondary storage**
   - Cost of searching for reading and writing data blocks that reside on secondary storage (disk)
   - Cost of searching records in a file depends on the type of access structures on that file (ordering, hashing, primary or secondary indexes)
   - Whether file blocks are allocated contiguously on same disk cylinder or scattered on the disk

2. **Storage cost**
   - Cost of storing any intermediate files that are generated by execution strategy for the query

COST COMPONENTS FOR QUERY EXECUTION (2/3)

3. **Computation cost**
   - Cost of performing in-memory operations on data buffers during query execution
   - Includes searching for and sorting records, merging records for a join, and performing computations on field values

4. **Memory usage cost**
   - Cost pertaining to number of memory buffers needed during query execution

5. **Communication cost**
   - Cost of shipping the query and its results from the database site to the site or terminal where the query originated

COST COMPONENTS FOR QUERY EXECUTION (3/3)

- For large database
  - Main emphasis is on minimizing access **cost to secondary storage**
  - Simple cost functions ignore other factors and compare different query execution strategies in terms of the number of block transfers between disk and main memory
- For smaller database
  - Minimizing **computation cost** as most of data in files involved in query can be completely stored in memory
- In distributed database
  - **Communication cost** must be minimized as many sites are involved

CATALOG INFORMATION USED IN COST FUNCTIONS (1/2)

- Size of the file :
  - Number of records (r)
  - Record size (R)
  - Number of blocks (b)
  - Blocking factor (bfr)

- Access methods and indexes
  - No of levels (x) of each multilevel index (primary, secondary, or clustering)
  - No of first-level index blocks (b_{I1}) – for some cost functions

CATALOG INFORMATION USED IN COST FUNCTIONS (2/2)

- **Selectivity (sl)** is the fraction of records satisfying an equality condition on the attribute

- **Selection Cardinality (s)** is the average number of records that will satisfy an equality selection condition on that attribute, calculated as $s = sl * r$
- **No of distinct values (d)**

- For a key attribute,

  $d = r$, $sl = 1/r$, and $s = 1$

- For a nonkey attribute, assume that d distinct values are uniformly distributed among the records , we have

  $sl = (1/d)$ hence, $s = (r/d)$

EXAMPLES OF COST FUNCTIONS FOR SELECT (1/8)

- We denote cost for method S_i as C_{si}
- **S1: Linear search**
  - Search all file blocks to retrieve all records satisfying the selection condition
  - Hence, $C_{S1a} = b$
  - For equality condition on a key, only half the file blocks are searched on the average before finding the record

    $C_{S1b} = (b/2)$ if record is found

    $C_{S1b} = b$ if no record satisfies condition

EXAMPLES OF COST FUNCTIONS FOR SELECT (2/8)

- **S2 : Binary search**
  - Search accesses approx.

$$C_{S2} = \log_2 b + [(s/bfr)] - 1 \text{ file blocks}$$

  - This reduces to $\log_2 b$ if equality condition is on a unique (key) attribute, because $s = 1$ in this case

EXAMPLES OF COST FUNCTIONS FOR SELECT (3/8)

- **S3 : Using primary index (S3a) or hash key (S3b) to retrieve single record**
  - For primary index
    - Retrieve one more block than the no of index levels

$$C_{S3a} = x+1$$

  - For hashing

$C_{S3b}=1$ for static or linear hashing

$C_{S3b}=2$ for extendible hashing

EXAMPLES OF COST FUNCTIONS FOR SELECT (4/8)

- **S4: Using ordering index to retrieve multiple records**
  - If comparison condition is >,>=,<,<= on a key field with ordering index, roughly half the file records will satisfy the condition

$$C_{S4}=x+(b/2)$$

EXAMPLES OF COST FUNCTIONS FOR SELECT (5/8)

- **S5: Using clustering index to retrieve multiple records**
  - Given equality condition, s records will satisfy condition, where s is selection cardinality of indexing attribute
  - This means that [(s/bfr)] file blocks will be accessed

$$C_{S5}=x+ [(s/bfr)]$$

EXAMPLES OF COST FUNCTIONS FOR SELECT (6/8)

- **S6: Using a secondary index**
  - On equality comparison, s records will satisfy the condition, where s is selection cardinality of indexing attribute
  - Since index is non clustering, each of the records may reside on a different block, hence

$$C_{S6a}=x+s$$

  - This reduces to x+1 for a key indexing attribute

EXAMPLES OF COST FUNCTIONS FOR SELECT (7/8)

- If comparison condition is <,>=,<,<= and half of the file records are assumed to satisfy the condition, then roughly half the first-level index blocks are accessed plus half the file records via the index

$$C_{S6b}=x+(b_{l1}/2) + (r/2)$$

- r/2 factor can be refined if better selectivity estimates are available

EXAMPLES OF COST FUNCTIONS FOR SELECT (8/8)

- **S7: Conjunctive selection**
  - **Use either S1 or one of the methods S2 to S6**
  - In latter case, we use one condition to retrieve the records and then check in the memory buffer whether each retrieved record satisfies the remaining conditions in the conjunction

- **S8: Conjunctive selection using composite index**
  - **Same as S3a, S5, S6a depending on type of index**

EXAMPLES OF USING COST FUNCTIONS (1/7)

- EMPLOYEE file has r_E=10,000 records stored in b_E=2000 disk blocks with blocking factor bfr_E=5 records/block and the following access paths :
  1. A clustering index on Salary, with levels x_{Salary}=3 and average selection cardinality s_{Salary}=20
  2. A secondary index on key attribute Eno with x_{Eno}=4 (s_{Eno}=1)
  3. A secondary index on the nonkey attribute Dno, with x_{Dno}=2 and first level index blocks b_{I1Dno}=4. There are d_{Dno}=125 distinct values for Dno, so the selection cardinality of Dno is $s_{Dno}=(r_E/d_{Dno})=80$
  4. A secondary index on Sex, with x_{Sex}=1. There are d_{Sex}=2 values for the sex attribute, so the average selection cardinality is $s_{Sex}=(r_E/d_{Sex})=5000$

EXAMPLES OF USING COST FUNCTIONS (2/7)

- Illustrate the use of cost functions with the examples below

  (OP1) : $\sigma_{\text{Eno='123456789'}}$ (EMPLOYEE)
  (OP2) : $\sigma_{\text{Dno>5}}$ (EMPLOYEE)
  (OP3) : $\sigma_{\text{Dno=5}}$ (EMPLOYEE)
  (OP4) : $\sigma_{\text{Dno=5 AND Salary>30000 AND Sex='F'}}$ (EMPLOYEE)

- Cost of linear search option S1 estimated as

  $C_{S1a}=b_E=2000$ (for nonkey attribute)
  $C_{S1b}=(b_E/2)=1000$ (average cost for selection on key attribute)

EXAMPLES OF USING COST FUNCTIONS (3/7)

(OP1) : $\sigma_{Eno='123456789'}$ (EMPLOYEE)

- Use either S1 (C_{S1b}=1000)
- Method S6

$$C_{S6a} = x_{Eno}+1$$
$$= 4+1$$
$$= 5$$

- Method S6 is chosen

EXAMPLES OF USING COST FUNCTIONS (4/7)

(OP2) : $\sigma_{Dno>5}$ (EMPLOYEE)

- Method S1 (C_{S1a}=2000)

- Method S6

$$C_{S6b} = x_{Dno} + (b_{I1Dno}/2) + (r_E/2)$$
$$= 2 + (4/2) + (10000/2)$$
$$= 5004$$

- Method S1 is chosen

EXAMPLES OF USING COST FUNCTIONS (5/7)

(OP3) : $\sigma_{\text{Dno}=5}$ (EMPLOYEE)

- Method S1 (C_{S1a}=2000)
- Method S6

$$C_{S6a} = x_{Dno} + s_{Dno}$$
$$= 2 + 80$$
$$= 82$$

- Method S6 is chosen

EXAMPLES OF USING COST FUNCTIONS (6/7)

(OP4) : $\sigma_{\text{Dno=5 AND Salary>30000 AND Sex='F'}}$ (EMPLOYEE)

- It has conjunctive selection, estimate the cost of using any one of the three components of selection condition to retrieve records plus brute force approach (linear search)
- Method S1 (C_{S1a}=2000)
- For (Dno=5), cost estimate is C_{S6a}=82
- For (Salary>30000), cost estimate is

$$
\begin{aligned}
C_{S4} &= x_{Salary}+(b_E/2) \\
&= 3+(2000/2) \\
&= 1003
\end{aligned}
$$

EXAMPLES OF USING COST FUNCTIONS (7/7)

- For (Sex='F'), cost estimate is

$$\begin{aligned} CS6a &= x_{Sex} + s_{Sex} \\ &= 1+5000 \\ &= 5001 \end{aligned}$$

- Method S6a for (Dno=5) is chosen

- Hence, condition (Dno=5) is used to retrieve records, and remaining part of conjunctive condition (Salary>30000 AND Sex='F') is checked for each selected record after it is retrieved into memory

EXAMPLES OF COST FUNCTIONS FOR JOIN (1/7)

- Need to have estimate for size (number of tuples) of file that results after JOIN
- Usually kept as a *ratio of the size (number of tuples) of resulting join file to the size of Cartesian product file*, if both are applied to same input files, and is called **join selectivity (js)**
- If number of tuples of a relation R is |R|, we have

  $js = |(R \bowtie_c S)| \,/\, |(R \times S)| = |(R \bowtie_c S)| \,/\, (|R| * |S|)$
- If no join condition c is there, then js=1. The join is same as the CARTESIAN PRODUCT.
- If no tuples from relations satisfy join condition, then js=0

EXAMPLES OF COST FUNCTIONS FOR JOIN (2/7)

- Hence, in general 0<=js<=1

For a join where condition c is equality comparison R.A=S.B, we get

Case 1: If A is a key of R, then

$|(R \bowtie_c S)| <= |S|$, so $js<=(1/|R|)$

Case 2: If B is a key of S, then

$|(R \bowtie_c S)| <= |R|$, so $js<=(1/|S|)$

EXAMPLES OF COST FUNCTIONS FOR JOIN (3/7)

- For estimating size of resulting file if size of two input files are known use formula below

$$|(R \bowtie_c S)| = js * |R| * |S|$$

- Assume that R has b_R blocks and S has b_S blocks

EXAMPLES OF COST FUNCTIONS FOR JOIN (4/7)

- **J1: Nested-loop join**
  - Suppose R is outer loop

$$C_{J1}= b_R+ (b_R*b_S)+ ((js*|R|*|S|)/bfr_{RS})$$

  - Last part of formula is cost of writing resulting file to disk

EXAMPLES OF COST FUNCTIONS FOR JOIN (5/7)

- **J2: Single-loop join (using access structure)**
  - **If index exists for join attribute B of S with index levels x_B,** we can retrieve each record s in R and then use index to retrieve all matching records t from S that satisfy t[B]=s[A]
  - **Cost depends on type of index**
  - Secondary index where s_B is selection cardinality for join attribute B of S, we have

$$C_{J2a} = b_R + ((|R| * (x_B + s_B)) + ((js * |R| * |S|)/bfr_{RS})$$

EXAMPLES OF COST FUNCTIONS FOR JOIN (6/7)

- **Clustering index** where s_B is selection cardinality of B

$$C_{J2b} = b_R + ((|R| * (x_B + (s_B / bfr_B))) + ((js * |R| * |S|) / bfr_{RS})$$

- **Primary index**

$$C_{J2c} = b_R + ((|R| * (x_B + 1)) + ((js * |R| * |S|) / bfr_{RS})$$

- If **hash** key exists for one of the two attributes, say B of S

$$C_{J2d} = b_R + (|R| * h) + ((js * |R| * |S|) / bfr_{RS})$$

where h>=1 is average number of block accesses to retrieve a record, given hash key value

EXAMPLES OF COST FUNCTIONS FOR JOIN (7/7)

- **J3: Sort-merge join**
  - If sorted on join attributes, cost function is

    $$C_{J3a} = b_R + b_S + ((js * |R| * |S|)/bfr_{RS})$$

  - If files need to be sorted, then cost of sorting is also added.
  - **Sorting cost is $(2*b)+(2*b*\log_2 b)$** for a file of b blocks

$$C_{J3b} = (2*b_R*(1+\log_2 b_R)) + (2*b_S*(1+\log_2 b_S)) + b_R + b_S + ((js*|R|*|S|)/bfr_{RS})$$

EXAMPLES OF USING COST FUNCTIONS (1/4)

- EMPLOYEE file has r_E=10,000 records stored in b_E=2000 disk blocks with blocking factor bfr_E=5 records/block and the following access paths :
  1. A clustering index on Salary, with levels x_{Salary}=3 and average selection cardinality s_{Salary}=20
  2. A secondary index on key attribute Eno with x_{Eno}=4 (s_{Eno}=1)
  3. A secondary index on the nonkey attribute Dno, with x_{Dno}=2 and first level index blocks b_{I1Dno}=4. There are d_{Dno}=125 distinct values for Dno, so the selection cardinality of Dno is $s_{Dno}=(r_E/d_{Dno})=80$
  4. A secondary index on Sex, with x_{Sex}=1. There are d_{Sex}=2 values for the sex attribute, so the average selection cardinality is $s_{Sex}=(r_E/d_{Sex})=5000$

EXAMPLES OF USING COST FUNCTIONS (2/4)

- We have DEPARTMENT file consisting of $r_D=125$ records stored in $b_D=13$ disk blocks
- There is a primary index on DNumber of DEPARTMENT with $x_{DNumber}=1$ level

$$EMPLOYEE \bowtie_{Dno=DNumber} DEPARTMENT$$

- The join selectivity for OP6 is $js_{OP6}=(1/|DEPARTMENT|)= 1/125$ since DNumber is key of DEPARTMENT
- Assume blocking factor for resulting file is $bfr_{ED}=4$ records per block

EXAMPLES OF USING COST FUNCTIONS (3/4)

1. Using Method J1 with EMPLOYEE as outer loop

   $C_{J1} = b_E + (b_E * b_D) + ((js_{OP6} * r_E * r_D) / bfr_{ED})$

   $= 2000 + (2000 * 13) + (((1/125) * 10000 * 125) / 4)$

   $= 30500$

2. Using method J1 with DEPARTMENT as outer loop

   $C_{J1} = b_D + (b_E * b_D) + ((js_{OP6} * r_E * r_D) / bfr_{ED})$

   $= 13 + (2000 * 13) + (((1/125) * 10000 * 125) / 4)$

   $= 28513$

EXAMPLES OF USING COST FUNCTIONS (4/4)

3. Using method J2 with EMPLOYEE as outer loop

$C_{J2}=b_E+(r_E*(x_{DNumber}+1))+((js_{OP6}*r_E*r_D)/bfr_{ED})$

$= 2000+(10000*2)+(((1/125)*10000*125)/4)$

$= 24500$

4. Using method J2 with DEPARTMENT as outer loop

$C_{J2}=b_D+(r_D*(x_{Dno}+s_{Dno}))+((js_{OP6}*r_E*r_D)/bfr_{ED})$

$= 13+(125*(2+80))+(((1/125)*10000*125)/4)$

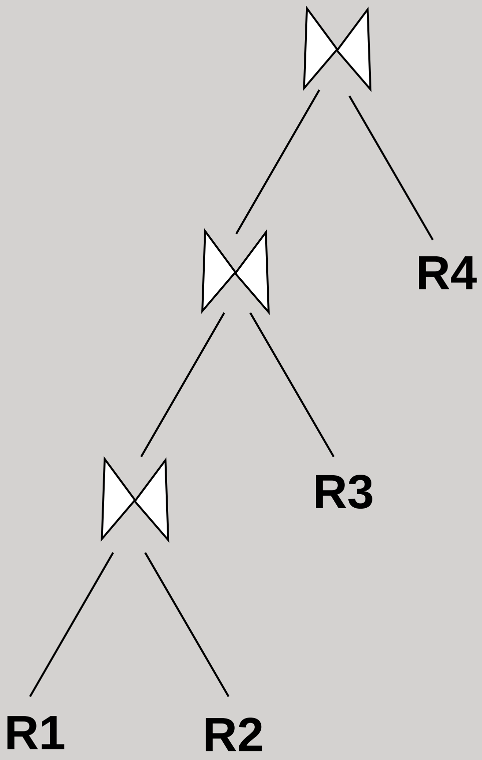
$= 12763$

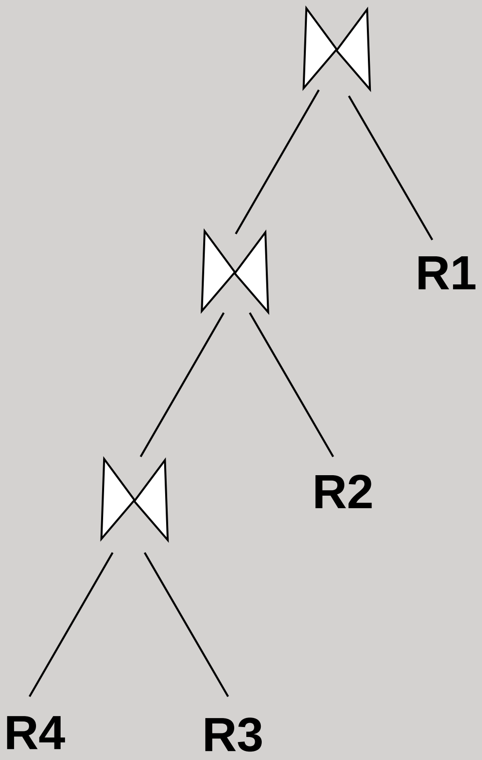
- Since case 4 has lowest cost estimate it is chosen

MULTIPLE RELATION QUERIES AND JOIN ORDERING – KINDLY GO THROUGH YOURSELF

- **A query that joins N relations will have N-1 join operations**
- Estimating cost of every possible join tree for a query with large number of joins will require huge amount of time by query optimizer
- Query optimizer limits the structure of a (join) query tree to that of left-deep (or right-deep) trees
- **Left-deep tree is a binary tree where right child of each nonleaf node is always a base relation**
- **Optimizer will choose the particular left-deep tree with lowest estimated cost**

LEFT-DEEP JOIN TREE (1/2) – PLEASE STUDY YOURSELF





LEFT-DEEP JOIN TREE (2/2) – PLEASE STUDY YOURSELF

- With left-deep trees, the **right child is considered to be the inner relation when executing a nested-loop join**

- Advantages of left-deep (or right-deep) tree
  - They are amenable to pipelining
  - Having a base relation as one of the inputs of each join allows the optimizer to utilize any access paths on that relation that may be useful in executing the join