Database Systems

Query Processing

Last Lecture…

- Indexes

- Any questions?

This Lecture…

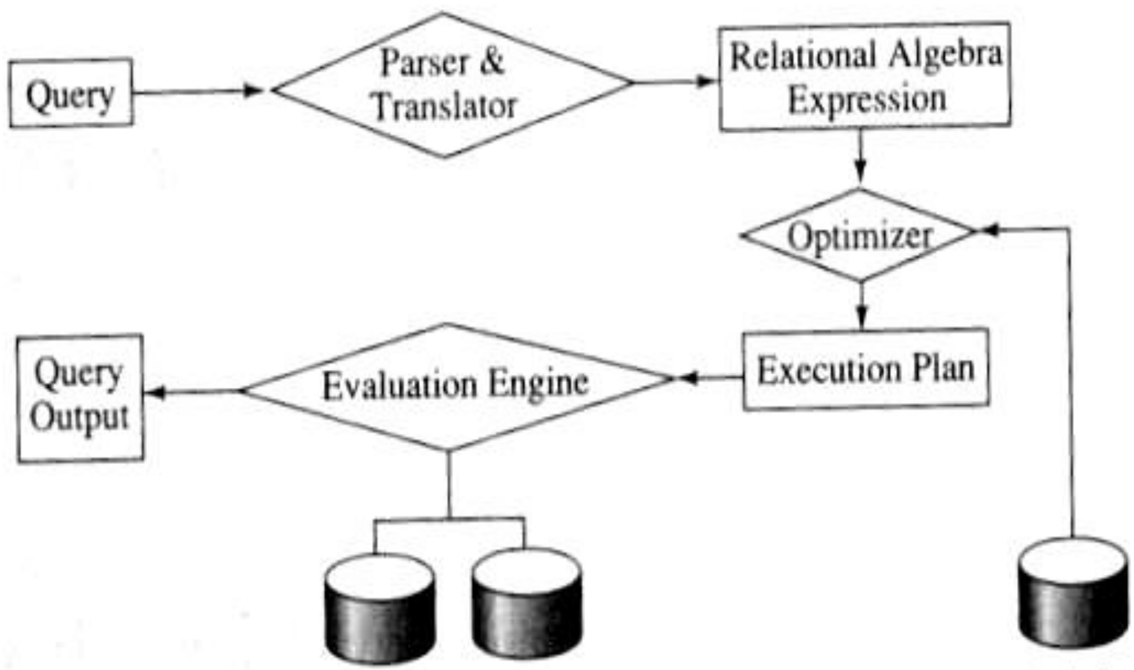
- Query Processing: Overview
- What?
- Why?
- How? (Basics)

Query Processing

- What happens to a query inside the DBMS?
- Query processing considers this issue.

Steps in QP…

- The steps involved in query processing include…
  - Parsing and translation
  - Optimization
  - Evaluation



Steps in QP… (contd.)

- Parsing & translation step verifies the correctness of the query and converts the query into an internal form (usually an extended relational algebra expression)

- This internal form is either a tree (i.e. query tree) or a graph (i.e. query graph)

Steps in QP… (contd.)

- Next, an efficient execution strategy for retrieving the results of the query from the database files is generated. This step is called query optimization.
- This is the heart of query processing in a relational database system
- The evaluation engine executes the query according to the chosen plan

Parsing & Translating…

- The first step in query processing is to convert the query into an form that can be executed
- Consider the following schema

  S(sno, sname, status, city)

  P(pno, pname, colour, weight)

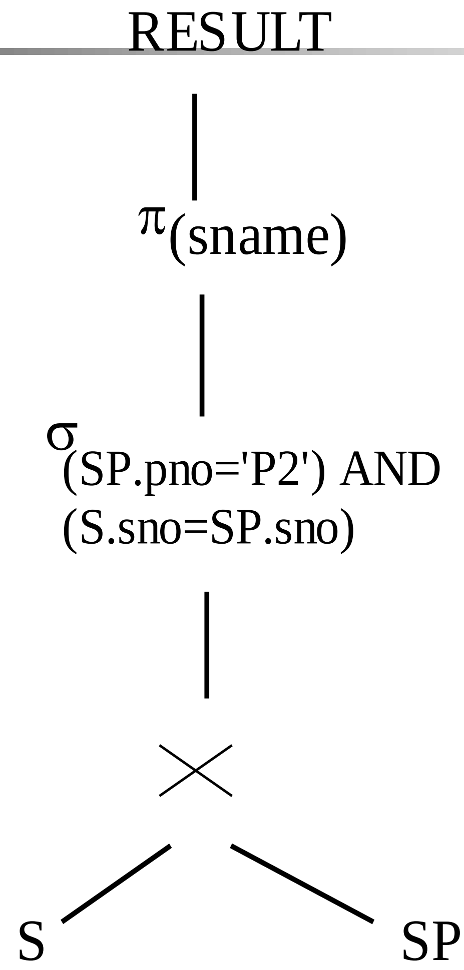
  SP(sno, pno, qty)

Parsing & Translating… (contd.)

```
SELECT   s.sname
FROM     S, SP
WHERE    S.sno = SP.sno AND
         SP.pno = 'P2'
```

We can express this query as a relational algebra as follows…

Parsing & Translating… (contd.)



Parsing & Translating… (contd.)

- Usually, a SELECT-FROM-WHERE-GROUP BY called a ***query block*** is converted an extended relational algebra expression

- There can be many query blocks (i.e. with nested queries) in a complex query

Why do we need Query Optimization?

- Consider the following number of tuples for S and SP

S	–	100 pages
SP	-	10000 pages

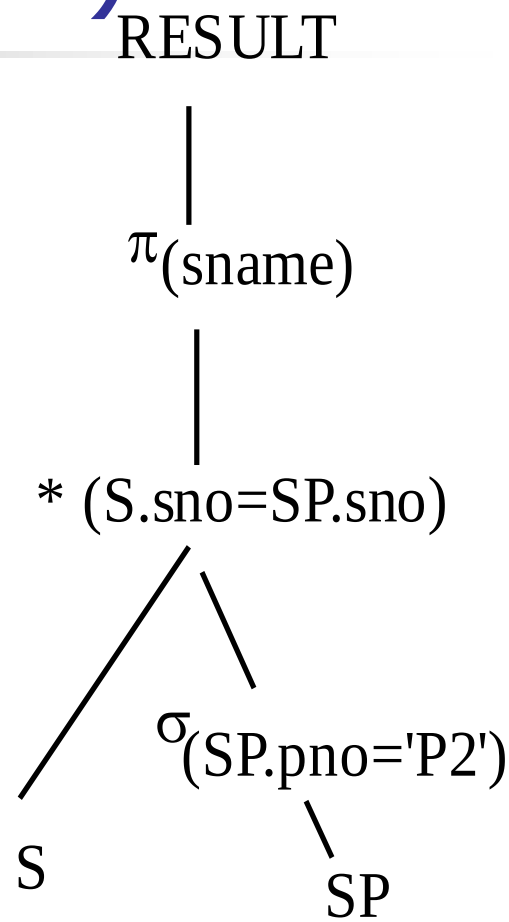
- Cost for computing Cartesian Product:
  - read 10,000 * 100 pages
  - write 1,000,000 pages (intermediate result)
- Selection
  - read 1,000,000 pages
  - keep 50 tuples (assuming 50 tuple SP.pno = ‘P2’)
- Total number of disk I/O ~ 3,000,000 disk I/Os

Optimization? (contd.)

- Consider the following relational algebra, which produces the same result

- Selection operator
  - Read 10,000 pages
  - Keep the result, 50 tuples in memory
- Join with S
  - Read 100 pages

Total cost 10,100 disk I/Os



Heuristic Optimization

- A query can have many equivalent query trees
- Optimizer must find an efficient query plan to execute
- Heuristic rules are used for algebraic optimization

Equivalence Rules…

- To transform a relational algebra expression from to an equivalent efficient query expression, certain equivalence rules are used.

Equivalence Rules… (contd.)

- Conjunctive selection operations can be deconstructed into a sequence of individual selections. This transformation is referred to as a cascade of σ.

$$\sigma_{C1 \wedge C2}(E) \quad = \quad \sigma_{C1} \; (\sigma_{C2} \, (E))$$

- Selection operations are commutative.

$$\sigma_{C1} \; (\sigma_{C2} \, (E)) \quad = \quad \sigma_{C2} \; (\sigma_{C1} \, (E))$$

- Cascade of Π.

$$\Pi_{L1} \, (\, \Pi_{L2} \, (\ldots \, (\Pi_{Ln} \, (E) \ldots)) \quad = \quad \Pi_{L1} \, (\, E \,)$$

Equivalence Rules… (contd.)

- Selections can be combined with Cartesian products and theta joins.

  a. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E2$

- This expression is just the definition of the theta join.

  b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta 2} E2) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$

Etc.

Equivalence Rules (contd.)

- Equivalence rules states that two expressions are equal
- It does not state, which one is better

- A large number of plans are possible! Estimating the cost for each is prohibitively expensive

- The optimizer uses heuristic rules to prune the plan space (reduce the number of plans to be considered)
  - E.g. Bring selects down, avoid cartesian products, etc.

Indexes and cost of query plans…

- Using an index does not necessarily mean efficient query plan.
- Can you think of an instance where using an index is inefficient?
- Query optimizer estimates costs to compare different execution plans

Cost estimation

- Execution plan has
  - A set of relational algebra operators (query plan) to obtain the result of the query
  - Algorithm used to evaluate each relational algebra operators

- Some relational algebra operators have many possible ways (algorithms).

- Costs may differ significantly based on the chosen algorithm

- We will study algorithms some algorithms used for simple selections and joins

Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

Reserves:

- Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.

- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

Simple Selections

```
SELECT  *
FROM    Reserves R
WHERE   R.rname < 'C%'
```

- Of the form $\sigma_{R.attr\ op\ value}(R)$
- Size of result approximated as *size of R * reduction factor*; we will consider how to estimate reduction factors later.
- With no index, unsorted: Must essentially scan the whole relation; cost is M (#pages in R).
- With an index on selection attribute: Use index to find qualifying data entries, then retrieve corresponding data records. (Hash index useful only for equality selections.)

Using an Index for Selections

- Cost depends on #qualifying tuples, and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!

Equality Joins With One Join Column

```
SELECT  *
FROM    Reserves R1, Sailors S1
WHERE   R1.sid=S1.sid
```

- In algebra: R $\bowtie$ S. Common! Must be carefully optimized. R $\times$ S is large; so, R $\times$ S followed by a selection is inefficient.

- Assume: M tuples in R, p_R tuples per page, N tuples in S, p_S tuples per page.
  - In our examples, R is Reserves and S is Sailors.

- *Cost metric*: # of I/Os. We will ignore output costs.

Simple Nested Loops Join

foreach tuple r in R do
    foreach tuple s in S do
        if $r_i == s_j$ then add <r, s> to result

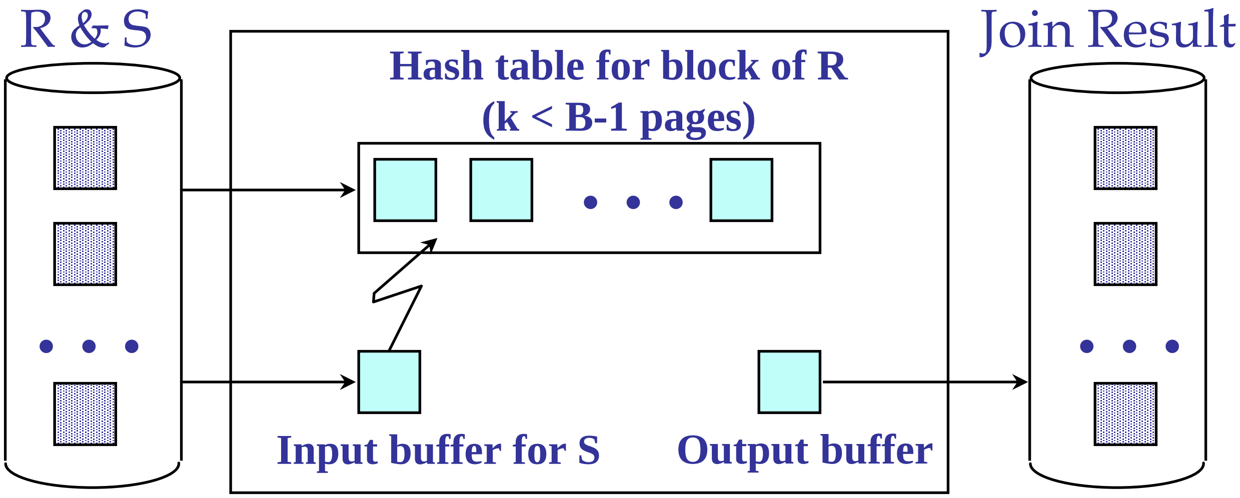
- For each tuple in the *outer* relation R, we scan the entire *inner* relation S.
  - Cost: $M + p_R * M * N = 1000 + 100*1000*500$ I/Os.

Page-oriented Nested Loop Join

- Page-oriented Nested Loops join: For each *page* of R, get each *page* of S, and write out matching pairs of tuples <r, s>, where r is in R-page and S is in S-page.
  - Cost: M + M*N = 1000 + 1000*500
  - If smaller relation (S) is outer, cost = 500 + 500*1000

Block Nested Loops Join

- Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold ``block” of outer R.
  - For each matching tuple r in R-block, s in S-page, add <r, s> to result. Then read next R-block, scan S, etc.



Examples of Block Nested Loops

- Cost: Scan of outer + #outer blocks * scan of inner
  - #outer blocks = $\lceil \# \ of \ \ pages \ of \ \ outer \ / \ blocksize \rceil$
  - Block size = available buffers - 2
- With Reserves (R) as outer, and 100 pages of R:
  - Cost of scanning R is 1000 I/Os; a total of 10 *blocks*.
  - Per block of R, we scan Sailors (S); 10*500 I/Os.
  - If space for just 90 pages of R, we would scan S 12 times.
- With 100-page block of Sailors as outer:
  - Cost of scanning S is 500 I/Os; a total of 5 blocks.
  - Per block of S, we scan Reserves;   5*1000 I/Os.

Index Nested Loops Join

foreach tuple r in R do
    foreach tuple s in S where $r_i == s_j$ do
        add <r, s> to result

- If there is an index on the join column of one relation (say S), can make it the inner and exploit the index.
  - Cost: $M + ((M*p_R) *$ cost of finding matching S tuples)

- For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.
  - Clustered index: 1 I/O (typical), unclustered: upto 1 I/O per matching S tuple.

Sort-Merge Join $(R \bowtie_{i=j} S)$

- Sort R and S on the join column, then scan them to do a "merge" (on join col.), and output result tuples.
  - Advance scan of R until current R-tuple >= current S tuple, then advance scan of S until current S-tuple >= current R tuple; do this until current R tuple = current S tuple.
  - At this point, all R tuples with same value in Ri (*current R group*) and all S tuples with same value in Sj (*current S group*) *match*; output <r, s> for all pairs of such tuples.
  - Then resume scanning R and S.

Example of Sort-Merge Join

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

- R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)
- Cost: O(M log M) + O(N log N) + (M+N)
  - The cost of scanning, M+N, could be M*N (very unlikely!)

Cost-based Optimization

- The fact that there are many algorithms for relational operators, choosing a good query plan using heuristics is not enough

- We can calculate the cost of each candidate query tree with the possible algorithms for each operator (& the difference can be significant)

- To compare such query plans, cost-based optimization techniques are used

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Statistics and Catalogs

- Need information about the relations and indexes involved. ***Catalogs*** typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.

Statistics and Catalogs

- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- *Reduction factor (RF)* associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples * product of all RF's.
  - Implicit assumption that *terms* are independent!

  System R:
  - Term *col=value* has RF *1/NKeys(I),* given index I on *col*
  - Term *col1=col2* has RF *1/MAX(NKeys(I1), NKeys(I2))*
  - Term *col>value* has RF *(High(I)-value)/(High(I)-Low(I))*

Summary

- Query Processing consists of several steps
- Query optimization is an important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query:
  - Consider a set of alternative plans (*heuristics are used to reduce the number of possible plans to consider*).
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues*: Statistics, indexes, operator implementations.