Query optimization

Iztok Savnik, FAMNIT

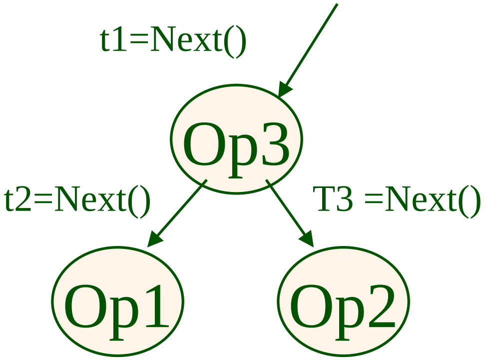
Slides & Textbook

- Textbook:
  - Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems, McGraw-Hill, 3rd ed., 2007.*
- *Slides:*
  - *From „Cow Book“: R.Ramakrishnan, http://pages.cs.wisc.edu/~dbbook/*

Introduction to query optimization

- Contents
  - Query evaluation plan
  - Translate SQL into RA
  - Cost estimation
  - Problem definition
  - Solution space
  - Enumeration of plans
  - Search algorithms
  - System R

Program in DBMS

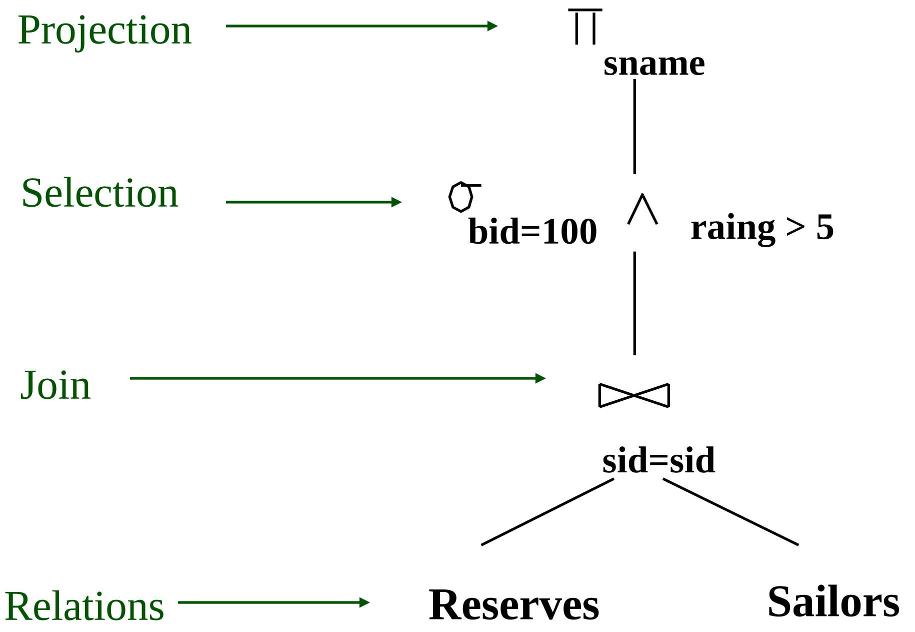


- *What is program in DBMS?*
  - Tree of relational algebra operators annotated with the algorithms.
  - Algorithm is determined for each particular operation.
  - Also called query evaluation plan (QEP).
- Tree of iterators
  - Operators are implemented as iterators (scans)
  - Interface: open(), next(), close()
  - Call next() triggers calls next() at children
  - Results of children are processed
  - The constructed tuples are send to parents

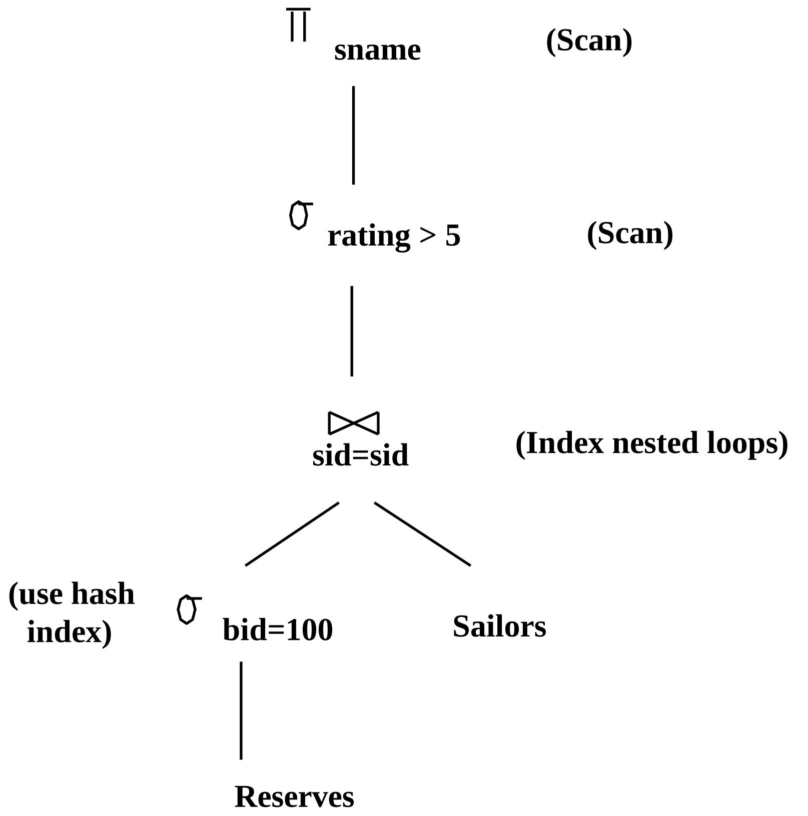
Program in DBMS

- It is a tree structured program (often pipelines) where streams of tuples flow among operators
  - At some points the flow can stop for sometime to materialize a table.
  - Disk pages are read in the leafs of query plans.
  - The tuples constructed in RA operators are sent up to the parent operators.

Example of RA operator tree



RA operator tree annotated with algorithms (Query Evaluation Plan)



Query optimization

- How to optimize a tree of RA operators?
  - Logical and physical optimization (there are many interactions!)
  - Optimal query evaluation plan is the result
- Logical optimization
  - Searching for logically equivalent query expressions
  - To determine the cost of a query we need to have physical QEP
- Physical optimization
  - Physical algebra is used that contains only two operators: (1) access paths and (2) joins.
  - Searching for optimal methods for evaluating RA operators.
  - Physical optimization intervenes with the logical optimization.

Translating SQL into RA

- Decompose SQL queries into blocks
  - Query block is a sequence of joins.
  - SQL block is treated as a procedure.
  - Nested block must be called in each iteration of the subsuming block.
- Query block is translated to a RA expression
  - A block and its nested blocks are optimized separately.
  - No optimization among blocks.
  - Join structured queries have more chance to be properly optimized.

Equivalences of RA expressions

- Query optimization of RA expressions
  - Enumerate all possible equivalent RA expressions.
  - Assign all possible implementations to operators.
  - Estimate the cost of all query plans (QEP) and select the best one.
- Enumerate equivalent RA expressions
  - *Logical algebra*: the rules are presented shortly.
  - *Physical algebra*: join re-ordering, pushing down Π and σ.
- Algebraic properties of RA operations allow optimization.

Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- Selections: $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots\ \sigma_{cn}(R))$ (Cascade)

  $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (Commute)
- Projections: $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ (Cascade)
- Joins: $R \bowtie (S \bowtie T) \quad (R \bowtie S) \bowtie T$ (Associative)

  $(R \bowtie S) \quad (S \bowtie R)$ (Commute)

☞ Show that: $R \bowtie (S \bowtie T) \quad (T \bowtie R) \bowtie S$

More Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection based on attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with ⋈.

$$\delta(R \bowtie S) \equiv \delta(R) \bowtie S$$

- Similarly, if a projection follows a join, we can `push' it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

$$\Pi(R \bowtie S) \equiv \Pi(R) \bowtie S$$

Cost of a query

- Result is always an estimation.
- Statistics is stored in the system catalogs.
- Statistics is used for estimating the cost.
  - Cost of query: response time, total time
  - Sizes of the intermediate results
- The costs for the particular operations were presented in previous lecture.
  - Sequential scan, Index-based scan, NL join, Index NL join, sort-merge join, itd.
- Take into account the cost of CPU and I/O.
  - We use # of block read/write operations
  - More and more processing in RAM recently

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.
- 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.

Selectivity estimations for RA operations

- *Assumptions:*
  - Attribute values are distributed uniformly.
  - Attributes and terms are mutually independent.
- Selection:
  - Term A=value has SP = 1 / Nkeys(A)
  - Term A_1=A_2 has SP = 1 / MAX(Nkeys(A_1),Nkeys(A_2))
  - Term A>value has SP = (High(A)-value)/(High(A)-Low(A))
  - $SP(p(A_i) \wedge p(A_j)) = SP(p(A_i))*SP(p(A_j))$
  - $SP(p(A_i) \vee p(A_j)) = SP(p(A_i)) + SP(p(A_j)) – (SP(p(A_i)) * SP(p(A_j)))$
  - $SP(A \in \{value\}) = SP(A= value) * card(\{values\})$
- Projection:
  - Selectivity of projection is SP = 1.

Selectivity estimations for RA operations

- Join:
  - $R \bowtie_{A_1=A_2} S$, $A_1 \in R$ *and* $A_2 \in S$.
  - Join is defined with condition $A_1=A_2$.
  - Term $A_1=A_2$ *has* SP = $1 / MAX(Nkeys(A_1),Nkeys(A_2))$.
  - *Remember assumptions.*

Size estimations for RA operations

- Selection
  - $card(\sigma_F(R)) = SP(F) * card(R)$
- Projection
  - $card(\Pi_A(R))=card(R)$
- Cartesian product
  - $card(R \times S) = card(R) * card(S)$
- Join
  - *A* is a key of *R* and a foreign key of *S:* $card(R \bowtie_{A=B} S) = card(S)$
  - General case: $card(R \bowtie S) = SP * card(R) * card(S)$

Size estimations for RA operations

- Union
  - Upper bound: $card(R \cup S) = card(R) + card(S)$
  - Lower bound: $card(R \cup S) = max\{card(R), card(S)\}$
- Razlika
  - Upper bound: $card(R – S) = card(R)$
  - Lower bound: 0

Cost of access path

- Access path to a table has input data:
  - Logical expression including atomic terms used as parameters for accessing tables.
  - Indexes defined on table need to be considered.
  - Enumerate all possible access paths!
- Cost of an access path depends on
  - Selectivity of atomic terms used for access.
  - Available indexes.
- Cost of AP = cost of reading a fraction of table from disk using an access method.
  - The best AP selects the least number of tuples

Cost Estimates for Single-Relation Plans

- Index I on primary key matches selection:
  - *Cost is Height(I)[+1] for a B+ tree, about 1.2[+1] for hash index.*
- Clustered index I matching one or more selects:
  - *(NPages(I)+NPages(R)) * product of SP's of matching selects.*
- Non-clustered index I matching one or more selects:
  - *(NPages(I)+NTuples(R)) * product of SP's of matching selects.*
- Sequential scan of file:
  - *NPages(R).*

☞ ***Note:*** *Typically, no duplicate elimination on projections! (Exception: Done on answers if user says* DISTINCT*.)*

Cost of plans on multiple relations

```
SELECT select-list
FROM relation-list
WHERE  term1 AND...AND termk
```

- Query block:
- Maximal # tuples in the results is a product of the sizes of the input relations.
- *Selectivity* of each term influences the size of the final result.
  - *Size of result* = Max # tuples * product of all SPs.
- Multi-relational plans are often built from a plan for a sequence of joins by adding one more relation to the end of a join sequence.
  - Cost of join, cost of access path, estimation of size of the result.

Schema for Examples

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

- Similar to old schema; *rname* added for variations.
- 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.

Example 1: Selectivity of an access method

- Hash-based index H on <sid> from table Sailors
- Size of index: 160*1.25 = 200 pages
  - Data entries: //ptr+int// (8+8)*40000/4000 = 160 pages
- Selection condition: “sid=19“
- Catalog: Nkeys(H), Npages(Sailors)
- Selectivity: 1/Nkeys(H) = 1/40000
- Cost estimation:
  - Clustered: (200+500) * 1/40000 = 1 page
  - Unclustered: (200+40000) * 1/40000 = 2 pages

Example 2: Cost of an access method

- Range selectivity
- Condition: day > 12/12/12 on Reserves
- B+ tree T on attribute day: 400*1.5 = 600 pages
  - 1000*100*(8+8) /4000 = 400 pages (approxim. # no index pgs)
- Selectivity: (High(T) – value)/(High(T)-Low(T))
  - Assume: first reservation on 1/1/2000
  - (2023-2012) / (2023-2000) = //years only// ½
- Cost
  - Clustered: (600+1000) * ½ = 800 pages
  - Unclustered: (600+100000) * ½ = 50300 pages

Query optimization - definitions

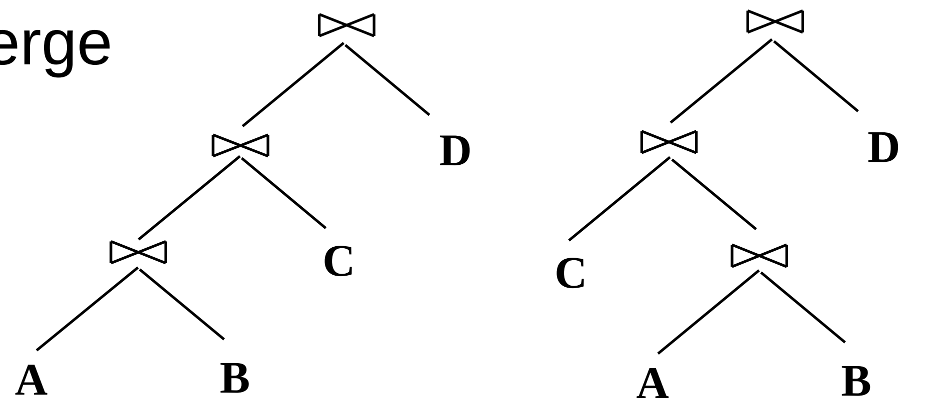
- Let *q* be a query issued on the database *B*
- Find the cheapest evaluation plan
  - Optimization algorithm
    - System R uses dynamic programming
  - *Cost estimation function* computes the cost of plan.
  - For each operation estimate the cost of all the algorithms (for evaluation of op.) that can be employed; best algorithm is selected.
- Optimization algorithms
  - Exhaustive search
  - Dynamic programming (System R)
  - Random search, genetic algorithms, etc.

Query optimization - definitions

- Cost estimation function should take into account
  - Number of blocks read from disk.
  - CPU used for computation of RA operators (we ignore CPU cost).
  - Storing intermediate temporary tables.
  - The amount of RAM used in particular algorithm.
- We use only number of read/write disk pages.

Abstraction of the problem

- Physical operations
- Access method query node
  - Access to the tuples
  - AM with index, AM with sorting, AM file scan, ...
  - Includes also the selection and projection operations
- Join query node
  - Various algorithms for join
  - Nested loops joins, sort-merge join, hash join, ...
  - Includes also the selection and projection operations



Solution space

- Space of equivalent query expressions
  – Equivalent class of a query
- Properties of relational algebra allow transformation of queries
  – Transformed query returns the same result!
  – Transformed query allows different QEP
  – The algorithms for RA ops need to be determined (again)
- We are searching for an expression that
  – Reads the least number of blocks
  – Best (least) execution time

Typical relational optimizer

- Opt. algorithm based on dynamic programming
  - Optimal plans are built bottom-up
  - Optimal solutions to problem with n joins
    - Adding optimally one join to optimal query with n-1 joins
  - Restrict solution space, left-deep plans, no materialization possible
  - Exhaustive search would enumerate all permutations
  - Dynamic programming is still exponential
- Optimizer of System R

Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans

Queries on one relation

- Combination of selection, projection and aggregation operations.
  - No joins!
- Enumerate and check all possible access methods.
  - Methods without indexes
    - File scan, sorted file scan
  - Methods with indexes
    - Index scan, index-only, more than one index, sorted index
  - Selection and projection are integrated
  - Results (in sorted order) pipelined to aggregation
    - Either existing ordering (last result), or sorting is used
- The method with the lowest price selected.

Example 3

```
SELECT  M.sid
FROM  Sailors M
WHERE  M.rating=8
```

- Tree index on attribute rating
  - (1/NKeys(I)) * NRecords(R) = (1/10) * 40000 = 4000 records.
  - Clustered index

    Index size: 12 //4+8// * 40000 / 4000 // page size // -> 120 *1.5 = 180

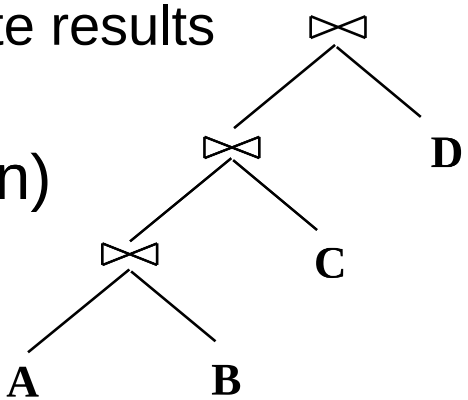
    Assume index 67% full

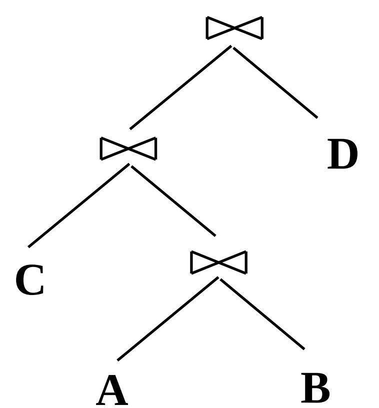
    (1/NKeys(I)) * (NPages(I)+NPages(R)) = (1/10) * (180+500) = 68 pages
  - Unclustered index

    (1/NKeys(I)) * (NPages(I)+ NRecords(R)) = (1/10) * (180+40000) = 4018 pg
- Index on s*id*:
  - All pages of index and file must be read !
  - Hash index, size = 40000*(8 + 8)/4000 = 160 pages * 1.25 = 200 pgs
  - Clustered index = 200+500 pages,
    Unclustered index = 200+40000 pages, Not good !
- Sequential scan
  - |Sailors| = 500 pages

Queries Over Multiple Relations

- Solution space is too big so we constraint it to some sub-space
  - The use of heuristics, or restrict the structure of RA trees.
  - Exhaustive search is also used... (<10 joins)
- Which part of the space we choose?
  - Depends on the search algorithm
  - System R uses solely left-deep plans
    - Left-deep trees allow the implementation of the pipeline
    - No need to store intermediate results
  - Zig-zag trees, bushy trees
    (also allow parallel execution)





Enumeration of Left-Deep Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

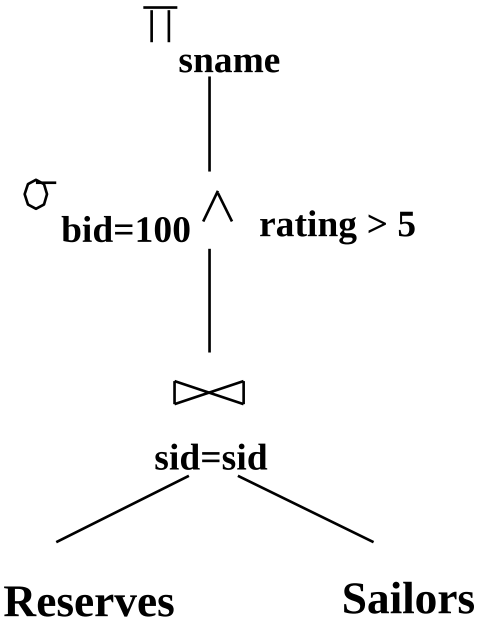
Enumeration of Plans (Contd.)

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an `interestingly ordered' plan or an addional sorting operator.
- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- In spite of pruning plan space, this approach is still exponential in the # of tables.

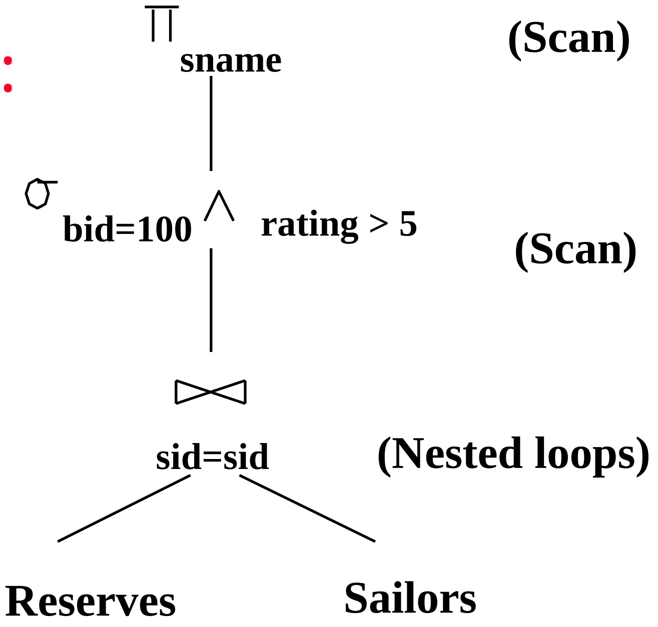
Example 4

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

RA tree:

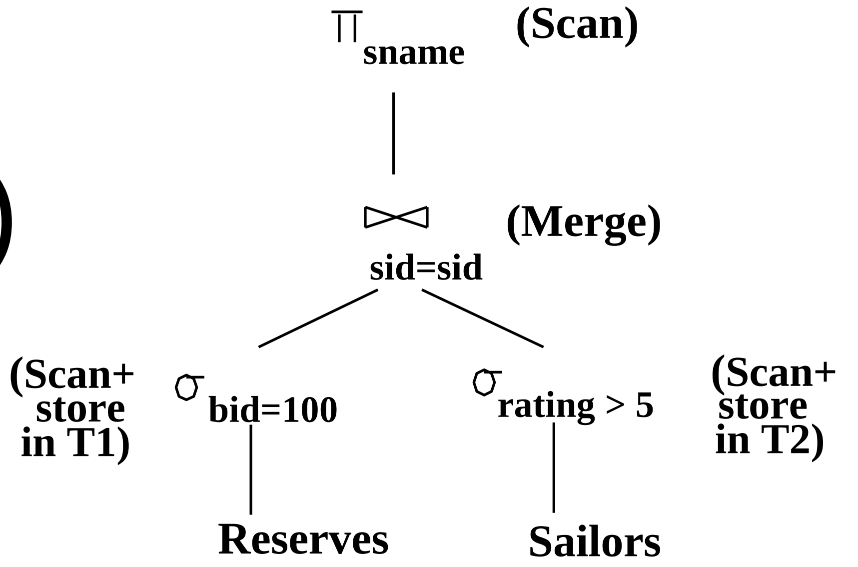


Plan:



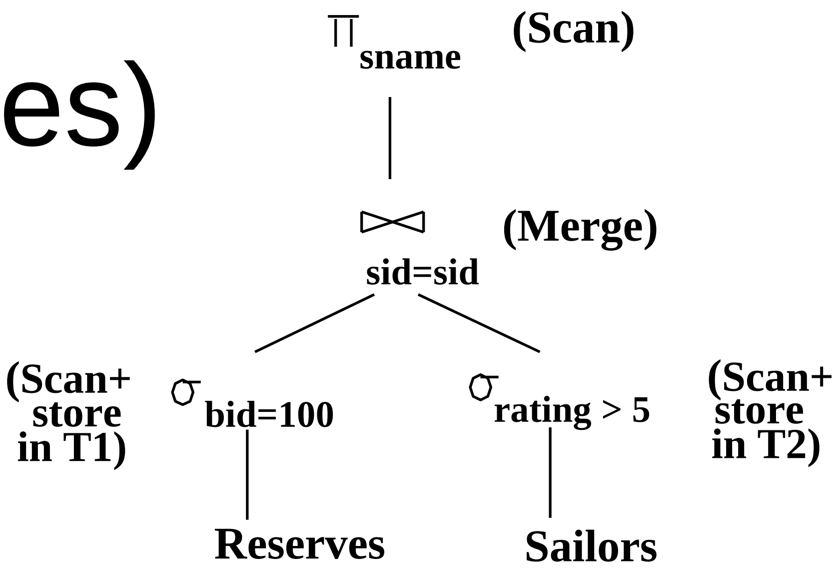
- Worst plan!
- Price: 1000+1000*500[*100] V/I blocks
- Selections can be pushed towards the leafs.
- No indexes
- *Optimization goal:* Search for optimal plan that computes the same result.

Plan 1
(without indexes)



- Main difference
  - Push down selections.
- Price is better
  - Table after selection of Reserves is small.
  - Table after selection on Sailors is smaller than original.
  - Join is done on small tables.
- Not all table attributes are moved among the nodes.
  - Push down projections.
  - Only those attributes that are needed are retained.
  - Example: T1 includes the attribute s*id,* T2 has attributes *sid* and *sname.*
  - Size of tuples contributes to the cost.

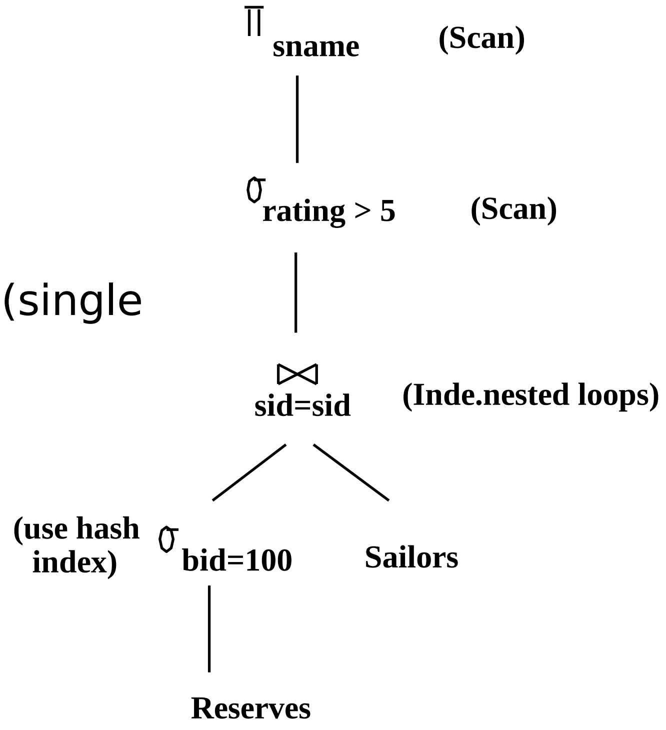
Plan 1
(without indexes)



- Plan cost:
  - Scan Reserves (1000) + write T1 (10 pages, 100 boats), unform distribution).
  - Scan Sailors (500) + write T2 (250 pages, 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*3*250), merge (10+250)
  - Sum: 1010+750+40+1500+260 = 3560 I/O blocks

Plan 2 (with index)

- Speed up selection on Reserves
  - Attribute b*id* in table Reserves is very selective (single boat).
  - Use hash index on b*id* (clustered/unclustered)
  - Size of HI: 16*1000*100/4000->400*1.25=500
- Speed up join
  - Hash index on *sid* from table Sailors
  - Small num. of tuples from outer relation
  - Projection is often joined with selection
- Access Reserves
  - Clustered = (500+1000)/100=15
  - Unclustered = (500+1000*100)/100=1005
- Index nested loops join
  - Cost = Scan Reserves + 1000 //sel.tuples// * (1.2 [+1])
  - Cost = 1215 – 3205 I/O blocks



Nested Queries

```
SELECT  S.sname
FROM  Sailors S
WHERE EXISTS
   (SELECT  *
    FROM  Reserves R
    WHERE  R.bid=103
    AND  R.sid=S.sid)
```

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of `calling' nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered. *The non-nested version of the query is typically optimized better.*

Nested block to optimize:

```
SELECT  *
FROM  Reserves R
WHERE  R.bid=103
   AND  S.sid= outer value
```

Equivalent non-nested query:

```
SELECT  S.sname
FROM Sailors S, Reserves R
WHERE  S.sid=R.sid
   AND R.bid=103
```

Summary

- 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.
    - Must prune search space; typically, left-deep plans only.
  - 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.

Summary (Contd.)

- Single-relation queries:
  - All access paths considered, cheapest is chosen.
  - *Issues*: Selections that *match* index, whether index key has all needed fields and/or provides tuples in a desired order.
- Multiple-relation queries:
  - All single-relation plans are first enumerated.
    - Selections/projections considered as early as possible.
  - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
  - Next, for each 2-relation plan that is `retained', all ways of joining another relation (as inner) are considered, etc.
  - At each level, for each subset of relations, only best plan for each interesting order of tuples is `retained'.