Query Optimization

Cost-Based

SWEN 304

Trimester 2, 2017

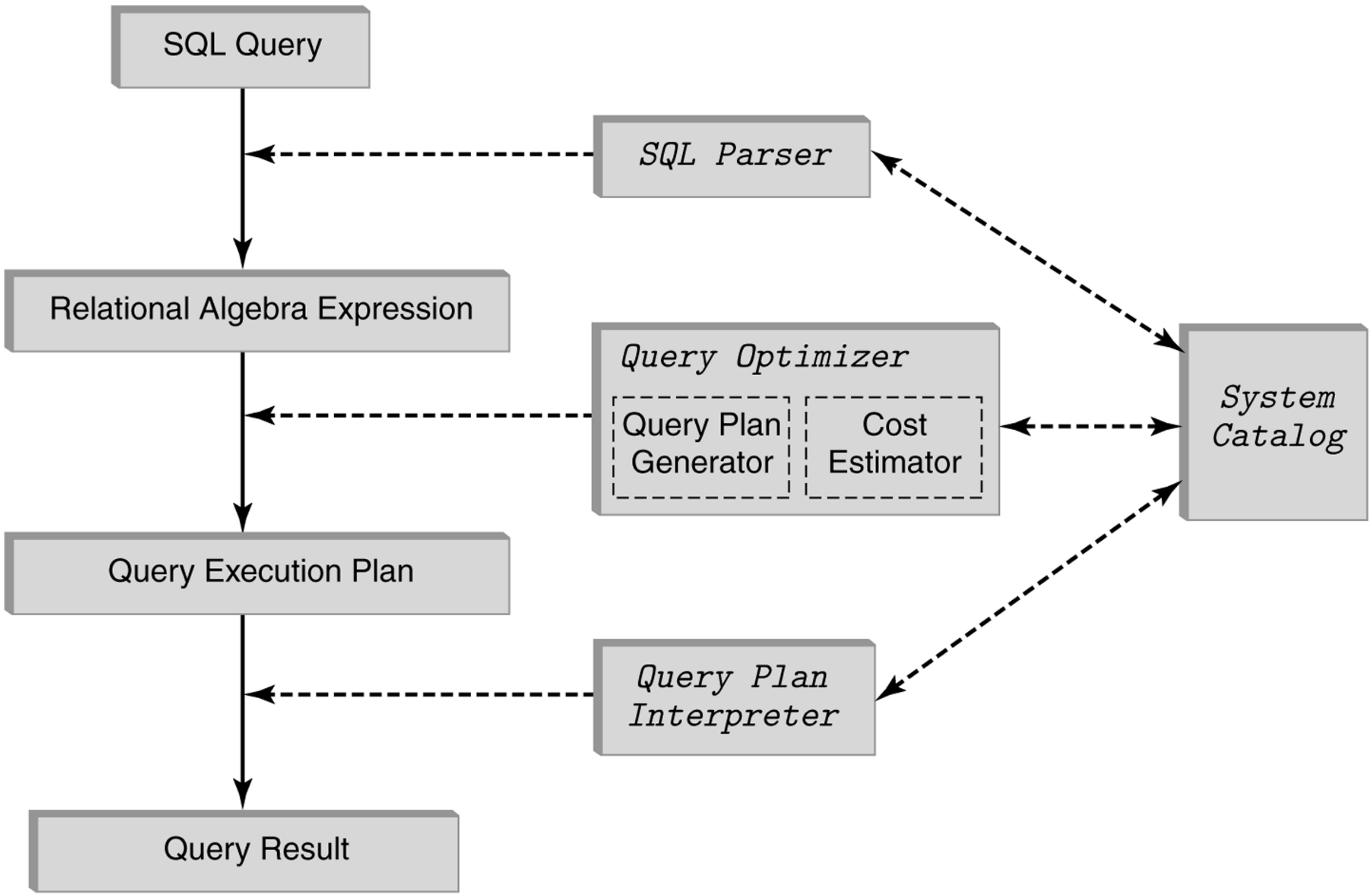
Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Slides by: Pavle Morga & Hui Ma

Outline – Query Evaluation in DBMS



Outline – Cost-based Query Optimization

- Why cost-based optimization?
- How to measure cost of query operations?
  - Cost function of a projection, selection, join, …
- How to evaluate query operations?
  - Algorithms for projection, selection, join, …
- Query tree of physical operators
    - Reading: chapters 17, 18, 19 of 6/E of the textbook
    - Supposed knowledge File Organization

Example

R (Reserves)

sid	bid	day
58	101	10/09/13
22	103	11/09/13
31	103	11/09/13

S (Sailor)

sid	sname	rating	age
22	Jerry	7	25
31	Tom	8	30
58	Minny	10	22

B (Boats)

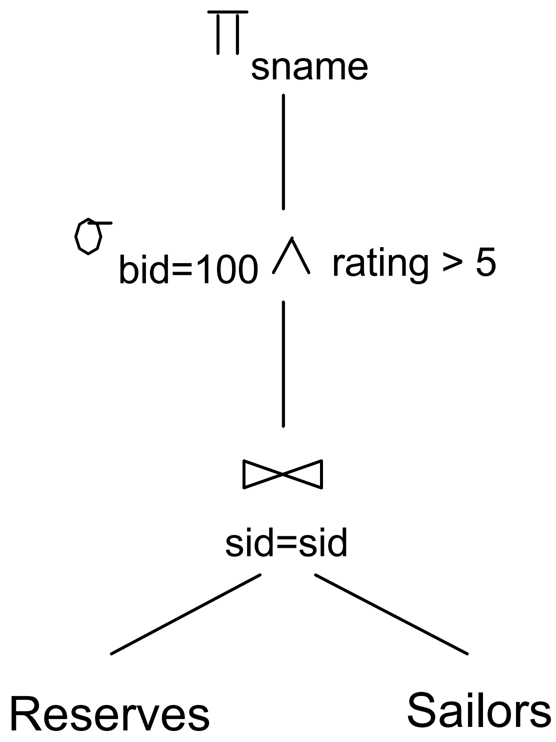
bid	bname	color
101	Dragon	blue
102	Moon	red
103	Star	green
104	Clipper	blue
105	Mary	green

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

Example

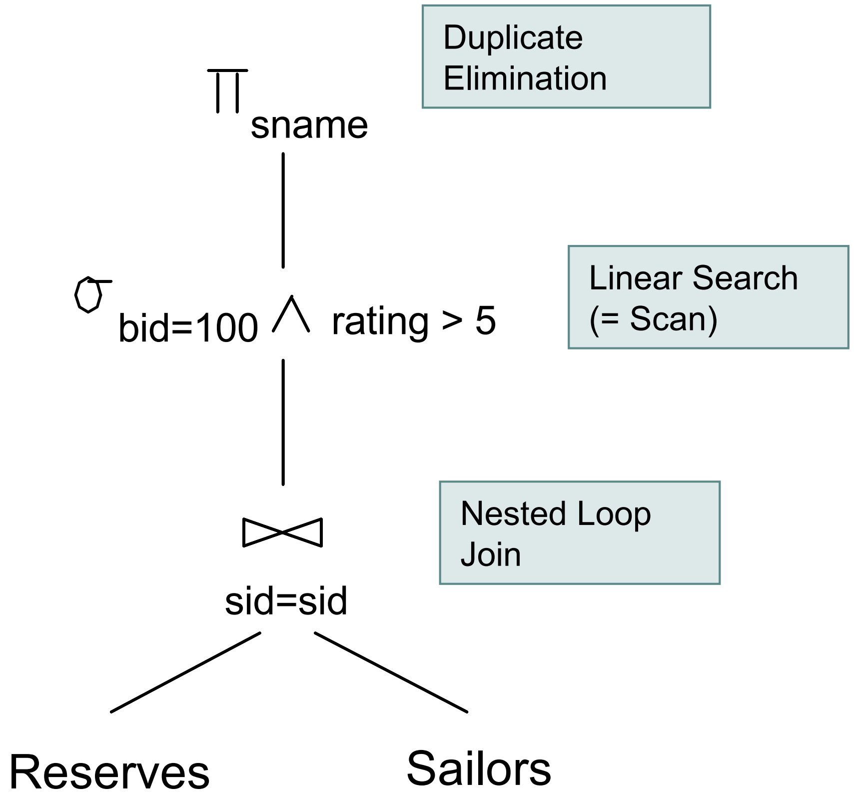
- Here is a query tree for the SQL query
  - Is this optimal?
  - How good is it?
  - How to measure "goodness"?
  - Need to estimate its costs!

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



Example

- Here is an execution plan:
  - How to evaluate the operations?
  - Need to pick an evaluation algorithm for each operation
  - DBMS typically offer multiple evaluation algorithms
    - that have been implemented by the DBMS programmers



Cost-Based Optimization

- A good query optimizer does not rely solely on heuristic rules
- It chooses that query execution plan which has the lowest cost estimate
  - Or at least one whose costs are reasonably good
- After heuristic rules are applied to a query, there still remain a number of alternative ways to execute it
  - These alternative ways relay on the existence of different auxiliary data structures and algorithms
- Query optimizer estimates the cost of executing each of alternative ways, and chooses the one with the lowest cost

Cost Components of a Query Execution

- Secondary storage access cost:
  - Reading data blocks during data searching,
  - Writing data blocks on disk, and
  - Storage cost (cost of storing intermediate files)
- Computation cost (CPU cost)
- Main memory cost (buffer cost)
- Communication cost
- Very often, only secondary storage access cost is considered
- So, the cost *C* will be the number of disk accesses

Cost Related Catalog Content

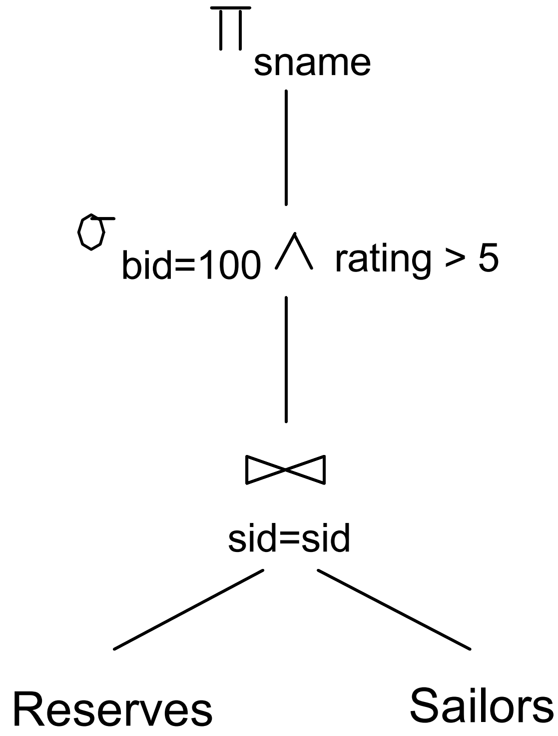
- For the purpose of a query cost estimate, a Catalog should contain following information for each base relation:
  - Number of tuples (= records) *r*
  - Number of blocks *b*
  - Blocking factor *f* (= the number of tuples that fit into one block
  - Available access methods and access attributes:
    - Access methods: sequential, indexed, hashed
    - Access attributes: primary key, indexing attributes, sort key
  - The number of levels *h* of each index
  - The number of distinct values *d* of each attribute

Some Assumptions

- To make it simpler, we shall suppose that:
  - All tuple fields are of a fixed size (although variable field size tuples are very frequent in practice)
  - All the intermediate query results are materialized (although there are some advanced optimizers that apply pipelined approach)
    - Materialized: intermediate query results are stored on a disk as temporary relations
    - Pipelining: tuples of the intermediate results are subjected to all subsequent operations without temporary storing
  - Intermediate results produced by unary operations are retained in blocks of the size as the initial files

Example

- Materialized evaluation:
  - evaluate one operation at a time, starting from the bottom
  - intermediate results are materialized into temporary relations (write to disk)
  - E.g., the result of the join is materialized, the temporary relation is then read from disk to compute the selection
  - Similarly, the result of the selection is materialized, and then from disk to compute the projection



Example

- Materialized evaluation is always applicable
  - Cost of writing results to disk and reading them back can be quite high
- Pipelined evaluation:
  - evaluate several operations together, passing the results of one operation on to the next
  - E.g., don't store result of join, but pass them on to selection
  - Similarly, don't store result of selection, but pass them on to projection
  - Much cheaper than materialization: no need to store a temporary relation to disk

Cost Function of a Project Operation

- Suppose the <attribute_list> of a project operation
  - contains a relation schema key, or
  - the keyword DISTINCT is not used in the SQL SELECT command,
- Then project operation
  - Reads all b_1 blocks, containing r tuples of the size n from the secondary storage into memory, and
  - Writes back b_2 blocks, containing r tuples of the size m ($m < n$), on the secondary storage
- Since $m < n$, it follows $b_2 < b_1$
  - The cost function is

$$C = b_1 + b_2$$

- Complexity $O(r)$

Projection: Evaluation of DISTINCT

- Suppose the <attribute_list> of the SELECT clause does not contain a relation schema key and the keyword DISTINCT is used
- Algorithm:
  - Read b_1 blocks form the base relation,
  - Drop unwanted columns and write b_2 blocks with duplicate tuples back
  - Look for duplicate tuples by reading in each of b_2, and comparing its tuples with tuples in the other blocks,
    - Number of reads: $\sum_{i=2}^{b_2} i = \frac{b_2^2 + b_2 - 2}{2}$
  - Complexity O(r^2)

Projection: Reduce Cost of DISTINCT with Sorting

- Algorithm:
  - Read b_1 blocks form the base relation,
  - Drop unwanted columns and write b_2 blocks with duplicate tuples back
  - Sort b_2 blocks (Complexity O($r*logr$))
  - Read successive blocks from the sorted file and drop all duplicates (except one)
  - Write back b_3 blocks without duplicates

- Overall complexity O($r*logr$)

Projection: Reduce Cost of DISTINCT with Index

- For SELECT DISTINCT <attr_list> a costly sort can be avoided if there exists an appropriate secondary index on the whole <attr_list>
- Then, the query optimizer only has to traverse index, to retrieve all different secondary key values, and write them back as an intermediate file
- So, let $d(Y)$ ($\leq r$) be the number of different Y = <attr_list> values, and m the number of node entries, and suppose index is implemented as a B^+-tree
- The total cost will be:

$$C = \lceil d(Y) / m \rceil + \lceil d(Y) / f \rceil$$

- Complexity: $O(d)$ or $O(r)$

Selection: Attribute Selection Cardinality

- If an attribute A of a relation schema N has $d(A)$ actual distinct values, then its selection cardinality $s(A)$ is

$$s(A) = r / d(A)$$

- For a key K, $d(K) = r$, and $s(K) = 1$
- If an attribute A is not a key, then

$$s(A) = (r / d(A)) \geq 1$$

- Selection cardinality $s(A)$ of the attribute A, allows us to compute how many tuples is expected to contain a given value

$$a \in \pi_A(N)$$

- We always assume a uniform distribution

Cost Function of a Select Operation

- Let s ($0 \le s \le r$) be the number of tuples that satisfy selection condition
- In the case of a linear search, select operation is performed by
  - reading b blocks in, and
  - writing s tuples as $\lceil s/f \rceil$ blocks back
- The cost function is

$$C = b + \lceil s / f \rceil$$

- Complexity O(r)

Cost Functions of Join Operation

- The join operation is one of the most time consuming operations in query processing
- We shall consider joins $N \bowtie_{jc} M$, where N and M are relational variables, and jc is a join condition of the form $N.Y = M.Y$
- N is called outer loop relation, and M is called inner loop relation
- Of the four basic join algorithms, we consider in more detail only nested-loop join

The Size of a Join Result

- We start from a simplifying supposition that the join condition $jc \equiv N.Y = M.Y$ is based on a foreign key/primary key pair, and that the corresponding referential integrity constraint $M[Y] \subseteq N[Y]$ is satisfied
- Then and only then, the size of a join result is

$$|N \bowtie_{jc} M| \leq r_M$$

- The number of blocks is

$$\lceil r_M / f \rceil$$

where f is the blocking factor

Question for You

- If Y is the primary key of N, and the referential integrity constraint $M[Y] \subseteq N[Y]$ is satisfied, then

$$|N \bowtie_{jc} M| \leq r_M$$

- Why is the statement true?

Nested-Loop Join

- Algorithm:

  For each tuple t in N (outer loop relation),

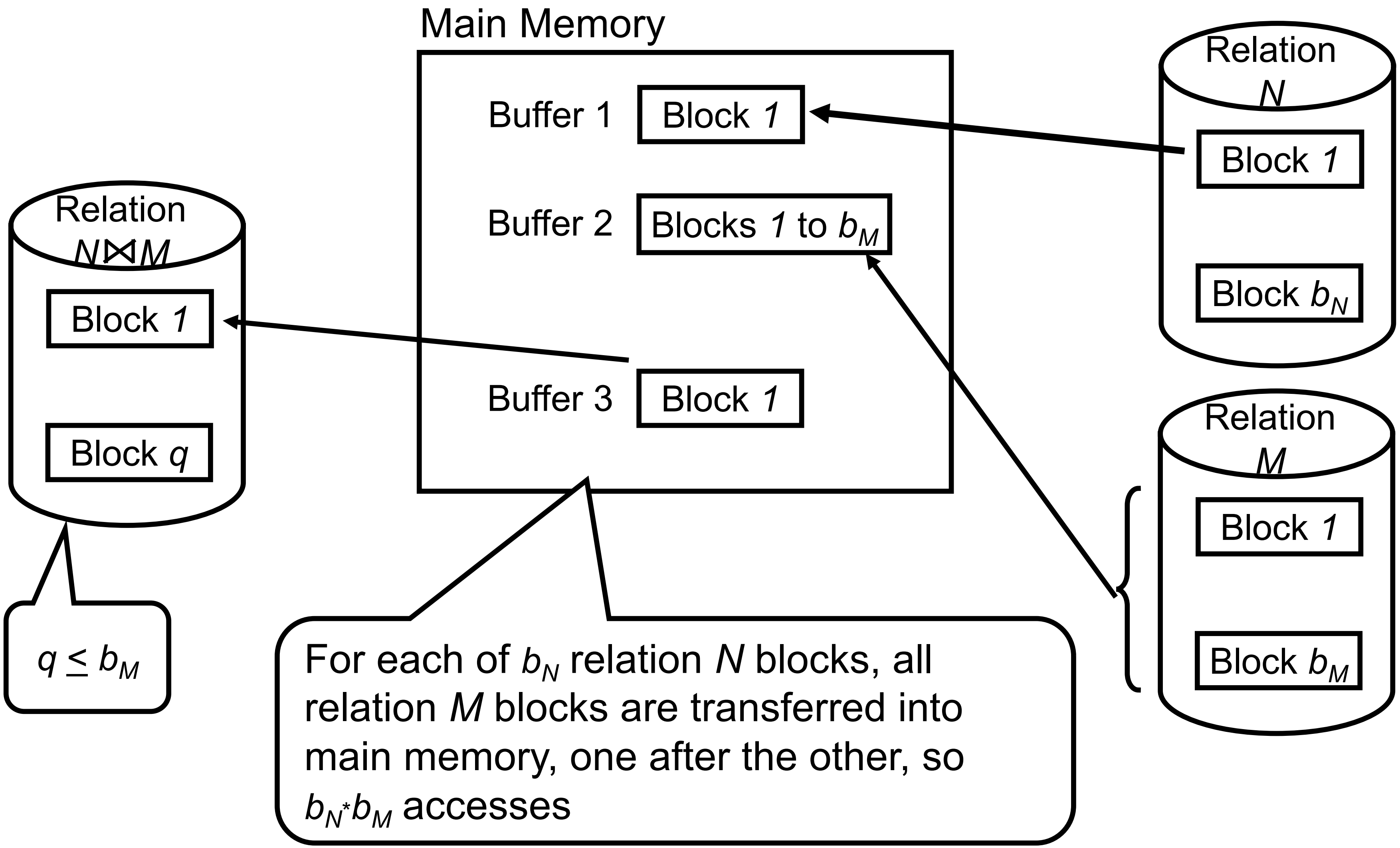
  retrieve each tuple u from M (inner loop relation),
  test whether the two tuples satisfy join condition jc

  (whether $t[Y] = u[Y]$)

  If yes, concatenate t and u and include them into the join result
- Let n ($n > 2$) be the number of buffers available for storing:
  - $n - 2$ out of b_N outer relation blocks at once
  - 1 of b_M inner relation blocks, and
  - 1 of $\lceil r_M / f \rceil$ result blocks

  in the main memory

Nested-Loop Join – Three Buffers



Cost of Nested-Loop Join

- The cost of nested-loop join:

$$C = b_N + b_M \lceil b_N / (n - 2) \rceil + \lceil r_M / f \rceil$$

With total number of blocks read for outer relation is b_N and total number of blocks read for inner relation is $b_M \lceil b_N / (n - 2) \rceil$

- The number of buffers n has considerable impact on the number of disk accesses C
- Since

$$b_M \lceil b_N / (n - 2) \rceil \approx b_N \lceil b_M / (n - 2) \rceil$$

the number of disk accesses C will be smaller if

$$b_N < b_M$$

- Complexity $O(r^2)$

Other Join Algorithms

- Single-loop join
  - Uses index on inner relation
  - Only exceptionally better than the others
- Sort-merge join
  - Relies on sorting,
  - A reasonably good choice
- Hash join
  - Relies on hashing
  - Greedy on memory
  - If provided enough memory, the best choice

Single-Loop Join (Homework)

- The prerequisite:
  - An index (or hash key) on join attribute *M.Y* of the inner relation *M*
- Algorithm:
  - Retrieve each tuple of the outer loop relation *N* and use the access structure to retrieve directly all matching tuples of the inner loop relation *M*
- Complexity $O(r * \log r)$ (large constants neglected)
- Needs at least four buffers, but additional buffers bring only a slight improvement
- Can't apply after select and project

Sort-Merge Join (Homework)

- Algorithm:
  - Sort the N and M relations ($m + 1$ buffers needed, $m \geq 2$)
  - Read successive and sorted blocks of N and M into memory
  - Compare successive N and M tuples from the blocks read in
  - Include the tuples that match into join result
- Complexity $O(r * log\ r)$
  - Which is the complexity of sorting
- Frequently the best choice

Hash Join (Homework)

- Consider a function h that will be applied on the join attributes $N.Y$ and $M.Y$
- Both relations N and M are split (one after the other) into m partitions using the same hash function h
- That way, partitions N_i and M_i contain tuples that are equivalent with regard to h, and a tuple from N_i may join only with some tuples from M_i
- Pairs (N_i , M_i) of partitions are joined one after the other and stored into join result (m iterations needed)
- Complexity O(r)

Question for You

- We introduced four join algorithms:
  - Nested-Loop Join (exhaustive search),
  - Single-Loop Join (B-tree, not applicable after select or project),
  - Sort-Merge Join (sort is expensive),
  - Hash Join (high demand on memory buffers)
- How does a query processor find the most effective one?

a) Randomly takes one

b) Calculates the cost of using each of them and takes the least expensive

Efficiency Improvement Techniques

- The efficiency of executing relational algebra operations can be improved using:
  - Indexes (select, project with DISTINCT),
  - More memory (join, sort)
  - Sorting:
    - project with DISTINCT,
    - set theoretic operations,
    - aggregates with GROUP BY
- More on this in the tutorial

Combining the Optimization Techniques

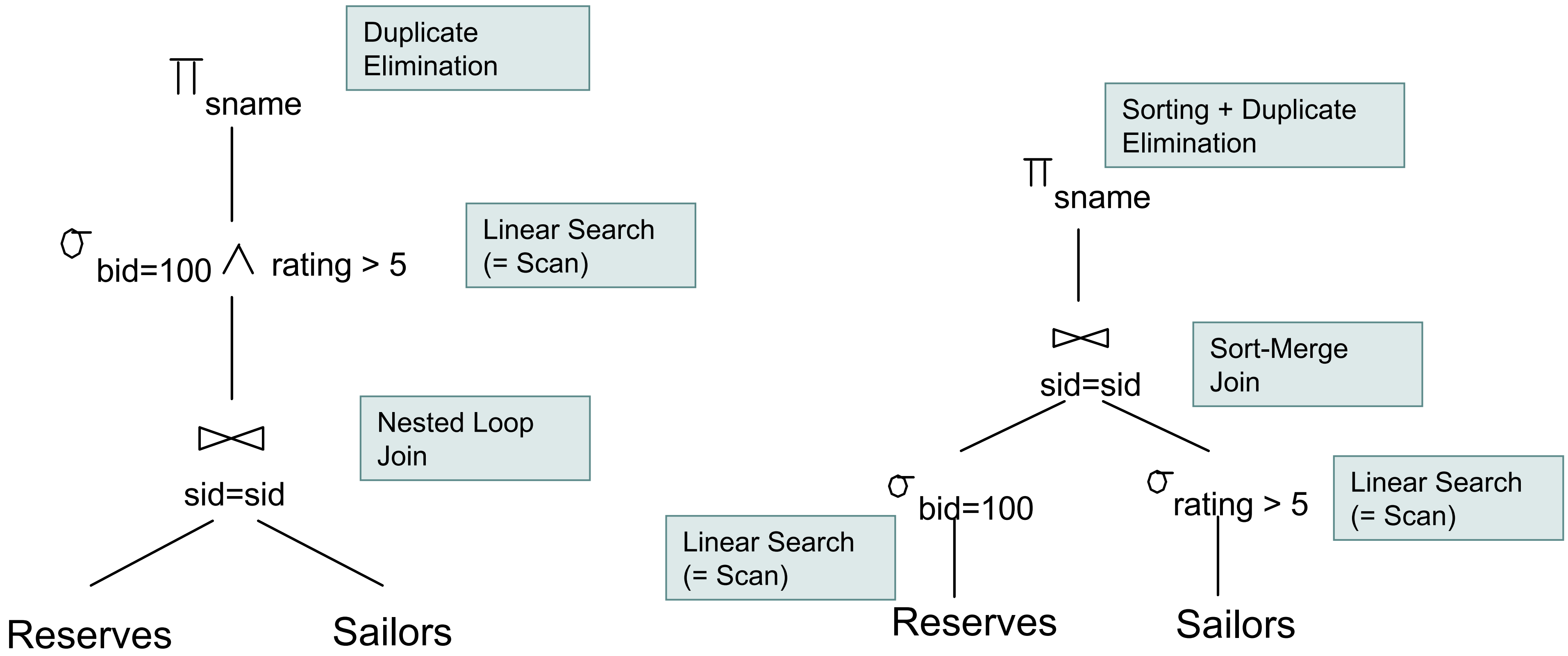
- The cost based optimization is applied onto the query tree that is a result of the heuristic optimization
- Cost based optimization mainly examines:
  - Implementation methods of operations, and
  - The order of multiple joins and their implementation methods
- Query tree of physical operators is produced when the cheapest execution plan is provided with access methods and algorithms to be used in executing the relational algebra operations

Nested Query Optimization

- Optimization of a nested query depends on whether it is correlated or not:
  - In a non correlated nested query, the result of the inner SELECT has to be computed only once, and each tuple of the outer SELECT is compared to that result
  - In a correlated nested query, inner SELECT is evaluated for each tuple of the outer SELECT
    - Namely, result of the inner SELECT depends on the attribute values of the current outer SELECT tuple

Example

- Which execution plan is better?



Summary

- DBMS processes a declarative query by converting it to the query tree of logical operators, and by optimizing it
  - looks for a reasonably efficient strategy to implement a query
- Heuristic optimization and cost based optimization are two basic optimization techniques
- Cost based optimization is applied to the result of heuristic optimization
  - exhaustive analyze the number of disk accesses of alternative available methods and algorithms to execute a query
- Techniques that improve query cost:
  - Indexing,
  - Using larger memory,
  - Sorting