VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Query Optimization

Lecturer Dr Pavle Mogin

COMP302
Database Systems

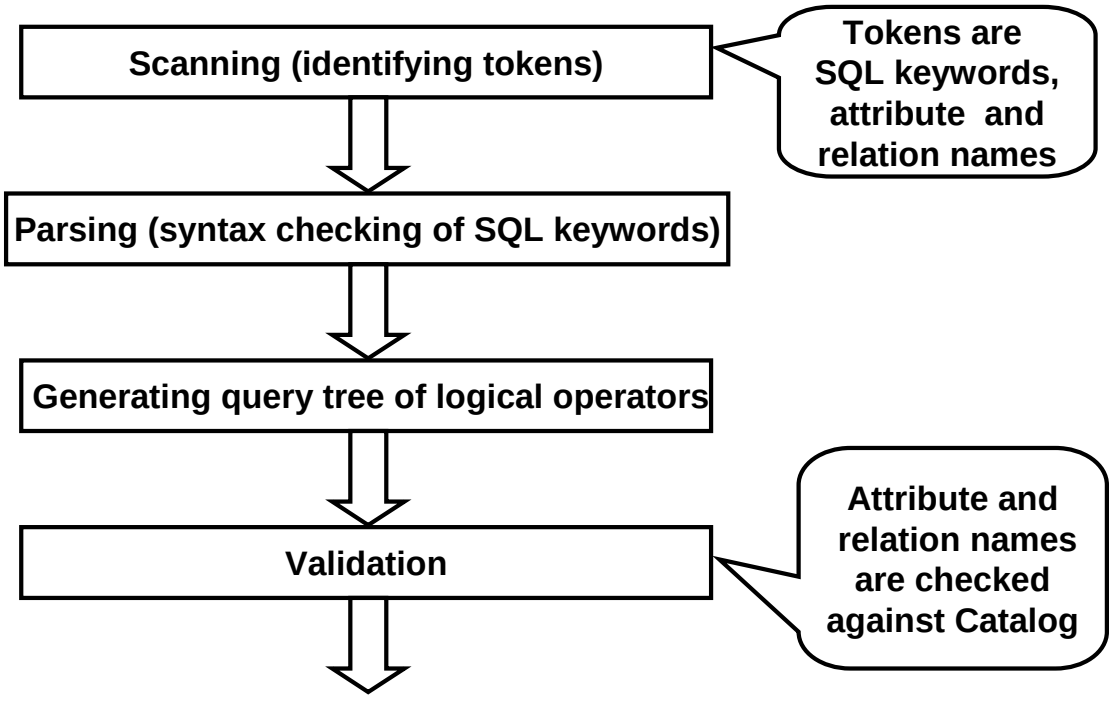
Plan for the Query Optimization topic

- Steps in declarative query processing
- Query optimization techniques
- Heuristic optimization
  - translating SQL into relational algebra
  - Reordering of operations
  - Heuristic optimization algorithm
- Cost based optimization
  - Cost functions of relational algebraic operations
  - Left deep tree
  - The cost based optimization procedure
- *Readings from the textbook:*
  - *Chapter 15,*
  - *Chapters 13, and 14*
    - *Sections: 13.2, to 13.8, and*
    - *Sections: 14.1 to 14.5*
  - *File Organization – COMP201*

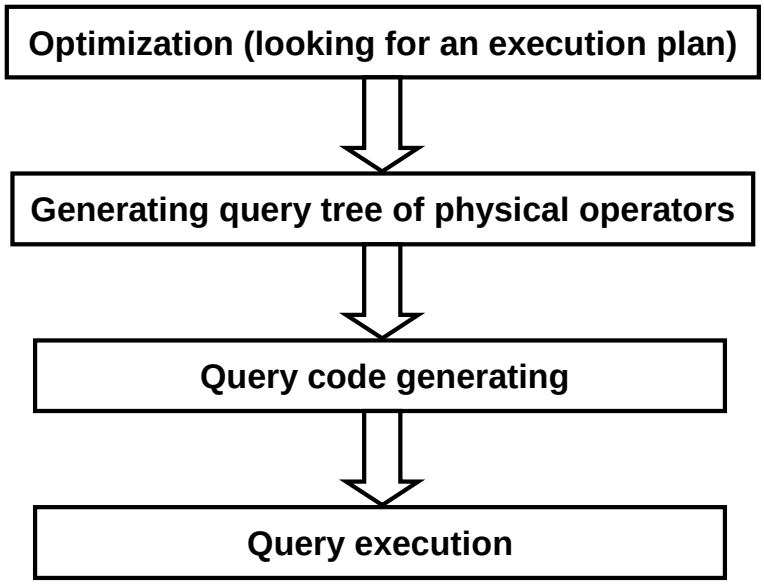
What is Query Optimization and Why

- Whereas declarative query languages (including SQL) offer a great comfort for users, they place a considerable burden on a Query Processor
- The Query Processor is responsible to produce an execution plan that will guarantee an acceptable response time
- Choosing a query execution plan is called Query Optimization and it mainly means making decisions about data access methods
- Query Optimization strongly relies on File Organization techniques

Typical Steps in Query Processing



Typical Steps in Query Processing



Query Optimization Techniques

- Of the eight query processing steps, we shall explicitly consider only:
  - generating query tree of logical operators, and
  - optimization step

  and implicitly
  - Generating query tree of physical operators
- SQL queries are typically:
  - first decomposed into *query blocks*
  - translated to an equivalent relational algebra expression, and represented as a *query tree of logical operators*
  - blocks are separately optimized
  - only nested queries have to be decomposed into query blocks, where each SELECT command constitutes one block

Query Optimization Techniques

- There exist two main query optimization techniques
- One relies on the heuristic reordering of the relational algebra operations, and
- The other involves systematic estimating the cost of the different execution plans, and choosing one with the lowest cost
- Cost based optimization combines both techniques, starting with heuristic and finishing with cost based technique

Translating SQL into Relational Algebra

1. For each SQL query, query parser generates an initial query tree of logical operators
2. It is called a canonical query, as well, and it is not optimized
3. In most cases, direct execution of a canonical query would be very inefficient

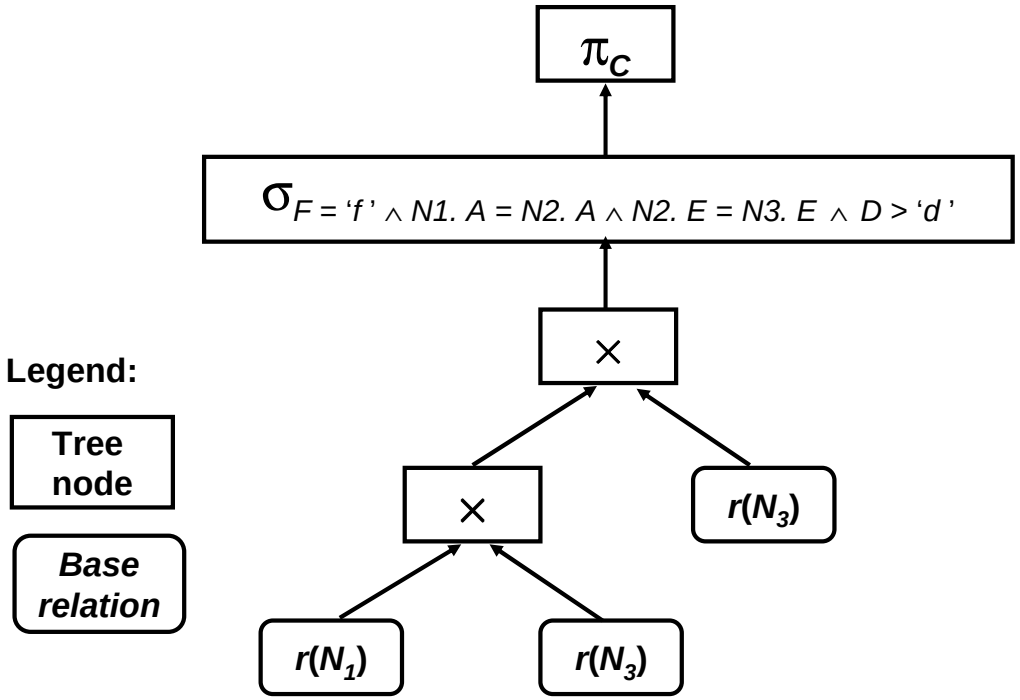
Heuristic Optimization - an Example

- Consider the following relation schemas
  $N_1(\{A, B, C, D\}, \{A\})$,
  $N_2(\{A, E, Q\}, \{AE\})$,
  $N_3(\{E, F, G\}, \{E, F\})$

  and the SQL query

  SELECT *C*
  FROM N_1, N_2, N_3
  WHERE F = '*f*' AND $N_1.A = N_2.A$ AND $N_2.E = N_3.E$
  AND D > '*d*';

Initial Query Tree



Structure of the Initial Query Tree

- A query tree of logical operators is a binary tree
- The nodes of that tree are logical (relational algebra) operators
- Lower level nodes, starting from leaves, contain Cartesian product operators
- These are applied onto relations from the SQL FROM clause
- After that are (relational) select and join conditions from SQL WHERE clause to upper tree nodes applied
- Finally is project operator of the SELECT clause attribute list to tree root applied

A Question for You

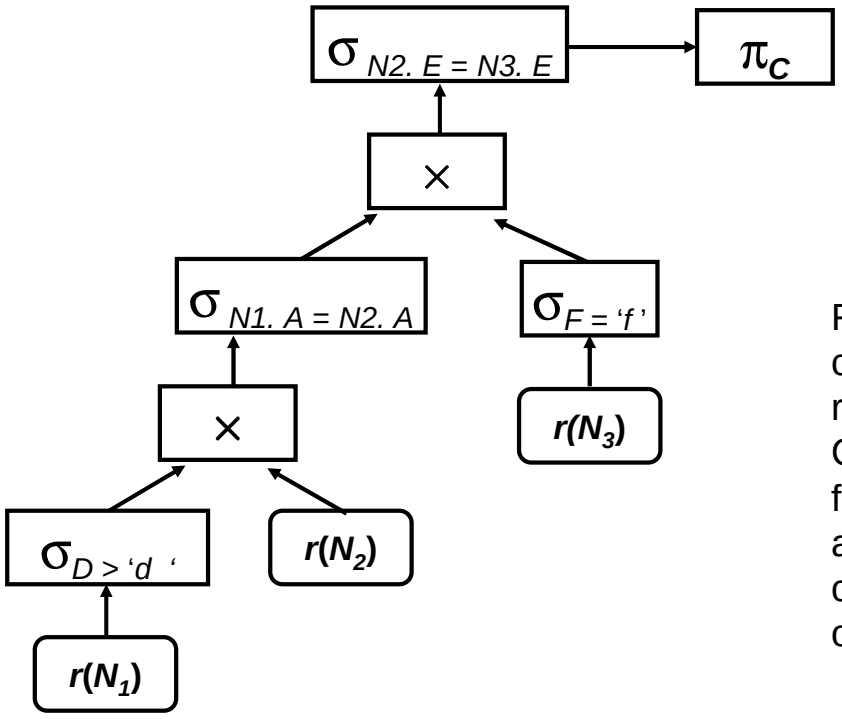
- For how many N_3 tuples asks the query from our example:

  a) Many
  b) Only Pavle knows
  c) At most one

Analysis of the Initial Query Tree

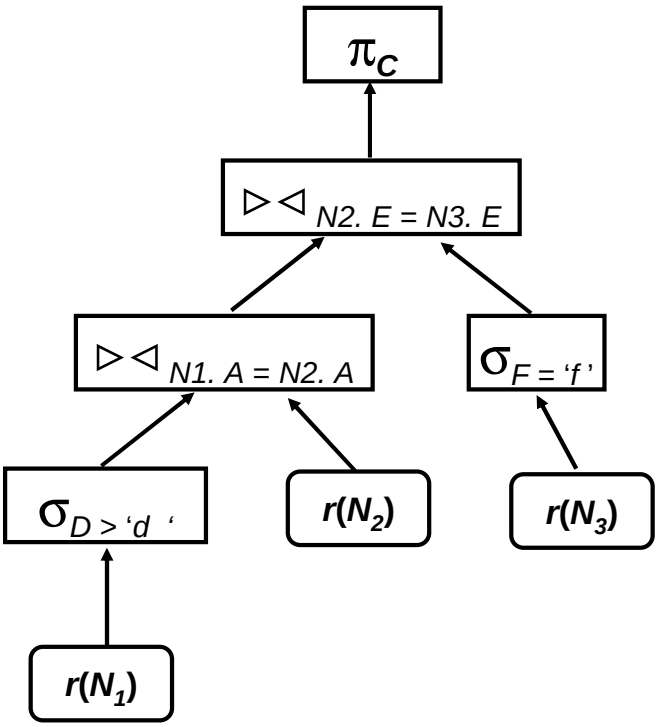
- According to the initial query tree structure, two Cartesian products should be executed first
- But this query asks for only a few tuples from $r(N_1)$ ($D >$ '*d*'), and even for at most one tuple from $r(N_3)$ ($F =$ '*f*')
- Main heuristic rule is to apply unary operations ***select*** and ***project*** before binary operations like join and set theoretic operations, and before aggregate functions
- Hence, move select operations down the tree

Query Tree After Moving Down Selects



Further improvement can be achieved by replacing each Cartesian product followed by a select according to a join condition with a join operator

Query Tree After Introducing Joins



Next improvement can be achieved by switching the positions of N_1 and N_3, so that the very restrictive select operation

$$\sigma_{F = 'f'}$$

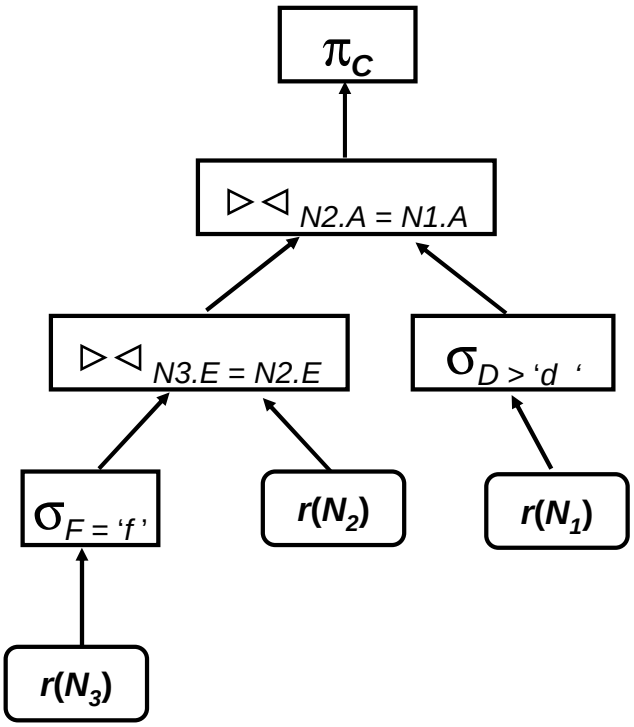
could be applied as early as possible

Question for You

- What is the rational behind an attempt to apply a restrictive select operation as early as possible:

1. All further operations will use less tuples and thus perform faster
2. The result will be more accurate
3. The optimization will be more complex to understand

Query Tree After Switching Positions

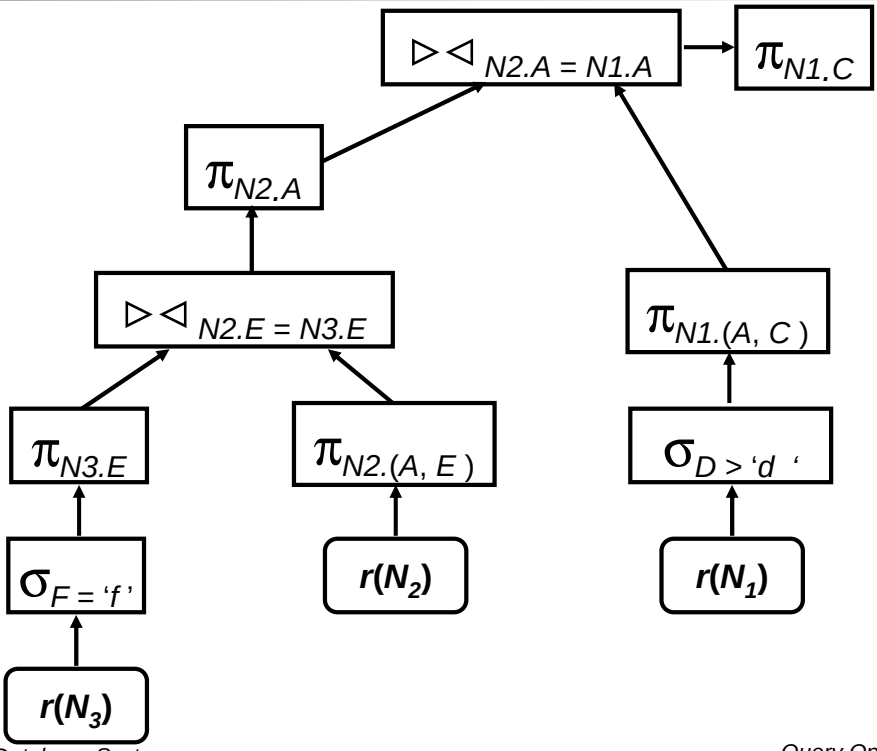


Final improvement can be achieved by keeping in intermediate relations only the attributes needed by subsequent operations This can be accomplished by applying defined, or even introducing new - undefined (but logically implied) project

(π)

operations as early as possible

Query Tree After Introducing Project Ops



Effect of Project Operations

- Performing project operations as early as possible brings an improvement in the efficiency of query execution
- Because tuples get shorter after projection, more of them will fit into the block of the same size
- Hence, after projection, the same number of tuples will be contained in a smaller number of blocks
- As there will be less blocks to be processed by subsequent operations, the query execution will be faster

Cost Based Optimization

- A good declarative query optimizer does not rely solely on heuristic rules
- It chooses that query execution plan which has the *lowest estimated cost*
- After heuristic rules are applied to a query, there still remains a number of alternative ways to execute it
- Query optimizer estimates the cost of executing each of alternative ways, and chooses the cheapest one

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 estimating, a Catalog should contain following information for each base relation:
  - the number of tuples (records) *r*
  - the number of blocks *b*
  - the blocking factor *f*
  - the primary access method and primary access attributes:
    - unordered
    - ordered (just sorted, or indexed, or hashed)
- secondary indexes and indexing attributes
- the number of levels of each index
- the number of distinct values *d* of each attribute

Some Assumptions

- For the sake of simplicity, 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
    - Whereas pipelining means that the tuples of the intermediate results are subjected to all subsequent operations without temporary storing
  - Intermediate results of unary operations retain the same block size as the initial files

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 b_1 blocks, containing r tuples of the size n, from the secondary storage unit into main memory, and
  - Writes back b_2 blocks, containing r tuples of the size m ($m < n$) on to secondary storage
  - Since $m < n$, it follows $b_2 < b_1$
  - The cost function is

$$C = b_1 + b_2$$

Cost Function of a Select Operation

- Let s ($0 \leq s \leq r$) be the number of tuples that satisfy selection condition
- In a general case, select operation is performed by reading a number p ($s \leq p \leq r$) of tuples, contained in b blocks, where $\lceil p/f \rceil \leq b$, and writing back s tuples as $\lceil s/f \rceil$ blocks
- Cost function of a select operation strongly depends on:
  - how restrictive the condition of select operation is, and
  - available access methods

Cost Functions of Join Operation

- The join operation is one of the most time consuming operations in query processing
- We shall consider joins $N \triangleright\triangleleft_{jc} M$, where N and M are relations, 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
- There are four basic join algorithms:
  - Nested - loop join
  - Single - loop join
  - Sort - merge join
  - Hash join

Join Selectivity

- Join selectivity, denoted as *js*, is

$$js = |N \triangleright\triangleleft_{jc} M| / |N \times M|$$

$$js = |N \triangleright\triangleleft_{jc} M| / (r_N * r_M)$$

$$0 \leq js \leq 1$$

- The size of the result of join is

$$|N \triangleright\triangleleft_{jc} M| = js * (r_N * r_M)$$

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

$$|N \triangleright\triangleleft_{jc} M| \leq r_M,$$

  and $js \leq 1/r_N$
- But, if Y is not a key of N, or referential integrity constraint $M[Y] \subseteq N[Y]$ is not satisfied, then *js* has to be determined according to the circumstances given

Question for You

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

$$|N \triangleright\triangleleft_{jc} M| \leq r_M,$$

- That statement is true:

a) Because each M tuple joins at most one N tuple
b) According to the first law of Black Magic
c) Because nobody can understand it

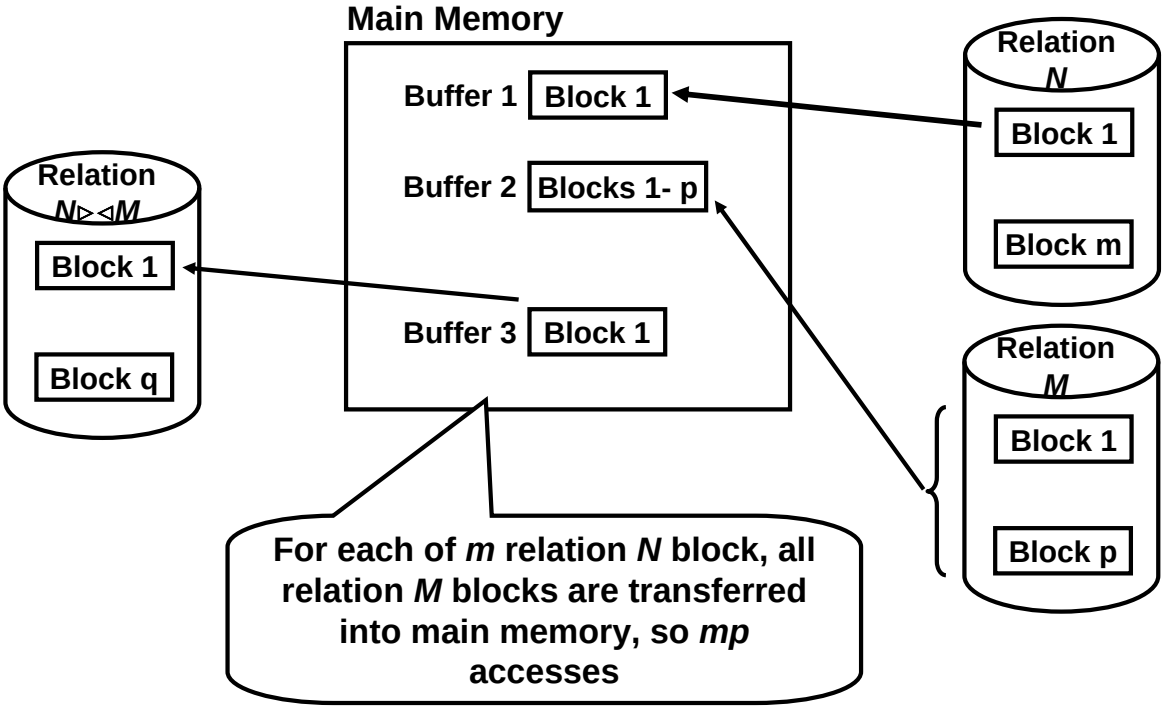
Nested - Loop Join

- Algorithm

  For each tuple t in N (outer loop relation), retrieve each tuple u from M (inner loop relation), and test whether the two tuples satisfy join condition jc (whether $t[N.Y] = u[M.Y]$)
- 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 $\lceil (js*(r_N*r_M))/f \rceil$ result blocks

  in the main memory

Nested Loop Join – Three Buffers



Cost of Nested - Loop Join

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

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

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

the number of disk accesses C will be smaller if

$$b_N < b_M$$

Single - Loop Join

- Algorithm:
  - Suppose an index (or hash key) exists for join attribute $M.Y$ of the inner relation M
  - Then 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
- The single loop join cost function will be

$$C = b_N + r_N*f(index) + \lceil (js*(r_N*r_M))/f\rceil$$

where $f(index)$ depends on the relation M index type

Single Loop Join

- To implement the algorithm, there are at least four buffers needed:
  - For each relation one,
  - One for index, and
  - One for the join result
- The efficiency is mostly influenced by the product $r_N * f(index)$
- After a relation is selected or projected, it can not be used as an inner loop relation in the single loop join algorithm, since the index addresses relate to the blocks of the original relation

Sort - Merge Join

- 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 red in
  - Put the tuples that match into join result
- Cost of sorting the N and M relation is

$$C = 2b_N(1 + \lceil log_m\, b_N \rceil) + 2b_M(1 + \lceil log_m\, b_M \rceil)$$

- Cost of the comparisons and of the writing back the result is

$$C = b_N + b_M + \lceil (js * (r_N * r_M))/f \rceil$$

Hash Join

- The hash join may be an efficient algorithm if there is enough available memory space
- There are many variants of the hash-join algorithm
- We shall consider three of them:
  - Partition Hash Join (lecture),
  - In Memory Partition Join (tutorial), and
  - Hybrid Hash-Join (tutorial)
- Generally, a hash join consists of two phases:
  - Partitioning phase, and
  - Probing (joining) phase
- Consider an equi-join of relations N and M on A and B

  $$N \triangleright\triangleleft_{A=B} M$$

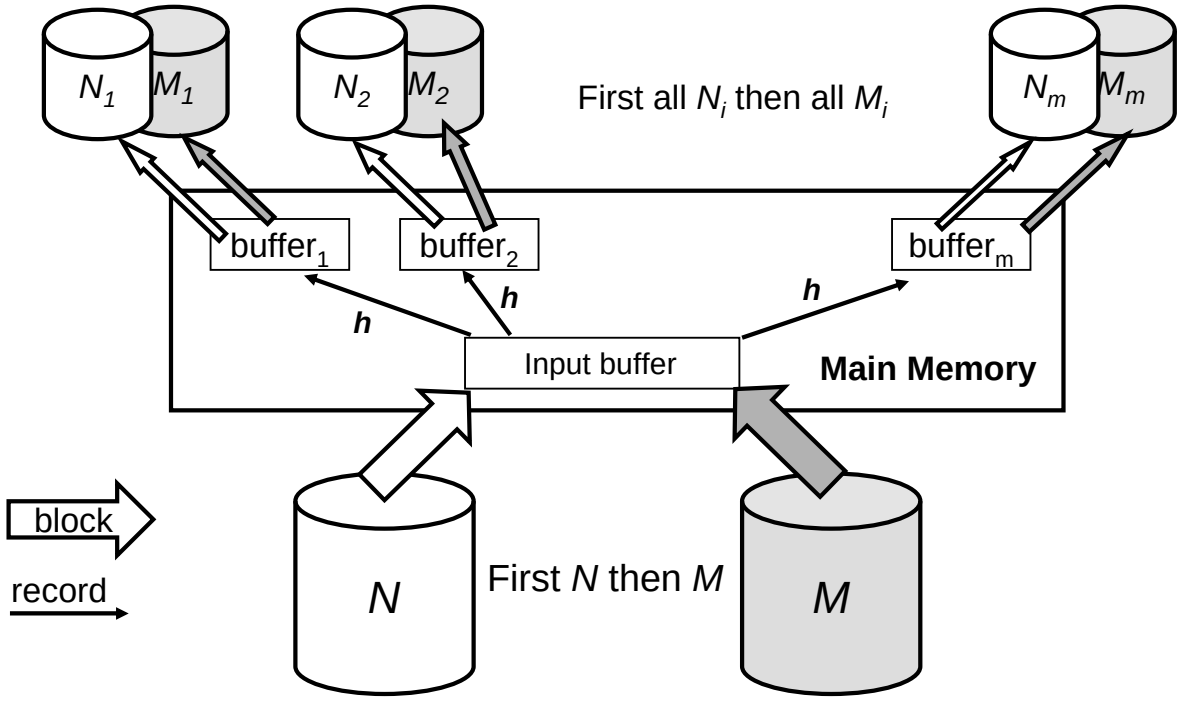
  and a function h that hashes each N or M tuple into one of m partitions
- Hashing is done on the join attributes

Partition Hash Join (partitioning phase)

- 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
- The partitioning phase requires $m + 1$ memory buffers:
  - One buffer for the input block, and
  - m buffers for partitions
- The number of disk accesses in the partitioning phase is:

  $$2(b_N + b_M)$$

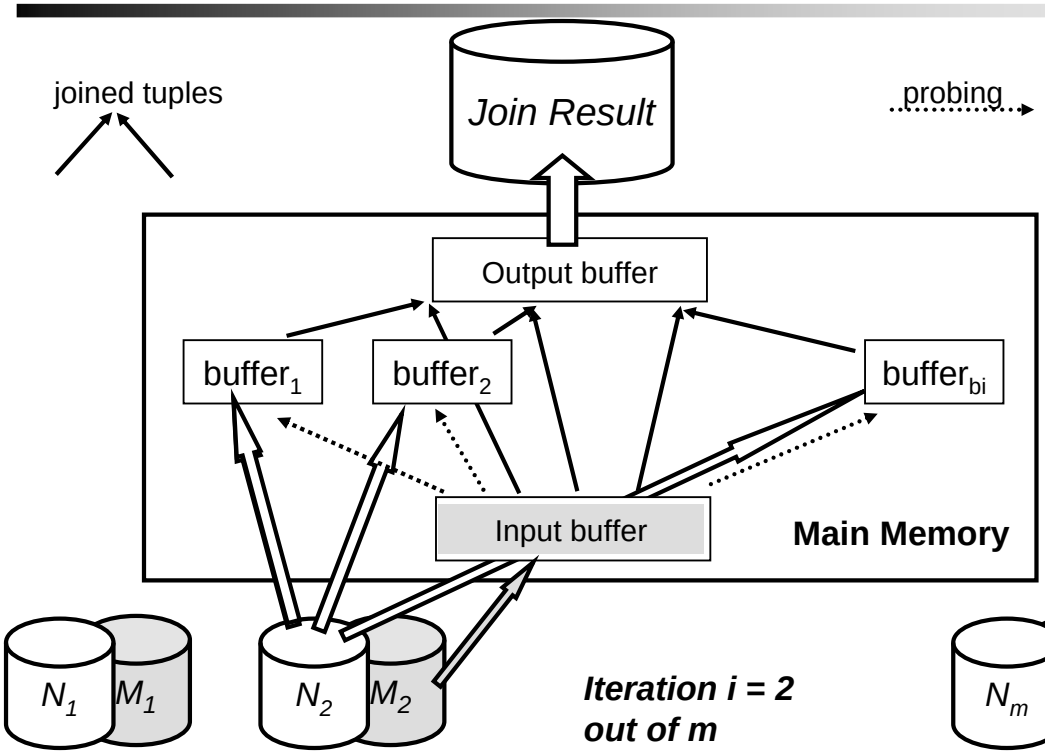
Partitioning Phase - Diagram



Probing (joining) Phase

- Pairs (N_i, M_i) of partitions are joined one after the other and stored into join result (m iterations needed)
- Probing phase needs $n = b_i + 2$ buffers:
  - b_i buffers to store all the blocks of the largest partition of the smaller relation,
  - 1 input buffer to store a block of the corresponding partition of the other (larger) relation, and
  - 1 buffer for the output (join result)
- The algorithm:
  - Reads in the whole partition of the smaller relation (say N_i), first
  - Then reads in one block after the other of the partition M_i,
  - Takes each tuple from the input buffer and probes for the matching tuples in the b_i buffers
  - The matching tuples are joined and placed into the output buffer
- The cost of the probing phase is $b_N + b_M + b_{result}$
- So, the total cost is $C = 3(b_N + b_M) + b_{result}$

Probing Phase - Diagram



Question for You

- We considered 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 to find the most effective?
  a) Take any, the result will be wrong anyway
  b) Copy from a peer
  c) Calculate the cost of using each of them and take the least expensive

Combining the Optimization Techniques

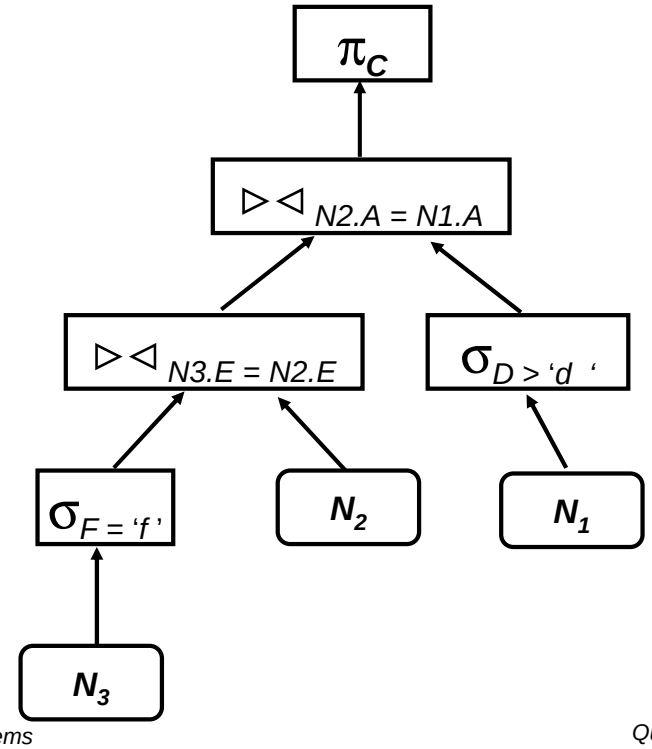
- The cost based optimization is applied onto the query tree that is a result of the heuristic optimization
- By means of the cost based optimization, there are mainly:
  - Implementation methods of select operations, and
  - The order of multiple joins and their implementation methods

  examined

Left Deep Trees

- Joining m ($m > 1$) relations requires ($m - 1$) joins
- These ($m - 1$) joins can be accomplished in many different orders
- Optimizer considers only left (or right) deep join trees, and takes into account the heuristic optimization rules
- A *left deep join tee* is a binary tree where the right child of each non - leaf node is a base relation or the result of a select or project (but not of another join) operation

Left Deep Tree - An Example



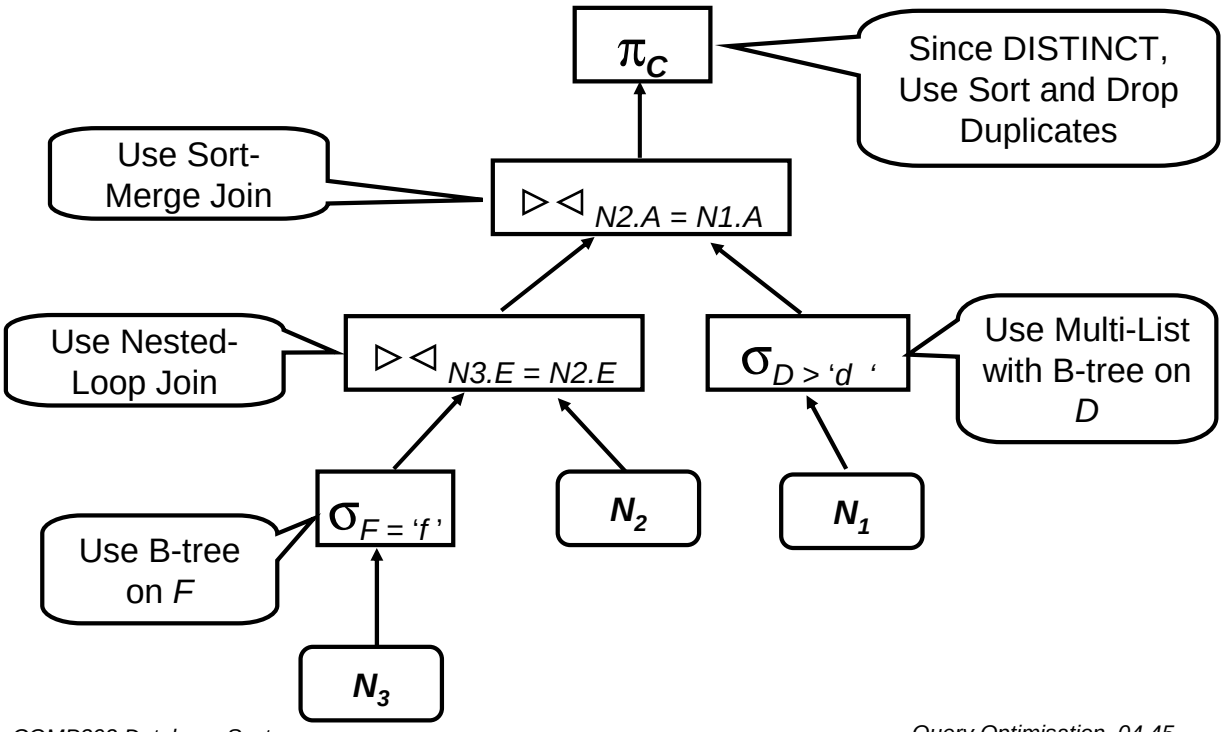
Nested Query Optimization

- 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

Query Tree of Physical Operators

- 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

Query Tree of Physical Operators (Example)



Summary

- DBMS processes the declarative query by converting it to a query tree of logical operators, and by optimizing it
- Query optimization is a looking for a reasonably efficient strategy to implement a query
- Heuristic optimization and cost based optimization are two basic optimization techniques

Summary (heuristic optimization)

- Heuristic optimization converts a declarative query to a canonical algebraic query tree, that is then gradually transformed using certain rules
- The main heuristics is to perform unary relational operations (selection and projection) before binary operations (joins, set theoretic), and aggregate functions with (or without) grouping

Summary (cost based optimization)

- Cost based optimization is applied to the result of heuristic optimization
- By means of cost based optimization are mainly:
  - implementation methods of select operation, and
  - order of multiple joins, and their implementation methods

  defined
- Cost based optimization means a rather exhaustive analyzing the number of disk accesses of alternative available methods and algorithms to execute a query

Plan for Transaction Processing topic

- Transaction processing basics
- Database transaction
- How transactions influence database consistency
  - Lost update problem
  - Dirty read problem
  - Unrepeatable read problem
- Schedule
- Serial and serializable schedules
- Serializability test
  - *Readings from the textbook:*
    - *Chapter 19*

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Query Optimisation Tutorials

Lecturer Dr Pavle Mogin

COMP302
Database Systems

Relationship Between b, r, f

- The relationship between the number of blocks b, number of records r, and blocking factor f is

$$b = \lceil r / f \rceil$$

- Let $(a_1, \ldots, a_n)$ be a tuple, and l_i the size of a_i in bytes, then

$$L = \sum_{i=1}^{n} l_i$$

is the storage capacity needed to store a tuple

The Effect of Projection

- The storage capacity needed to store one block of tuples is $B = L*f$
- Let $B_1 = L_1*f_1$, and $B_2 = L_2*f_2$ be given
- If $L_1 > L_2$, and $B_1 = B_2$, then

$$f_2 \geq f_1$$

- Let r_1, L_1, f_1, and r_2, L_2, f_2 be given
- If $r_1 = r_2$, $L_1 > L_2$, and $B_1 = B_2$, then

$$b_1 \geq b_2$$

- So, the project operation drops some fields in tuples, there will be less blocks after projection

Evaluation of DISTINCT Project

- Suppose the <attribute_list> of project operation does not contain a relation schema key and the keyword DISTINCT is used in the SQL SELECT command, then:
  - Reading of b_1 blocks, and writing back b_2 blocks occurs again (as in the case when DISTINCT is not specified), but the duplicate tuples have to be eliminated from b_2 blocks, as well
  - Eliminating duplicate tuples asks sorting of b_2 blocks
  - Finally, omitting duplicate tuples will require reading in b_2 blocks and writing back b_3, where b_3 is the number of blocks with duplicates omitted

Cost of DISTINCT Project Operation

- The cost of initial reading of b_1 blocks, and writing back b_2 blocks is:

$$C_1 = b_1 + b_2$$

- The worst case cost function of sorting b_2 blocks is

$$C_2 = 2\, b_2\, (1 + \lceil log_2\, b_2 \rceil)$$

- Omitting duplicate tuples will require

$$C_3 = b_2 + b_3,$$

where b_3 is the number of blocks with duplicates omitted

- Assuming the m-way sorting ($m \geq 2$), the total cost will be:

$$C = b_1 + b_2\, (4 + 2 \lceil log_m\, b_2 \rceil) + b_3$$

Attribute Selection Cardinality

- If an attribute A of relation schema N has $d(A)$ 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, for $r > 1$,

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

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

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

Selection Cardinality

- Consider relation *Student* having *1,000* tuples, then
  - $d(StudID) = 1{,}000$, and $s(StudID) = 1$
  - $d(Sex) = 2$ and $s(Sex) = 500$
  - $d(StudName) = 800$, and $s(StudName) = 1.25$
- Consider relation *Grades*, having *10,000* tuples, and suppose
  - $d(PapID) = 20$,
  - $Grade \in$ {A+, A, A-, B+, B, B-, C+, C},
- then:
  - $s(StudID) = 10$
  - $s(PapID) = 500$
  - $s(Grade) = 1{,}250$
  - $s(StudID, PapID) = 1$

Cost Functions of Select Operation

- *Linear search* (no indexes neither hash functions provided)

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

- *Unique key index*:
  - and $K = k$ (* selection condition *)

$$C = x + 1 + \lceil 1/f \rceil$$

  - and $k_1 \leq K \leq k_2$ (*suppose s tuples satisfy condition*)

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

- *Hash key*, and $K = k$ (all synonyms fit into *1* bucket)
  - $C = 1 + \lceil 1/f \rceil$ (*static hashing*)
  - $C = 2 + \lceil 1/f \rceil$ (*extendible hashing*)

Cost Functions of Select Operation

- *Clustering index*, and secondary key $Y = y$
  ($s(Y)$ successive tuples satisfy condition)

$$C = x + \lceil s(Y)/f \rceil + \lceil s(Y)/f \rceil$$

- *Secondary index*, and secondary key $Y = y$
  ($s(Y)$ random tuples satisfy condition)

$$C = x + s(Y) + \lceil s(Y)/f \rceil$$

Select Operation Methods

- A select operation may be executed either using a linear search or an index algorithm
- Suppose $b_1 = \lceil r_1 / f_1 \rceil$, s (selection cardinality), and the height x of the index tree are given
- The index search algorithm will be more efficient if the inequality

$$x + s < b_1$$

  evaluates true

Avoiding Sorting with DISTINCT

- A costly sort can be avoided if there exists an appropriate secondary index on the whole SELECT DISTINCT <attr_list>
- Then, the query optimizer has only to perform index search, to retrieve the secondary key values, and write them back as an intermediate file
- So, let x be the height of the secondary index on Y = <attr_list>
- The total cost will be:

$$C = x + s(Y) + \lceil s(Y)/f \rceil$$

The Order of Select and Project Ops

- One may pose the question whether a select or a project operation should be executed first in order to achieve minimal overall query cost
- Suppose $b_1 = \lceil r_1/f_1 \rceil$, s (selection cardinality), and f_2 (blocking factor after projection) are given
- Select first using linear search, then project

$$C(\sigma,\pi) = \lceil r_1/f_1 \rceil + \lceil s/f_1 \rceil + \lceil s/f_1 \rceil + \lceil s/f_2 \rceil$$

- Project first, then select

$$C(\pi, \sigma) = \lceil r_1/f_1 \rceil + \lceil r_1/f_2 \rceil + \lceil r_1/f_2 \rceil + \lceil s/f_2 \rceil$$

- So

$$C(\sigma,\pi) < C(\pi, \sigma) \Leftrightarrow \lceil s/f_1 \rceil < \lceil r_1/f_2 \rceil$$

The Order of Select and Project Ops

- Suppose $b_1 = \lceil r_1 / f_1 \rceil$, s (selection cardinality), and f_2 (blocking factor after projection) are given
- Select first using index search, then project

  $C(\sigma,\pi) = x + s + \lceil s / f_1 \rceil + \lceil s / f_1 \rceil + \lceil s / f_2 \rceil$

- Project first, then select using index search is not possible, since after projection the index cannot be used any more

Join Selectivity (an Example)

N

A	B
1	1
0	2
5	3

$\triangleright\triangleleft$

M

B	C
1	1
1	2
2	3
2	4
3	5
3	6

=

$N \triangleright\triangleleft M$

A	B	B	C
1	1	1	1
1	1	1	2
0	2	2	3
0	2	2	4
5	3	3	5
5	3	3	6

$r_N = 3$, $r_M = 6$,

Since relation N key is B, and referential integrity $M[B] \subseteq N[B]$ is satisfied, $| N \triangleright\triangleleft M | = r_M = 6$

$js = | N \triangleright\triangleleft M | / (r_N * r_M) = r_M / (r_N * r_M)$,

hence $js = 1 / r_N = 0.33$

Join Selectivity (an Example)

N

A	$\underline{B}$
1	1
0	2
5	3

$\sigma_{A=0}(N) =$ N_1

A	$\underline{B}$
0	2

B is still the key of N_1, but referential integrity $M[B] \subseteq N_1[B]$ is not satisfied, so join selectivity is

$$js = 1 / d_M(B)$$

where $d_M(B)$ is the number of (distinct) B values in M, and an even distribution of B values in M is supposed

N_1

A	$\underline{B}$
0	2

$\triangleright\triangleleft$

M

B	$\underline{C}$
1	1
1	2
2	3
2	4
3	5
3	6

$=$

$N_1 \triangleright\triangleleft M$

A	B	B	$\underline{C}$
0	2	2	3
0	2	2	4

The Size of the Join Result Block

- The result of an equi - join is a set of tuples over a concatenation of relation N and relation M attribute sets
- So, if the relation N tuple size is L_N, and the relation M tuple size is L_M, then the size L of the join result tuple is:

$$L = L_N + L_M$$

- If the blocking factor f of the join result is given, the size of the join result block will be

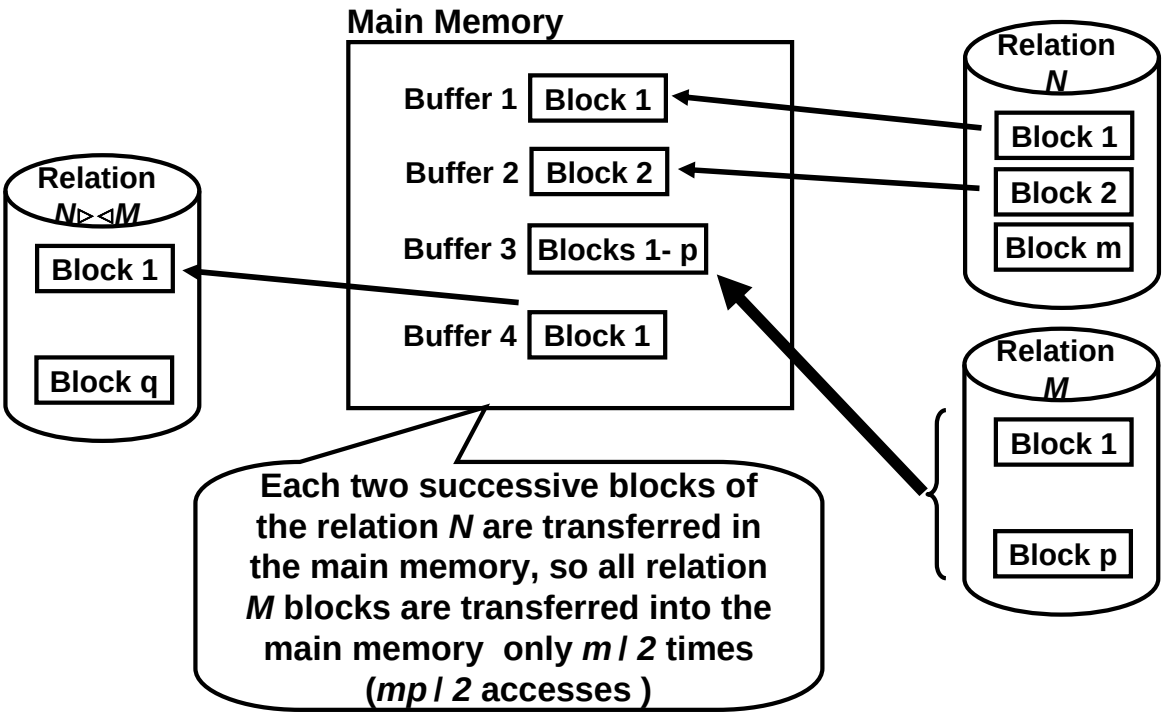
$$B = L * f$$

- If the size of the join result block is given, then

$$f = \lfloor B / L \rfloor$$

- Note that block sizes of the N and M relations may be equal to B, but may be different, as well

Nested Loop Join – Four Buffers



Cost of Nested Loop Join (an Example)

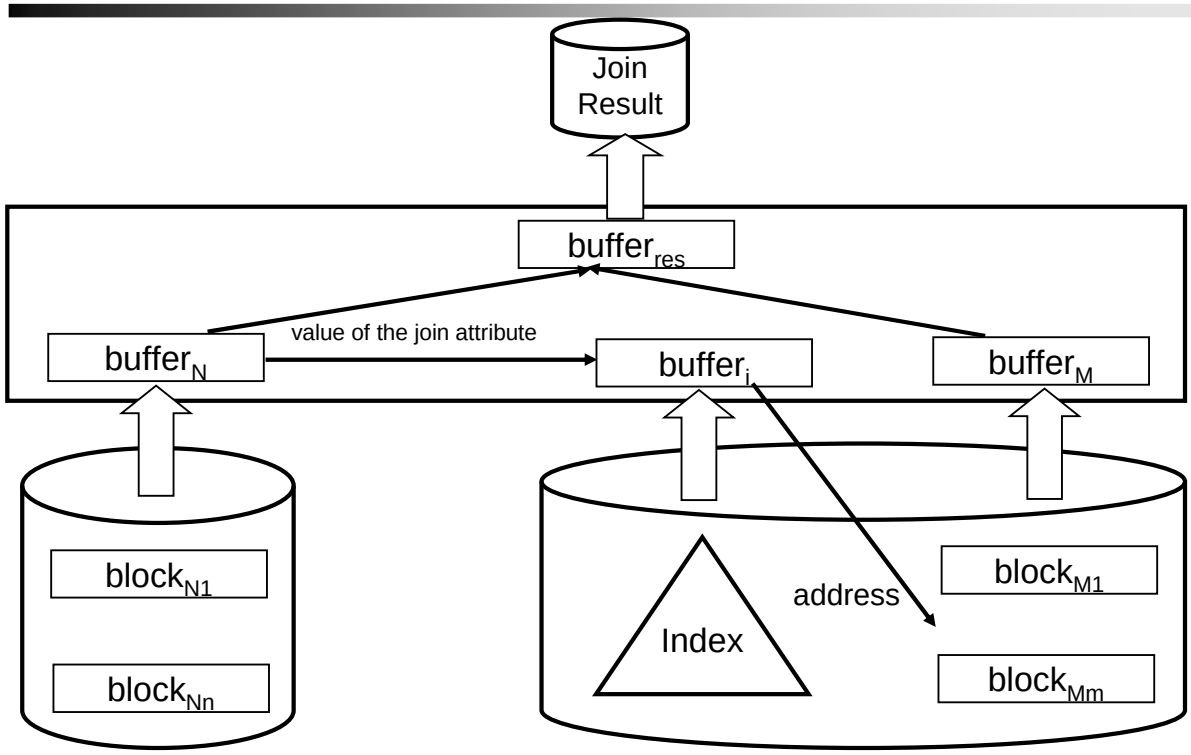
- Let:
  - $r_N = 600$, $b_N = 60$,
  - $r_M = 5000$, $b_M = 1000$,
  - $n = 5$,
  - blocking factor of join result $f = 10$,
  - join condition attribute Y be the key of N, and
  - referential integrity $M[Y] \subseteq N[Y]$ satisfied
- Then:
- join selectivity $js = 1 / 600$
  - cost of $N \triangleright\triangleleft M$ is:

$60 + 1000\lceil 60 / 3 \rceil + ((1 / 600)(600 * 5000) / 10) = 20,560$

  - cost of $M \triangleright\triangleleft N$ is:

$1000 + 60\lceil 1000 / 3 \rceil + ((1 / 600)(600 * 5000) / 10) = 21,540$

Single Loop Join



Index Function f(index)

- $f(index) = x + 1$, for a unique key index on $M.Y$,
- $f(index) = h$, and $1 \leq h \leq 2$ for a hash key on $M.Y$,
- $f(index) = x + s(M.Y)/f$, for a clustering index on $M.Y$,
- $f(index) = x + s(M.Y)$, for a secondary index on $M.Y$,

Nested Loop Versus Sort/Merge

- $C(\text{nest}) = b_1 + b_2\lceil b_1/(n-2)\rceil + \lceil |N \triangleright\triangleleft M|/f \rceil$
- $C(\text{sort}) = b_1(3 + 2\lceil \log_m b_1 \rceil) + b_2(3 + 2\lceil \log_m b_2 \rceil) + \lceil |N \triangleright\triangleleft M|/f \rceil$
- Suppose $b_1 = b_2 = b$, then
  $C(\text{nest}) < C(\text{sort}) \Leftrightarrow b < (n-2)(5 + 4\lceil \log_m b \rceil)$
- Suppose $b_1 << b_2$, then
  $C(\text{nest}) < C(\text{sort}) \Leftrightarrow b_1 < (n-2)(3 + 2\lceil \log_m b_2 \rceil)$

Comparing Nested and Single Loop

- $C(\text{nest}) = b_1 + b_2\lceil b_1/(n-2)\rceil + \lceil |N \triangleright\triangleleft M|/f_n \rceil$
- $C(\text{single}) = \lceil r_o/f_o \rceil + r_o * f(index_i) + \lceil |N \triangleright\triangleleft M|/f_s \rceil$
- Let it be:
  - $f_o * f(index_i) \geq 10$, (* then is $r_o * f(index_i) >> b_o = \lceil r_o/f_o \rceil$ *)
  - $b_1 << b_2$, and
  - the difference between $\lceil |N \triangleright\triangleleft M|/f \rceil$ values for single and nested loop joins is negligible
- Then, when the following inequality evaluates true, it is worth examining in more detail replacing a nested loop with the corresponding single loop join

$$r_o * f(index_i) \leq b_2\lceil b_1/(n-2)\rceil$$

Sort/Merge Versus Single Loop

- C(sort) = $b_1(3 + 2\lceil log_m b_1 \rceil) + b_2(3 + 2\lceil log_m b_2 \rceil) + \lceil |N \triangleright\triangleleft M| / f_m \rceil$
- C(single) = $\lceil r_o / f_o \rceil + r_o * f(index_i) + \lceil |N \triangleright\triangleleft M| / f_s \rceil$
- Let it be:
  - $f_o * f(index_i) \geq 10$, (then is $r_o * f(index_i)$ >> $b_o = \lceil r_o / f_o \rceil$)
  - $b_1 << b_2$, and
  - the difference between $\lceil |N \triangleright\triangleleft M| / f \rceil$ values for single loop and sort/merge joins is negligible
- Then, when the following inequality evaluates true, it is worth examining in more detail replacing a sort/merge with the corresponding single loop join

$$r_o * f(index_i) \leq b_2(3 + 2\lceil log_m b_2 \rceil)$$

In Memory Partition Hash

- If the number of buffers available is $n > b_N + 2$, then is the partitioning phase not needed
- The smaller relation is loaded into memory and stored in a hash table
- The blocks of the larger relation are read into memory one after the other
- Each tuple of the current block is hashed on its join attribute and the matching tuples are sought in the corresponding bucket of the hash table
- The total cost is $C = b_N + b_M + b_{result}$

Hybrid Hash Join

- Suppose the number of buffers available is

  $$m + b_1,$$

  where b_1 is the number of blocks in the partition N_1 of the (smaller) relation N
- In the first pass of the partition phase, partition N_1 is computed and loaded into b_1 buffers, whereas $m - 1$ buffers are used to build partitions N_2 to N_m and store them on disk
- In the second pass of the partitioning phase, partition N_1 still occupies buffers, and $m - 1$ buffers are used to build partitions M_2 to M_m and store them on disk
- Whenever a M tuple hashes to M_1, it is probed against N_1 tuples and (eventual) join is placed in the result buffer
- In the probing phase are remaining $m - 1$ partition pairs joined

Hash-Join Versus Other Algorithms

- Providing that there is enough memory buffers available, partition hash-join may outperform all the other join algorithms
- Suppose $b_1 << b_2$, then
  - C(hash) < C(nested) $\Leftrightarrow 3 < \lceil b_1/(n-2) \rceil$
  - C(sort) < C(hash) $\Leftrightarrow \perp$
  - C(hash) < C(single) $\Leftrightarrow 3b_2 < r_o * f(index_i)$

Hash-Join Buffer Requirement

- But, even the Partition Hash-Join that is the modest one regarding memory requirements, needs

$$n = b_i + 2$$

buffers for the probing phase, where

$$b_i = \lceil b_N/(n - 1) \rceil$$

is the number of blocks in the largest partition of the smaller relation N, and $m = n - 1$ is the number of partitions

- So, supposing a perfect uniform distribution of tuples among the partitions, the following has to be satisfied

$$(n - 2)(n - 1) \geq b_N$$

in order to perform a Partition Hash-Join

Comparing Costs of pjp and jp

- It is not always justified to perform project of the inner loop join relation before performing join
- Consider:

$$C(pjp) = C(prjM) + C(NjoinM_1) + C(prj(NjoinM_1))$$

and

$$C(jp) = C(NjoinM) + C(prj(NjoinM))$$

in the case of nested loop join algorithm

Cost of the Set Theoretic Operations

- Generally, union, set difference, and intersection require first to sort relations N and M
- After sorting, relations have to be red into main memory, block by block
- Then the tuples from these blocks are compared
- Tuples that satisfy the condition of the operation in question, are output into result relation
- The cost function is computed in a very similar fashion as for sort - merge join
- The only difference is in computing the number of result relation blocks b

Cost of the set theoretic operations

- The number of blocks b of the result relation depends on the type of the set theoretic operation
  - For union $N \cup M$ operation, $0 \leq b \leq b_N + b_M$
  - For set difference $N \setminus M$ operation, $0 \leq b \leq b_N$
  - For intersection $N \cap M$ operation, $0 \leq b \leq min(b_N, b_M)$
- Due to the huge cost, the optimizer avoids execution of the Cartesian product operation, unless forced to (if FROM clause contains more relations, and there is no join condition in the WHERE clause)

Cost of aggregate functions

- Executing just one aggregate function (without grouping) requires reading all b blocks into main memory and outputing just one result block, so

$$C = b + 1$$

- Executing an aggregate function with *grouping* requires preliminary sorting of the relation, unless there exist appropriate access structure on the whole <group_list> (like cluster index) that makes sorting unnecessary
- Then reading all b blocks into main memory, and
- Finally (supposing that grouping attributes have selection cardinality s) to output $\lceil s/f \rceil$ blocks

Left Deep Tree

- Even for $m = 3$, there are, in principle, 6 left deep join trees to be considered:
  - $(N_1 \triangleright\triangleleft N_2) \triangleright\triangleleft N_3$ $(N_2 \triangleright\triangleleft N_1) \triangleright\triangleleft N_3$
  - $(N_1 \triangleright\triangleleft N_3) \triangleright\triangleleft N_2$ $(N_3 \triangleright\triangleleft N_1) \triangleright\triangleleft N_2$
  - $(N_2 \triangleright\triangleleft N_3) \triangleright\triangleleft N_1$ $(N_3 \triangleright\triangleleft N_2) \triangleright\triangleleft N_1$
- The number of joins to be analyzed can be smaller, if there is a select σ_C operation applied to say N_1, so that:

$$\sigma_C(N_1) < N_2 \text{ and } \sigma_C(N_1) < N_3$$

- Then only

$$(N_1 \triangleright\triangleleft N_2) \triangleright\triangleleft N_3,$$
$$(N_1 \triangleright\triangleleft N_3) \triangleright\triangleleft N_2,$$

  have to be considered

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Query Optimization
Extra Tutorial

Lecturer Dr Pavle Mogin

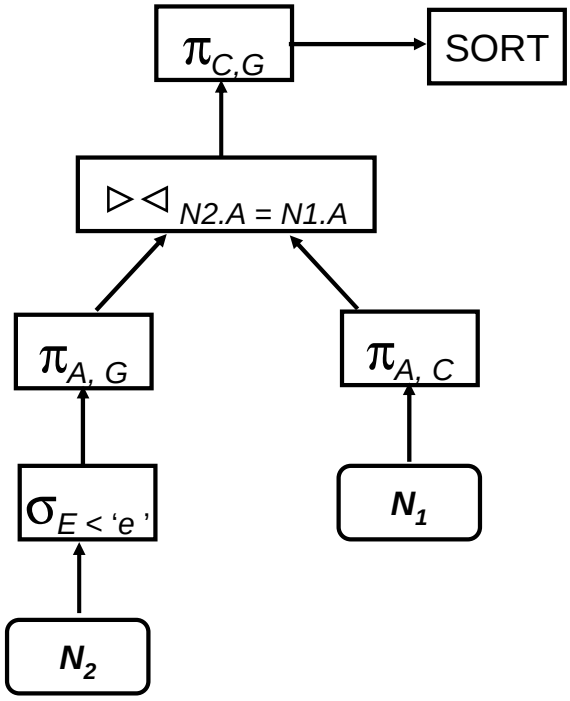
COMP302
Database Systems

Query Optimization Example

$N_1(\{A, B, C, D\}, \{A\})$
$N_2(\{E, F, G, A\}, \{F\})$

```
SELECT C, G FROM N1, N2
WHERE N1.A = N2.A AND E < 'e'
ORDER BY C;
```

Heuristic Optimization Tree



Cost Optimization

- Cost optimization should follow closely (but not strictly) result of the heuristic optimization
- So, given:
  - Tuple sizes, numbers of tuples,
  - Size of the buffer pool, maximum size of a block,
  - Selection cardinalities,
  - Indexes,…

  calculate the smallest cost of:
  - $\sigma_{E < 'e'}(N_2)$
  - $\pi_{A, G}(N_2)$
  - $\pi_{A, C}(N_1)$
  - $\triangleright\triangleleft_{N2.A = N1.A}$
  - $\pi_{C, G}(N_1 \triangleright\triangleleft N_2)$

Cost of a SELECT

- First calculate:
  - The number of N_1 and N_2 blocks
  - The number of buffers available
- Calculate $\sigma_{E<\text{`}e\text{'}}(N_2)$:
  - Suppose perfect uniform distribution and find how many N_2 tuples satisfy E < 'e' condition
  - Choose between:
    - A linear search, and
    - An index search
  - Using formulae:
    - C(linear) < C(index) $\Leftrightarrow$ b < x + s (for secondary index)
    - C(linear) < C(index) $\Leftrightarrow$ b < x + $\lceil$ s / f $\rceil$ (for cluster index)

Cost of a Project

- When calculating the cost of $\pi_{A,\,G}(N_2)$, the number of input blocks should be determined by the output from the previous operation (which was $\sigma_{E<\text{`}e\text{'}}(N_2)$
- Use formulae:
  - C(project) = b_1 + b_2

Cost of a Join

- Inputs are outputs of previous operations (two projects in the case considered)
- First calculate the join selectivity and the blocking factor of the output
- Mind that equi join retains all columns
- Then compare the costs of two (out of four) join algorithms
- Compare the cost of the winner with the next
- Mind, the partition hash join is very greedy on memory space

Cost of a Sort

- If possible, perform sort at the end (after deleting all tuples and columns not needed in the result)
- The input in the sort algorithm is the output from the previous operation ($\pi_{C,\,G}(N_1 \rhd\lhd N_2)$ in the case considered)
- Apply m-way sort – merge algorithm (log_m)
- Mind, it is not always possible to defer the sort till the end of the query execution