Query Processing and Optimization

Strathmore UNIVERSITY

Wednesday, June 23, 2021

Reference Material

- Chapter 19-Fundamentals of Database Systems

Learning Outcome

- ✓ Introduction to Query Processing
- ✓ Translating SQL Queries into Relational Algebra
- ✓ Cost Based Estimation for Query Optimization
- ✓ Algorithms for Relational Algebra Operations
- ✓ Using Heuristics in Query Optimization

Introduction

A query expressed in a high-level query language such as **SQL** must be scanned, parsed, and validate.

- **Scanner:** identify the language tokens.
- **Parser:** check query syntax.
- **Validate**: check all attribute and relation names are valid.
- An internal representation **(query tree or query graph)** of the query is created after scanning, parsing, and validating.

Then DBMS must devise an **execution strategy** for retrieving the result from the database files

Introduction to Query Processing and Optimization

Strathmore UNIVERSITY

- **Query optimization**: the process of choosing a suitable execution strategy for processing a query.

- Two internal representations of a query
  - **Query Tree**
  - **Query Graph**

Techniques for Implementing Query Optimization

There are two main techniques for implementing query optimization.

1. Heuristic rules(Logical) -for re-ordering the operations in a query.

2. Systematically estimating the cost of different execution strategies and choosing the lowest cost estimate.

Processing a Query

- Typical steps in processing a high-level query
  1. *Query in a high-level query language like SQL*
  2. **Scanning, parsing, and validation**
  3. *Intermediate-form of query like query tree*
  4. **Query optimizer**
  5. *Execution plan*
  6. **Query code generator**
  7. *Object-code for the query*
  8. **Run-time database processor**
  9. *Results of query*

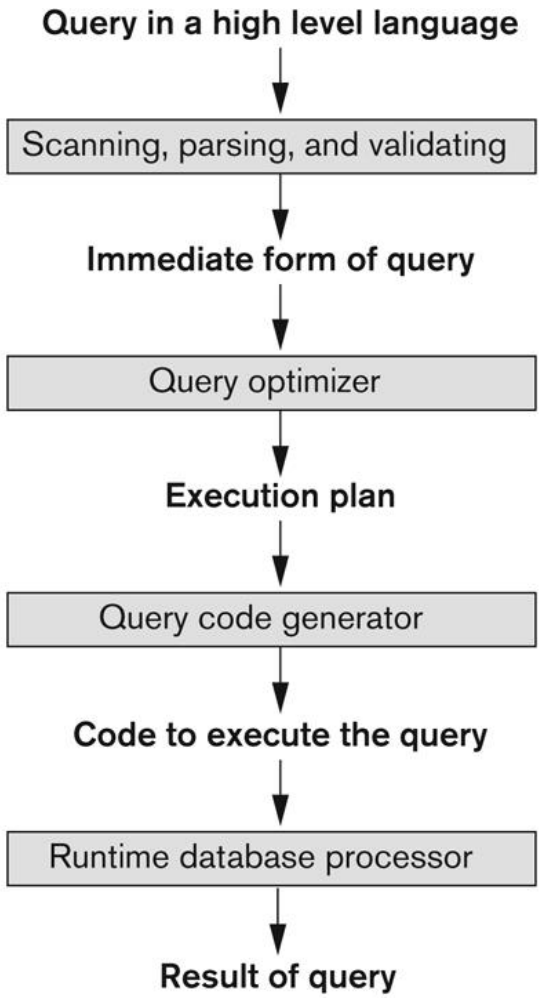


Figure 15.1
Typical steps when processing a high-level query.

Code can be:

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

SQL Queries and Relational Algebra

- SQL query is translated into an equivalent ***extended*** relational algebra expression --- represented as a **query tree**
- In order to transform a given query into a query tree, the query is decomposed into ***query blocks***
  - **Query block**:
    - The basic unit that can be translated into the algebraic operators and optimized.
  - A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.
- The query optimizer chooses an execution plan for each block

SQL Queries and Relational Algebra (1)

- Example

```
SELECT  Lname, Fname
FROM EMPLOYEE
WHERE Salary > ( SELECT MAX(Salary)
                 FROM EMPLOYEE
                 WHERE Dno = 5        )
```

Translating SQL Queries into Relational Algebra

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > (    SELECT    MAX (SALARY)
                        FROM      EMPLOYEE
                        WHERE     DNO = 5);
```

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > C
```

$\pi_{LNAME,\ FNAME}\ (\sigma_{SALARY>C}(EMPLOYEE))$

```
SELECT MAX (SALARY)
FROM      EMPLOYEE
WHERE     DNO = 5
```

$\mathcal{F}_{MAX\ SALARY}\ (\sigma_{DNO=5}\ (EMPLOYEE))$

Algorithms for Relational algebra Operations and Cost Estimation

Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - *disk accesses*, *CPU*, or even network *communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
    - Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful

Measures of Query Cost (Cont.)

- For simplicity we just use *number of block transfers from disk* as the cost measure
  - We ignore the difference in cost between sequential and random I/O for simplicity
  - We also ignore CPU costs for simplicity
- Costs depends on the size of the buffer in main memory
  - Having more memory reduces need for disk access
  - Amount of real memory available to buffer depends on other concurrent OS processes, and hard to determine ahead of actual execution
  - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account
- We do not include cost to writing output to disk in our cost formulae

Strathmore
UNIVERSITY

Algorithms Covered

- Selection Algorithms
- Sorting Algorithms
- Join Algorithms

Selection Operation

- **File scan** – search algorithms that locate and retrieve records that fulfill a selection condition.
- Algorithm **S1** (*linear search*). Scan each file block and test all records to see whether they satisfy the selection condition.
  - Cost estimate (number of disk blocks scanned) = b_r
    - b_r denotes number of blocks containing records from relation *r*
  - If selection is on a key attribute, cost = $(b_r/2)$
    - stop on finding record
  - Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices

Selection Algorithms

- **S1—Linear search -** : Retrieve *every record* in the file, and test whether its attribute values satisfy the selection condition.

Example 1

Suppose their exists an account relation with the following statistics:

10,000 records, blocking factor= 20, 50 distinct branches, 500 distinct a/c balance values:

a) How many disk blocks are needed to store all records?

b) Suppose a linear search is conducted on $\sigma_{account\_no=500}$ **(ACCOUNT) :**

I) How many disk accesses will be needed to find a record that matches the predicate?

II) Suppose no record matches the predicate, how many disk accesses will be needed to scan for a match?

Selection Operation (Cont.)

- **S2** *(binary search).* Applicable if selection is an equality comparison on the attribute on which file is ordered.
  - Assume that the blocks of a relation are stored contiguously
  - Cost estimate (number of disk blocks to be scanned):

$$E_{A2} = \lceil \log_2(b_r) \rceil + \left\lceil \frac{SC(A, r)}{f_r} \right\rceil - 1$$

  - $\lceil \log_2(b_r) \rceil$ — cost of locating the first tuple by a binary search on the blocks
  - $SC(A, r)$ — number of records that will satisfy the selection
  - $\lceil SC(A, r)/f_r \rceil$ — number of blocks that these records will occupy
- Equality condition on a key attribute: $SC(A, r) = 1$; estimate reduces to $E_{A2} = \lceil \log_2(b_r) \rceil$

Selection Operator Algorithms

Strathmore UNIVERSITY

S3a—Using a primary index.

If the selection condition involves an equality comparison on a key attribute with a primary index— use the primary index to retrieve the record.

Cost of operation

Number of index levels accessed plus 1

nLevelsA(I) + 1

NB: This condition retrieves a single record at most

Selection Operator Algorithms

Strathmore UNIVERSITY

S3b—Using a hash key.

If the selection condition involves an equality comparison on a key attribute with a hash key-Use a hash key

- If attribute A is the hash key, then we apply the hashing algorithm to calculate the target address for the tuple.

Cost Of Operation

- If there is no overflow, the expected cost is 1.
- If there is overflow, additional accesses may be necessary, depending on the amount of overflow and the method for handling overflow.

NB: This condition retrieves a single record at most

Selection Operator Algorithms

Strathmore UNIVERSITY

S4—Using a primary index to retrieve multiple records.

If the comparison condition is >, >=, <, or <= on a key field with a primary index

- Then we can first use the index to locate the tuple satisfying the predicate A = x. (Provided the index is sorted)
- Then the required tuples can be found by accessing all tuples before or after this one.

Example:

- For $\sigma_{A>=V}(R)$ use index to find first tuple v and scan relation sequentially from there
- For $\sigma_{A<=V}(R)$ just scan relation sequentially till first tuple > **v** do not use index

Cost of Operation

- Assuming uniform distribution, then we would expect half the tuples to satisfy the inequality, so the estimated cost is:

nLevelsA(I) + [nBlocks(R)/2]

Selection Operator Algorithms

Strathmore UNIVERSITY

S5—Using a clustering index to retrieve multiple records.

- If the predicate involves an equality condition on attribute A, which is not the primary key but does provide a clustering secondary index, then we can use the index to retrieve the required tuples. The estimated cost is:

  nLevelsA(I) + [SC_A(R)/bFactor(R)]

- The second term is an estimate of the number of blocks that will be required to store the number of tuples that satisfy the equality condition, which we have estimated as SC_A(R).

S6—Using a secondary index on an equality comparison

- This search method can be used to retrieve a single record if the indexing field is a key (has unique values) or to retrieve multiple records if the indexing field is not a key.
- This can also be used for comparisons involving >, >=, <, or <=.

Cost of operation

Key attribute= x +1 block accesses

- **Retrieve multiple records** if search-key is **not a candidate key**

Each of n matching records may be on a different block

Cost = (x + n) (n- number of records)

Strathmore
UNIVERSITY

Selection Operator Algorithms
(For Complex Selections)

Selection Algorithms (Complex)

Strathmore UNIVERSITY

Conjunctive Selection: Several condition with AND

- Single index: retrieve records satisfying some attribute condition (with index) and check remaining conditions
- Composite index (Hash Structure)
- Intersection of multiple indexes

Disjunctive Selections (Several Conditions with OR)

- Index/binary search if all conditions have access path and take union
- Linear search otherwise

Selection Operator Algorithms

Strathmore UNIVERSITY

S7—Conjunctive selection using an individual index.

- If an attribute involved in any single simple condition in the conjunctive select condition has an access path (ordered,/indexed) that permits the **use of one of the methods S2 to S6**, use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive select condition.

Selection Operator Algorithms

Strathmore UNIVERSITY

S8—Conjunctive selection using a composite index.

- If two or more attributes are involved in equality conditions in the conjunctive select condition and a **composite index** (or hash structure) exists on the combined fields— use the composite index

Example:

OP5: $\sigma_{Essn='123456789' \text{ AND } Pno=10}(WORKS\_ON)$

WORKS_ON file for—we can use the index directly.

Selection Operator Algorithms

Strathmore UNIVERSITY

S9—Conjunctive selection by intersection of record pointers.

- If **secondary indexes** (or other access paths) are available **on more than one of the fields** involved in simple conditions in the conjunctive select condition, and if the **indexes include record pointers** (rather than block pointers), then each index can be used to retrieve the set of record pointers that satisfy the individual condition. \
- The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive select condition, which are then used to retrieve those records directly.
- If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions..

In general, method S9 assumes that each of the indexes is on a nonkey field of the file, because if one of the conditions is an equality condition on a key field, only one record will satisfy the whole condition

Break Out Question

Consider a relation **s** over the attributes **A** and **B** with the following characteristics: 7,000 tuples with 70 tuples per block(page) and a hash index on attribute **A** .The values that the attribute A takes in relation **s** are integers that are uniformly distributed in the range 1 – 200.

i. Assuming that the aforesaid index on **A** is unclustered, estimate the number of disk accesses(Cost) needed to compute the query **σA=18(s).**

ii. What would be the cost estimate if the index were clustered? Explain your reasoning.

Sorting Algorithms

Sorting

- External sorting refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory.
- Use a sort-merge strategy, which starts by sorting small subfiles – called runs –of the main file and merges the sorted runs, creating larger sorted subfiles that are merged in turn.
- The algorithm consists of two phases: sorting phase and merging phase.

Sorting Algorithms

Sorting is one of the primary algorithms used for query processing

ORDER BY

DISTINCT

JOIN

- Relations that **fit in memory** — use techniques like quicksort, merge sort, bubble sort
- Relations that don’**t fit in memory** — external sort-merge

Sorting algorithm

- Note that sorting of a particular file may be avoided if an appropriate index—such as a primary or clustering index exists on the desired file to allow ordered access to the records of the file.
- The **sort-merge algorithm,** like other database algorithms, requires *buffer space* in main memory, where the actual sorting and merging of the runs is performed.
- Sort-Merge Algorithm consists of two phases: the **sorting phase and the merging phase**.
- The buffer space in main memory is part of the **DBMS cache**—an area in the computer's main memory that is controlled by the DBMS.
- The buffer space is divided into individual buffers, where each **buffer** is the same size in bytes as the size of one disk block. Thus, one buffer can hold the contents of exactly *one disk block*.

Sort-merge Algorithm- Sorting Phase

- ✓ In the sorting phase, runs (portions or pieces) of the file that can fit in the available buffer space are read into main memory, sorted using an internal sorting algorithm, and written back to disk as temporary sorted subfiles (or runs).
- ✓ The size of each run and the number of initial runs (nR) are dictated by the number of file blocks (b) and the available buffer space (nB).

Sort-Merge Continued---Example

- ✓ For example, if the number of available main memory buffers nB = 5 disk blocks and the size of the file b = 1024 disk blocks, then nR= $\lceil (b/nB) \rceil$ or 205 initial runs each of size 5 blocks (except the last run which will have only 4 blocks).
- ✓ Hence, after the sorting phase, 205 sorted runs (or 205 sorted subfiles of the original file) are stored as temporary subfiles on disk.

Sort-merge Algorithm- Merging Phase

- ✓ In the **merging phase**, the sorted runs are merged during one or more **merge passes**.
- ✓ Each merge pass can have one or more merge steps.
- ✓ The **degree of merging(d_M)** is the number of sorted subfiles that can be merged in each merge step.
- ✓ During each merge step, one buffer block is needed to hold one disk block from each of the sorted subfiles being merged, and one additional buffer is needed for containing one disk block of the merge result, which will produce a larger sorted file that is the result of merging several smaller sorted subfiles.
- ✓ Hence, d_M is the smaller of ($nB - 1$)and nR, and the number of merge passes is $[(\log d_M(nR))]$.

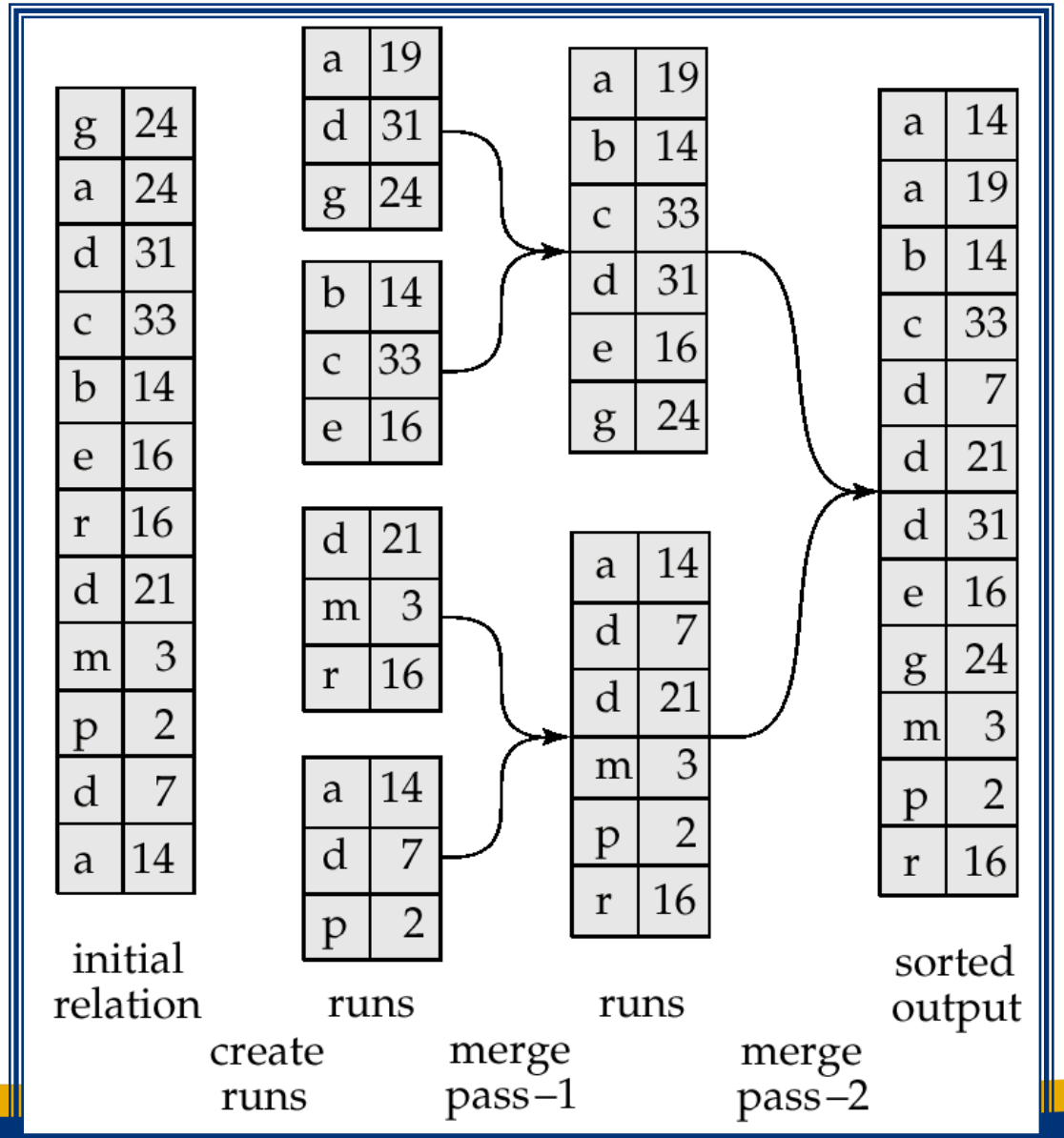
Sort-merge Algorithm- Sorting Phase Example

- ✓ In our example where nB =5, d_M = 4 (four-way merging), so the 205 initial sorted runs would be merged 4 at a time in each step into 52 larger sorted subfiles at the end of the first merge pass.
- ✓ These 52 sorted files are then merged 4 at a time into 13 sorted files, which are then merged into 4 sorted files, and then finally into 1 fully sorted file, which means that four passes are needed.

External Sort-Merge

```
set     i ← 1;
        j ← b;              {size of the file in blocks}
        k ← n_B;            {size of buffer in blocks}
        m ← ⌈(j/k)⌉;
{Sorting Phase}
while (i ≤ m)
do {
        read next k blocks of the file into the buffer or if there are less than k blocks
            remaining, then read in the remaining blocks;
        sort the records in the buffer and write as a temporary subfile;
        i ← i + 1;
}
{Merging Phase: merge subfiles until only 1 remains}
set     i ← 1;
        p ← ⌈log_{k-1} m⌉   {p is the number of passes for the merging phase}
        j ← m;
while (i ≤ p)
do {
        n ← 1;
        q ← ⌈j/(k-1)⌉ ;     {number of subfiles to write in this pass}
        while (n ≤ q)
        do {
            read next k-1 subfiles or remaining subfiles (from previous pass)
                one block at a time;
            merge and write as new subfile one block at a time;
            n ← n + 1;
        }
        j ← q;
        i ← i + 1;
}
```

Example: External Sorting Using Sort-Merge



External Merge Sort (Cont.)

- Cost analysis:

✓ The performance of the sort-merge algorithm can be measured in the number of disk block reads and writes (between the disk and main memory) before the sorting of the whole file is completed. The following formula approximates this cost:

$$(2 * b) + (2 * b * (\log d_M\ nR))$$

✓ The first term **(2 * *b*)** represents the number of block accesses for the sorting phase, since each file block is accessed twice: once for reading into a main memory buffer and once for writing the sorted records back to disk into one of the sorted subfiles.

✓ The second term represents the number of block accesses for the merging phase.

✓ During each merge pass, a number of disk blocks approximately equal to the original file blocks *b* is read and written. Since the number of merge passes is **($\log d_M\ nR$)**,we get the total merge cost of **(2 * *b* * (log*dM nR*)).**

Class Activity

A file of 4096 blocks is to be sorted with an available buffer space of 64 blocks.

a) How many passes will be needed in the merge phase of the external sort-merge algorithm?

b) Calculate the cost of sort-merge algorithm sorting of the file

Join Operation Algorithms

Join Operation

- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join
  - Hash-join
- Choice based on cost estimate
- Examples of Join Operation use the following information
  - Number of records of *customer*: 10,000 *depositor*: 5000
  - Number of blocks of *customer*: 400 *depositor*: 100

Join Operation Algorithms

Strathmore UNIVERSITY

- **Nested Loop Join** $r \bowtie_\theta s$

- For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition
- Outer loop should be smaller than inner loop
- Requires no indices and can be used with any kind of join condition.
- Expensive since it examines every pair of tuples in the two relations

Nested-Loop Join (Cont.)

- Examples use the following information
  - Number of records of *customer*: 10,000 *depositor*: 5000
  - Number of blocks of *customer*: 400 *depositor*: 100

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is
  $n_r * b_s + b_r$ disk accesses.

- If the smaller relation fits entirely in memory, use that as the inner relation. Reduces cost to
- $b_r + b_s$ disk accesses.
- Assuming worst case memory availability cost estimate is
  - 5000 ∗ 400 + 100 = 2,000,100 disk accesses with *depositor* as outer relation, and
  - 1000 ∗ 100 + 400 = 1,000,400 disk accesses with *customer* as the outer relation.
- If smaller relation (*depositor)* fits entirely in memory, the cost estimate will be 500 disk accesses.

Join Operation Algorithms

Strathmore UNIVERSITY

❑ Block Nested Loop Join

❑ Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation

Worst case estimate: $b_r * b_s + b_r$ block accesses.

– Each block in the inner relation *s* is read once for each *block* in the outer relation (instead of once for each tuple in the outer relation

- Best case: $b_r + b_s$ block accesses.

Join Operation Algorithms

Strathmore UNIVERSITY

❑ Indexed Nested Loop Join

❑ Index is available on inner loop's join attribute — use index to compute the join

- For each tuple t_r in the outer relation r, use the index to look up tuples in s that satisfy the join condition with tuple t_r.
- Worst case: buffer has space for only one page of r, and, for each tuple in r, we perform an index lookup on s.

- Cost of the join: $b_r + n_r * c$

  - Where c is the cost of traversing index and fetching all matching s tuples for one tuple or r
  - c can be estimated as cost of a single selection on s using the join condition.

Indexed Nested-Loop Join

- Index lookups can replace file scans if

  a) join is an equi-join or natural join and

  b) an index is available on the inner relation’s join attribute

Example of Nested-Loop Join Costs

- Compute *depositor* ⋈ *customer*, with *depositor* as the outer relation.
- Let *customer* have a primary B^+-tree index on the join attribute *customer-name*, which contains 20 entries in each index node.
- Since *customer* has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data
- *depositor* has 5000 tuples
- Cost of block nested loops join
  - 400*100 + 100 = 40,100 disk accesses assuming worst case memory (may be significantly less with more memory)
- Cost of indexed nested loops join
  - 100 + 5000 * 5 = 25,100 disk accesses.
  - CPU cost likely to be less than that for block nested loops join

Join Operation Algorithms-Class Activity

Strathmore UNIVERSITY

Question 1: Block Nested loop join algorithm

- Suppose you have two relations: Employee with 6000 records stored in 2000 blocks and Department, with 50 records stored in 10 blocks. Assume that the number of buffers available in main memory for implementing the join is nB = 7 blocks (buffers)., and a buffer is the same size as a block.

Questions

a) What will be the total number of block accesses made using Employee as the outer loop?

b) What will be the number of block accesses made using Department as the outer loop?

Join Operation Algorithms

Strathmore UNIVERSITY

❑ Hash-Join

- ❑ Only applicable in case of **equijoin or natural join**
- ❑ A **hash function *h*** is used to partition tuples of both relations into sets that have the **same hash value** on the **join attribute.**
- ❑ Tuples in the corresponding same buckets just need to be compared with one another and not with all the other tuples in the other buckets

Cost of operation:

- ❑ 3(Br +Bs) (hashing the values, writing values to disk, joining the attributes)

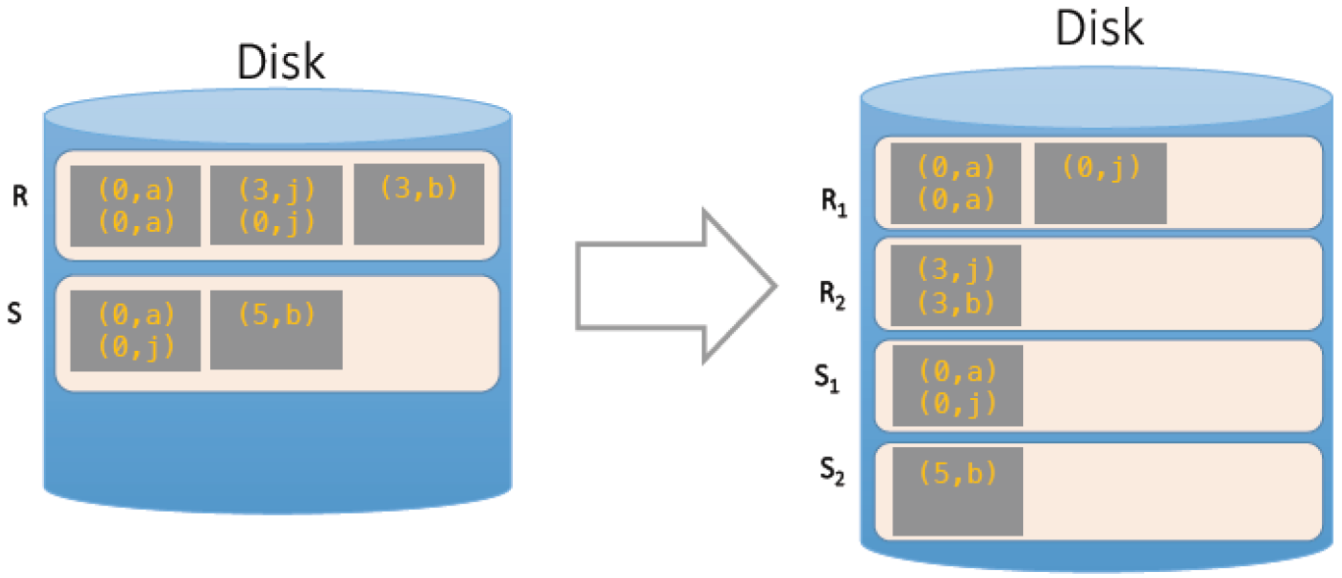
Partitioning phase: 2bR + 2bS

- Joining phase: bR + bS
- **Total: 3bR + 3bS**

Join Operation Algorithms

Strathmore UNIVERSITY

- **Hash-Join- Example**

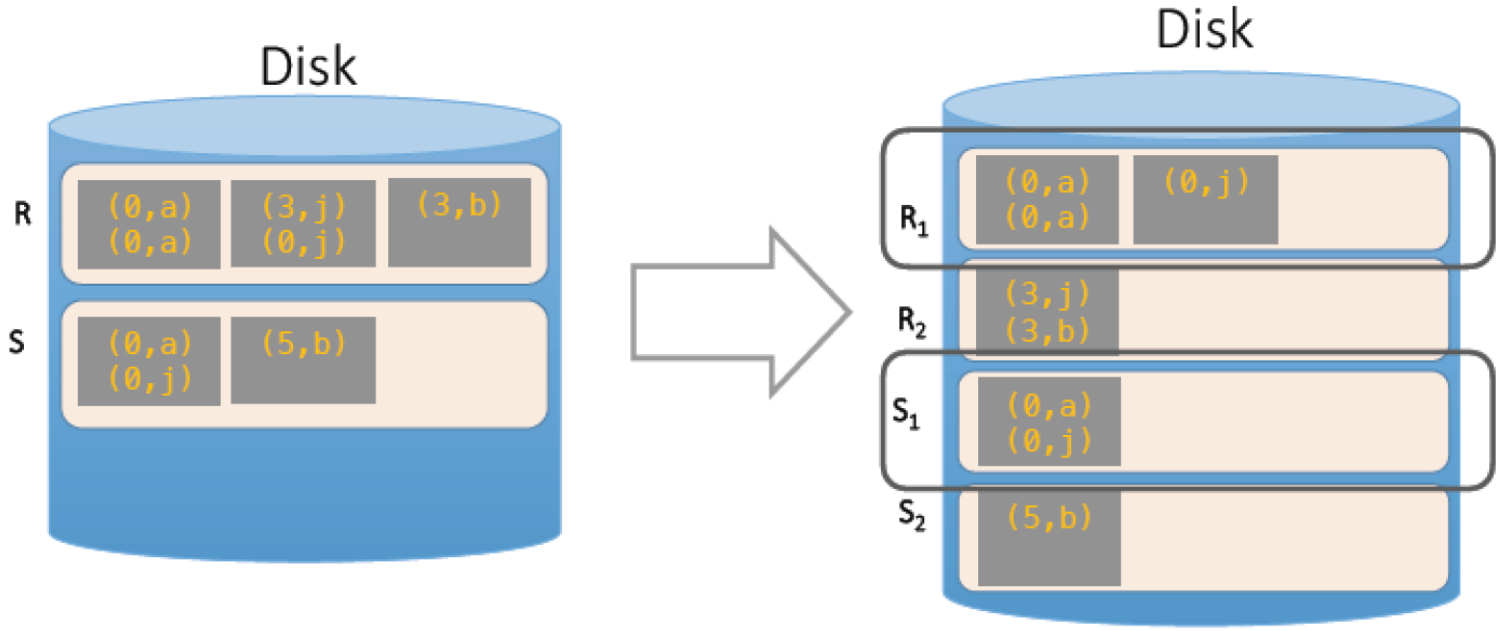


Step 1: Use hash function to partition into B buckets

Join Operation Algorithms

Strathmore UNIVERSITY

- **Hash-Join- Example**

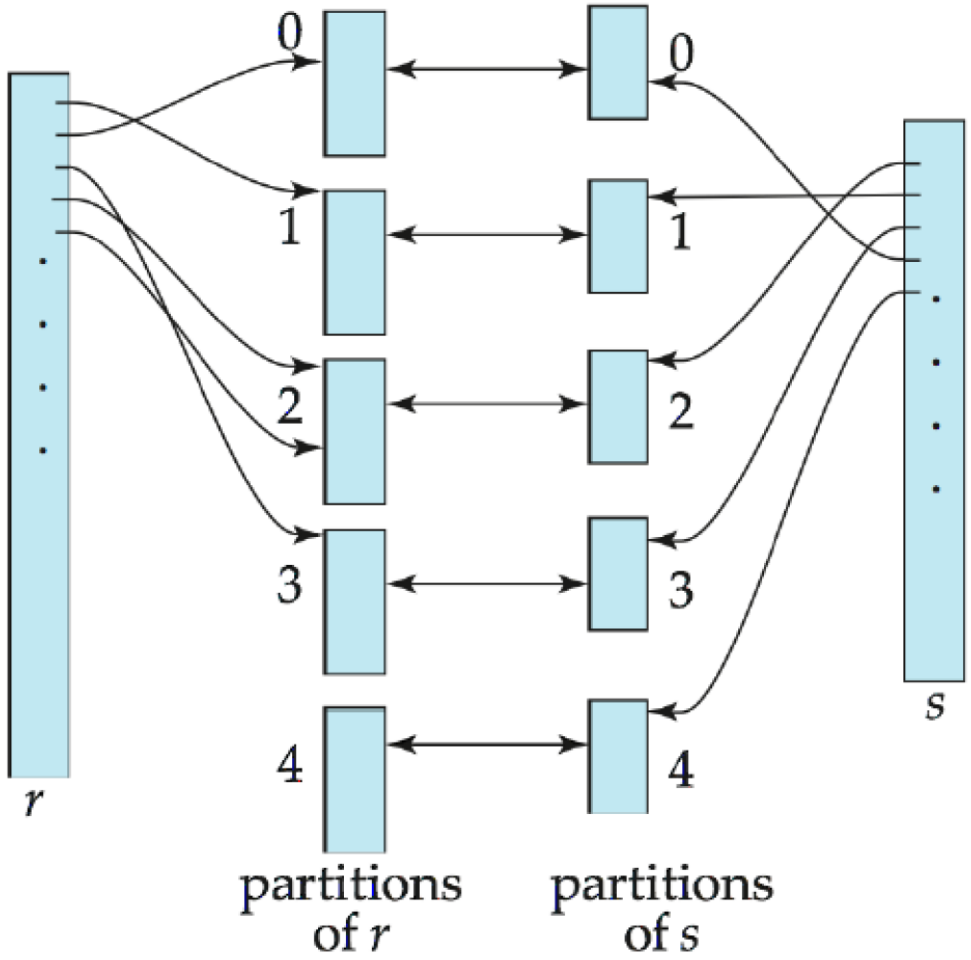


Step 2: Join matching buckets

Join Operation Algorithms

Strathmore UNIVERSITY

- **Hash-Join- Example**



Heuristic and Equivalence Rules

Equivalence Rules

Strathmore UNIVERSITY

❑ Commutative and Associative laws

❑ Commutative:

Order of arguments in an operator does not matter, the result is the same.

Example:

X + y =y + x

X * y = y * x

❑ Associative law

We may group two uses of the operator either from the left or the right.

Example:

(x + y) + z = x + (y + z)

Equivalence Rules

Strathmore UNIVERSITY

❑ Commutative and Associative laws

Several relational algebra operations are both associative and commutative. Particularly:

Example:

Commutative	Associative
$R \times S = S \times R$;	$(R \times S) \times T = R \times (S \times T)$
$R \cup S = S \cup R$;	$(R \cup S) \cup T = R \cup (S \cup T)$
$R \cap S = S \cap R$;	$(R \cap S) \cap T = R \cap (S \cap T)$
$R \bowtie S = S \bowtie R$;	$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Equivalence Rules

Strathmore UNIVERSITY

❑ Selection laws

- ❑ Selection reduces the size of a relation
- ❑ Move selections down the tree without changing what expression does
- ❑ If selection is complex (AND,OR) break it down to constituent parts

a. Splitting rule

$\sigma_{C1 \text{ AND } C2}(R) = \sigma_{c1}(\sigma_{c2}(R))$

$\sigma_{C1 \text{ OR } C2}(R) = \sigma_{c1}(R) \cup (\sigma_{c2}(R))$

b For a union the selection must be pushed to both arguments

$\sigma_{C}(R \cup S) = \sigma_{c}(R) \cup \sigma_{c}(S)$

c. For a difference, selection must be pushed to the first argument and optionally to the second

$\sigma_{C}(R - S) = \sigma_{c}(R) - (S)$ **OR** $\sigma_{C}(R - S) = \sigma_{c}(R) - \sigma_{c}(S)$

Equivalence Rules

Strathmore UNIVERSITY

❑ Selection laws

d. Selection can be pushed to one or more arguments.

$\sigma_C (R \times S) = \sigma_c (R) \times (S)$

$\sigma_C (R \cap S) = \sigma_c (R) \cap (S)$

$\sigma_C (R \bowtie S) = \sigma_c (R) \bowtie (S)$

$\sigma_C (R \bowtie_D S) = \sigma_c (R) \bowtie_D (S)$

If C has only attributes of S then we can write:

$\sigma_C (R \times S) = R \times \sigma_c (S)$ (Similar for the other operators: $\bowtie$, $\bowtie_D$, $\cap$)

Should relation R and S both happen to have all attributes of C then we can use the law :

$\sigma_C (R \bowtie S) = \sigma_c (R) \bowtie \sigma_c (S)$ (Impossible for X, $\bowtie_D$), since in those cases R and S have no shared attributes

Equivalence Rules

Strathmore UNIVERSITY

❑ Projection laws

- ❑ Projection reduces the size of the row
- ❑ Introduce a new projection somewhere below an existing projection

$\Pi_L(R \bowtie S) = \Pi_L((\Pi_M(R) \bowtie \Pi_N(S))$ – where M and N are the join attributes and the input attributes of L that are found among the attributes of R and S.

$\Pi_L(R \bowtie_C S) = \Pi_L((\Pi_M(R) \bowtie_c \Pi_N(S))$ - – where M and N are the join attributes (ie those mentioned in condition C) and the input attributes of L that are found among the attributes of R and S respectively.

Equivalence Rules

Strathmore UNIVERSITY

❑ Joins and Product laws

❑ Additional laws follow:

$R \bowtie_c S = \sigma_c (R) \times (S)$

$R \bowtie S = \Pi_L (\sigma_c (R \times S))$

Where C is the condition that equates each pair of attributes, from R and S with the same name, and L is a list that includes one attribute from each equated pair, and all other attributes of R and S

Strathmore
UNIVERSITY

Enumeration of alternative execution strategies

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

- When a join involves more than two relations the number of possible join trees grow rapidly.
- Fundamental to the efficiency of query optimization is the **search space** of possible execution strategies and the **enumeration algorithm** that is used to search this space for an optimal strategy.

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

- **Pipelining (stream based processing / on-the-fly processing)**

  - In this method of executing a query plan, tuples produced by one operation are passed directly to an operation that uses it without storing the intermediate tuples on disk. (Implemented by a network of operators.)
  - It saves disks I/O ‘s

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

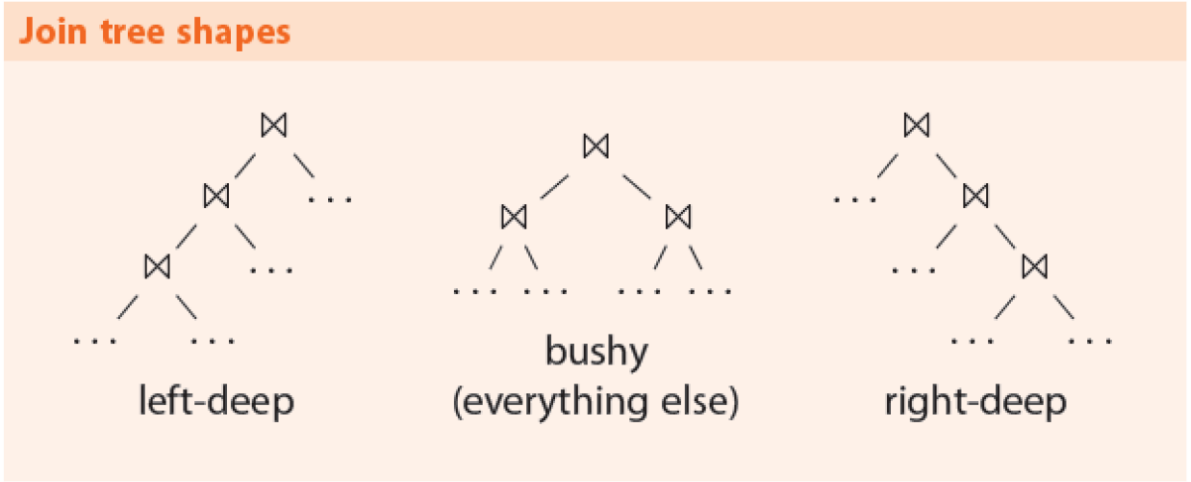
Materialization

- In this method of executing a query plan, the result of one operation are stored on a disk until it is needed by another operation. (Each intermediate operation is materialized on a disk.)

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

- **Join tree Shapes**



- A binary tree is a left deep tree if all right children are leaves. For a join relation the left child is the outer relation while the right child is the inner relation.
- A tree whose left children are leaves is called a right-deep tree.
- A tree which is neither a left deep tree or a right deep tree is called bushy tree.
- Left and right join trees are known as linear trees while bushy trees are known as non-linear trees.

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Left-deep join trees

❑ **Actual systems** often **prefer left-deep join** trees.1

- The inner (rhs) relation always is a base relation.
- Allows the use of index nested loops join.
- Easier to implement in a pipelined fashion

❑ Inner relations are always materialized since we need to compare inner relation for each tuple of outer relation.

❑ Left deep trees have the advantages of reducing the search space for the optimum strategy, and allowing the query optimizer to be based on dynamic processing techniques

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Reducing the search space

Search space is restricted in several ways which include:

1. Unary Operations (Projection and Selection) are processed on the fly. Selections are processed as relations are accessed for the first time; projections are processed as the results of other operations are generated.

2. Cartesian products are never formed unless the query itself specifies one

1. The inner operand of each join is a base relation, never an intermediate result. (This uses fact that with left-deep trees inner operand is a base relation and so already materialized)

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Enumerating left deep joins (Dynamic Programming)

❑ Dynamic programming is based on the assumption that the cost model satisfies the **principle of optimality.**

Idea:

- Find the cheapest plan for an n-way join in n **passes**.
- In each pass k, find the best plans for all k-relation **sub-queries**.
- **Construct** the plans in pass k from best i-relation and (k − i)-relation sub-plans found in **earlier passes** (1 6 i < k).

Assumption:

- To find the optimal **global plan**, it is sufficient to only consider the optimal plans of its **sub-queries** (“Principle of optimality”).

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Enumerating left deep joins using dynamic programming

- ❑ To ensure some potentially useful strategies are not discarded algorithm retains strategies with **interesting orders:**
- ❑ An intermediate result has an interesting order if it is sorted by a final ORDER BY attribute, GROUP BY attribute, or any attributes that participate in subsequent joins.

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p WHERE c.maxRent <
500 AND c.clientNo = v.clientNo AND
                v.propertyNo = p.propertyNo;
```

- Attributes c.clientNo, v.clientNo, v.propertyNo, and p.propertyNo are interesting.
- If any intermediate result is sorted on any of these attributes, then corresponding partial strategy must be included in search.

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Enumerating left deep joins using dynamic programming

- ❑ Algorithm proceeds from the **bottom up** and constructs all alternative join trees that satisfy the restrictions above, as follows:
- ❑ **Pass 1:**
- Enumerate the strategies for each base relation using a linear search and all available indexes on the relation.
- These partial strategies are partitioned into equivalence classes based on any interesting orders.
- An additional equivalence class is created for the partial strategies with no interesting order.

.

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

- **Enumerating left deep joins using dynamic programming**

- For each equivalence class**, strategy with lowest cost** is **retained for consideration in next pass.**
- Do not retain equivalence class with no interesting order if its lowest cost strategy is not lower than all other strategies.
- For a given relation R, any selections involving only attributes of R are processed on-the-fly. Similarly, any attributes of R that are not part of the SELECT clause and do not contribute to any subsequent join can be projected out at this stage (restriction 1 above).

.

Enumeration of alternative execution strategies

Strathmore UNIVERSITY

❑ Enumerating left deep joins using dynamic programming

❑ **Pass 2:**

- Generate all **2-relation strategies** by considering each strategy retained after Pass 1 as outer relation, discarding any Cartesian products generated (restriction 2 above).
- Again, any on-the-fly processing is performed and lowest cost strategy in each equivalence class is retained.

Pass n:

❑ Generate all **n-relation strategies** by considering each strategy retained after Pass (n – 1) as outer relation, discarding any Cartesian products generated.

❑ After pruning, now have lowest overall strategy for processing the query.

Heuristic Rules

Heuristic Rules

Strathmore UNIVERSITY

❑ Heuristic rules

- ❑ Cost-based optimization is **expensive**, even with dynamic programming.
- ❑ Systems may use **heuristics to reduce the number of choices** that must be made in a cost-based fashion.

▫Heuristic optimization transforms the query-tree by using a **set of rules** that typically (but not in all cases) improve execution performance:

- ❑ Perform selection early (reduces the number of tuples)
- ❑ Perform projection early (reduces the number of attributes)
- ❑ Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations.