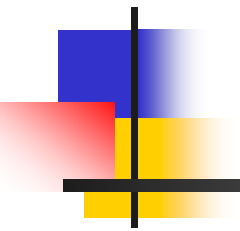
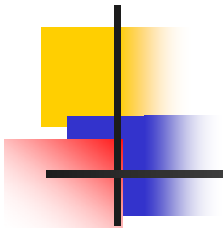


Algorithms for Query Processing and Optimization

Chapter 15



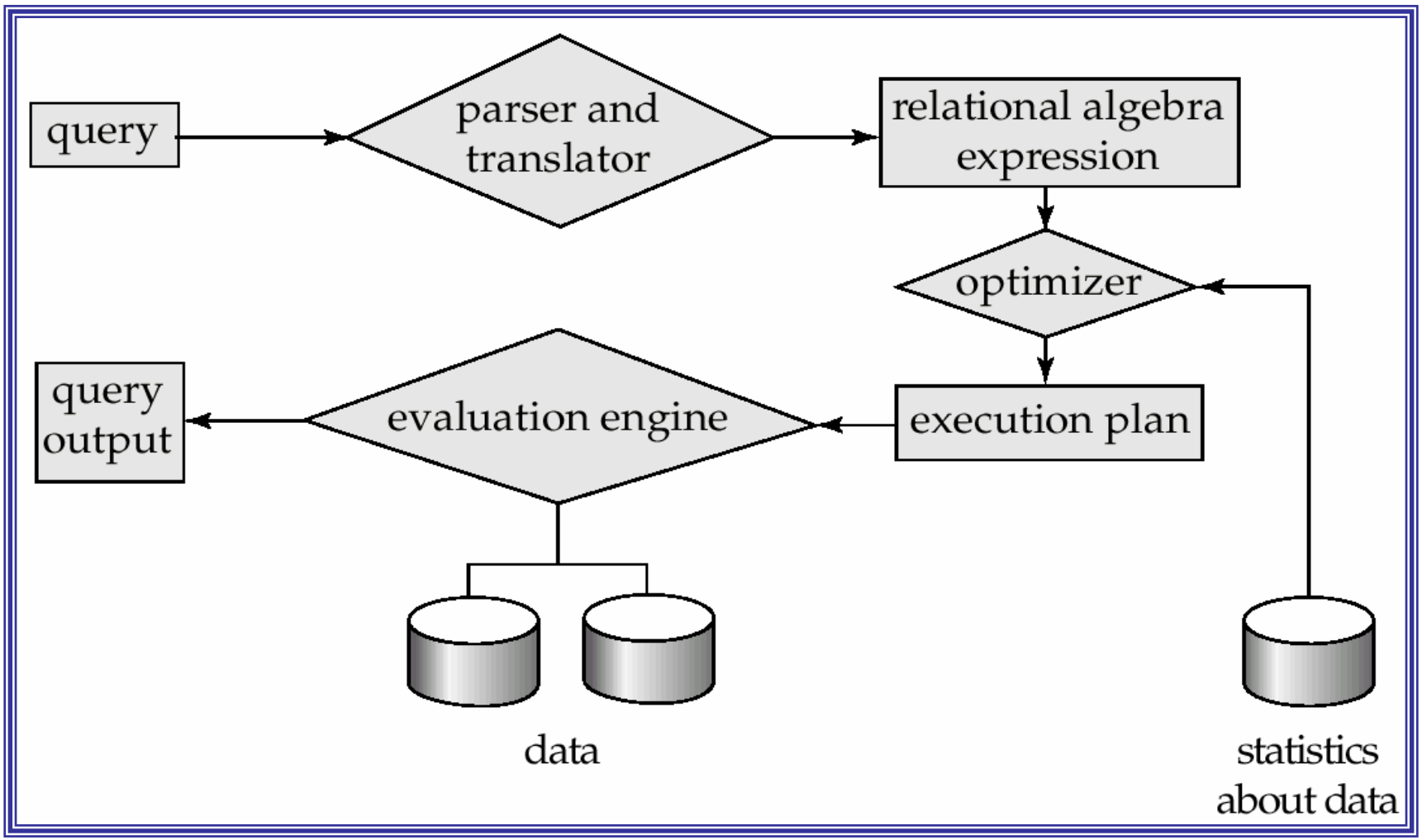
Chapter Objectives

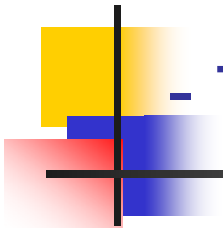
- Introduction to the process of choosing a suitable execution strategy for processing a query.

Chapter Outline

- Introduction to Query Processing
- Translating SQL Queries into Relational Algebra
- Algorithms for External Sorting
- Algorithms for the SELECT Operation
- Algorithms for the JOIN Operation
- Algorithms for the PROJECT Operation
- Using Heuristics in Query Optimization
- Using Selectivity and Cost Estimates in Query Optimization

- Introduction to Query Processing





- Translating SQL Queries into Relational Algebra …

- **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.
- **Nested queries** within a query are identified as separate query blocks.
- Aggregate operators in SQL must be included in the extended algebra.

… - 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
```

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

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

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

- Algorithms for External 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, such as most database files.

- **Sort-Merge strategy**: starts by sorting small subfiles (**runs**) of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.
  - Sorting phase: $n_R = \lceil (b/n_B) \rceil$
  - Merging phase: $d_M = \text{Min}\ (n_B\text{-}1, n_R)$; $n_P = \lceil (\log_{dM}(n_R)) \rceil$

  n_R: number of initial runs; b: number of file blocks;

  n_B: available buffer space; d_M: degree of merging;

  n_P: number of passes.

… - Algorithms for External Sorting

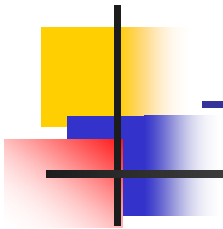
Figure 18.2 Outline of the sort-merge algorithm for external sorting.

```
set     i ← 1;
        j ← b;      {size of the file in blocks}
        k ← n_B;    {size of buffer in blocks}
        m ← ⌈(j/k)⌉;
{Sort 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 blocks in the buffer and write as a temporary subfile;
            i ← i + 1;
            }
{Merge 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;
                    n ← n + 1;
                      }
            j ← q;
            i ← i + 1;
                    }
```


Note: The above figure is now called Figure 15.2 in Edition 4

- The SELECT Operation

- Examples:
  - (OP1): $\sigma_{SSN='123456789'}(EMPLOYEE)$
  - (OP2): $\sigma_{DNUMBER>5}(DEPARTMENT)$
  - (OP3): $\sigma_{DNO=5}(EMPLOYEE)$
  - (OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}(EMPLOYEE)$
  - (OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}(WORKS\_ON)$



- Algorithms for a Simple SELECT Operation …

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

- S2. **Binary search**: If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search (which is more efficient than linear search) can be used. (See OP1).

- S3. **Using a primary index or hash key** to retrieve a single record: If the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key), use the primary index (or the hash key) to retrieve the record.

… - Algorithms for a Simple SELECT Operation …

- S4. **Using a primary index** to retrieve multiple records: If the comparison condition is >, ≥, <, or ≤ on a key field with a primary index, use the index to find the record satisfying the corresponding equality condition, then retrieve all subsequent records in the (ordered) file.

- S5. **Using a clustering index** to retrieve multiple records: If the selection condition involves an equality comparison on a non-key attribute with a clustering index, use the clustering index to retrieve all the records satisfying the selection condition.

- S6. **Using a secondary (B+-tree) index**: On an equality comparison, this search method can be used to retrieve a single record if the indexing field has unique values (is a key) or to retrieve multiple records if the indexing field is not a key. In addition, it can be used to retrieve records on conditions involving >,>=, <, or <=. (FOR **RANGE QUERIES)**

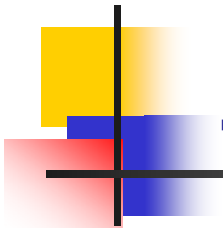
… - Algorithms for a Complex SELECT Operation …

- S7. **Conjunctive selection**: If an attribute involved in any single *simple condition* in the conjunctive condition has an access path 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 condition.

- S8. **Conjunctive selection using a composite index**: If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined field, we can use the index directly.

… - Algorithms for a Complex SELECT Operation …

- S9. **Conjunctive selection by intersection of record pointers**: This method is possible if secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the conjunctive condition and if the indexes include record pointers (rather than block pointers). Each index can be used to retrieve the *record pointers* that satisfy the individual condition. The *intersection* of these sets of record pointers gives the record pointers that satisfy the conjunctive 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.



… - Implementing SELECT Operation

- Whenever a **single condition** specifies the selection, we can only check whether an access path exists on the attribute involved in that condition. If an access path exists, the method corresponding to that access path is used; otherwise, the "brute force" linear search approach of method S1 is used. (See OP1, OP2 and OP3)

- For **conjunctive selection conditions**, whenever *more than one* of the attributes involved in the conditions have an access path, query optimization should be done to choose the access path that *retrieves the fewest records* in the most efficient way .

- Implementing the JOIN Operation ...

- Join (EQUIJOIN, NATURAL JOIN)
  - two–way join: a join on two files

    e.g. $R \bowtie_{A=B} S$
  - multi-way joins: joins involving more than two files.

    e.g. $R \bowtie_{A=B} S \bowtie_{C=D} T$

- Examples

  (OP6): EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT

  (OP7): DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

… - Implementing the JOIN Operation …

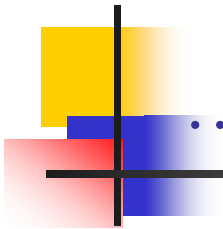
- J1. **Nested-loop join** (brute force): For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

- J2. **Single-loop join** (Using an access structure to retrieve the matching records): If an index (or hash key) exists for one of the two join attributes — say, B of S — retrieve each record t in R, one at a time, and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A].

… - Implementing the JOIN Operation …

- J3. **Sort-merge join:** If the records of R and S are *physically sorted* (ordered) by value of the join attributes A and B, respectively, we can implement the join in the most efficient way possible. Both files are scanned in order of the join attributes, matching the records that have the same values for A and B. In this method, the records of each file are scanned only once each for matching with the other file—unless both A and B are non-key attributes, in which case the method needs to be modified slightly.

- J4. **Hash-join:** The records of files R and S are both hashed to the *same hash file,* using the *same hashing function* on the join attributes A of R and B of S as hash keys. A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets. A single pass through the other file (S) then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from R.



… - Implementing the JOIN Operation …

- Factors affecting JOIN performance
  - Available buffer space
  - Join selection factor
  - Choice of inner VS outer relation

-- Partition hash join …

- **Partitioning phase**:
  - Each file (R and S) is first partitioned into M partitions using a partitioning hash function on the join attributes:

    R1 , R2 , R3 , …… Rm and S1 , S2 , S3 , …… Sm

  - A disk sub-file is created per partition to store the tuples for that partition.
- **Joining or probing phase:**
  - Involves M iterations, one per partitioned file. Iteration i involves joining partitions Ri and Si.

-- Partition hash join …

- Assume Ri is smaller than Si.

1. Copy records from Ri into memory buffers.

2. Read all blocks from Si, one at a time and each record from Si is used to *probe* for a matching record(s) from partition Si.

3. Write matching record from Ri after joining to the record from Si into the result file.

-- Partition hash join …

- Cost analysis of partition hash join:

1. Reading and writing each record from R and S during the partitioning phase: $(b_R + b_S)$, $(b_R + b_S)$
2. Reading each record during the joining phase: $(b_R + b_S)$
3. Writing the result of join: b_{RES}

Total Cost: $3^* (b_R + b_S) + b_{RES}$

- Algorithms for PROJECT Operation

- $\pi_{<\text{attribute list}>}(R)$

  1. If <attribute list> has a key of relation R, extract all tuples from R with only the values for the attributes in <attribute list>.

  2. If <attribute list> does NOT include a key of relation R, duplicated tuples must be removed from the results.

- Methods to remove duplicate tuples

  1. Sorting

  2. Hashing

- Algorithm for SET operations …

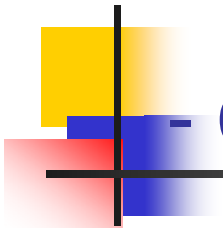
- **Set operations**: UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN PRODUCT.
- **CARTESIAN PRODUCT** of relations R and S include all possible combinations of records from R and S. The attribute of the result include all attributes of R and S.
- **Cost analysis** of CARTESIAN PRODUCT
  - If R has n records and j attributes and S has m records and k attributes, the result relation will have n*m records and j+k attributes.
- CARTESIAN PRODUCT operation is very **expensive** and should be avoided if possible.

… - Algorithm for SET operations …

- **UNION** (See Figure 15.3c)
  1. Sort the two relations on the same attributes.
  2. Scan and merge both sorted files concurrently, whenever the same tuple exists in both relations, only one is kept in the merged results.

- **INTERSECTION** (See Figure 15.3d)
  1. Sort the two relations on the same attributes.
  2. Scan and merge both sorted files concurrently, keep in the merged results only those tuples that appear in both relations.

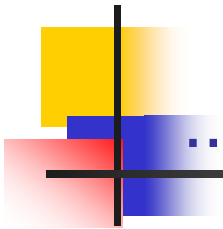
- **SET DIFFERENCE** R-S (See Figure 15.3e)
  - (keep in the merged results only those tuples that appear in relation R but not in relation S.)



Combining Operations using Pipelining …

- **Motivation**
  - A query is mapped into a sequence of operations.
  - Each execution of an operation produces a temporary result.
  - Generating and saving temporary files on disk is time consuming and expensive.

- **Alternative:**
  - Avoid constructing temporary results as much as possible.
  - Pipeline the data through multiple operations - pass the result of a previous operator to the next without waiting to complete the previous operation.



… - Combining Operations using Pipelining …

- **Example:** For a 2-way join, combine the 2 selections on the input and one projection on the output with the Join.
- Dynamic generation of code to allow for multiple operations to be pipelined.
- Results of a select operation are fed in a "**Pipeline**" to the join algorithm.
- Also known as **stream-based processing**.

- Using Heuristics in Query Optimization ...

- **Process for heuristics optimization**

  1. The parser of a high-level query generates an *initial internal representation*;
  2. Apply heuristics rules to optimize the internal representation.
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

- The **main heuristic** is to apply first the operations that reduce the size of intermediate results.

  E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

… - Using Heuristics in Query Optimization …

- **Query tree**: a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as *leaf nodes* of the tree, and represents the relational algebra operations as *internal nodes*.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

… - Using Heuristics in Query Optimization …

- **Example:**
  For every project located in 'Stafford', retrieve the project number, the controlling department number and the department manager's last name, address and birthdate.

  SQL query:

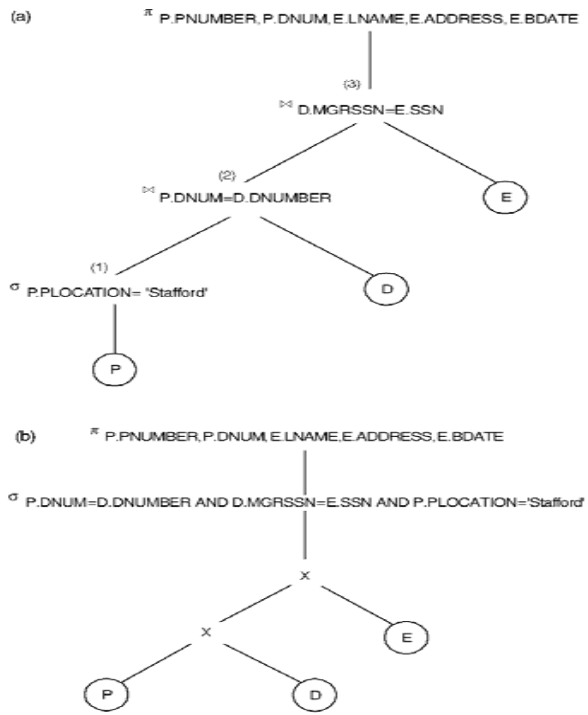
```
Q2: SELECT  P.NUMBER,P.DNUM,E.LNAME, E.ADDRESS, E.BDATE
    FROM    PROJECT AS P,DEPARTMENT AS D, EMPLOYEE AS E
    WHERE  P.DNUM=D.DNUMBER AND D.MGRSSN=E.SSN AND
           P.PLOCATION='STAFFORD';
```

  Relation algebra:

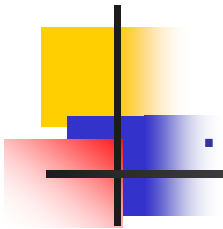
$$\pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(((\sigma_{\text{PLOCATION='STAFFORD'}}(\text{PROJECT})) \bowtie_{\text{DNUM=DNUMBER}} (\text{DEPARTMENT})) \bowtie_{\text{MGRSSN=SSN}} (\text{EMPLOYEE}))$$

… - Using Heuristics in Query Optimization …

Figure 18.4 Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2.



Note: The above figure is now called Figure 15.4 in Edition 4



… - Using Heuristics in Query Optimization …

Heuristic Optimization of Query Trees:

- The same query could correspond to many different relational algebra expressions — and hence many different query trees.

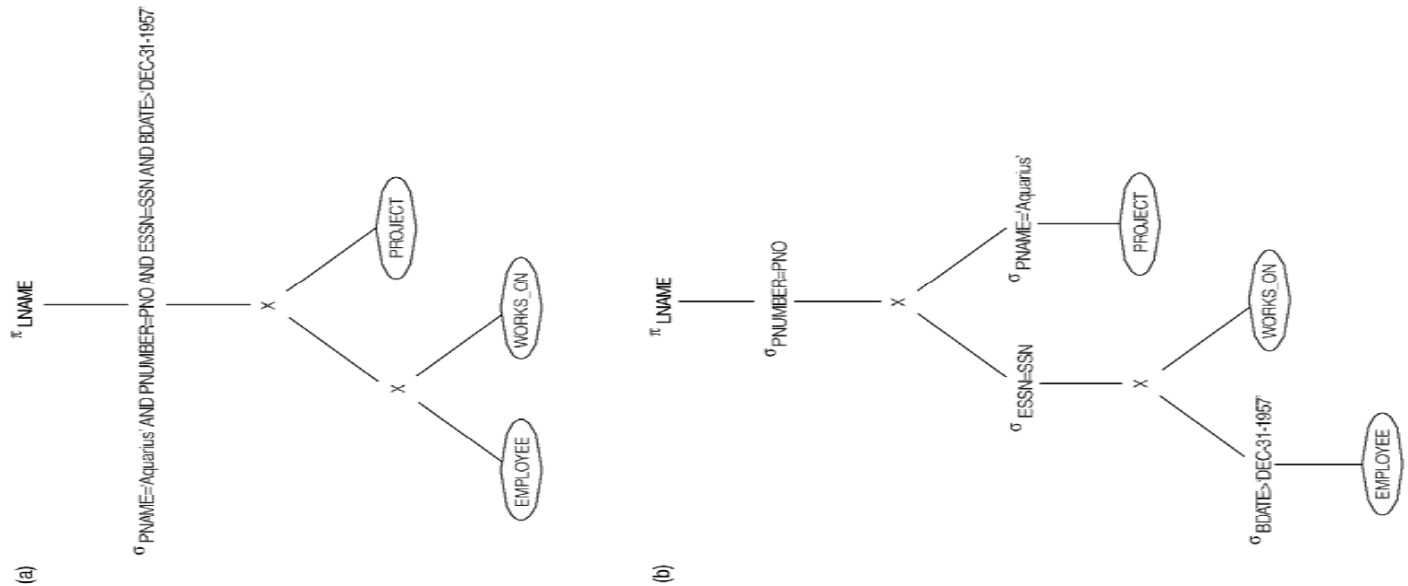
- The task of heuristic optimization of query trees is to find a **final query tree** that is efficient to execute.

- **Example**:

  Q: SELECT LNAME
  FROM EMPLOYEE, WORKS_ON, PROJECT
  WHERE PNAME = 'AQUARIUS'
  AND PNMUBER=PNO
  AND ESSN=SSN
  AND BDATE > '1957-12-31';

… - Using Heuristics in Query Optimization …

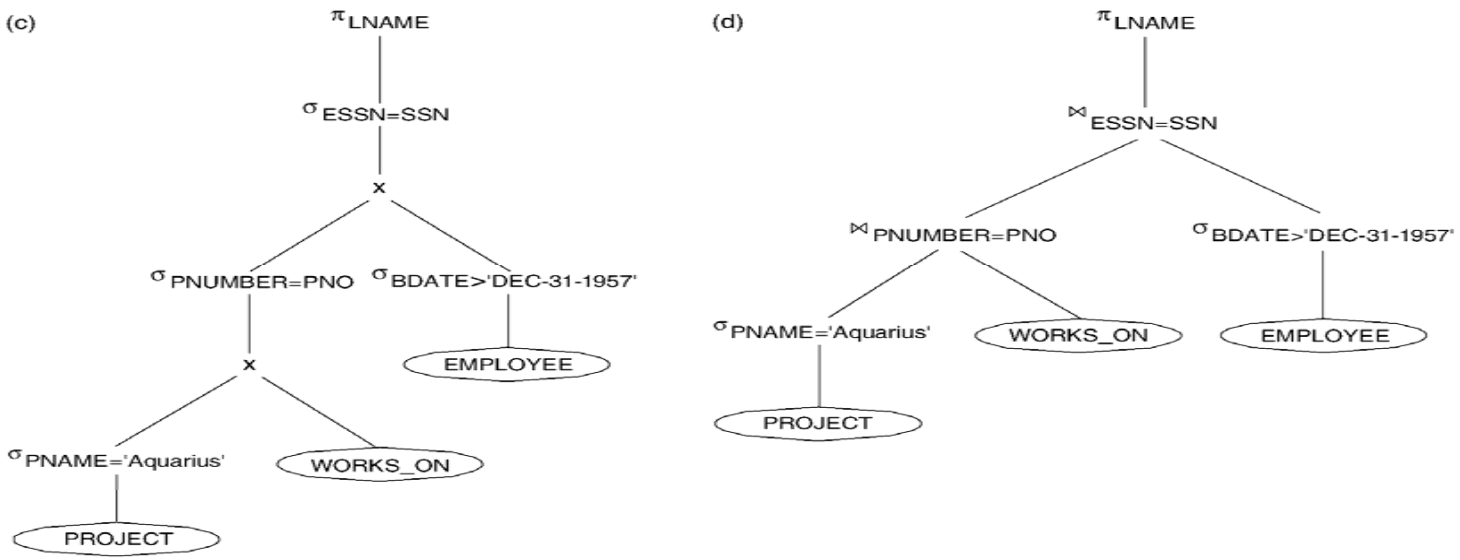
Figure 18.5 Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q. (b) Moving SELECT operations down the query tree.



Note: The above figure is now called Figure 15.5 in Edition 4

… - Using Heuristics in Query Optimization …

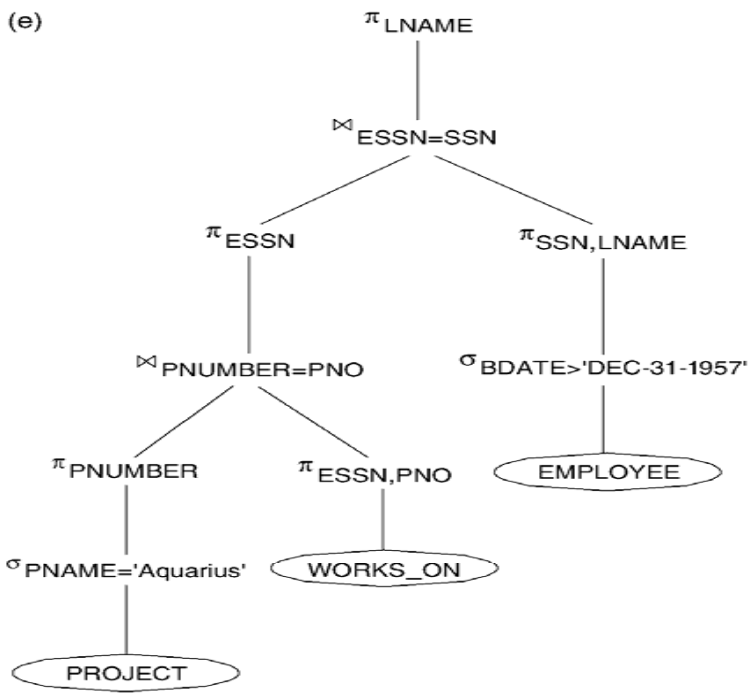
Figure 18.5 Steps in converting a query tree during heuristic optimization. (c) Applying the more restrictive SELECT operation first. (d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.



Note: The above figure is now called Figure 15.5(continued c, d) in Edition 4

… - Using Heuristics in Query Optimization …

Figure 18.5 Steps in converting a query tree during heuristic optimization. (e) Moving PROJECT operations down the query tree.



Note: The above figure is now called Figure 15.5(continued e) in Edition 4

-- General Transformation Rules for Relational Algebra Operations …

1. Cascade of s: A conjunctive selection condition can be broken up into a cascade (sequence) of individual s operations:
   $\sigma_{c1 \text{ AND } c2 \text{ AND } \ldots \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$

2. Commutativity of s: The s operation is commutative:
   $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$

3. Cascade of p: In a cascade (sequence) of p operations, all but the last one can be ignored:
   $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List1}(R)$

4. Commuting s with p: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
   $\pi_{A1, A2, \ldots, An}(\sigma_{c}(R)) = \sigma_{c}(\pi_{A1, A2, \ldots, An}(R))$

… -- General Transformation Rules for Relational Algebra Operations …

5. Commutativity of $\bowtie$ (and x): The $\bowtie$ operation is commutative as is the x operation: $R \bowtie_C S = S \bowtie_C R$; $R \times S = S \times R$

6. Commuting s with $\bowtie$ (or x): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows :

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

   Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:

   $\sigma_c (R \bowtie S) = (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$

… -- General Transformation Rules for Relational Algebra Operations …

7. Commuting p with (or x): Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

   $\pi_L (R \bowtie_C S) = (\pi_{A1, \ldots, An} (R)) \bowtie_C (\pi_{B1, \ldots, Bm} (S))$

   If the join condition c contains additional attributes not in L, these must be added to the projection list, and a final p operation is needed.

… -- General Transformation Rules for Relational Algebra Operations …

8. Commutativity of set operations: The set operations u and ? are commutative but – is not.

9. Associativity of ⋈, x, u, and ∩ : These four operations are individually associative; that is, if q stands for any one of these four operations (throughout the expression), we have:

   (R q S) q T = R q (S q T)

10. Commuting s with set operations: The s operation commutes with u , ∩ , and –. If q stands for any one of these three operations, we have

    $\sigma_c (R \; q \; S) = (\sigma_c (R)) \; q \; (\sigma_c (S))$

… -- General Transformation Rules for Relational Algebra Operations …

11. The p operation commutes with u.
    $p_L (R \cup S) = (p_L (R)) \cup (p_L (S))$

12. Converting a (s, x) sequence into ⋈: If the condition c of a s that follows a x Corresponds to a join condition, convert the (s, x) sequence into a ⋈ as follows:
    $(s_C (R \times S)) = (R \bowtie_C S)$

13. Other transformations

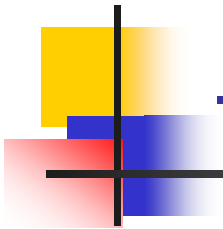
-- Outline of a Heuristic Algebraic Optimization Algorithm ...

1. Using rule 1, break up any select operations with conjunctive conditions into a cascade of select operations.
2. Using rules 2, 4, 6, and 10 concerning the commutativity of select with other operations, move each select operation as far down the query tree as is permitted by the attributes involved in the select condition.
3. Using rule 9 concerning associativity of binary operations, rearrange the leaf nodes of the tree so that the leaf node relations with the most restrictive select operations are executed first in the query tree representation.
4. Using Rule 12, combine a cartesian product operation with a subsequent select operation in the tree into a join operation.

… -- Outline of a Heuristic Algebraic Optimization Algorithm

5. Using rules 3, 4, 7, and 11 concerning the cascading of project and the commuting of project with other operations, break down and move lists of projection attributes down the tree as far as possible by creating new project operations as needed.

6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.

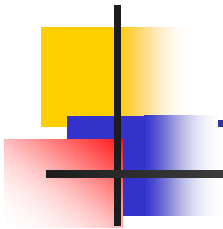


-- Summary of Heuristics for Algebraic Optimization

1. The main heuristic is to apply first the operations that reduce the size of intermediate results.

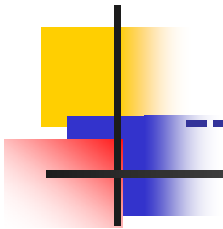
2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible.)

3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)



- Cost Based Optimization (CBO)

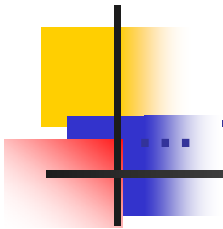
- **Cost-based query optimization**: Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimates
- In heuristic query optimization parameters such as the size of the operand tables is not considered.
- Issues in cost based optimization
  - Cost function
  - Number of execution strategies to be considered



-- Using Selectivity and Cost Estimates in CBO ...

- Cost Components for Query Execution
  1. Access cost to secondary storage
  2. Storage cost
  3. Computation cost
  4. Memory usage cost
  5. Communication cost

Note: Different database systems may focus on different cost components.



... -- Using Selectivity and Cost Estimates in CBO

- Catalog Information Used in Cost Functions
  - Information about the size of a file
    - number of records (tuples) (r),
    - record size (R),
    - number of blocks (b)
    - blocking factor (bfr)
  - Information about indexes and indexing attributes of a file
    - Number of levels (x) of each multilevel index
    - Number of first-level index blocks (b_{I1})
    - Number of distinct values (d) of an attribute
    - Selectivity (sl) of an attribute
    - Selection cardinality (s) of an attribute. (s = sl * r)

--- Examples of Cost Functions for SELECT ...

- **S1. Linear search (brute force) approach**

  C_{S1a} = b;

  For an equality condition on a key, C_{S1a} = (b/2) if the record is found; otherwise C_{S1a} = b.

- **S2. Binary search**:

  $C_{S2} = \log_2 b + \lceil (s/bfr) \rceil - 1$

  For an equality condition on a unique (key) attribute,

  $C_{S2} = \log_2 b$

- **S3. Using a primary index (S3a) or hash key (S3b) to retrieve a single record**

  C_{S3a} = x + 1; C_{S3b} = 1 for static or linear hashing;

  C_{S3b} = 1 for extendible hashing;

… --- Examples of Cost Functions for SELECT …

- **S4. Using an ordering index to retrieve multiple records:**

  For the comparison condition on a key field with an ordering index, $C_{S4} = x + (b/2)$

- **S5. Using a clustering index to retrieve multiple records**:

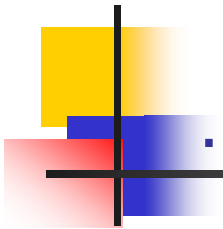
  $C_{S5} = x + \lceil (s/bfr) \rceil$

- **S6. Using a secondary (B^+-tree) index:**

  For an equality comparison, $C_{S6a} = x + s$;

  For an comparison condition such as >, <, >=, or <=,

  $C_{S6a} = x + (b_{I1}/2) + (r/2)$



… --- Examples of Cost Functions for SELECT

- **S7. Conjunctive selection:**

  Use either S1 or one of the methods S2 to S6 to solve.

  For the latter case, use one condition to retrieve the records and then check in the memory buffer whether each retrieved record satisfies the remaining conditions in the conjunction.

- **S8. Conjunctive selection using a composite index**:

  Same as S3a, S5 or S6a, depending on the type of index.

End