Chapter One

Query Processing and Optimization

(Reading in the Textbook: Chapters 18 & 19)

Outline

- Basics of Relational Data Model
- Translating SQL Queries into Relational Algebra
- Basic Algorithms for Executing Query Operations
- Using Heuristic in Query Operations
- Using Selectivity and Cost Estimates in Query Optimization
- Semantic Query Optimization

Relational Data Model

- Data models are of three types:
  - High-level data model/ conceptual data model
  - Medium-level data model/Implementation-level data model/ logical data model
  - low-level data model/physical data model
- Relational data model is one example of medium-level data models
  - A data model that is based on relations (also known as tables)
    - A row is called a tuple; a column header is called an attribute; the table is called a relation; the data types describing the types of values that can appear in each column is called a domain
  - Uses the concept of a mathematical relation (looks somewhat like a table of values)as its basic building block
  - Has its theoretical basis in set theory and first order predicate logic

Relational Algebra

- Relational Algebra refers the basic set of operations for the formal relational model
- A sequence of relational algebra operations forms **relational algebra expression**
- In the slides:
  - σ and S are used interchangeably for SELECT
  - π and P are used interchangeably for PROJECT
  - ρ and q are used interchangeably for RENAME
  - ∪ and D are used interchangeably for UNION
  - ∩ and C are used interchangeably for INTERSEECTION
  - …

Basic Relational Algebra Operations

The SELECT operation

$S_{<\text{selection condition}>}$ (R)

- S (sigma) is used to denote the SELECT operator
- The selection condition is a Boolean expression specified on the attributes of relation R
- R is a relational algebra expression whose result is a relation; the simplest expression is just the name of a database relation
- The relation resulting from the SELECT operation has the same attributes as R
- The number of tuples in the resulting relation is always less than or equal to the number of tuples in R
- Select operation is commutative
- We can always combine a cascade of SELECT operations into a single SELECT operation with a conjunctive (AND) condition

Basic Relational Algebra Operations…

The PROJECT operation

$P_{<attribute\ list>}\ (R)$

- P (pi) is used to denote the PROJECT operator
- <attribute list> is a list of attributes from the attributes of relation R
- The degree of the result of PROJECT operation is equal to the number of attributes in <attribute list>
- The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R
- If the projection list is a super key of R, i.e., it includes some key of R, the resulting relation has the same number of tuples as R

Basic Relational Algebra Operations…

The RENAME operation

$\rho_{S(B1, B2, .., Bn)}(R)$ or $\rho_{S}(R)$ or $\rho_{(B1, B2, .., Bn)}(R)$

- q (rhoi) is used to denote the RENAME operator
- S is the new relation name
- B1, B2,..,Bn are the new attribute names

Basic Relational Algebra Operations…

Set Theoric Operations

- Are used to merge the elements of two sets in various ways including UNION, INTERSECTION, and SET DIFFERENCE
- Are binary operations: each is applied to two sets
- The relations must be union compatible: have the same degree (the same umber of attributes)and $dom(A_i) = dom(B_i)$ for each attributes (each pair of corresponding attributes have the same domain)

Basic Relational Algebra Operations…

UNION

- R **D** S = S **D** R
- R **D** (S **D** T) = (R **D** S) **D** T
- A relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated

Basic Relational Algebra Operations…

INTERSECTION

- R **C** S = S **C** R
- (R **C** S) **C** T = R **C** (S **C** T)
- A relation that includes all tuples that are in both R and S

Basic Relational Algebra Operations…

SET DIFFERENCE

- R - S # S - R
- A relation that includes all tuples that are in R but not in S

Basic Relational Algebra Operations…

JOIN

- A binary set operation
- the relations do not have to be union compatible
- THETA JOIN
  - $R_{<\text{join condition}>}$ S
  - A general join condition is of the form: <condition> AND <condition> AND ….AND<condition>
  - Where each condition is of the form A_i **h** B_j, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and **h**(theta) is a comparison operator.
  - A JOIN operation on two relations $R(A_1,A_2,…,A_n)$ and $S(B_1,B_2,…,B_m)$

Basic Relational Algebra Operations…

JOIN

- EQUIJOIN
  - A JOIN that involves join conditions with equality comparisons only
  - we always have one or more pairs of attributes that have *identical values* in every tuple
- NATURAL JOIN (*)
  - Requires that the two join attributes have the sane name in both relations.
  - Renaming is necessary if the join attributes are not identical
  - Only tuples from R that have matching tuples in S –and vice versa- appear in the result
  - Tuples without a matching tuple are eliminated from the JOIN result
  - Tuples with null in the join attributes are also eliminated

Basic Relational Algebra Operations…

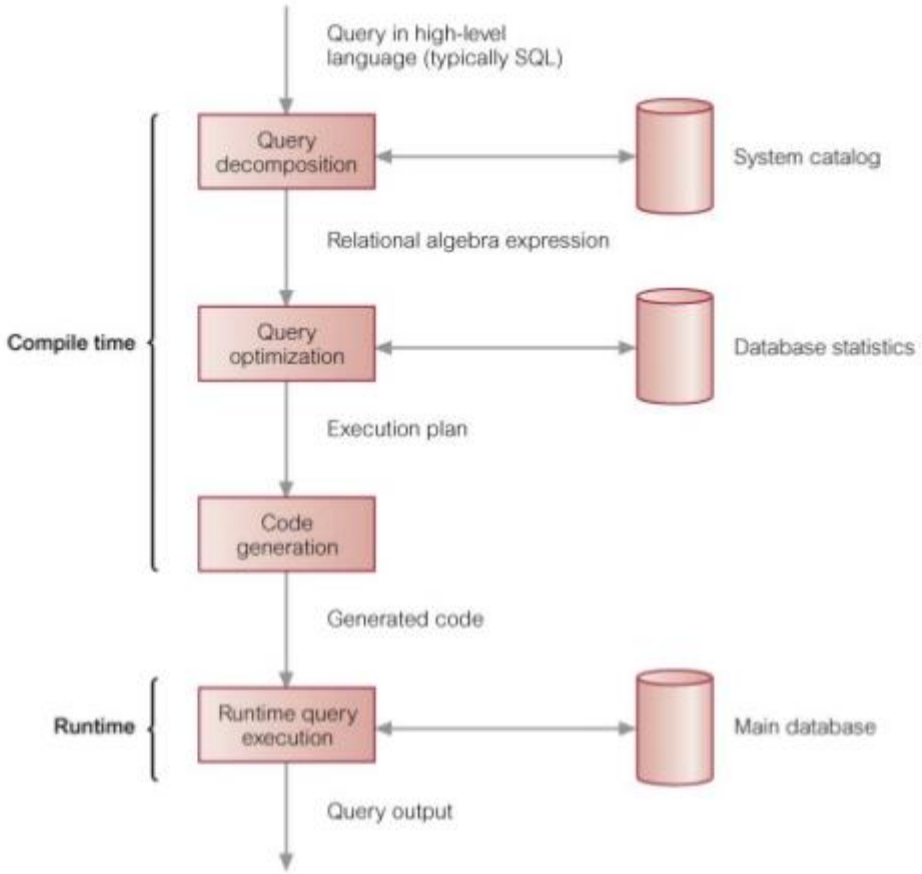
OUTER JOIN

- LEFT OUTER JOIN
  - Keeps every tuple in the left relation R in R S; if no matching tuple is found in S, then the attributes of S in the join result are filled with null values.
- RIGHT OUTER JOIN
  - Keeps every tuple in the right relation S in R S; if no matching tuple is found in R, then the attributes of R in the join result are filled with null values
- FULL OUTER JOIN
  - Keeps all tuples in both the left and the right relations when no matching tuples are found, filling them with null values as needed

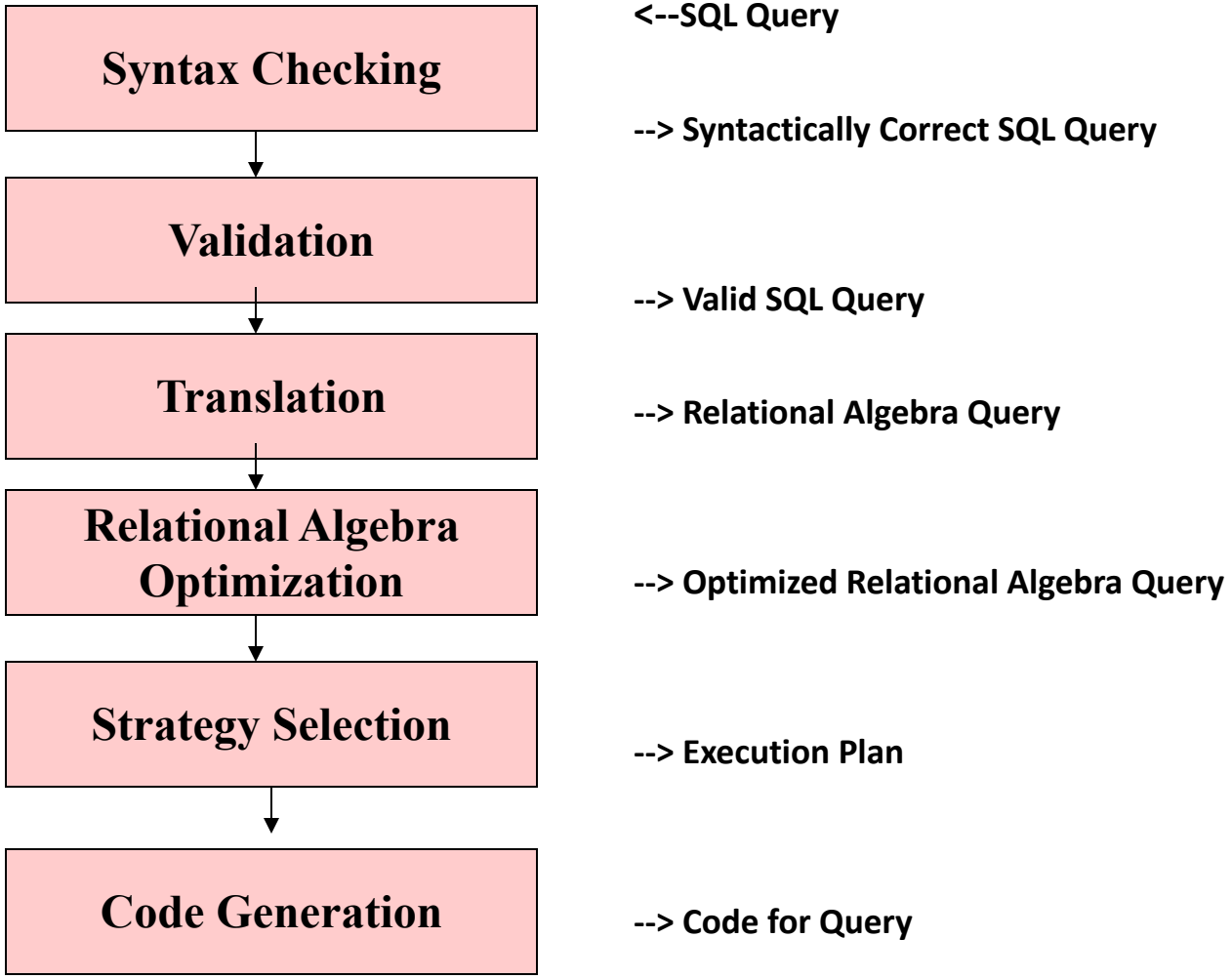
Query optimization

- A query typically has many possible execution strategies, and the process of choosing a suitable one for the processing a query is known as query optimization
- Query optimization is an activity conducted by a query optimizer in a DBMS to select the best available strategy for executing the query
- High-level query languages like SQL (for RDBMS) are more declarative in nature because they specifies what the intended results of the query are, rather than identifying the details of how the result should be obtained.
- SQL queries are translated into corresponding relational algebra for the optimization

Phases of Query Processing



Query Optimization…



Translating SQL Queries into Relational Algebra

- An SQL query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clauses if these are part of the block
- Nested queries within a query are identified as separate query blocks
- Inner blocks could be
  - uncorrelated nested query- the inner block needs to be evaluated only once
  - correlated nested query-a tuple variable from the outer block appears in the WHERE-clause of the inner block
- SQL queries are decomposed into query blocks, which form the basic units that can be translated into the algebraic operators and optimized
- The query optimizer would then choose an execution plan for each block
- SQL query is translated to an equivalent extended relational algebra expression , represented as a query tree data structure, that is then optimized

Translating SQL Queries into Relational Algebra

- Example:
  SELECT ALL
  FROM Employee
  WHERE salary > 3000
- **$S_{\text{salary ="3000"}}$ Employee**
- SELECT name,age
  FROM Employee
  WHERE salary >
  (SELECT MAX(salary))
  FROM Employee
  WHERE department="IT"
- **C= MAX salary ($S_{\text{department="IT"}}$ Employee)**
- **$P_{\text{name,age}}$($S_{\text{salary>C}}$ Employee)**

Represent Relational Algebra by Query tree

- A **query tree** is a tree data structure that corresponds to an extended relational algebra expression. It represents the input relations of the query as *leaf nodes* of the tree, and it 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.

Represent Relational Algebra by Query tree…

- The order of execution of operations *starts at the leaf nodes*, which represents the input database relations for the query, and *ends at the root node*, which represents the final operation of the query. The execution terminates when the root node operation is executed and produces the result relation for the query

Heuristic-Based Query Optimization: Example

- Query

"Find the last names of employees born after 1957 who work on a project named ‘Aquarius’."

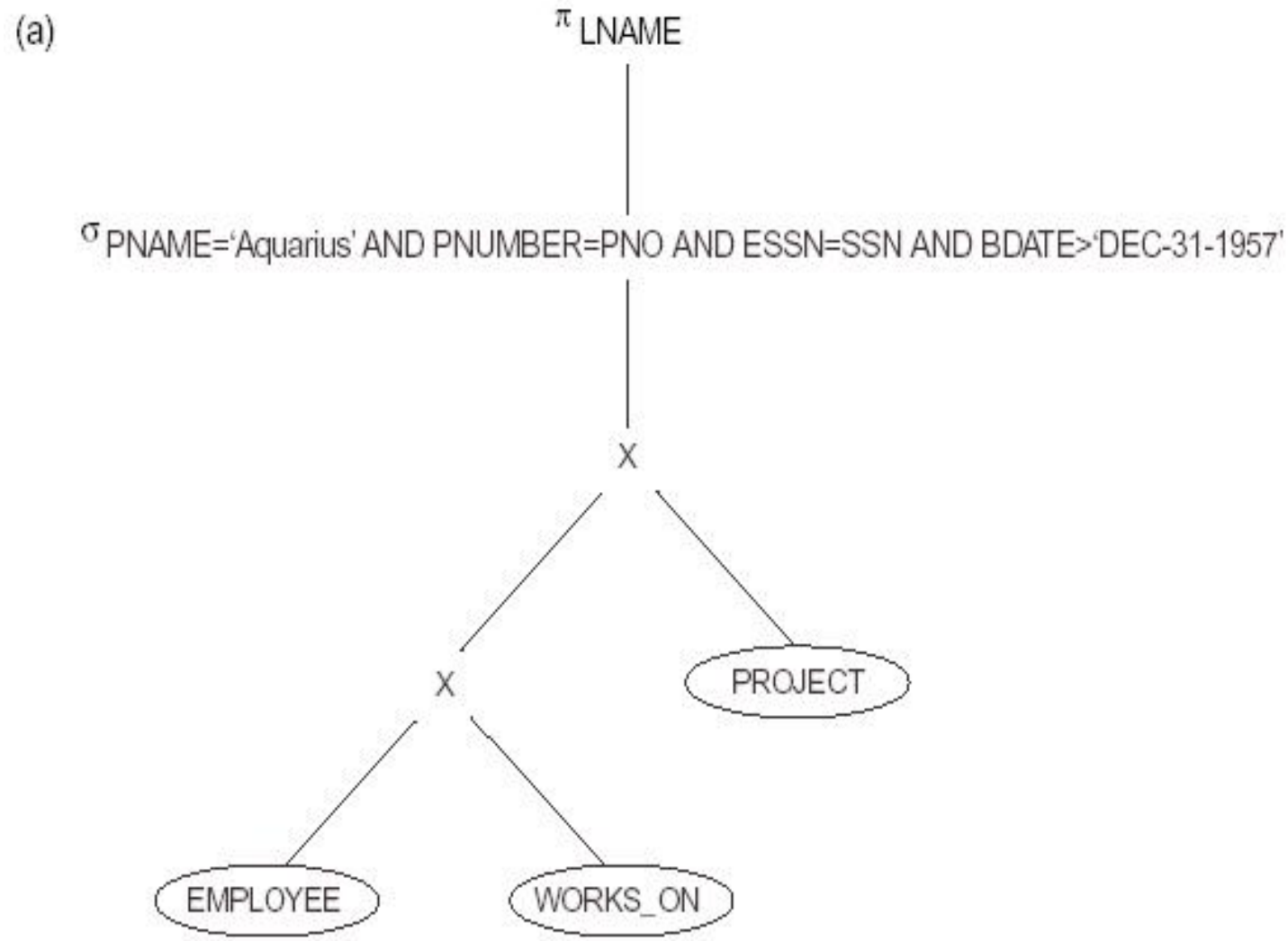
- SQL

SELECT LNAME

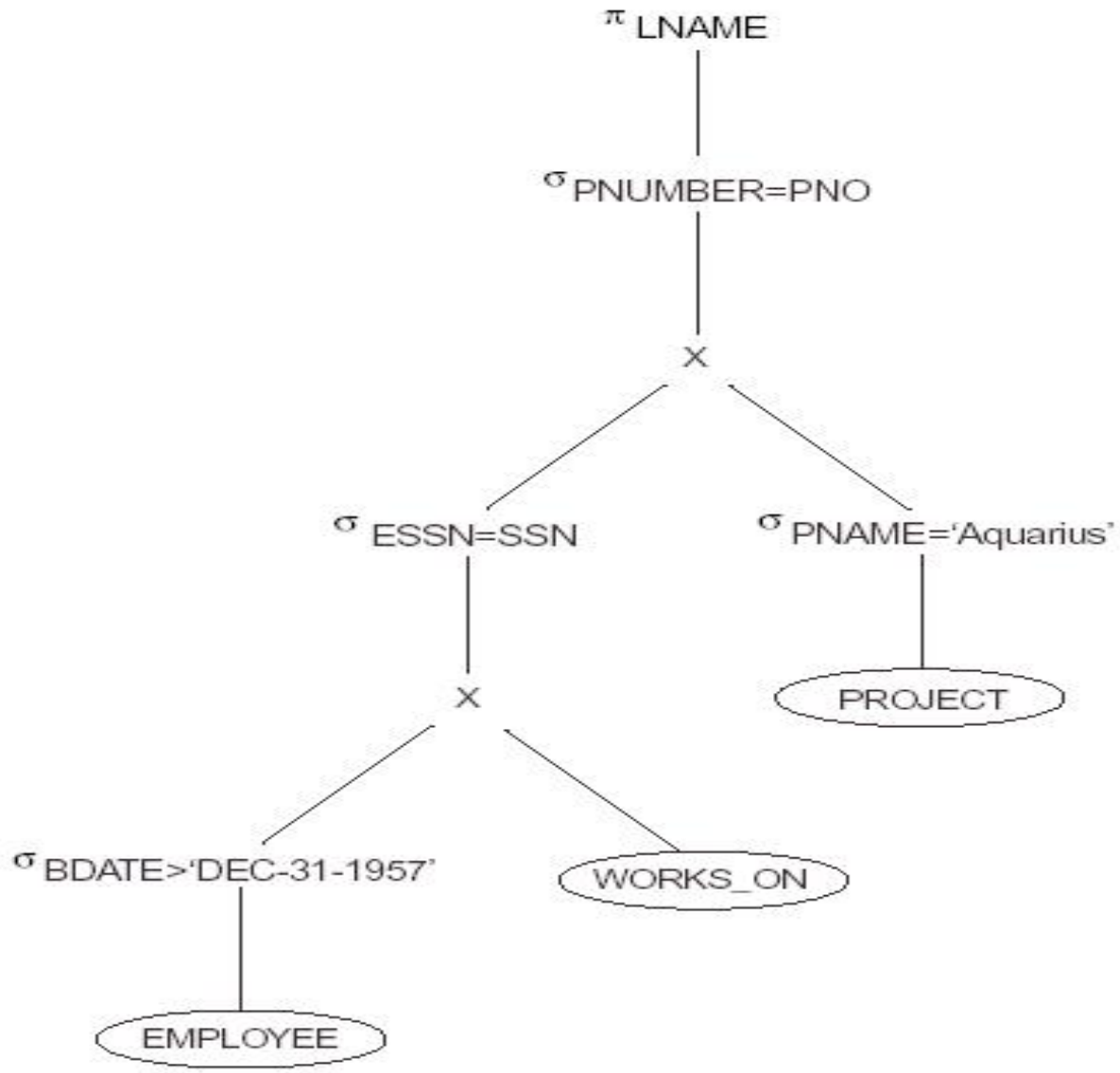
FROM EMPLOYEE, WORKS_ON, PROJECT

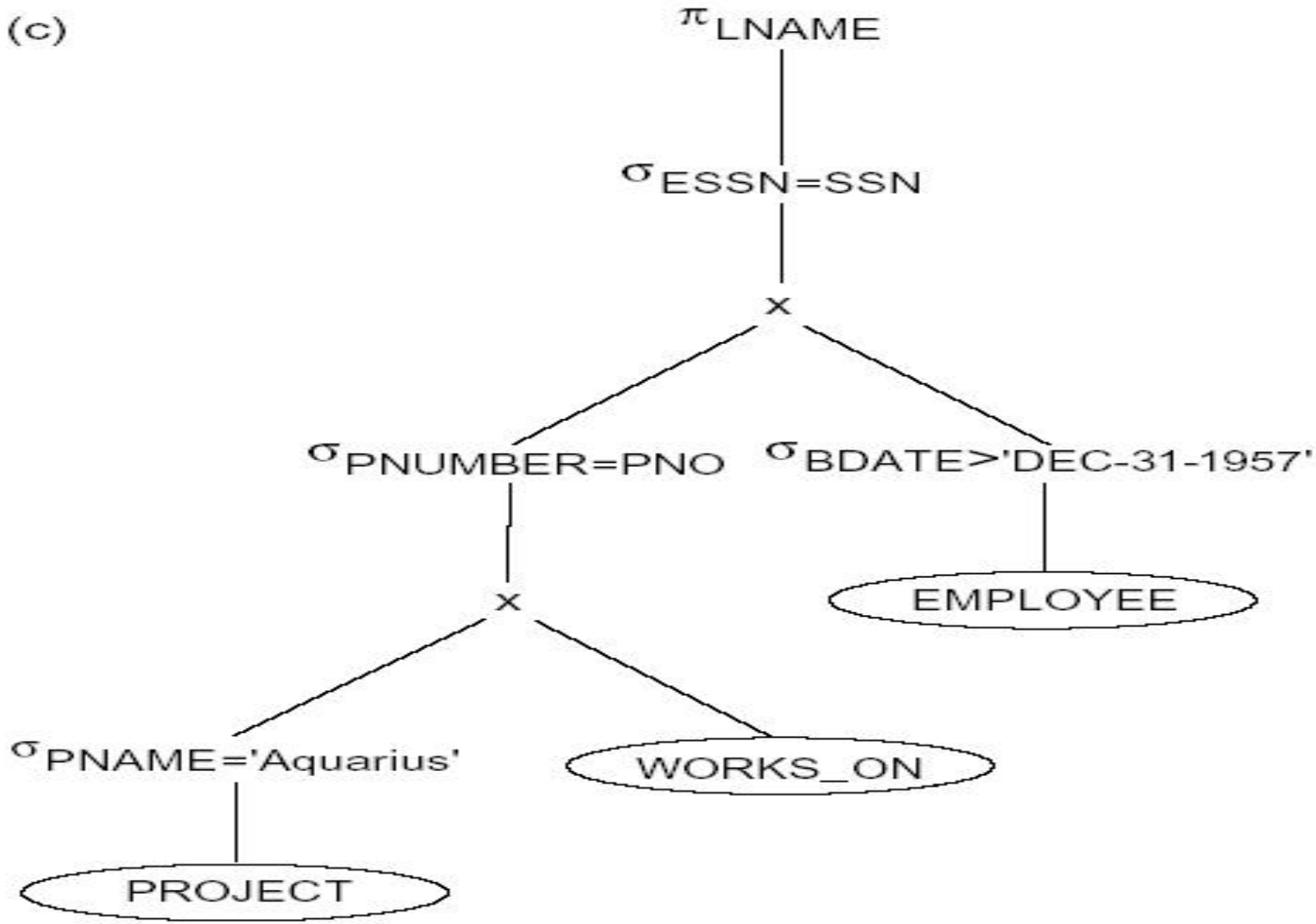
WHERE PNAME=‘Aquarius’ **AND** PNUMBER=PNO **AND** ESSN=SSN **AND** BDATE.‘1957-12-31’;

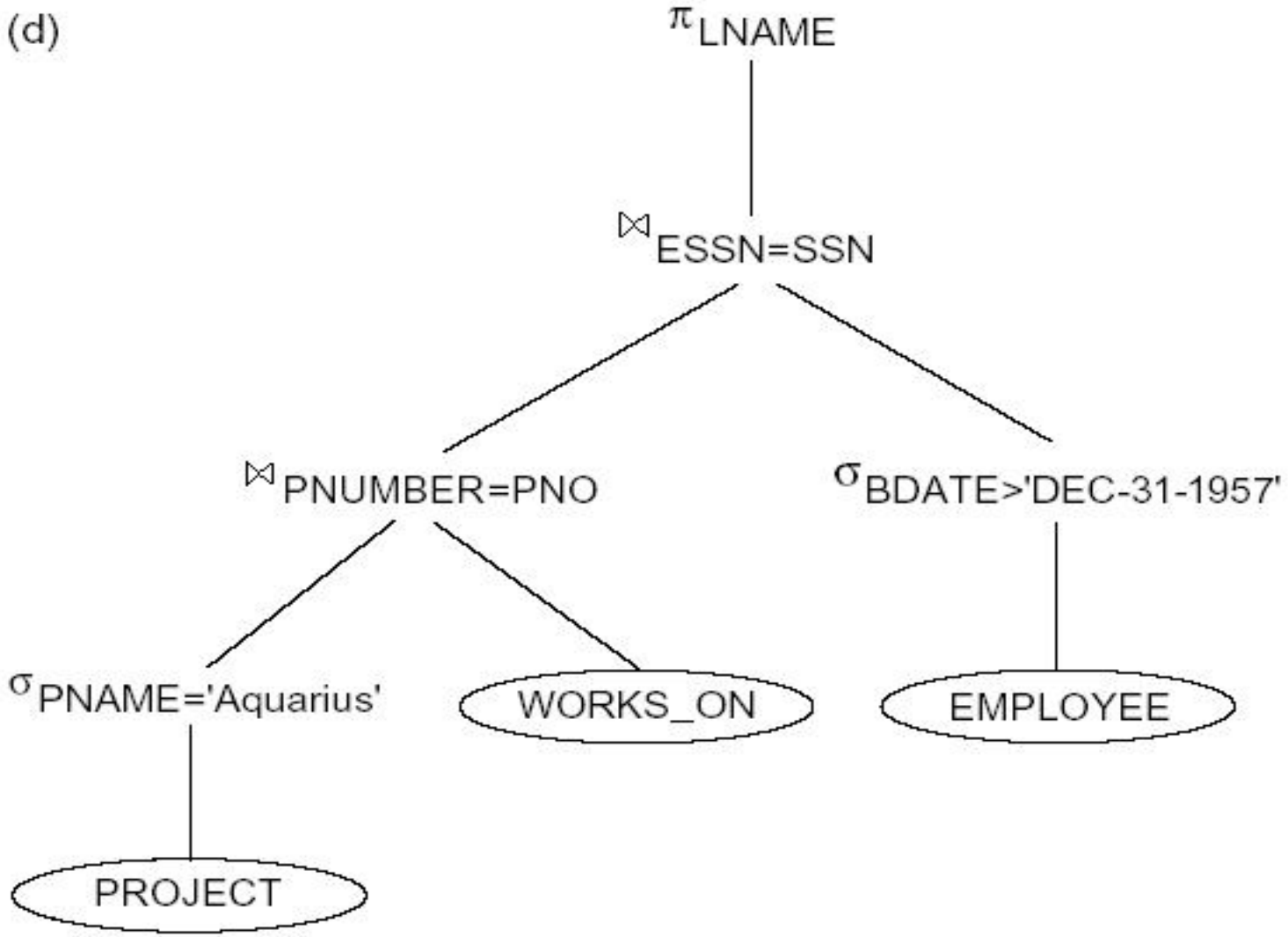
There are five different alternative query trees for this SQL in the next slides.

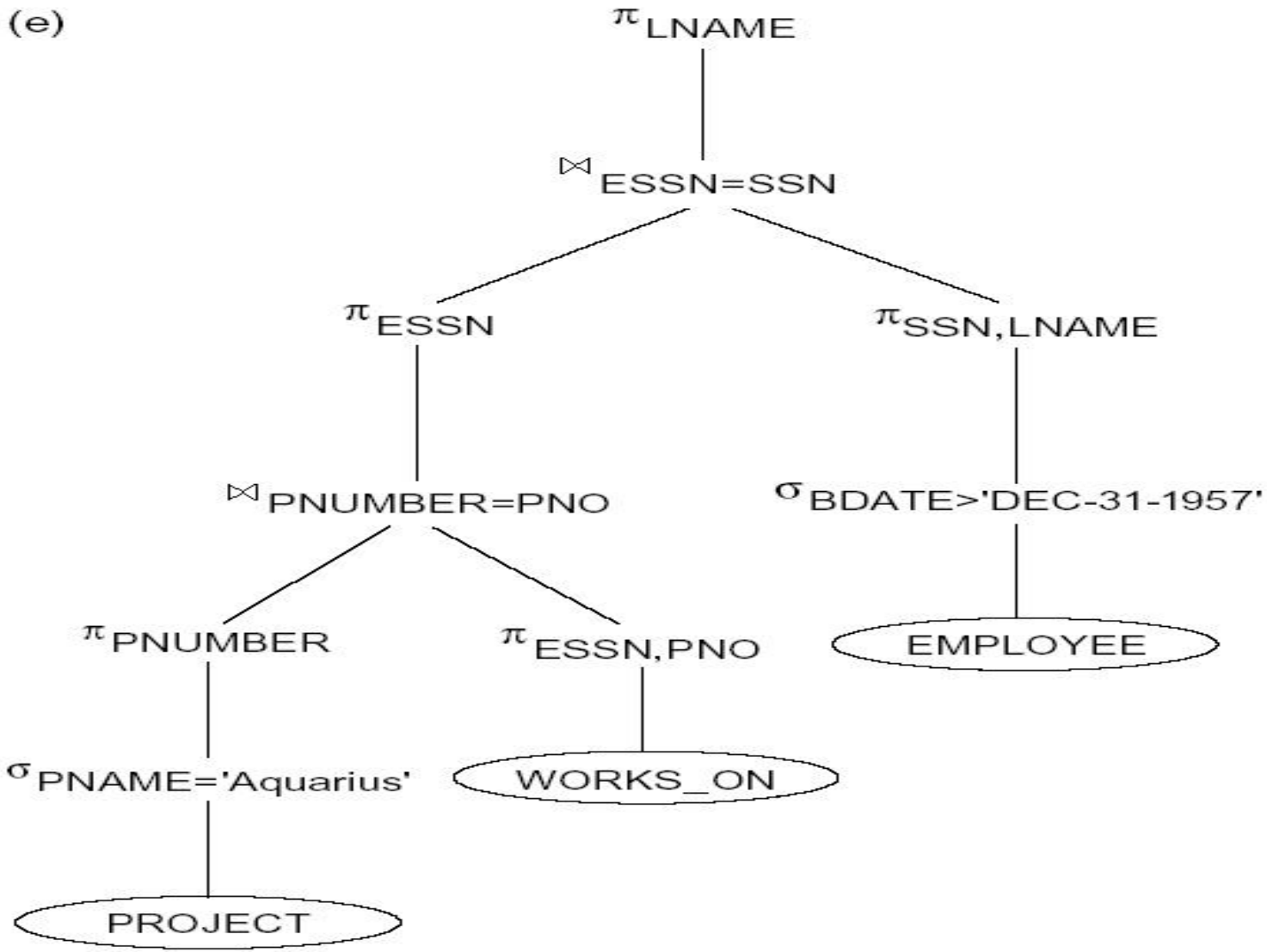

(a)
π LNAME
σ PNAME='Aquarius' AND PNUMBER=PNO AND ESSN=SSN AND BDATE>'DEC-31-1957'
X
X
PROJECT
EMPLOYEE
WORKS_ON

(b)


π LNAME
σ PNUMBER=PNO
X
σ ESSN=SSN
σ PNAME='Aquarius'
X
PROJECT
σ BDATE>'DEC-31-1957'
WORKS_ON
EMPLOYEE


(c)
πLNAME
σESSN=SSN
x
σPNUMBER=PNO
σBDATE>'DEC-31-1957'
x
EMPLOYEE
σPNAME='Aquarius'
WORKS_ON
PROJECT


(d)
πLNAME
⋈ESSN=SSN
⋈PNUMBER=PNO
σBDATE>'DEC-31-1957'
σPNAME='Aquarius'
WORKS_ON
EMPLOYEE
PROJECT


(e)
πLNAME
⋈ESSN=SSN
πESSN
πSSN,LNAME
⋈PNUMBER=PNO
σBDATE>'DEC-31-1957'
πPNUMBER
πESSN,PNO
EMPLOYEE
σPNAME='Aquarius'
WORKS_ON
PROJECT

Basic Algorithms for Executing Query Operations/Query processing

- It is all about algorithms that implements the relational algebra operations wed discussed above
- These are
  - External sorting
  - Select
  - Join
  - Project
  - Set operations

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
- The typical external sorting algorithm uses a sort-merge strategy, which starts by sorting small subfiles-called runs—of the main file and then merges the stored runs, creating larger sorted subfiles that are merged in turn
- The basic sort-merge algorithm consists of two phases
  - Sorting phase and
  - Merging phase

External Sorting…

- 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 a run and **number of initial runs** is dictated by the **number of file blocks (*b*)** and the **available buffer space .** For example, if = 5 blocks and the size of the file *b* = 1024 blocks, then = , or 205 initial runs each of size 5 blocks (except the last run which will have 4 blocks). Hence, after the sort phase, 205 sorted runs are stored as temporary subfiles on disk

External Sorting…

- In the **merging phase,** the sorted runs are merged during one or more **passes.**
- The **degree of merging** is the number of runs that can be merged together in each pass.
- In each pass, one buffer block is needed to hold one block from each of the runs being merged, and one block is needed for containing one block of the merge result

Implementing SELECT (read Ch 18 p.694 of the text book)

- Search methods for selection
  - Linear search
  - Binary search
  - Using a primary index
  - Using a primary index to retrieve multiple records
  - Using a clustering index to retrieve multiple records
  - Using a secondary index on an equality comparison
  - Conjunctive selection using an individual index
  - Conjunctive selection using a composite index
  - Conjunctive selection by intersection of record pointers

Implementing SELECT…

- When the optimizer is choosing between multiple simple conditions in a conjunctive select condition, it typically considers the selectivity of each condition
- **Selectivity (S)**: the ration of the number of records that satisfy the condition to the total number of records in the relation
- For example, for an equality condition on a key attribute of relation r(R), s= 1/r(R)
- For an equality condition on an attribute with I distinct values, s is estimated by (|r(R)|/i) /r(R) or 1/i
- The number of records satisfying a selection condition with selectivity s is estimated to be| r(R)| * s. The smaller this estimate is, the higher the desirability of using that condition first to retrieve records

Implementing the JOIN operation

- Our focus is n EQUIJOIN and NATURAL JOIN
- We base on the join operation $R_{A=B} S$ for the discussion of the algorithms of join
  - Where R and S are the relations to be joined and A and B are domain-compatible attributes of R and S respectively
- Methods for Implementing Joins
  - Nested-loop join
  - Single-loop join
  - Sort-merge join
  - Hash join

Implementing the JOIN operation

- **Nested-loop Join:**
  - 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]
- **Single-loop join**(using an access structure to retrieve the matching records)
  - If an index exists for one of the two join attributes –say, B of S- retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A]
- **Sort-merge join**
  - The records of R and S need to be sorted by value of the join attributes(or external sorting needs to be applied) before applying this algorithm
  - (assuming A or B or both are key attributes)
    - Pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file

Implementing the JOIN operation…

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
- This algorithms assumes that the smaller of the two files fits entirely into memory buckets after the first phase
- First, a single pass through the file with fewer records (say, R) hashes its records to the hash file buckets; this is called the partitioning phase since the records of R are partitioned into the hash buckets
- In the second phase, called the probing phase, a single pass through the other file (S) then hashes each of its records to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket

Implementing PROJECT

- Let us consider the A PROJECT operation $P_{<attribute\ list>}(R)$
- If <attribute list> includes a key of relation R, implementation of the above PROJECT is straightforward
  - The result of the operation will have the same number of tuples as R, but with only the values for the attributes in <attribute list> in each tuple
- If <attribute list> does not include a key of R, duplicate tuples must be eliminated
  - Duplicates can be eliminated either by sorting the result and eliminate duplicate tuples that appear consecutively after sorting or using hashing
- Recall that in SQL queries, the default is not to eliminate duplicates and we are supposed to use the keyword DISTINCT to eliminate duplicate records

Implementing set operations

- Set operations include UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN PRODUCT
- These operations are sometimes expensive to implement. The CARTESIAN PRODUCT R X S (R with n records and j attributes; S with m records and k attributes), for example, results in n * m records and j + k attributes. Hence it is important to avoid this operation and to substitute other equivalent operations during query optimization
- The three set operation (UNION, INTERSECTION and SET DIFFERENCE) apply only to union-compatible relations
  - Relations that have the same number of attributes and the same attribute domains
  - Can be implemented by using a variation of sort-merge techniques or by using hashing

Implementation of UNION: R **D** S

- Using sort-merge technique
  - Sort the two relations on the same attributes
  - Scan and merge both sorted files concurrently
    - Whenever the same tuple exits in both relations, keep only one of the tuple
- Using hashing
  - First hash (partition) the records of R
  - Then, hash (probe) the records of S, but do not insert duplicate records in the buckers

Implementation of INTERSECTION: R ∩ S

- Using sort-merge technique
  - Sort the two relations on the same attributes
  - Scan both sorted files concurrently so that
    - Whenever the same tuple exits in both relations, keep the tuple
- Using hashing
  - First hash (partition) the records of R to the hash table
  - Then, while hashing each record of S, probe to check if an identical record from R is found in the bucket, and if so add the record to the result file

Implementation of SET DIFFERENCE: R - S

- Using sort-merge technique
  - Sort the two relations on the same attributes
  - Scan both sorted files concurrently and
    - Add the tuples that are available only in R
- Using hashing
  - First hash (partition) the records of R to the hash table
  - Then, while hashing each record of S, probe to check if an identical record from R is found in the bucket, and if so remove that record from the bucket

Implementing Aggregate Operations…

- COUNT, AVERAGE, SUM
  - Dense index (if there is an index entry for every record in the main file)
    - Index seach can be used
    - The associated computation would be applied to the values in the index
  - Non-dense index
    - The actual number of records associated with each index entry must be used for a correct computation (except for COUNT DISTINCT, where the number of distinct values can be counted from the index itself)
- When a GROUP BY clause is used in a query, the aggregate operator must be applied separately to each group of tuples
  - The table must first be partitioned into subsets of tuples, where each partition (group) has the same value for the grouping attributes

Implementing Aggregate Operations

- The aggregate operators are MIN, MAX, COUNT, AVERAGE and SUM
- Either full table scan or index search could be used
- MAX
  - SELECT MAX(Salary)
    FROM Employee;
- If an (ascending) index on Salary exists for the Employee relation, then the optimizer can decide on using the index to search for the largest value by following the rightmost pointer in each index node from the root to the rightmost leaf
- The MIN aggregate can be handled in a similar manner, except that the leftmost pointer is followed from the root to the leftmost leaf

Implementing Outer Join

- Consider the following SQL query that is based on LEFT OUTER JOIN.
- *SELECT name, departmentName*
  FROM (Employee LEFT OUTER JOIN DEPARTMENT ON Dno = Dnumber);
- OUTER JOIN can be implemented either by
  - Modifying the join algorithms such as nested loop or single loop join
  - For example, to compute a *left* outer join, we use the left relation as the outer loop or single loop because every tuple in the left relation must appear in the result. If there are matching tuples in the other relation, the joined tuples are produced and saved in the result. However, if no matching tuple is found, the tuple is still included in the result but is padded with null value(s).
  - The sort-merge and hash join algorithms can also be extended to compute outer joins

Implementing Outer Join…

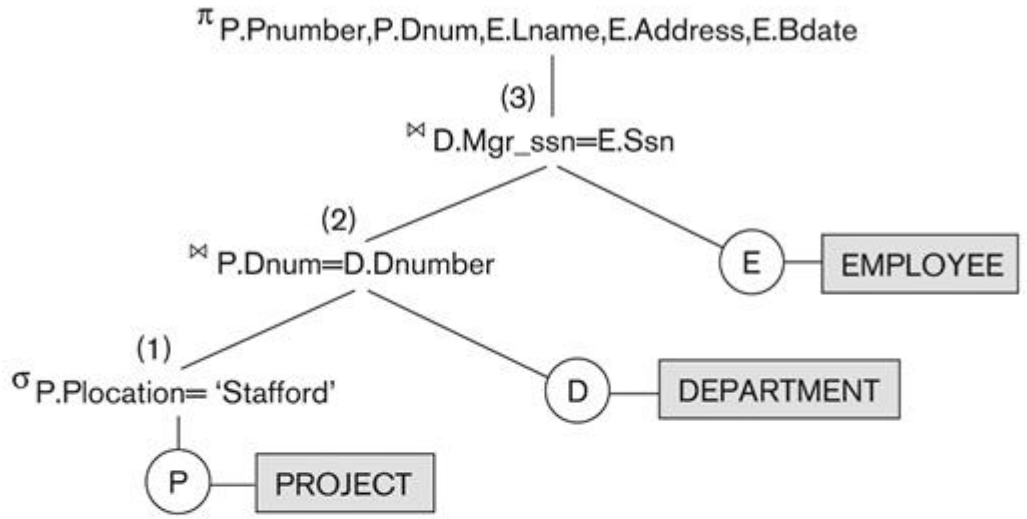
- Consider the following SQL query that is based on LEFT OUTER JOIN.
- *SELECT name, DName*
  FROM (Employee LEFT OUTER JOIN DEPARTMENT ON Dno = Dnumber);
- OUTER JOIN can be implemented by
  - by executing a combination of relational algebra operators. For example, the left outer join operation shown above is equivalent to the following sequence of relational operations:
    - 1. Compute the (inner) JOIN of the Employee and Department tables
    - 2. Find the Employee tuples that do not appear in the inner JOIN result
    - 3. Pad each tuple in the #2 with a null Dname field
    - 4. Apply the UNION operation to #1 and #2 to produce the LEFT OUTER JOIN result

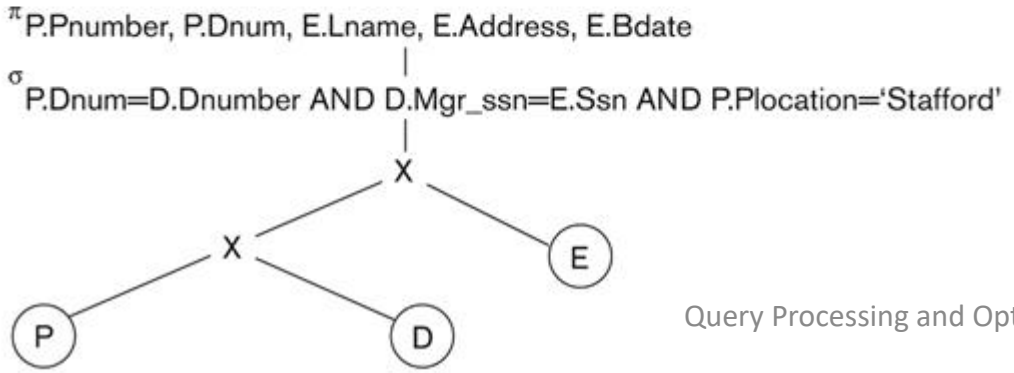
Using Heuristics in Query Optimization

- The steps that a high-level SQL query passes to give us a result are:
  - A high-level SQL is parsed and validated by a query parser and then, converted to an initial internal representation form of the SQL in query tree form (without any optimization)
  - The heuristic query optimizer applies different heuristic rules to the initial internal representation and optimize the query (more efficient to execute but gives the same result as the original one) and generate an other equivalent internal representation of the query, final query tree
  - The optimized internal representation form of the query is used to generate query execution plan

SELECT P.PNUMBER, P.DNUM, E.LNAME, E.ADDRESS, E.BDATE
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE P.DNUM = D **AND** D.MSGR = E.SSN **AND** P.PLOCATION = 'Stafford';

$$\mathbf{P}_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(((\mathbf{S}_{\text{PLOCATION='Stafford''}}(\text{PROJECT}))_{\text{DNUM=DNUMBER}}(\text{DEPARTMENT}))_{\text{MGRSSN=SSN}}(\text{EMPLOYEE}))$$





Example SQL query and equivalent Query tree

General transformation rules for Relational algebra operations

1. **Cascade of σ.** A conjunctive selection condition can be broken up into a cascade (that is, a sequence) of individual σ operations:

   $$\sigma_{c_1 \text{ AND } c_2 \text{ AND } \ldots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$$

2. **Commutativity of σ.** The σ operation is commutative:

   $$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

3. **Cascade of π.** In a cascade (sequence) of π operations, all but the last one can be ignored:

   $$\pi_{\text{List}_1}(\pi_{\text{List}_2}(\ldots(\pi_{\text{List}_n}(R))\ldots)) \equiv \pi_{\text{List}_1}(R)$$

General transformation rules for Relational algebra operations…

4. **Commuting σ with π.** If the selection condition c involves only those attributes $A_1, \ldots, A_n$ in the projection list, the two operations can be commuted:

$$\pi_{A_1, A_2, \ldots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \ldots, A_n}(R))$$

5. **Commutativity of ⋈ (and ×).** The join operation is commutative, as is the × operation:

$$R \bowtie_c S \equiv S \bowtie_c R$$
$$R \times S \equiv S \times R$$

Notice that although the order of attributes may not be the same in the relations resulting from the two joins (or two Cartesian products), the *meaning* is the same because the order of attributes is not important in the alternative definition of relation.

General transformation rules for Relational algebra operations…

6. **Commuting σ with ⋈ (or ×).** 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:

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

Alternatively, if the selection condition c can be written as (c_1 AND c_2), where condition c_1 involves only the attributes of R and condition c_2 involves only the attributes of S, the operations commute as follows:

$$\sigma_c (R \bowtie S) \equiv (\sigma_{c_1} (R)) \bowtie (\sigma_{c_2} (S))$$

The same rules apply if the ⋈ is replaced by a × operation.

General transformation rules for Relational algebra operations…

7. **Commuting π with $\bowtie$ (or $\times$).** Suppose that the projection list is $L = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$, where $A_1, \ldots, A_n$ are attributes of R and $B_1, \ldots, B_m$ are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

$$\pi_L (R \bowtie_c S) \equiv (\pi_{A_1, \ldots, A_n} (R)) \bowtie_c (\pi_{B_1, \ldots, B_m} (S))$$

General transformation rules for Relational algebra operations…

8. **Commutativity of set operations.** The set operations $\cup$ and $\cap$ are commutative, but $-$ is not.

9. **Associativity of $\bowtie$, $\times$, $\cup$, and $\cap$.** These four operations are individually associative; that is, if both occurrences of θ stand for the same operation that is any one of these four operations (throughout the expression), we have:

   $$(R \,\theta\, S) \,\theta\, T \equiv R \,\theta\, (S \,\theta\, T)$$

10. **Commuting σ with set operations.** The σ operation commutes with $\cup$, $\cap$, and $-$. If θ stands for any one of these three operations (throughout the expression), we have:

    $$\sigma_c (R \,\theta\, S) \equiv (\sigma_c (R)) \,\theta\, (\sigma_c (S))$$

General transformation rules for Relational algebra operations…

11. **The π operation commutes with ∪.**

$$\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$$

12. **Converting a (σ, ×) sequence into ⋈.** If the condition c of a σ that follows a × corresponds to a join condition, convert the (σ, ×) sequence into a ⋈ as follows:

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

General transformation rules for Relational algebra operations…

13. **Pushing σ in conjunction with set difference.**

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

    However, σ may be applied to only one relation:

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

14. **Pushing σ to only one argument in ∩.**

    If in the condition σ_c all attributes are from relation R, then:

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

15. **Some trivial transformations.**

    If S is empty, then $R \cup S = R$

    If the condition c in σ_c is true for the entire R, then $\sigma_c (R) = R$.

Outline of a heuristic algebraic optimization Algorithm

- We can now outline the steps of an algorithm that utilizes some of the above rules to transform an
  initial query tree into a final tree that is more efficient to execute (in most cases)
- The steps of the algorithm are as follows:

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. This permits a greater degree of freedom in moving SELECT operations down different branches of the tree.
2. Using Rules 2, 4, 6, and 10, 13, 14 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. If the condition involves attributes from *only one table*, which means that it represents a *selection condition*, the operation is moved all the way to the leaf node that represents this table. If the condition involves attributes from *two tables*, which means that it represents a *join condition*, the condition is moved to a location down the tree after the two tables are combined.

Outline of a heuristic algebraic optimization Algorithm…

3. Using Rules 5 and 9 concerning commutativity and associativity of binary operations, rearrange the leaf nodes of the tree using the following criteria. First, position the leaf node relations with the most restrictive SELECT operations so they are executed first in the query tree representation. The definition of *most restrictive* SELECT can mean either the ones that produce a relation with the fewest tuples or with the smallest absolute size.[4] Another possibility is to define the most restrictive SELECT as the one with the smallest selectivity; this is more practical because estimates of selectivities are often available in the DBMS catalog. Second, make sure that the ordering of leaf nodes does not cause CARTESIAN PRODUCT operations; for example, if the two relations with the most restrictive SELECT do not have a direct join condition between them, it may be desirable to change the order of leaf nodes to avoid Cartesian products.[5]
4. Using Rule 12, combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation in the tree into a JOIN operation, if the condition represents a join condition.

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. Only those attributes needed in the query result and in subsequent operations in the query tree should be kept after each PROJECT operation.

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

Using Selectivity and Cost Estimates in Query Optimization

- Query optimizer should not depend only on heuristic rules
  - It should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate
  - This approach is called cost-based query optimization, an approach that uses traditional optimization techniques that search the solution space to a problem for a solution that minimizes an objective cost function
  - The cost functions used in query optimization are estimates and not exact cost functions, so the optimization may select a query execution strategy that is not optimal one

Cost Components for Query Execution

The cost of executing a query includes the following components:

- **Access cost to secondary storage**: the cost of searching for, reading and writing data blocks that resides on secondary storage, mainly on disk
- **Storage cost**: the cost of storing any intermediate files that are generated by an execution strategy for the query
- **Computation cost**: the cost of performing in-memory operations on the data buffers during query execution.
  - Such operations include searching for and sorting records, merging records for a join, and performing computations on field values

Cost Components for Query Execution…

The cost of executing a query includes the following components..:

- **Memory usage cost**: the cost pertaining to the number of memory buffers needed during query execution
- **Communication cost:** the cost of shipping the query and its results from the database site or terminal where the query originated

Notes:

- For large databases, the main emphasis is on minimizing the access cost to secondary storage, thus, different query strategies are compared in terms of the number of block transfers between disk and main memory
- For smaller databases, the emphasis is on minimizing computation cost
- In distributed databases, communication cost must be minimized also

(Type of information needed in cost functions) Catalog Information used in Cost Functions

- To estimate the costs of various execution strategies, we must keep track of any information that is needed for the cost functions
- This information may be stored in the DMBS catalog, where it is accessed by the query optimizer
- Number of records (tuples) (r)
- The average record size (R)
- Number of blocks (b)

(Type of information needed in cost functions) Catalog Information used in Cost Functions

- Blocking factor (bfr)
- Primary access method of each file
- Primary access attributes of each file
- Number of levels (x) of each multilevel index
- Number of first-level index blocks
- Number of distinct values (d) of an attribute
- Selectivity (sl) of an attribute
- Selection cardinality (s) of an attribute

Examples of cost functions for SELECT

- The cost is in terms of number of blocks transfers between memory and disk (the estimates for computation time, storage cost and other factors are ignored)
- The cost for each of the methods s1 to s9 is referred to as *block accesses*
- S1: Linear search (brute force) approach: We search all the file blocks to retrieve all records satisfying the selection condition, hence, =b. For an equality condition on a key, only half the file blocks are searched on the average before finding the record, so =(b/2) if the record is found; if no record satisfies the condition, =b.

Examples of cost functions for SELECT…

- S2: Binary search: This search accesses approximately = $\log_2 b$ + (s/bfr) - 1 file blocks. This reduces to $\log_2 b$ if the equality condition is on a unique (key) attribute, because s = 1 in this case.
- S3: Using a) a primary index or b) hash key to retrieve a single record: for a primary index, retrieve one more block than the number of index levels; hence = x +1. For hashing, the cost function is approximately =1 for static hashing or linear hashing, and it is 2 for extended hashing
- S4: Using an ordering index to retrieve multiple records: If the comparison condition is >, >=, < or <= on a key field with an ordering index, roughly half the file records will satisfy the condition. This gives a cost function of = x + (b/2)

Examples of cost functions for SELECT…

- S5: Using a clustering index to retrieve multiple records: Given an equality condition, s records will satisfy the condition, where s is the selection cardinality of the indexing attribute. This means that (s/bfr) file blocks will be accessed, giving = x+ (s/bfr)
- S6: Using a secondary (-tree) index:
  - a) For equality comparison: because the index is non-clustering, the worst case cost estimate is x + S; In fact, this reduces to x + 1 for a key indexing attribute.
  - b) For the comparison conditions >,>=, < or <=, if half the file records are assumed to satisfy the condition, then half the first-level index blocks are accessed, plus half the file records via index. The cost estimate for this case, approximately, is = x +half the first-level index + (r/2)

Examples of cost functions for SELECT…

- S7: Conjunction selection: We can use either S1 or one of the methods S2 to S6 discussed above. In the later case, we 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

Example

- Given the following information, compute the cost of the relational algebra expressions in the next slide using the possible algorithms discussed
- Suppose we have an Employee relation that has 10,000 records stored in 2000 disk blocks with blocking factor of 5 records/block
- Access paths:
  - A clustering index on SALARY, with levels = 3 and average selection cardinality = 20.
  - A secondary index on the key attribute SSN, with = 4 (= 1).
  - A secondary index on the non-key attribute DNO, with = 2 and first-level index blocks = 4. There are = 125 distinct values for DNO, so the selection cardinality of DNO is S_{DNO} = = 80.
  - A secondary index on SEX, with = 1. There are = 2 values for the sex attribute, so the average selection cardinality is = = 5000

Example…

- Q1. $S_{SSN =“123”}$ (Employee)
- Q2. $S_{DNO >5}$ (Employee)
- Q3. $S_{DNO =5}$ (Employee)
- Q4. $S_{DNO =5 \text{ AND } SALARY > 3000 \text{ AND } SEX =“F”}$ (Employee)

Solutions

- Question #1 ($S_{SSN =\text{“}123\text{”}}$ (Employee)):
- we can use method S1 or method s6a (where SSN is non-clustered key)
- Using method S1 (linear search): SSN is a key; the average cost of linear search with a key attribute will be estimated as half of the total number of disk blocks
  - As we have a total of 2000 blocks, the average cost is **1000**
- Using method S6a, the general form was x + s where x is the primary index level and s is the selection cardinality (number of records that satisfy the condition of the attribute). In our case s is 1 as SSN is a key. Furthermore, X is 4 as we have been given that the primary index level of SSN is 4. Thus the cost using method S6a is 4 + 1 = 5
- Then, the optimal method would be method 56a

Solutions…

- Question #2 ($S_{DNO>5}$ (Employee)):
  - DNO is non key attribute, with index levels 2 and first-level index blocks of 4
  - If we employ the method S1, it will cost us 2000 as DNO is non key
  - If we employ method S6b, index levels of the att + ½ (first-level index blocks) + ½ (total records in the relation) :2+ 4/2 + 10,000 /2= 5004 blocks

  Thus, we will go for linear search method for this relational algebra
- Question #3 ($S_{DNO=5}$ (Employee)):
  - Employing method S1 costs us 2000 blocks accesses
  - Employing method S2 ??
  - Employing method S6a, as DNO not a key attribute more than 1 record could satisfy the condition. Thus, S =80; because the index is non-clustering, each of the records may reside on a different block at the worst case, thus, x (index levels) = 2 is applied.
    - X + S = 2 + 80 = 82

Solutions…

- Question #4 ($S_{DNO =5 \text{ AND } SALARY > 3000 \text{ AND } SEX =\text{“F”}}$ (Employee)):
- In this case, the relational algebra has a conjunctive selection condition. The linear search approach results in 2000 block accesses.
- As an alternative, we need to estimate the cost of using any one of the three components of the selection condition to retrieve the records.
  - Using the condition (DNO = 5) first gives the cost estimate of 82 (using S6a)
  - Using the condition (SALARY > 30,000) first gives a cost estimate of 3 + (2000/2) = 1003 (Using S4)
  - Using the condition (SEX = “F”) first gives a cost estimate of 1+ 5000 = 5001 (using S6a)
  - The Query optimizer would then choose method S6a on the secondary index on DNO because it has the lowest cost estimate/
  - The condition (DNO = 5) is used to retrieve the records, and the remaining part of the conjunctive condition (SALARY > 30,000 AND SEX =‘F’) is checked for each selected record after it is retrieved into memory

Semantic Query Optimization

- This is a technique that uses constraints specified on the database schema
  - Like unique attributes and other more complex constrain
- Consider the following SQL:

```
SELECT E.LastName, M. LastName
FROM Employee AS E, Employee AS M
WHERE E.SuperSSN = M.SSN AND E.Salary > M.Salary
```

Semantic Query Optimization…

- The above SQL retrieves names of employees who earn more than their supervisors.
- Suppose that we had a constraint on the database schema that stated that no employee can earn more than his or her direct supervisor.
- If the semantic query optimizer checks for the existence of this constraint, it need not execute the query at all because it knows that the result of the query will be empty. This may save considerable time if the constraint checking can be done efficiently.

Semantic Query Optimization…

- However, searching through many constraints to find those that are applicable to a given query and that may semantically optimize it can also be quite time-consuming.
- With the inclusion of active rules in database systems , semantic query optimization techniques may eventually be fully incorporated into the DBMSs of the future.

Semantic Query Optimization…

- Consider another example:

```
SELECT Lname, Salary
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.Dno = DEPARTMENT.Dnumber and
EMPLOYEE.Salary>100000
```

- In this example, the attributes retrieved are only from one relation: EMPLOYEE; the selection condition is also on that one relation. However, there is a referential integrity constraint that Employee.Dno is a foreign key that refers to the primary key Department.Dnumber

Semantic Query Optimization…

- Therefore, this query can be transformed by removing the DEPARTMENT relation from the query and thus avoiding the inner join as follows:

```
SELECT Lname, Salary
FROM EMPLOYEE
WHERE EMPLOYEE.Dno IS NOT NULL and     EMPLOYEE.Salary>100000
```

- This type of transformation is based on the primary-key/foreign-key relationship semantics, which are a constraint between the two relations