Heuristic Query Optimization Techniques Essay

135114_Abigail Muthoni_ICSA

Heuristics in Query optimization

➔ What is **heuristic**?

Heuristic is a technique used to solve problems faster than classical methods at a cheaper rate.

➔ Optimization strategies that use heuristic principles to improve the predicted performance of a query's internal representation are covered, which is commonly in the form of a **query tree or query graph data structure.**

➔ An SQL query's scanner and parser **first create a data structure** that corresponds to the query's initial representation, which is **later optimized using heuristic methods**.

➔ This results in a query representation that is optimal for the query execution approach.

➔ Then, **from the access pathways** available on the files included in the query, a query execution plan is constructed to execute groups of actions.

➔ One of the most important heuristic rules is to use **SELECT** and **PROJECT** operations **before using JOIN** or other procedures because, the size of the file produced by a binary operation—such as JOIN—is frequently a multiplicative function of the sizes of the input files, binary operations are utilized. The SELECT and PROJECT operations **reduce file size**, they should be used before a join or other binary operation.

➔ We introduced the query tree and query graph notations in the context of relational algebra and calculus, respectively. These can be used as the foundation for the data structures that are utilized to describe queries internally.

➔ A **query graph** is used to represent a relational calculus expression, whereas a **query tree** is used to describe a relational algebra or extended relational algebra statement.

➔ We illustrate how to use heuristic optimization methods to **turn an initial query tree into an equivalent query tree**, which reflects a new relational algebra expression that is more efficient to run but produces the same result as the original tree. The equivalence of several relational algebra expressions is also discussed.

➔ Finally the query execution plan generation will be examined.

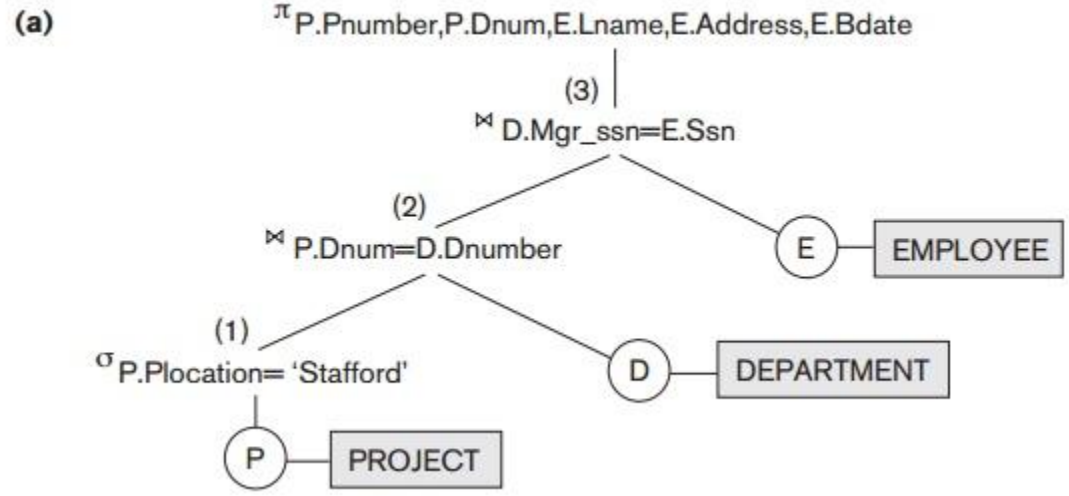
Query Trees and Graphs

- ➔ A **query tree** is a data structure in the form of a tree that represents a relational algebra statement.
- ➔ The query's input relations are represented as **tree leaf nodes**, while the relational algebra operations are represented as **internal nodes**. When the operands for an internal node operation are accessible, the query tree is executed by replacing that internal node with the relation that results from executing the operation.
- ➔ The order in which operations are executed begins with the leaf nodes, which represent the query's input database relations, and ends with the root node, which represents the query's final operation. When the root node action is completed, the execution ends and the result relation for the query is produced.
- ➔ The figure below depicts a query tree for query Q2:
  Retrieve the project number, the managing department number, and the department manager's last name, address, and birthday for each project in 'Stafford.' The following relational algebra expression corresponds to the query specified on the COMPANY relational schema.

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$

This corresponds to the following SQL query:

```
Q2: SELECT  P.Pnumber, 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.Mgr_ssn=E.Ssn AND
            P.Plocation= 'Stafford';
```



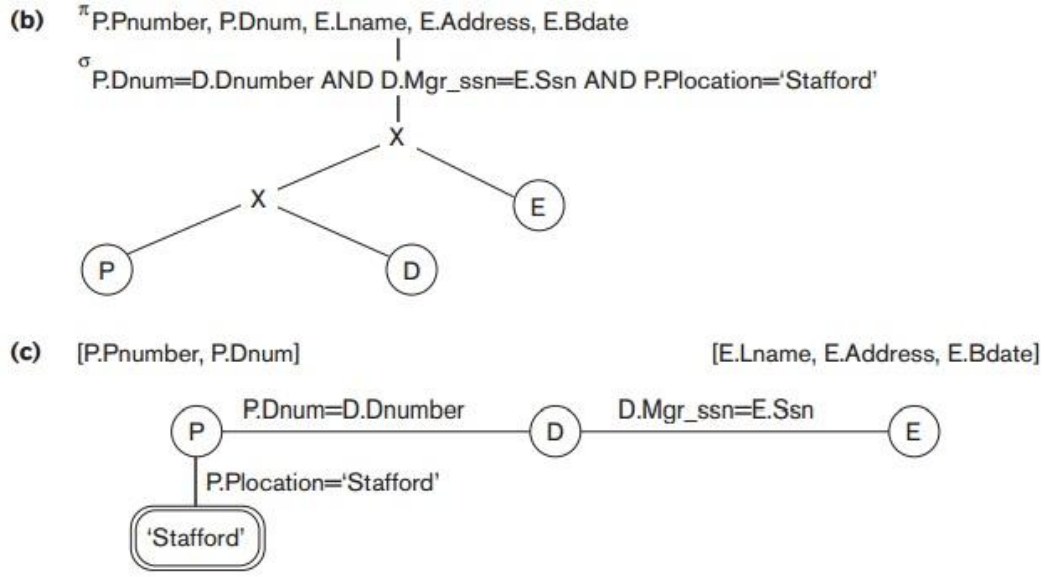


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

- ➔ The leaf nodes P, D, and E represent the three relations PROJECT, DEPARTMENT, and EMPLOYEE, respectively, while the interior tree nodes reflect the expression's relational algebra operations. Because part of the resulting tuples of operation (1) must be ready before we can begin executing operation (2), the node designated (1) in the figure above must begin execution before node (2). Likewise, node (2) must begin running and providing results before node (3), and so on, can begin executing.
- ➔ The query tree, as we can see, represents a precise order of actions for running a query. The query graph notation provides a **more neutral data structure** for representing a query.
- ➔ The query graph for question (Q2) is given in the above figure. **Relation nodes**, which are depicted as single circles, **reflect relations in the query**. **Constant nodes**, which are depicted as double circles or ovals, **reflect constant values**, which are often from the query selection criteria. The graph edges,as illustrated above, reflect the selection and join criteria. Finally, the properties to be outputted from each relation are shown above each relation in square brackets.

➔ The query graph model does not specify which operations should be performed first. Each query has just one graph associated with it. Although some optimization strategies used query graphs, query trees are now widely acknowledged as the preferred method since, in fact, the query optimizer must display the sequence of operations for query execution, which is not achievable with query graphs.

Heuristics optimization of Query Trees

➔ Many distinct relational algebra expressions—and hence many alternative query trees—can be comparable in terms of representing the same question.

➔ Without any optimization, the query parser will normally construct a standard starting query tree to match to a SQL query. The beginning tree for a SELECT-PROJECT-JOIN query, such as Q2. The CARTESIAN PRODUCT of the relations indicated in the FROM clause is applied first, followed by the WHERE clause selection and join criteria, and then the projection on the SELECT clause attributes. Because of the CARTESIAN PRODUCT procedures, such a canonical query tree reflects a relational algebra expression that is inefficient if run directly.

➔ If the PROJECT, DEPARTMENT, and EMPLOYEE relations, respectively, had record sizes of 100, 50, and 150 bytes and included 100, 20, and 5,000 tuples, the CARTESIAN PRODUCT result would have 10 million tuples with record sizes of 300 bytes a piece. The first query tree in the figure above is, on the other hand, in a format that may be easily generated from the SQL query. It will never be carried out. This initial query tree will be transformed into an analogous final query tree that is efficient to run by the heuristic query optimizer.

➔ The optimizer must provide rules for relational algebra expression equivalence that may be used to turn the initial tree into the final, optimized query tree. First the **query tree will be changed by heuristics**, and then we go over generic transformation rules and how they may be utilized in an algebraic heuristic optimizer.

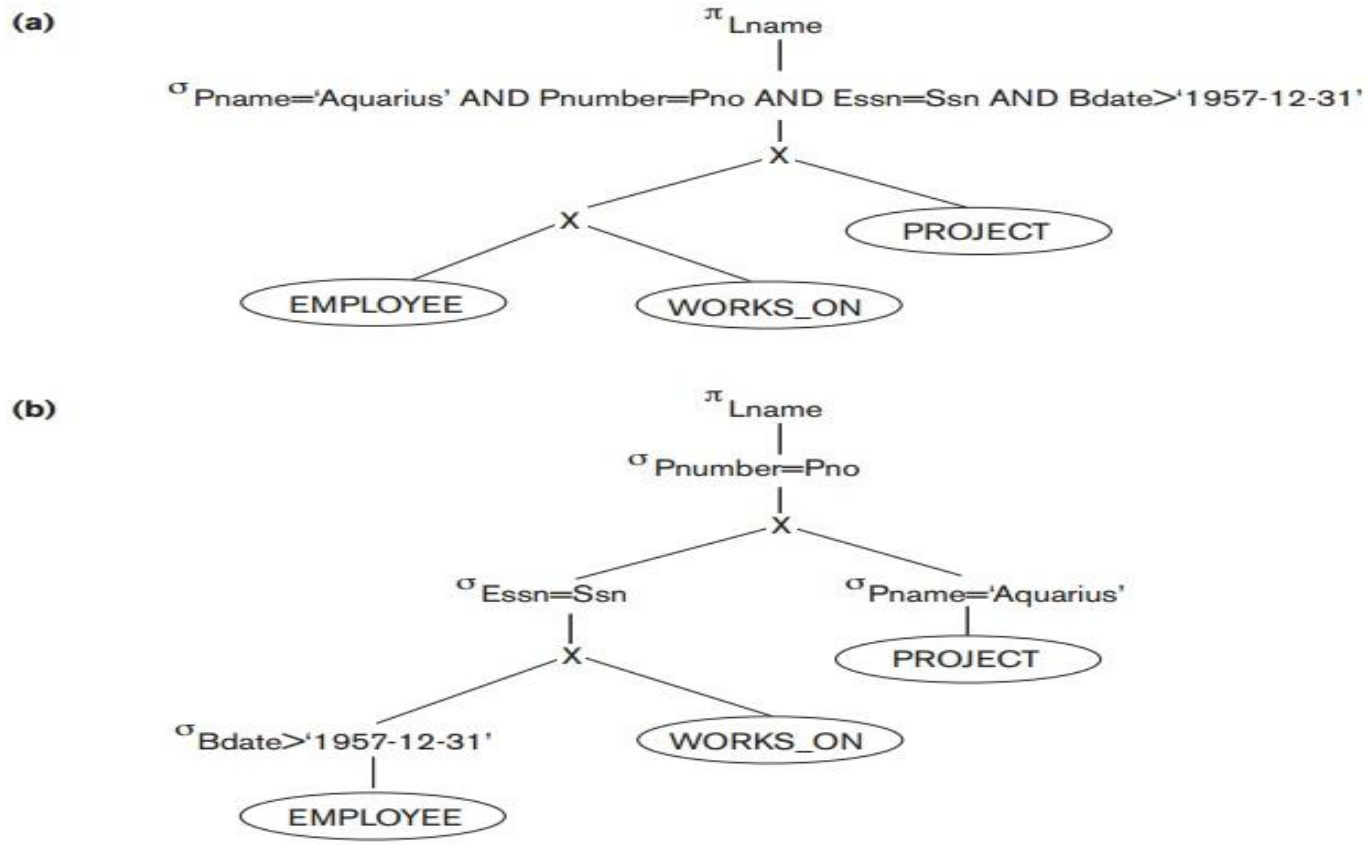
➔ An example of query transformation. Consider the following query Q on figure above: Find the last names of employees who started working on the 'Aquarius' project after 1957. This query may be written as follows in SQL:

SELECT Lname
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE Pname='Aquarius' AND Pnumber=Pno AND Essn=Ssn
AND Bdate > '1957-12-31';

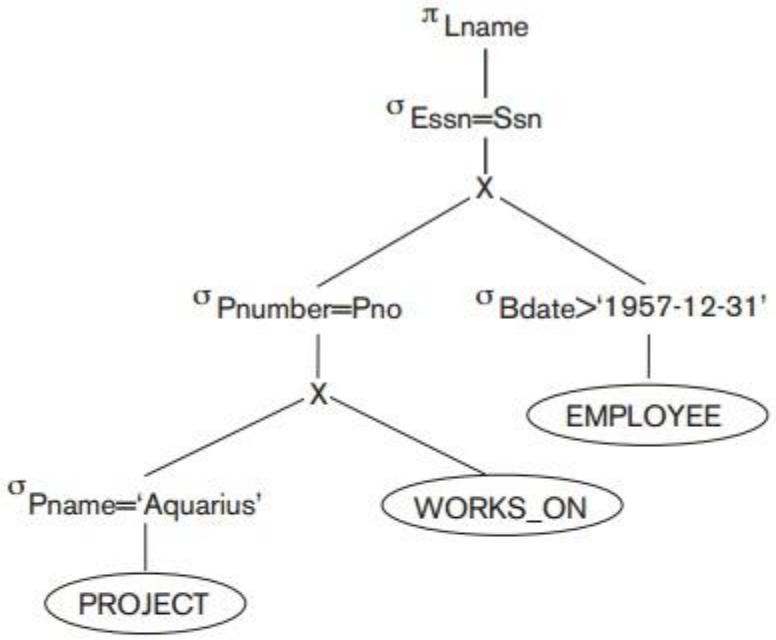
➔ The figure above shows the initial query tree for Q. (a). Directly executing this tree produces a very big file that contains the CARTESIAN PRODUCT of the whole EMPLOYEE, WORKS ON, and PROJECT files. As a result, the initial query tree is never run, but rather changed into a more efficient equivalent tree.

Steps in converting a query tree during heuristic optimization

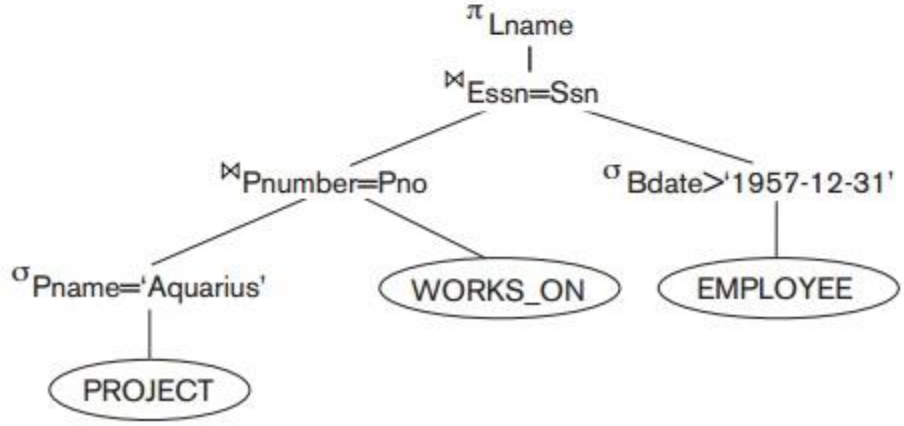
1. Initial (canonical) query tree for SQL query Q.
2. Moving SELECT operations down the query tree.
3. Applying the more restrictive SELECT operation first.
4. Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
5. Moving PROJECT operations down the query tree.

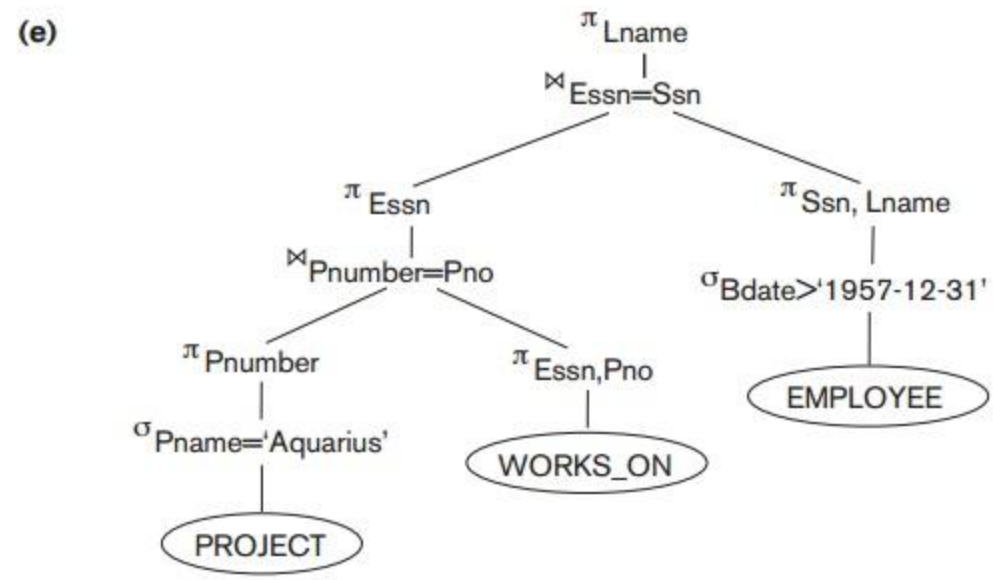


(c)


π Lname
σ Essn=Ssn
X
σ Pnumber=Pno
σ Bdate>'1957-12-31'
X
EMPLOYEE
σ Pname='Aquarius'
WORKS_ON
PROJECT

(d)


π Lname
⋈ Essn=Ssn
⋈ Pnumber=Pno
σ Bdate>'1957-12-31'
σ Pname='Aquarius'
WORKS_ON
EMPLOYEE
PROJECT



➔ This particular query only needs one record from the PROJECT relation—for the 'Aquarius' project—and only the EMPLOYEE records for individuals whose date of birth is after '1957-12-31' are required for this query. The figure demonstrates an updated query tree that reduces the amount of tuples in the CARTESIAN PRODUCT by using SELECT operations first.

➔ Switching the placements of the EMPLOYEE and PROJECT connections in the tree, as illustrated in figure above, improves the situation much further (c). The fact that Pnumber is a key property of the PROJECT relation means that the SELECT operation on the PROJECT relation will only return one entry. As illustrated, we may optimize the query tree even further by replacing every CARTESIAN PRODUCT action that is followed by a join condition with a JOIN operation (d). Another enhancement is to maintain just the characteristics required by following operations in intermediary relations, as illustrated, by adding PROJECT () procedures as early as practicable in the query tree (e).

➔ The intermediary relations' characteristics (columns) are reduced, while the SELECT operations reduce the amount of tuples (records).

➔ A query tree can be changed into an equivalent query tree that is more efficient to run, as seen in the prior example. However, we must ensure that the transformation stages always result in a query tree that is equal. The query optimizer needs to know which transformation rules retain this equivalence in order to do so.

Advantages of Heuristic Optimizer over Cost-based optimization

➔ It is cheaper.

REFERENCES

Using Heuristics in Query Optimization. (2018). BrainKart.

https://www.brainkart.com/article/Using-Heuristics-in-Query-Optimization_11540/

What is heuristic optimization in DBMS? (2022). Tutorialspoint.

https://www.tutorialspoint.com/what-is-heuristic-optimization-in-dbms