COS221 - L29 - Query processing and optimisation (Part 2)

Linda Marshall

16 May 2023

Using Heuristics for Query Optimisation

- Heuristic rules are used to modify the internal representation (query tree/graph data structure) of a query to improve performance.
- The scanner and parser of an SQL query generates a data structure that corresponds to the initial query representation.
- This representation is then optimised using heuristic rules resulting in an optimised query representation.
- A query execution plan is then defined to execute groups of operations based on the access paths available to the files involved in the query.
- One of the main heuristics is to apply SELECT and PROJECT operations before a JOIN - or any other binary operation. SELECT and PROJECT reduce the size of the files involved in the JOIN - or any other binary operation.

Using Heuristics for Query Optimisation

Consider the following query:

```
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';
```

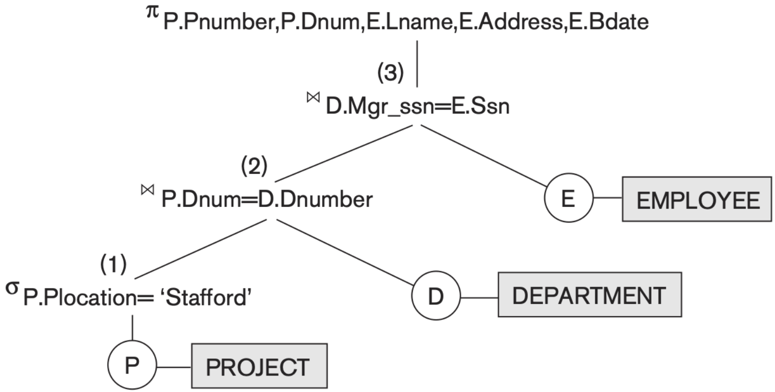
The corresponding RA is given by:

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

Using Heuristics for Query Optimisation

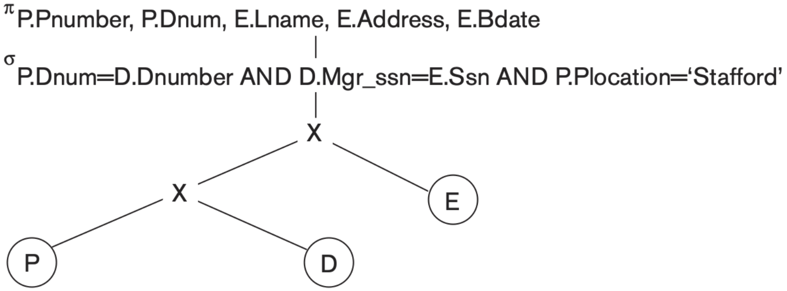
$$\pi_{Pnumber,Dnum,Lname,Address,Bdate}(((\sigma_{Plocation='Stafford'}(PROJECT)) \bowtie_{Dnum=Dnumber} (DEPARTMENT)) \bowtie_{Mgr\_ssn=SSn} (EMPLOYEE))$$

The query tree for the RA expression is given by:



Using Heuristics for Query Optimisation

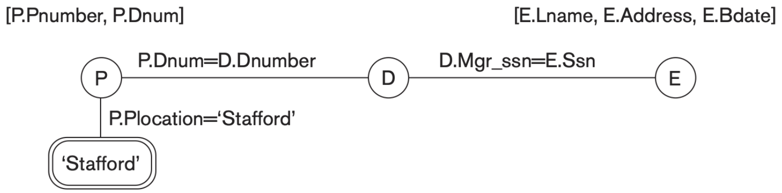
- It is possible that multiple equivalent query trees exist for a query.
- The tree below is an example of the initial query tree of the RA before optimisation.



π P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
σ P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation='Stafford'
X
X
E
P
D

Using Heuristics for Query Optimisation

A query graph, which is more neutral in its representation of the query is given by the following, where P, D and E are relation nodes and 'Stafford' a constant node. Selection and join conditions are given on the edges with attributes to be retrieved (projected) given for each relation. The order in which the operations are to be done is not given.

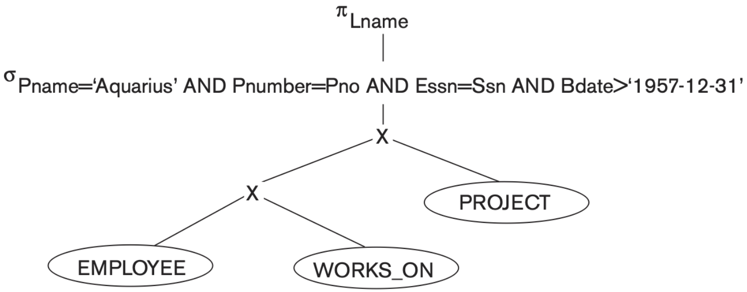


Using Heuristics for Query Optimisation

Consider the following query:

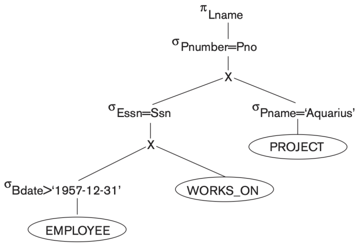
```
SELECT Lname
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE Pname='Aquarius' AND Pnumber=Pno AND Essn=SSn
      AND Bdate > '1957-12-31';
```

The initial tree is given by:


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

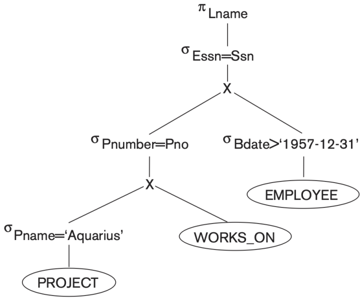
Using Heuristics for Query Optimisation

This query requires one record from PROJECT where Pname is 'Aquarius' and employee records with date of birth after 31 December 1957. To reduce the search space, the SELECT operations can be applied first. It would however be better if the more restrictive select is applied first (i.e. Pname='Aquarius').



π_{Lname}
$\sigma_{\text{Pnumber=Pno}}$
X
$\sigma_{\text{Essn=Ssn}}$
$\sigma_{\text{Pname='Aquarius'}}$
PROJECT
X
$\sigma_{\text{Bdate>'1957-12-31'}}$
WORKS_ON
EMPLOYEE

Move SELECT statements down the tree

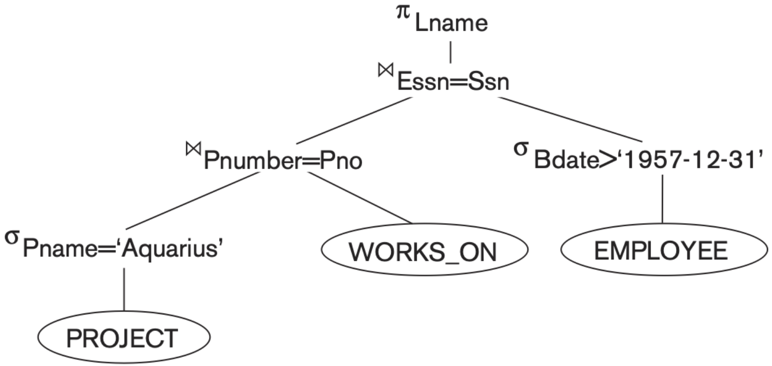


π_{Lname}
$\sigma_{\text{Essn=Ssn}}$
X
$\sigma_{\text{Pnumber=Pno}}$
$\sigma_{\text{Bdate>'1957-12-31'}}$
X
EMPLOYEE
$\sigma_{\text{Pname='Aquarius'}}$
WORKS_ON
PROJECT

Apply the more restrictive SELECT first.

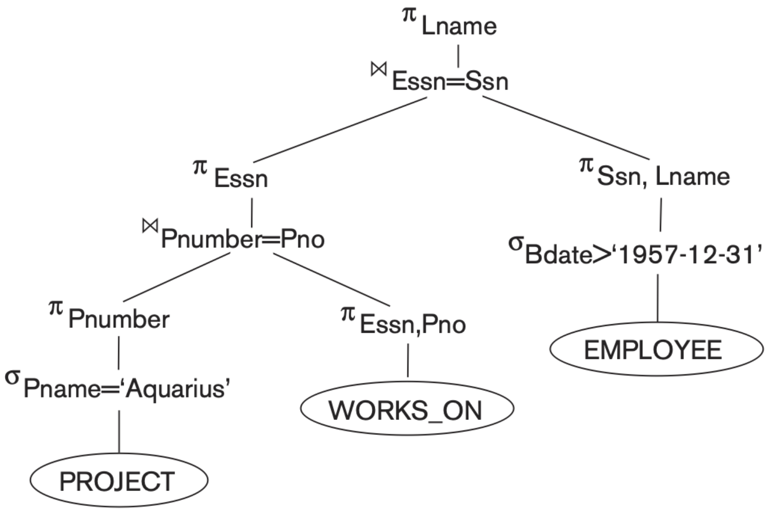
Using Heuristics for Query Optimisation

Replace any Cartesian product followed by a select operation with a join operation



Using Heuristics for Query Optimisation

Keep only the attributes needed by subsequent operations in the intermediate relations by including project operations as early as possible.



Using Heuristics for Query Optimisation

- It is important that the query tree transformation preserves equivalence.
- Once a query tree has been created, it is converted into a query execution plan.

General Transformation Rules for RA Operations

1. Cascade of σ:
   $\sigma_{C_1 \text{ AND } C_2 \text{ AND } \ldots \text{ AND } C_n}(R) \equiv \sigma_{C_1}(\sigma_{C_2}(\ldots(\sigma_{C_n}(R))\ldots))$
2. Commutivity of σ:
   $\sigma_{C_1}(\sigma_{C_2}) \equiv \sigma_{C_2}(\sigma_{C_1})$
3. Cascade of π: In a list of projects, all except the last one can be ignored
   $\pi_{List_1}(\pi_{List_2}(\ldots(\pi_{List_n}(R)))) \equiv \pi_{List_1}(R)$
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.
   $\pi_{A_1, A_2, \ldots A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \ldots A_n}(R))$

General Transformation Rules for RA Operations

5. Commutativity of $\bowtie$ (and $\times$) : Note, the order of the attributes may not be the same in the relations, but the meaning is.
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$
6. Commuting σ with $\bowtie$ (or $\times$):
   $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$
   If c comprises of c_1 and c_2 with c_1 being attributes of R and c_2 attributes of S, then the commute takes place as follows:
   $\sigma_c(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$

General Transformation Rules for RA Operations

7. Commuting π with $\bowtie$ (or $\times$):
   $\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,A_2,...,A_n}(R)) \bowtie_c (\pi_{B_1,B_2,...,B_m}(S))$, where $L = \{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m\}$ and A_i is an attribute of R and B_j is an attribute of S.
   If c contains attributes not in L, then the commuting requires another project on the RHS to result in only attributes of L.
   $\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A_1,...,A_n,A_{n+1},...,A_{n+k}}(R)) \bowtie_c (\pi_{B_1,...,B_m,B_{m+1},...,B_{m+l}}(S)))$
8. Commutivity of set operations: UNION and INTERSECTION are commutative, but SET DIFFERENCE is not.

General Transformation Rules for RA Operations

9. Associativity of $\bowtie$, $\times$, $\cup$ and $\cap$:
   Let θ stand for these operations, then $(R\theta S)\theta T \equiv R\theta(S\theta T)$
10. Commuting σ with set operations:
    $\sigma_c(R\theta S) \equiv (\sigma_c(R))\theta(\sigma_c(S))$
11. The π operation commutes with $\cup$:
    $\pi_L(R\theta S) \equiv (\pi_L(R))\theta(\pi_L(S))$
12. Converting a (*sigma*, *times*) sequence into $\bowtie$:
    $(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

DeMorgan's Law
$NOT(c_1 \text{ AND } c_2) \equiv (NOT c_1) \text{ OR } (NOT c_2)$
$NOT(c_1 \text{ OR } c_2) \equiv (NOT c_1) \text{ AND } (NOT c_2)$

Heuristic Algebraic Optimisation Algorithm

- Step 1. Use Rule 1 to break SELECT operations with conjunctive conditions into cascading SELECT operations. This makes it easier to move SELECT operations down the branches of the tree.
- Step 2. Use rules 2, 4, 6 and 10 to move the SELECT as far down the query tree as permitted. A selection condition on one table is moved to the leaf node that represents the table. A selection condition on two tables requires a join and the select is moved to after the join.
- Step 3. Use rules 5 and 9 to arrange the leaf nodes based on the following criteria:
  - position the leaf node relations so that the most restrictive SELECT operation is executed first.
  - ensure that the positioning of leaf nodes does not result in Cartesian product type operations

Heuristic Algebraic Optimisation Algorithm - cont.

- Step 4. Use rule 12 to combine a Cartesian product with subsequent SELECT into a join operation.
- Step 5. Use rules 3, 4, 7 and 11 to break down and move projection attributes as far down the tree as possible.
- Step 6. Identify sub-trees that represent groups of operations that can be executed by a single algorithm.

Using Selectivity and Cost Estimates in Query Optimisation

- Query optimisation does not solely depend in heuristic rules. The cost of executing the query using different execution strategies and algorithms also plays a role. The lowest cost estimate is chosen and therefore needs to be accurate.
- The number of execution strategies should not be so vast that time taken in choosing the strategy takes away from the time taken to optimise.
- The cost estimation approach is best suited for compiled queries. If used with interpreted queries, full-blown optimisation will take too much from the available runtime.
- *Cost-based query optimisation* is an optimisation technique that searches the solution space of a problem to minimise the objective (cost) function. The cost function in query optimisation is an estimate and therefore it is possible that the best solution is not chosen.

Using Selectivity and Cost Estimates in Query Optimisation

The cost of executing a query includes the following components:

- Access cost to secondary storage - the cost of reading/writing blocks to and from secondary storage as well as the cost of searching for information.
- Disk storage cost - the cost of storing intermediate files generated by an execution strategy
- Computation cost - the cost of performing operation on the records in memory. Also referred to as CPU cost.
- Memory usage cost - the cost relating to the number of main memory buffers needed to execute the query.
- Communication cost - the cost of moving the query and its results between the database and the site from where the query originated.

The exercise is about minimising cost. For large databases minimising access cost to secondary storage is important. For small databases the minimising of computation cost is the focus.

Using Selectivity and Cost Estimates in Query Optimisation

- The information used by the cost function is kept in the database catalog. The following information is typically stored:
  - size of file
  - number of tuples/records
  - record size
  - number of file blocks
  - blocking factor
- The primary file organisation for each file is also kept, such as unordered or ordered on an attribute. If ordered, whether a primary or clustering index is kept on a key attribute or is the access hashed! Information on secondary indexes.

Using Selectivity and Cost Estimates in Query Optimisation

- The number of distinct values of an attribute and the attribute selectivity are kept and used to calculate the selection cardinality. Counters or selectivity values can be stored for specific attribute values.
- Cost functions are calculated for each algorithm - e.g. the selection algorithms each have a cost function associated with it as do the join algorithms.
- Join queries can very quickly produce any number of alternative query trees. Join query trees are often limited to left-deep or right-deep trees.

Semantic Query Optimisation

- Semantic query optimisation makes use of constraints specified on the database schema.
- For example, if a constraint is placed on the database that an employee may not earn more than their direct supervisor, a query requiring such information will return an empty result because it is not possible in terms of the data.
- Techniques used are:
  - Join elimination
  - Predicate introduction
  - Order optimisation
  - Exploiting uniqueness