Example query

\[ \pi_{P.Pnumber, P.Dnum, P.Lname, E.Address, E.Bdate} \mid D.MgrSSN=E.SSN \mid P.Dnum=D.Dnumber \mid P.Plocation='Stafford' \]

Turned into canonical form

\[ \pi_{P.Pnumber, P.Dnum, P.Lname, E.Address, E.Bdate} \sigma_{P.Dnum=D.Dnumber \land D.MgrSSN=E.SSN \land P.Plocation='Stafford'} \]

Query Graph

For completeness (we won’t use it in this chapter.)

Example of optimization

Given SQL query:

```
SELECT LNAME
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE Pname='Aquarius' \land Pnumber=Pno \land ESSN=SSN \land Bdate>='1957-12-31'
```

Generate canonical tree and optimize
Canonical Tree

First optimization
Move selects down the tree
Only need one project from all projects, only
need employees born after 1957-12-31

Graph with first opt applied

Second optimization
Do joins on key attributes early.
Pnumber = Pno and ESSN=SSN are two
examples of joins which can be done early.

Of these two relations, Pnumber=Pno after
\( \sigma_{\text{Pname='Aquarius'}} \) results in only one record
(do it first!)
Also move this join earlier than
EMPLOYEE, WORKS_ON join.

Graph with 2nd opt applied

Third optimization
Reduce attributes as early as possible. Fewer
attributes to carry around, few attributes to
add in join operations.
Push selects down to earliest place possible.
Always beware

Make sure that each optimization doesn’t change the nature of the query.

Leads us to General Transformation Rules for Relational Algebra Operations.

Rule 1.

Cascade of $\sigma$

$\sigma_{c_1 \land c_2 \land c_3 \cdots c_n}(R) = \sigma_{c_n}(\sigma_{c_{n-1}}(\cdots(\sigma_{c_2}(\sigma_{c_1}(R))))$

Replace the several and’ed conditions with a cascade of conditions.

(used to break one condition into many smaller conditions)

Rule 2

Commutativity of $\sigma$

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

The order of the conditions can be exchanged.

Rule 3

Cascade of $\pi$

$\pi_{list_1}(\pi_{list_2}(\cdots(\pi_{list_n}(R)))) = \pi_{list_n}(R)$

All project operations except for the last, can be ignored.

Rule 4

Commuting $s$ with $p$

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

The order of select and project is interchangeable if and only if all the arguments used in the condition $c$ are in \{A_1, A_2 \ldots A_n\}.

Rule 5

$\bowtie$ and $X$ are commutative.

$R \bowtie S = S \bowtie R$ and $R \times S = S \times R$

Order of attributes is different (but equivalent see 7.1.2)
Rule 6
Commute $\sigma$ with $\bowtie$ or $X$
$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$
If C’s attributes are totally in R’s attributes,
If c can be divided into c1 and c2 (c1 in R’s attributes and c2 in S’s attributes)
$\sigma_{c1}(R \bowtie S) = \sigma_{c1}(R) \bowtie \sigma_{c2}(R)$

Rule 7
Commute $\pi$ with $\bowtie$ or $X$
$\pi_c(R \bowtie S) = \pi_{L1}(R) \bowtie \pi_{L2}(S)$
Where $L1 \subseteq R$’s attributes, $L2 \subseteq S$’s attributes.
If c contains attributes not in L then these attributes must be added to $L1$ and $L2$ and the final select must be applied to both.

Rule 8
Union and Intersection are commutative. Set difference is not.

Rule 9
$\bowtie$, $\times$, $\cap$, and $\cup$ are associative
$(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$
Where is $\bowtie$ the same operation over the entire expression. This is not the case if $\bowtie$ varies over the expression (that is $(A \times B) \cup C$? 
$A \times (B \cup C)$ may not necessarily be true.)

Rule 10
Commuting $\sigma$ with set operations
$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie \sigma_c(S)$

Rule 11
Commuting $\pi$ with $\cup$
$\pi_{L1}(R \cup S) = \pi_{L1}(R) \cup \pi_{L1}(S)$
Rule 12

\[ \sigma_c (R \times S) = (R \bowtie_c S) \]

If \( c \) in \( \sigma_c \) is a join condition, convert \( \sigma X \) into \( \bowtie_c \).

Other rules

Many more including the DeMorgan’s Laws and others.

How to apply rules to make better queries.

Reminder these are “potentially” better queries.

Algorithm

1. Start with canonical form.
2. Break \( \sigma \) into cascade of \( \sigma \) (Rule 1).
3. Use rules 2, 4, 6, and 10 to move \( \sigma \) as far down tree as possible. Use \( c \) to guide in selection of \( \sigma \) placement.
4. Apply rules 5 and 9 to rearrange leaf nodes so most restrictive \( \sigma \) are executed first (produce fewest tuples or smallest absolute size).

Algorithm cont.

4. Second, if results from \( \sigma \) are not join compatible, rearrange nodes to remove need for \( X \).
5. Apply rule 12, (convert SELECT-CP to join)
6. Apply rules 3, 4, 7, and 11 to move PROJECTs down. Should keep only the bare minimum attributes to perform query.

Application of algorithm

We already applied to to solve the problem earlier.
Execution Plans

The final goal for a query tree is to be used in an execution plan.

Given the SQL query

```
SELECT Fname, Lname, Address
FROM DEPARTMENT, EMPLOYEE
WHERE Dname='Research' and Dnumber=Dno;
```

Use optimization to convert canonical query to optimized query.

```
πFname, Lname, Address
σDname='Research'

DEPARTMENT

EMPLOYEE

σDnumber=Dno
```

Optimizer

Takes the query tree and figures out how to implement it given the database structure.

e.g.
1. An index search on DEPARTMENT.
2. A table scan access method for EMPLOYEE.
3. Nested-loop join.
4. Scan of join for select.

Evaluations

Materialized evaluation - generates intermediate tables for each result (physically materialized).
Pipelined evaluation - as tuples are produced, they are fed into waiting operations. (best since we don’t write intermediate results to disk or read them out.)

Summarize heuristic optimization

Better than nothing.
Done for interactive sessions. (interpreted, not the best, but still very good).
Exploit selectivity to reduce # of block that need to be transferred.

Cost-based query optimization

Better way to optimize queries.
Find the algorithm with the lowest cost estimate (only as good as your estimates).
Works best for compiled queries (vs. interpreted).

Search solution space of opportunities. Try to minimize a cost function.
Typical costs
Cost to access secondary storage - seek, read, write a block time. Influenced by physical layout.
Storage costs - intermediate files.
Computation costs - algorithm cost (sort etc.)
Memory usage costs - how much primary memory.
Communication costs - query and result.

Depend on size and layout of database
Large DB - minimize secondary storage access.
Small in memory DB - minimize computation cost.
Distributed DB - minimize communications costs.

Definitions used for costs calculations
Per table
r - number of tuples
R - average record size (bytes)
Bfr - blocking factor (records/block)
primary access methods and attributes.
x - number of level in multi-level index
b1 - number of first-level index blocks

More definitions
D - number of distinct values
sl - fraction of records satisfying equality on attribute.
s - selection cardinality (sl*r) (avg. # of records selected.)
Some values change, some don’t.