2. Textbook Query Optimization

- Algebra Revisited
- Canonical Query Translation
- Logical Query Optimization
- Physical Query Optimization

Algebra Revisited

The algebra needs some more thought:

- correctness is critical for query optimization
- can only be guaranteed by a formal model
- the algebra description in the introduction was too cursory

What we ultimately want to do with an algebraic model:

- decide if two algebraic expressions are equivalent (produce the same result)

This is too difficult in practice (not computable in general), so we at least want to:

- guarantee that two algebraic expressions are equivalent (for some classes of expressions)

This still requires a strong formal model. We accept false negatives, but not false positives.

Tuples

Tuple:

- a (unordered) mapping from attribute names to values of a domain
- sample: [name: "Sokrates", age: 69]

Schema:

- a set of attributes with domain, written $\mathcal{A}(t)$
- sample: {(`name`,string),(age, number)}

Note:

- simplified notation on the slides, but has to be kept in mind
- domain usually omitted when not relevant
- attribute names omitted when schema known

Tuple Concatenation

- notation: $t_1 \circ t_2$
- sample: [name: "Sokrates", age: 69]$\circ$[country: "Greece"]
  $=$ [name: "Sokrates", age: 69, country: "Greece"]
- note: $t_1 \circ t_2 = t_2 \circ t_1$, tuples are unordered

Requirements/Effects:

- $\mathcal{A}(t_1) \cap \mathcal{A}(t_2) = \emptyset$
- $\mathcal{A}(t_1 \circ t_2) = \mathcal{A}(t_1) \cup \mathcal{A}(t_2)$

Tuple Projection

Consider $t =$ [name: "Sokrates", age: 69, country: "Greece"]

Single Attribute:

- notation $t.a$
- sample: $t.name =$ "Sokrates"

Multiple Attributes:

- notation $t_{|A}$
- sample: $t_{|\{name,age\}} =$ [name: "Sokrates", age: 69]

Requirements/Effects:

- $a \in \mathcal{A}(t)$, $A \subseteq \mathcal{A}(t)$
- $\mathcal{A}(t_{|A}) = A$
- notice: $t.a$ produces a value, $t_{|A}$ produces a tuple

Relations

Relation:

- a set of tuples with the same schema
- sample: {[name: "Sokrates", age: 69], [name: "Platon", age: 45]}

Schema:

- schema of the contained tuples, written $\mathcal{A}(R)$
- sample: {(name,string),(age, number)}

Sets vs. Bags

- relations are sets of tuples
- real data is usually a multi set (bag)

Example:

```
select age
from student
```

age
23
24
24
...

- we concentrate on sets first for simplicity
- many (but not all) set equivalences valid for bags

The optimizer must consider three different semantics:

- logical algebra operates on bags
- physical algebra operates on streams (order matters)
- explicit duplicate elimination ⇒ sets

Set Operations

Set operations are part of the algebra:

- union ($L \cup R$), intersection ($L \cap R$), difference ($L \setminus R$)
- normal set semantic
- but: schema constraints
- for bags defined via frequencies (union $\rightarrow +$, intersection $\rightarrow$ min, difference $\rightarrow -$)

Requirements/Effects:

- $\mathcal{A}(L) = \mathcal{A}(R)$
- $\mathcal{A}(L \cup R) = \mathcal{A}(L) = \mathcal{A}(R)$, $\mathcal{A}(L \cap R) = \mathcal{A}(L) = \mathcal{A}(R)$, $\mathcal{A}(L \setminus R) = \mathcal{A}(L) = \mathcal{A}(R)$

Free Variables

Consider the predicate *age* $= 62$

- can only be evaluated when *age* has a meaning
- *age* behaves a free variable
- must be bound before the predicate can be evaluated
- notation: $\mathcal{F}(e)$ are the free variables of e

Note:

- free variables are essential for predicates
- free variables are also important for algebra expressions
- dependent join etc.

Selection

Selection:

- notation: $\sigma_p(R)$
- sample: $\sigma_{a \geq 2}(\{[a:1],[a:2],[a:3]\}) = \{[a:2],[a:3]\}$
- predicates can be arbitrarily complex
- optimizer especially interested in predicates of the form $attrib = attrib$ or $attrib = const$

Requirements/Effects:

- $\mathcal{F}(p) \subseteq \mathcal{A}(R)$
- $\mathcal{A}(\sigma_p(R)) = \mathcal{A}(R)$

Projection

Projection:

- notation: $\Pi_A(R)$
- sample: $\Pi_{\{a\}}(\{[a:1,b:1],[a:2,b:1]\}) = \{[a:1],[a:2]\}$
- eliminates duplicates for set semantic, keeps them for bag semantic
- note: usually written as $\Pi_{a,b}$ instead of the correct $\Pi_{\{a,b\}}$

Requirements/Effects:

- $A \subseteq \mathcal{A}(R)$
- $\mathcal{A}(\Pi_A(R)) = A$

Rename

Rename:

- notation: $\rho_{a \rightarrow b}(R)$
- sample:
  $\rho_{a \rightarrow c}(\{[a:1,b:1],[a:2,b:1]\}) = \{[c:1,b:1],[c:2,b:2]\}$?
- often a pure logical operator, no code generation
- important for the data flow

Requirements/Effects:

- $a \in \mathcal{A}(R), b \notin \mathcal{A}(R)$
- $\mathcal{A}(\rho_{a \rightarrow b}(R)) = \mathcal{A}(R) \setminus \{a\} \cup \{b\}$

Join

Consider $L = \{[a : 1], [a : 2]\}, R = \{[b : 1], [b : 3]\}$

Cross Product:

- notation: $L \times R$
- sample: $L \times R = \{[a : 1, b : 1], [a : 1, b : 3], [a : 2, b : 1], [a : 2, b : 3]\}$

Join:

- notation: $L \bowtie_p R$
- sample: $L \bowtie_{a=b} R = \{[a : 1, b : 1]\}$
- defined as $\sigma_p(L \times R)$

Requirements/Effects:

- $\mathcal{A}(L) \cap \mathcal{A}(R) = \emptyset, \mathcal{F}(p) \in (\mathcal{A}(L) \cup \mathcal{A}(R))$
- $\mathcal{A}(L \times R) = \mathcal{A}(L) \cup \mathcal{A}R$

Equivalences

Equivalences for selection and projection:

$$
\begin{aligned}
\sigma_{p_1 \wedge p_2}(e) &\equiv \sigma_{p_1}(\sigma_{p_2}(e)) && (1)\\
\sigma_{p_1}(\sigma_{p_2}(e)) &\equiv \sigma_{p_2}(\sigma_{p_1}(e)) && (2)\\
\Pi_{A_1}(\Pi_{A_2}(e)) &\equiv \Pi_{A_1}(e) && (3)\\
& \quad \text{if } A_1 \subseteq A_2 \\
\sigma_p(\Pi_A(e)) &\equiv \Pi_A(\sigma_p(e)) && (4)\\
& \quad \text{if } \mathcal{F}(p) \subseteq A \\
\sigma_p(e_1 \cup e_2) &\equiv \sigma_p(e_1) \cup \sigma_p(e_2) && (5)\\
\sigma_p(e_1 \cap e_2) &\equiv \sigma_p(e_1) \cap \sigma_p(e_2) && (6)\\
\sigma_p(e_1 \setminus e_2) &\equiv \sigma_p(e_1) \setminus \sigma_p(e_2) && (7)\\
\Pi_A(e_1 \cup e_2) &\equiv \Pi_A(e_1) \cup \Pi_A(e_2) && (8)
\end{aligned}
$$

Equivalences

Equivalences for joins:

$$
\begin{aligned}
e_1 \times e_2 &\equiv e_2 \times e_1 && (9)\\
e_1 \bowtie_p e_2 &\equiv e_2 \bowtie_p e_1 && (10)\\
(e_1 \times e_2) \times e_3 &\equiv e_1 \times (e_2 \times e_3) && (11)\\
(e_1 \bowtie_{p_1} e_2) \bowtie_{p_2} e_3 &\equiv e_1 \bowtie_{p_1} (e_2 \bowtie_{p_2} e_3) && (12)\\
\sigma_p(e_1 \times e_2) &\equiv e_1 \bowtie_p e_2 && (13)\\
\sigma_p(e_1 \times e_2) &\equiv \sigma_p(e_1) \times e_2 && (14)\\
& \quad \text{if } \mathcal{F}(p) \subseteq \mathcal{A}(e_1) &&\\
\sigma_{p_1}(e_1 \bowtie_{p_2} e_2) &\equiv \sigma_{p_1}(e_1) \bowtie_{p_2} e_2 && (15)\\
& \quad \text{if } \mathcal{F}(p_1) \subseteq \mathcal{A}(e_1) &&\\
\Pi_A(e_1 \times e_2) &\equiv \Pi_{A_1}(e_1) \times \Pi_{A_2}(e_2) && (16)\\
& \quad \text{if } A = A_1 \cup A_2, A_1 \subseteq \mathcal{A}(e_1), A_2 \subseteq \mathcal{A}(e_2) &&
\end{aligned}
$$

Canonical Query Translation

Canonical translation of SQL queries into algebra expressions.
Structure:

select distinct $a_1, \ldots, a_n$
from $R_1, \ldots, R_k$
where p

Restrictions:

- only **select distinct** (sets instead of bags)
- no **group by**, **order by**, **union**, **intersect**, **except**
- only attributes in **select** clause (no computed values)
- no nested queries, no views
- not discussed here: NULL values

From Clause

1. Step: Translating the **from** clause

Let $R_1, \ldots, R_k$ be the relations in the **from** clause of the query.
Construct the expression:

$$F = \begin{cases} R_1 & \text{if } k = 1 \\ ((\ldots(R_1 \times R_2) \times \ldots) \times R_k) & \text{else} \end{cases}$$

Where Clause

2. Step: Translating the **where** clause

Let p be the predicate in the **where** clause of the query (if a **where** clause exists).
Construct the expression:

$$W = \begin{cases} F & \text{if there is no } \textbf{where} \text{ clause} \\ \sigma_p(F) & \text{otherwise} \end{cases}$$

Select Clause

3. Step: Translating the **select** clause

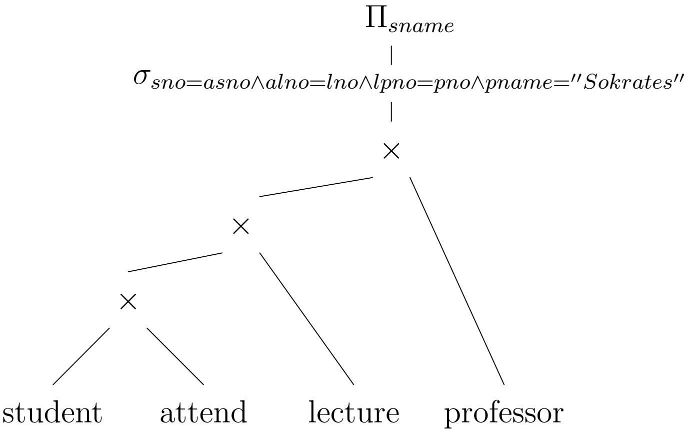
Let $a_1, \ldots, a_n$ (or "*") be the projection in the **select** clause of the query. Construct the expression:

$$S = \begin{cases} W & \text{if the projection is "*"} \\ \Pi_{a_1,\ldots,a_n}(W) & \text{otherwise} \end{cases}$$

4. Step: S is the canonical translation of the query.

Sample Query

select distinct $s.sname$
from $student\ s,\ attend\ a,\ lecture\ l,\ professor\ p$
where $s.sno = a.asno$ **and** $a.alno = l.lno$ **and**
$l.lpno = p.pno$ and $p.pname ='' Sokrates''$



Extension - Group By Clause

2.5. Step: Translating the **group by** clause. Not part of the "canonical" query translation!

Let $g_1, \ldots, g_m$ be the attributes in the **group by** clause and *agg* the aggregations in the **select** clause of the query (if a **group by** clause exists). Construct the expression:

$$G = \begin{cases} W & \text{if there is no } \textbf{group by} \text{ clause} \\ \Gamma_{g_1,\ldots,g_m;agg}(W) & \text{otherwise} \end{cases}$$

use G instead of W in step 3.

Optimization Phases

Textbook query optimization steps:

1. translate the query into its canonical algebraic expression
2. perform logical query optimization
3. perform physical query optimization

we have already seen the translation, from now one assume that the algebraic expression is given.

Concept of Logical Query Optimization

- foundation: algebraic equivalences
- algebraic equivalences span the potential search space
- given an initial algebraic expression: apply algebraic equivalences to derive new (equivalent) algebraic expressions
- note: algebraic equivalences do not indicate a direction, they can be applied in both ways
- the conditions attached to the equivalences have to be checked

Algebraic equivalences are essential:

- new equivalences increase the potential search space
- better plans
- but search more expensive

Performing Logical Query Optimization

Which plans are better?

- plans can only be compared if there is a *cost function*
- cost functions need details that are not available when only considering logical algebra
- consequence: logical query optimization remains a heuristic

Performing Logical Query Optimization

Most algorithms for logical query optimization use the following strategies:

- organization of equivalences into groups
- directing equivalences

Directing means specifying a preferred side.
A *directed equivalences* is called a *rewrite rule*. The groups of rewrite rules are applied sequentially to the initial algebraic expression. Rough goal:

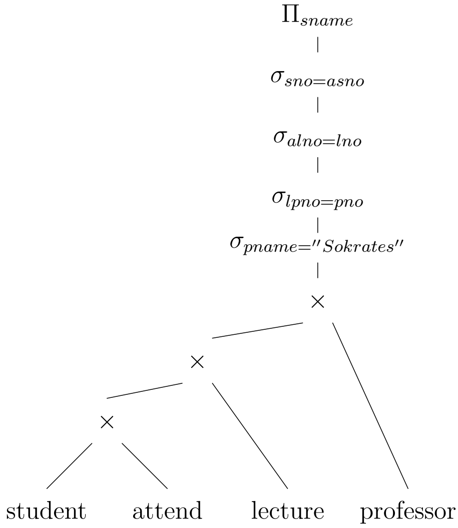
reduce the size of intermediate results

Phases of Logical Query Optimization

1. break up conjunctive selection predicates
   (equivalence (1) →)
2. push selections down
   (equivalence (2) →, (14) →)
3. introduce joins
   (equivalence (13) →)
4. determine join order
   (equivalence (9), (10), (11), (12))
5. introduce and push down projections
   (equivalence (3) ←, (4) ←, (16) →)

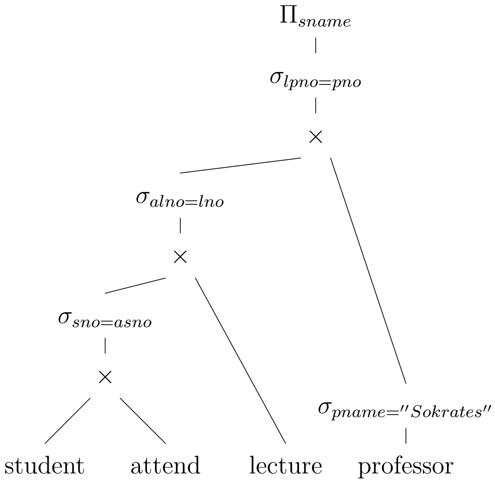
Step 1: Break up conjunctive selection predicates

- selection with simple predicates can be moved around easier



Step 2: Push Selections Down

- reduce the number of tuples early, reduces the work for later operators



Π_{sname}

$\sigma_{lpno=pno}$

$\times$

$\sigma_{alno=lno}$

$\times$

$\sigma_{sno=asno}$

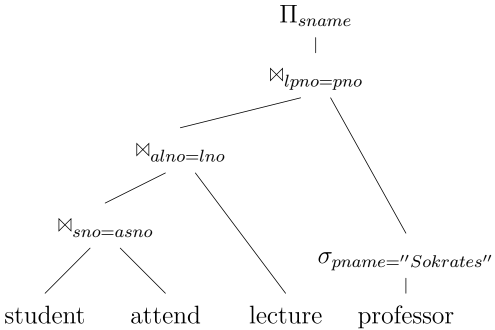
$\times$

$\sigma_{pname=''Sokrates''}$

student attend lecture professor

Step 3: Introduce Joins

- joins are cheaper than cross products



Step 4: Determine Join Order

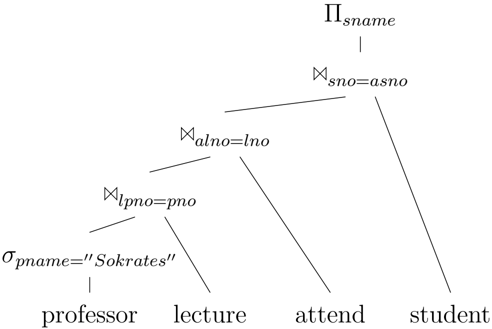
- costs differ vastly
- difficult problem, NP hard (next chapter discusses only join ordering)

Observations in the sample plan:

- bottom most expression is $student \bowtie_{sno=asno} attend$
- the result is huge, all students, all their lectures
- in the result only one professor relevant $\sigma_{name=''Sokrates''}(professor)$
- join this with lecture first, only lectures by him, much smaller

Step 4: Determine Join Order

- intermediate results much smaller



Π_{sname}

$\bowtie_{sno=asno}$

$\bowtie_{alno=lno}$

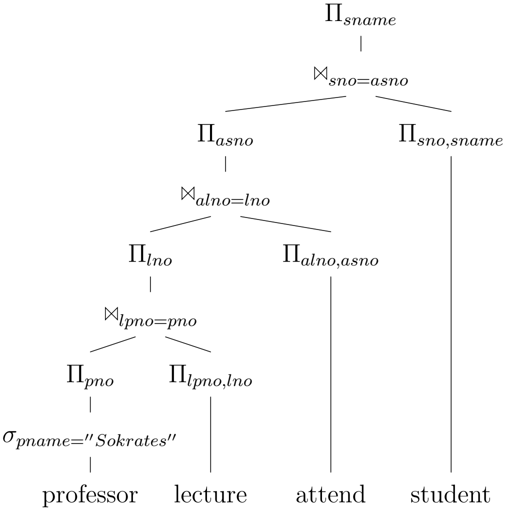
$\bowtie_{lpno=pno}$

$\sigma_{pname=''Sokrates''}$

professor lecture attend student

Step 5: Introduce and Push Down Projections

- eliminate redundant attributes
- only before pipeline breakers

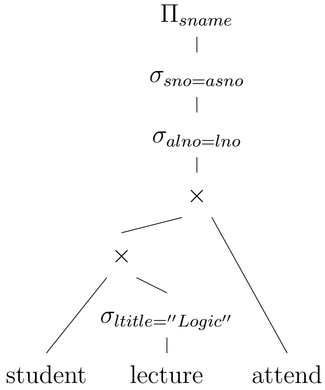


Limitations

Consider the following SQL query

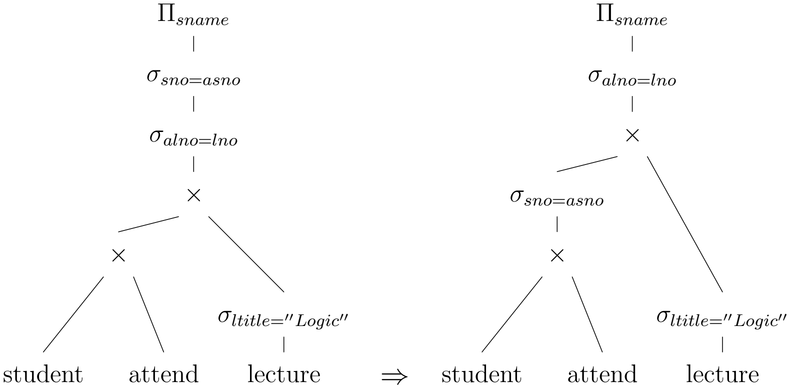
select distinct *s.sname*
from *student s, lecture l, attend a*
where $s.sno = a.asno$ **and** $a.alno = l.lno$ **and** $l.ltitle =''Logic''$

Steps 1-2 could result in plan below. No further selection push down.



Limitations

However a different join order would allow further push down:



- the phases are interdependent
- the separation can loose the optimal solution

Physical Query Optimization

- add more execution information to the plan
- allow for cost calculations
- select index structures/access paths
- choose operator implementations
- add property enforcer
- choose when to materialize (temp/DAGs)

Access Paths Selection

- scan+selection could be done by an index lookup
- multiple indices to choose from
- table scan might be the best, even if an index is available
- depends on selectivity, rule of thumb: 10%
- detailed statistics and costs required
- related problem: materialized views
- even more complex, as more than one operator could be substitued

Operator Selection

- replace a logical operator (e.g. $\bowtie$) with a physical one (e.g. $\bowtie^{HH}$)
- semantic restrictions: e.g. most join operators require equi-conditions
- $\bowtie^{BNL}$ is better than $\bowtie^{NL}$
- $\bowtie^{SM}$ and $\bowtie^{HH}$ are usually better than both
- $\bowtie^{HH}$ is often the best if not reusing sorts
- decission must be cost based
- even $\bowtie^{NL}$ can be optimal!
- not only joins, has to be done for all operators

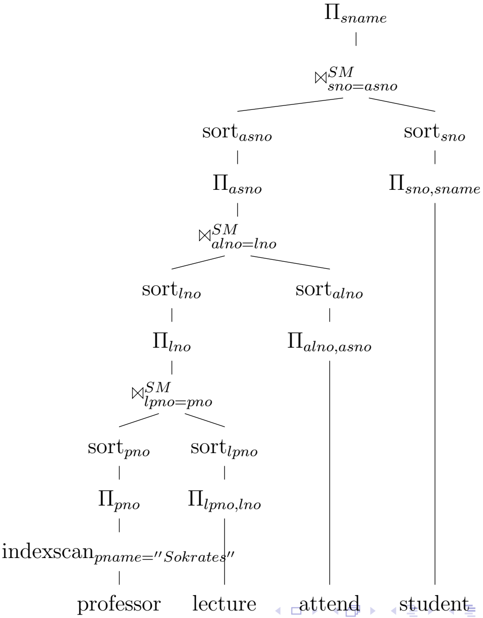
Property Enforcer

- certain physical operators need certain properties
- typical example: sort for $\bowtie^{SM}$
- other example: in a distributed database operators need the data locally to operate
- many operator requirements can be modeled as properties (hashing etc.)
- have to be guaranteed as needed

Materializing

- sometimes materializing is a good idea
- temp operator stores input on disk
- essential for multiple consumers (factorization, DAGs)
- also relevant for $\bowtie^{NL}$
- first pass expensive, further passes cheap

Physical Plan for Sample Query



Π_{sname}

$\bowtie^{SM}_{sno=asno}$

sort_{asno} sort_{sno}

Π_{asno} $\Pi_{sno,sname}$

$\bowtie^{SM}_{alno=lno}$

sort_{lno} sort_{alno}

Π_{lno} $\Pi_{alno,asno}$

$\bowtie^{SM}_{lpno=pno}$

sort_{pno} sort_{lpno}

Π_{pno} $\Pi_{lpno,lno}$

$\text{indexscan}_{pname=''Sokrates''}$

professor lecture attend student

Outlook

- separation in two phases looses optimality
- many decissions (e.g. view resolution) important for logical optimization
- textbook physical optimization is incomplete
- did not discuss cost calculations
- will look at this again in later chapters