Query optimization part 1
Algebraic rewriting

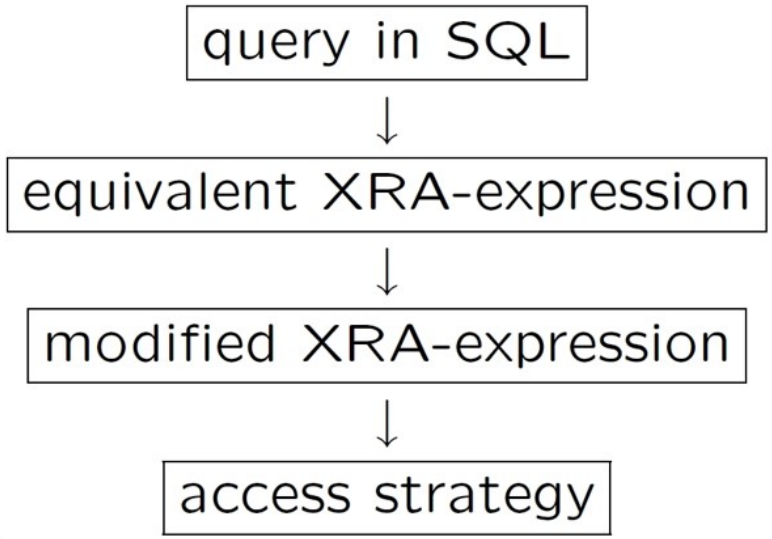
Hans Philippi

August 5, 2024

Query processing

From SQL query to result table ...
Let's have a look under the hood

Query processing



- XRA stands for eXtended Relational Algebra, dealing with multisets, sorting and aggregates
- We will not go into detail with respect to step 1
- In this lecture, we will focus on step 2
- We will deal with step 3 in another lecture

Bags aka multisets

- Set like collections allowing for duplicates
- Some examples ...

Union:

$$\{a, b, c\} \cup \{a, a, c, d\} = \{a, a, a, b, c, c, d\}$$

Intersection:

$$\{a, a, a, b, c, c, d\} \cap \{a, a, b, e\} = \{a, a, b\}$$

Minus:

$$\{a, a, a, b, c, c, d\} - \{a, a, b, e\} = \{a, c, c, d\}$$

Extended projection

Extended projection supports simple expressions as projected attributes

R	
A	B
1	4
2	5
3	6

π_{A+B}(R)	
A	AplusB
1	5
2	7
3	9

Ordering

Projection may involve ordering: τ

$\uparrow$ ascending (default)
$\downarrow$ descending

R	
A	B
1	3
2	6
4	3
5	3

$\tau_{\uparrow B \downarrow A}(\mathrm{R})$	
B	A
3	5
3	4
3	1
6	2

Grouping and aggregate functions

Symbol: Γ

Shopping		
Store	Product	Price
AH	Tomato polpa	2.25
Jumbo	Linguine	2.52
Aldi	Tomato polpa	1.29
AH	Mozzarella	1.95
Jumbo	Tomato polpa	2.25
AH	Linguine	2.79

$\Gamma_{Product,MIN(Price)}$(Shopping)	
Product	MinPrice
Tomato polpa	1.29
Linguine	2.52
Mozzarella	1.95

Query processing and algebraic optimization

Basic ideas

- Rationale: main-memory is limited; data traffic between main-memory and external memory should be minimized
- Unary operations (σ, π) are rather cheap
- Selection is even cheaper if a suitable index is present
- Binary operations, especially join-like operations, are expensive
- Apply unary operations as early as possible to reduce the size of the operands of binary operations
- Size: number of tuples as well as number of attributes count
- Many algebraic optimization rules are rules of thumb, but they also might be based on cost models (see following lectures)

Algebraic properties

In retrospect, some basic algebraic properties concerning real numbers ...

Commutativity

- $a + b = b + a$
- $a * b = b * a$
- $a - b \neq b - a$

Associativity

- $(a + b) + c = a + (b + c)$
- $(a * b) * c = a * (b * c)$
- $(a - b) - c \neq a - (b - c)$

Distributivity

- $a * (b + c) = a * b + a * c$
- $a * (b - c) = a * b - a * c$

Algebraic rewriting

Note that the unary operators σ and π generally can be calculated by a single table scan

Cascading and commuting selections

- $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R)) = \sigma_{p \wedge q}(R)$

Cascading projections

- $\pi_{L1}(\pi_{L2}(R)) = \pi_{L1}(R)$ with condition $L1 \subseteq L2$

Commuting selections and projections

- $\pi_L(\sigma_p(R)) = \sigma_p(\pi_L(R))$ with condition $attr(p) \subseteq L$

Commutativity and associativity of binary operators

- $R \cup S = S \cup R$
- $R \cap S = S \cap R$
- $R \bowtie S = S \bowtie R$

- $(R \cup S) \cup T = R \cup (S \cup T)$
- $(R \cap S) \cap T = R \cap (S \cap T)$
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Distributivity

The following rule plays a crucial role in query optimization, formalizing the notion of "applying unary operators early"

- $\sigma_{p_1 \wedge p_2 \wedge p_3}(R \bowtie S) = \sigma_{p_3}(\sigma_{p_1}(R) \bowtie \sigma_{p_2}(S))$

with conditions $attr(p_1) \subseteq attr(R), attr(p_2) \subseteq attr(S)$

Several other distributive rules are valid, for example

- $\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$

You will find more examples in the exercises

Associativity and join order

Given the associativity of the join, we have a lot of possibilities to calculate the result of long join chains

Take for example $R \bowtie S \bowtie T \bowtie U$, then we have ...

- $((R \bowtie S) \bowtie T) \bowtie U$
- $R \bowtie (S \bowtie (T \bowtie U))$
- $R \bowtie ((S \bowtie T) \bowtie U)$
- $(R \bowtie S) \bowtie (T \bowtie U)$
- $S \bowtie ((R \bowtie T) \bowtie U)$
- ...

...although a natural join degenerates to a cartesian product if the tables do not have any attributes in common

Associativity and join order

Given the associativity of the join, we have a lot of possibilities to calculate the result of long join chains $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$

- The number of possibilities grows exponentially with n
- Determining an "optimal" solution is possible for limited values of n using dynamic programming techniques
- For larger values of n, heuristics are used
- For the size of tables, we can adopt the product of the width in bytes and the cardinality (number of tuples)
- In another lecture, we will show how to estimate the size of a join result of two tables, given the size of the operands
- A common greedy heuristic determines the size of the join result of each pair of two tables and iteratively chooses the smallest one

Literature

This lecture is mainly based on chapter 16 from:

Database Systems, the Complete Book, 2nd edition
Garcia-Molina, Ullman & Widom