OPTIMIZATION OF RELATIONAL QUERIES

Dr. Gajdos Sándor – Dr. Erős Levente
November 2014 – November 2017
BME–TMIT
CONTENTS

- Overview
- Catalog cost estimation
- Optimization approaches
  - Cost based optimization
  - Heuristic optimization
OVERVIEW

query \rightarrow \text{interpreter and compiler} \rightarrow \text{relational algebra expression} \rightarrow \text{optimizer} \rightarrow \text{execution plan} \rightarrow \text{evaluator engine} \rightarrow \text{query output}

\text{data} \rightarrow \text{statistics about data}
STEPS

1. (Syntactical) analysis, compilation
   Does it make sense?
   Input: SQL statement
   Result: relational algebra expression

2. Cost optimization

3. Evaluation
CATALOG BASED COST ESTIMATION

- The following statistical data is stored in the so-called catalog
  - Catalog data about relations
  - Catalog data about indexes
  - Query cost
- Cost is estimated based on catalog data.
CATALOG DATA ABOUT RELATIONS

- $n_r$: number of records in relation $r$
- $b_r$: number of blocks storing the records of relation $r$
- $s_r$: size of a record
- $f_r$: how many records fit in a data block
CATALOG DATA ABOUT RELATIONS

- $V(A, r)$: how many different values attribute $A$ has in relation $r$ (cardinality).
  - $V(A, r) = |\pi_A(r)|$
  - If $A$ is a key, then $V(A, r) = n_r$

- $SC(A, r)$: (Selection Cardinality) average number of records that satisfy a selection condition.
  - If $A$ is a key, then $SC(A, r) = 1$
  - In general $SC(A, r) = \frac{n_r}{V(A,r)}$

- If the records of a relation are physically stored together, then:
  $$b_r = \left\lfloor \frac{n_r}{f_r} \right\rfloor$$
CATALOG DATA ABOUT INDEXES

- $f_i$: number of pointers going out of a node in case of a tree index, like B* tree
- $HT_i$: number of index levels (Height of Tree)
  - $HT_i = \left\lfloor \log_{f_i} V(A, r) \right\rfloor$ (B* tree)
  - $HT_i = 1$ (hash)
- $LB_i$: a number of leaf blocks (Lowest level index Block)
COST OF QUERY

Definition:

- Number of block reading and writing operations from the disc (without writing out the result).
- Takes the most time by far – good metric
  - By orders of magnitude more costly then performing operations in the memory, etc.
COST OF OPERATIONS – OUTLINE

- Selection
  - Selection algorithms (basic, indexed, comparison based)
  - complex selection

- Join
  - Types
  - Size estimation
  - Join algorithms

- Other
  - Filtering repetitions
  - Union, intersection, subtraction
BASIC SELECTION ALGORITHMS (=)

A1: Linear search
- Cost:
  \[ E_{A1} = b_r \]

A2: Binary search
- Requirements:
  - Blocks are located continuously on the disk
  - The file is ordered by attribute A
  - The selection condition is equality on attribute A
- Cost:
  \[ E_{A2} = \lceil \log_2 (b_r + 1) \rceil + \left\lceil \frac{SC(A, r)}{f_r} \right\rceil - 1 \]
INDEXED SEARCH ALGORITHMS

**Primary index** – requires the data file to be physically ordered by the index attribute. Everything else is a **secondary index**

**A3:** Using primary index, if the equality condition is defined on the key

- \[ E_{A3} = HT_i + 1 \]

**A4:** Using primary index, if the equality condition is defined on a non-key attribute (the primary index is on the non-key attribute)

- \[ E_{A4} = HT_i + \left[ \frac{SC(A,r)}{f_r} \right] \]

**A5:** Using secondary index.

- \[ E_{A5} = HT_i + SC(A, r) \]
- \[ E_{A5} = HT_i + 1, \text{ if } A \text{ is a key} \]
COMPARISON BASED SELECTION – $\sigma_{A \leq v(R)}$

Estimation of the number of result records:

- If $v$ is unknown: $\frac{n_r}{2}$

- If $v$ is known, and the distribution is uniform:

  $$n_{\text{average}} = n_r \cdot \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$
COMPARISON BASED SELECTION – $\sigma_{A \leq v}(R)$

**A6:** With primary index.
- If $v$-t is unknown:
  
  $$E_{A6} = HT_i + \frac{b_r}{2}$$

- If $v$ is known:

  $$E_{A6} = HT_i + \left\lfloor \frac{c}{f_r} \right\rfloor,$$

  Where $c$ is the number of records for which $A \leq v$

**A7:** With secondary index

$$E_{A7} = HT_i + \frac{LB_i}{2} + \frac{n_r}{2}$$
JOIN OPERATION

Definition:

\[ r_1 \bowtie_\theta r_2 = \sigma_\theta (r_1 \times r_2) \]

Types:

- **Natural join**
  \[ r_1 \bowtie r_2 = \pi_{A \cup B} (\sigma_{R_1.X = R_2.X} (r_1 \times r_2)) \]

- **Outer join**
  - Left outer join: \( r_1 \bowtie (+) r_2 \)
  - Right outer join: \( r_1 (+) \bowtie r_2 \)
  - Full outer join: \( r_1 (+) \bowtie (+) r_2 \)

- **Theta join**:
  \[ r_1 \bowtie_\theta r_2 = \sigma_\theta (r_1 \times r_2) \]
NESTED-LOOP JOIN

Given are two relations, \( r \) and \( s \):

FOR each record \( t_r \in r \) DO BEGIN
    FOR each record \( t_s \in s \) DO BEGIN
        test if pair \( (t_r, t_s) \), fulfills join condition \( \theta \)
        IF yes, THEN add record \( t_r . t_s \) to the result
    END
END

- „worst case” cost: \( n_r \cdot b_s + b_r \)
- If at least one relation fits in the memory, then its cost is: \( b_r + b_s \)
BLOCK NESTED-LOOP JOIN

FOR each block $b_r \in r$ DO BEGIN
    FOR each block $b_s \in s$ DO BEGIN
        FOR each record $t_r \in b_r$ DO BEGIN
            FOR each record $t_s \in b_s$ DO BEGIN
                test pair $(t_r, t_s)$
            END
        END
    END
END
END

- „worst-case” költsége: $b_r \cdot b_s + b_r$
- with a lot of memory: $b_r + b_s$
INDEXED NESTED-LOOP JOIN

For one of the relations \( s \) we have an index

Let us put the indexed relation to the inner cycle of the first algorithm

\[ \Rightarrow \text{Using the index, the search can be performed at a lower cost} \]

Cost:
\[ b_r + n_r \cdot c, \]
where \( c \) is the cost of selection on \( s \).
FURTHER JOIN IMPLEMENTATIONS

- sorted merge join
  - order relations by the attributes provided in the join condition

- hash join
  - one of the relations is accessed through a hash table when looking for its records matching the records of the other relation
FURTHER OPERATIONS

- **Filtering repetitions** (ordering, and then deleting)
- **Projection** (projection, the filtering repetitions)
- **Union** (ordering of both relations, and filtering duplications during merge)
- **Intersection** (ordering both relations, then leaving only the repetitions during merge)
- **Subtraction** (ordering both relations, and then during merge, we only leave those elements in the result set, which are only present in the first relation)
METHODS FOR EVALUATING EXPRESSIONS

- **Materialization**
  - Evaluating a single operation of a complex expression at a time

- **Pipelining**
  - Multiple operations are evaluated in parallel
  - The result of an operation is immediately transferred to the input of the next operation
MATERIALIZATION

- Canonical format:
  \[ \pi_{\text{customer\_name}}(\sigma_{\text{balance} < 2500}(account) \bowtie \text{customer}) \]

- Query tree:

- Final cost: cost of operations + cost of storing sub-results
- Advantage: Easy implementation
- Drawback: Many storage operations (block operations)
PIPELINING

- Parallel evaluation
- Operations create sub-results for the following operation, based on the sub-results of the preceding operation
- Does not calculate the whole relation (sub-result) at a time

Advantage:
- No temporary storage needed
- Low memory requirement

Drawback:
- Narrows down the algorithms that can be used
CHOOSING THE EXECUTION PLAN

Many execution plans are possible for the same result
Questions to be answered:
• Which operations?
• In what order?
• By which algorithm?
• By which workflow?

One exact execution plan:
COST-BASED OPTIMIZATION

Greedy and wrong strategy:
- Listing all possible equivalent execution plans
- Evaluating each plan
- Choosing the optimal one

Example: In case of expression \( r_1 \bowtie r_2 \bowtie r_3 \rightarrow 12 \) equivalent expressions

In more general: to join \( n \) relations, \( \frac{(2(n-1))!}{(n-1)!} \) equivalent expressions exist.

This would mean too much load for the system.

Solution: Heuristic cost-based optimization
I. HEURISTIC, RULE BASED OPTIMIZATION

- Manipulating the query tree
- Example:

EMPLOYEE (EMPLOYEE_ID, LAST_NAME, FIRST_NAME, BIRTH_DATE, ...)
PROJECT (PROJECT_ID, PNAME, ...)
WORKS_ON (PROJECT_ID, EMPLOYEE_ID)

select last_name  
 from employee, works_on, project  
where employee.birth_date > '1957.12.31'  
 and works_on.project_id = project.project_id  
 and works_on.employee_id = employee.employee_id  
 and project.pname = 'Aquarius'

2014. nov.
A POSSIBLE RELATIONAL ALGEBRA EXPRESSION EQUIVALENT

\[ \pi_{\text{LAST\_NAME}} \left( \left( \sigma_{\text{BIRTH\_DATE}>"1957.12.31"}(\text{EMPLOYEE}) \right) \bowtie \text{EMPLOYEE\_ID}=\text{EMPLOYEE.EMPLOYEE\_ID} (\text{WORKS\_ON}) \bowtie \text{PROJECT\_ID}=\text{PROJECT.PROJECT\_ID} \sigma_{\text{PNAME}="Aquarius"}(\text{PROJECT}) \right) \]
GOAL: TO CHOOSE THE QUICKEST EQUIVALENT

Starting from: canonical format (Descartes, selection, projection)

\[ \sigma_{PNAME = "Aquarius" \land PROJECT.ID = PROJECT.PROJECT_ID \land EMPLOYEE.ID = EMPLOYEE.EMPLOYEE_ID \land BIRTH.DATE > "1957.12.31"} \]

\[ \pi_{LAST\_NAME} \]
STEP 2: SINKING SELECTIONS

\[
\pi_{\text{LAST\_NAME}} \\
\sigma_{\text{PROJECT\_ID}=\text{PROJECT\_PROJECT\_ID}} \\
\times \\
\sigma_{\text{EMPLOYEE\_ID}=\text{EMPLOYEE\_EMPLOYEE\_ID}} \\
\times \\
\sigma_{\text{BIRTH\_DATE}>"1957.12.31"} \\
\times \\
\sigma_{\text{PNAME}="Aquarius"} \\
\times \\
\text{PROJECT} \\
\times \\
\text{WORKS\_ON} \\
\times \\
\text{EMPLOYEE}
\]
STEP 3: REARRANGING LEAVES

\[ \pi_{\text{last.name}} \]

\[ \sigma_{\text{employee_id}=\text{employee.employee_id}} \]

\[ \times \]

\[ \sigma_{\text{project_id}=\text{project.project_id}} \]
\[ \sigma_{\text{birth_date}>"1957.12.31"} \]

\[ \times \]

\[ \sigma_{\text{pname}="\text{Aquarius"}} \]

\[ \text{works_on} \]

\[ \text{project} \]

\[ \text{employee} \]
STEP 4: JOIN

\[ \pi_{\text{LAST NAME}} \]

\[ \bowtie \text{EMPLOYEE_ID}=\text{EMPLOYEE.EMPLOYEE_ID} \]

\[ \bowtie \text{PROJECT_ID}=\text{PROJECT.PROJECT_ID} \]

\[ \sigma_{\text{PNAME}="\text{Aquarius"}} \]

\[ \sigma_{\text{BIRTH DATE} > "1957.12.31"} \]

\[ \text{WORKS_ON} \]

\[ \text{EMPLOYEE} \]

\[ \text{PROJECT} \]
STEP 5: SINKING PROJECTIONS
WHEN ARE TWO TREES EQUIVALENT?
RELATIONAL ALGEBRA TRANSFORMATIONS I.

- \( \sigma_{c_1 \land c_2 \land \ldots \land c_n}(r) \equiv \sigma_{c_1} \left( \sigma_{c_2} \left( \ldots \left( \sigma_{c_n}(r) \right) \ldots \right) \right) \)
- \( \sigma_{c_1} \left( \sigma_{c_2}(r) \right) \equiv \sigma_{c_2} \left( \sigma_{c_1}(r) \right) \)
- \( \pi_{List_1} \left( \pi_{List_2} \left( \ldots \left( \pi_{List_n}(r) \right) \ldots \right) \right) \equiv \pi_{List_1}(r) \)
- \( \pi_{A_1,A_2,\ldots,A_n} \left( \sigma_{c}(r) \right) \equiv \sigma_{c} \left( \pi_{A_1,A_2,\ldots,A_n}(r) \right) \)
WHEN ARE TWO TREES EQUIVALENT?

RELATIONAL ALGEBRA TRANSFORMATIONS II.

- \( r \bowtie_c s \equiv s \bowtie_c r \)
- \( \sigma_c (r \bowtie s) \equiv (\sigma_c (r)) \bowtie s \)
- \( \pi_L (r \bowtie_c s) \equiv (\pi_{A_1,...,A_n} (r)) \bowtie_c (\pi_{B_1,...,B_m} (s)) \)
- \( \pi_L (r \bowtie_c s) \equiv \\
  \pi_L \left( (\pi_{A_1,...,A_n,A_{n+1},...,A_{n+k}} (r)) \bowtie_c (\pi_{B_1,...,B_m,B_{m+1},...,B_{m+p}} (s)) \right) \)

Set operations (union, intersection) are commutative.

Join, Cartesian product, union, and intersection are associative:

\[(r \theta s) \theta t \equiv r \theta (s \theta t)\]
WHEN ARE TWO TREES EQUIVALENT?
RELATIONAL ALGEBRA TRANSFORMATIONS III.

\[ \sigma_C (r \theta s) \equiv (\sigma_C (r)) \theta (\sigma_C (s)) \]
\[ \pi_L (r \theta s) \equiv (\pi_L (r)) \theta (\pi_L (s)) \]

Further rules:
\[ c \equiv \neg (c_1 \land c_2) \equiv (\neg c_1) \lor (\neg c_2) \]
\[ c \equiv \neg (c_1 \lor c_2) \equiv (\neg c_1) \land (\neg c_2) \]
RULES, SUMMARY

- Conjunctions in selections are transformed to a series of selections.
- Selections are swapped with the other operations.
- Query tree leaves are re-arranged.
- Cartesian products and the selection (join) condition above them are transformed to a single operation (theta join)
- Projections are swapped with the other operations