10. Query Processing

10.1 Introduction

- descriptive formulation, based on the conceptual schema

- hence support for the idea of data independence: query of the user is not dependent on the physical schema (i.e., from the storage structure)

- problem: translation of the descriptive formulation into an efficient, executable access plan

- Usually a (trivial) evaluation algorithm can already be associated to the descriptive formulation of a query (e.g. in relational algebra, relational calculus, SQL, etc.).
example: query formulation is expression of the relational algebra
→ execute the operations in the given order and create intermediate result relations (temporary relations) after each operation, but: evaluation is usually inefficient

The goal is therefore to find an improvement of the trivial evaluation method. This process is known as the optimization of a relational query. It is important to understand that the result is in no way optimum.

Unfortunately, it is impossible to find the fastest of all execution alternatives in an efficient way. The strategy is to use some kind of “try and error” methods which create alternatives in a more or less targeted manner and which estimate their execution time (or costs) with the aid of a cost model.

measures of query cost
– disk accesses: number of transfers of blocks from and to disk
– CPU time to execute a query
– cost of communication in parallel and distributed DBMS

Most systems and optimization algorithms only consider disk accesses, because they are very slow compared to in-memory operations.

Optimization is necessary and especially worthwhile, because the costs for a bad (trivial) and good access plan can differ enormously.
10.2 Phases of translation/optimization

- goal: syntactical and semantical analysis of the query
- given: query in a relational query language, e.g. SQL
- step 1: translation of the query into an equivalent, **internal representation**, where views are replaced by the defining query
- requirements/properties of the internal representation:
  - abstract, without specialties of the concrete syntax
  - near to a DB language
  - sufficiently expressive to be able to represent all queries
  - neutral as much as possible, no decisions for optimization in advance
- The internal representation is mostly an equivalent expression of the relational algebra that is usually represented as an **operator tree**.
- example:
  - algebra expression \( \pi_A(\sigma_{D=99}(\sigma_{B=C}(R \times S))) \)
  - operator tree
- step 2: transformation of the internal representation into a **standard form**
  - task of the **algebraic/logical optimization**
  - use of a general **heuristics** for the transformation of the internal representation into a better (i.e., more efficient) one with maintenance of equivalence
  - **Heuristics** represent rules of thumb about the meaningful application of specific transformation rules and serve for the control of alternatives for generating access plans.
  - foundation: laws about equivalence transformations of relational algebra expressions
  - example of a heuristics: “selections as soon as possible“
− no consideration of the internal representation of relations, index structures, etc. so far

- **step 3:** generation of access plans from the standard form
  - task of the **physical optimization**
  - For a descriptive operator frequently several different implementations (i.e., translation alternatives) into an **executable operator** are possible.
  - An algorithm for the implementation of a descriptive operator can be considered as an operator of a **physical algebra**.
  - assignment of the possible executable operations to the components of the internal representation (the nodes of the operator tree)
  - in general many alternatives possible
  - consideration of relation representations, index structures etc.

- **step 4:** cost analysis for choosing the “most favorable” access plan
  - task of **cost estimation**
  - The **query optimizer** tries to determine a most possible efficient access plan for the query with the aid of **cost models**.
  - A cost model works on the basis of schema information, the knowledge about costs (run time and space complexity) of employed algorithms and statistics about relations, index structures, and the distribution of attribute values.
– Often steps 3 and 4 are executed in an interlaced manner.

- **step 5**: creation of **program code** for realizing the access plan
  - Access plan can either be compiled and stored in the database or interpreted directly for interactive (“ad hoc”) queries.

- **summary:**

```
scaner
parser
view resolution
```

```
descriptive query
```

```
algebraic expression
```

```
query
optimizer
```

```
access plan
```

```
code generation
execution
```
10.3 Algebraic Optimization/Logical Optimization

Introduction

- task: equivalent transformation of expressions of the relational algebra
- goal: reduction of intermediate results with respect to number and size of tuples
- foundations
  - optimization on the basis of the algebraic normal form: An SQL query of the general form `select distinct A_1, A_2, ..., A_n from R_1, R_2, ..., R_m where F` is transformed into the algebraic expression \( \pi_{A_1, A_2, ..., A_n}(\sigma_F(R_1 \times R_2 \times ... \times R_m)) \).
The leaves of the operator tree represent basis relations, the inner nodes the operators of the relational algebra.

- laws of the relational algebra

- used heuristics like e.g.
  - Selections should be executed as soon as possible.
    ⇒ Shift selections in the expression as much as possible to the interior of the expression.
  - Shift projections to the interior of the expression, but not before selections on base relations.
  - Change the execution order, possibly also the order of binary operators, if this is useful for the shift of selections and projections.

- example:
  - SQL query
    ```sql
    select title
    from professors, lectures
    where name = "Sokrates" and pers-id = held_by
    ```
  - relational algebra expressions
    \[
    \pi_{title}(\sigma_{name="Sokrates" \land pers-id=held_by}(professors \times lectures))
    \]
  - relation professors 7 tuples, relation lectures 10 tuples
expensive evaluation: due to cross product access to \( 7 \cdot 10 = 70 \) tuples, one tuple satisfies selection condition

simple improvement: first find “right” professor, then form the cross product

\[
p_{\text{title}}(\sigma_{\text{pers-id}=\text{held_by}(\sigma_{\text{name}=\text{Sokrates}}(\text{professors}) \times \text{lectures}))}
\]

selection on relation professors (7 tuples), result 1 tuple, cross product with relation lectures (10 tuples), in total access to 17 tuples necessary

Equivalences in the relational algebra

- The following laws relate to the view of a tuple as a set of mappings from attributes to domains, i.e., the order of the tuple components is irrelevant.

- Let \( R_1, R_2, \ldots \) be relations (base relations or derived relations, i.e., intermediate results). Let \( F, F_1, F_2 \) be conditions, \( \text{attr} \) be the mapping of conditions to the set of attributes contained in them (e.g. \( \text{attr}(\text{name} = \text{“Sokrates”}) = \{\text{name}\} \)).

- **rule 1:** commutativity of join, union, intersection and cross product

\[
\forall \omega \in \{\bowtie, \cup, \cap, \times\} : R_1 \omega R_2 = R_2 \omega R_1
\]

- **rule 2:** associativity of join, union, intersection and cross product

\[
\forall \omega \in \{\bowtie, \cup, \cap, \times\} : R_1 \omega (R_2 \omega R_3) = (R_1 \omega R_2) \omega R_3
\]
Commutativity and associativity are especially important after the algebraic optimization phase, namely for the creation of access plans. This allows the query optimizer to choose a favorable execution order from many possible ones.

- **Rule 3**: Permutability of selections
  \[
  \sigma_{F_1}(\sigma_{F_2}(R)) = \sigma_{F_2}(\sigma_{F_1}(R))
  \]

- **Rule 4**: Cascade of selections
  \[
  \sigma_{F_1 \land F_2 \land \ldots \land F_n}(R) = \sigma_{F_1}(\sigma_{F_2}(\ldots(\sigma_{F_n}(R))\ldots))
  \]

- **Rule 5**: Cascade of projections
  \[
  \pi_{A_1}(\pi_{A_2}(\ldots(\pi_{A_n}(R))\ldots)) = \pi_{A_1}(R) \quad (A_1 \subseteq A_2 \subseteq \ldots \subseteq A_n \subseteq R)
  \]

- **Rule 6**: Permutation of a projection with a selection to the interior of an expression, if the projection does not remove attributes from the selection condition
  \[
  \pi_A(\sigma_F(R)) = \sigma_F(\pi_A(R)), \text{ if } attr(F) \subseteq A
  \]

- **Rule 7**: Permutation of a projection with a selection to the interior of an expression
  \[
  \pi_A(\sigma_F(R)) = \pi_A(\sigma_F(\pi_A \cup B(R))), \text{ if } B = attr(F) - A
  \]