Query Optimization

Ilaria Battiston *

Winter Semester 2020-2021

*All notes are collected with the aid of material provided by T. Neumann. All images have been retrieved by slides present on the TUM Course Webpage.

Contents

1 Introduction

Queries involve multiple steps to be performed. All techniques obviously depend on the physical characteristic of hardware, however there are algorithms and data structures able to help making them faster.

Query optimization is specifically important since SQL-like languages are declarative, hence do not specify the exact computation. Furthermore, the same query can be performed in multiple ways, and choosing the correct one is not trivial (depending on size, indices and more).

For instance, looking at cardinality of joins can either cause a Cartesian product of a hundred or several thousands, depending on the order in which they are executed. Often the human-written sequential order of instructions is not the most efficient one.

1.1 Query Processing

Query processing consists in:

1. Taking a text query as input;
2. Compiling and optimizing;
3. Extract the execution plan;
4. Executing the query.

Roughly, it can ve differentiated between compile time and runtime system. Most systems strongly separate those phases, for example using so-called prepared queries.

```
SELECT S.NAME
FROM STUDENTS S
WHERE S.ID = ?
```

Compilation takes time and limits the number of queries which can be ran in parallel. This kind of query can be executed over and over providing a value `Q1(123)` without the compilation overhead.

Embedded SQL is another technique to optimize compilation time in the case of large periods of time between the two phases. The programming language compiler takes care of the SQL part as well, optimizing it.

Specifically, the steps executed in compile time are:

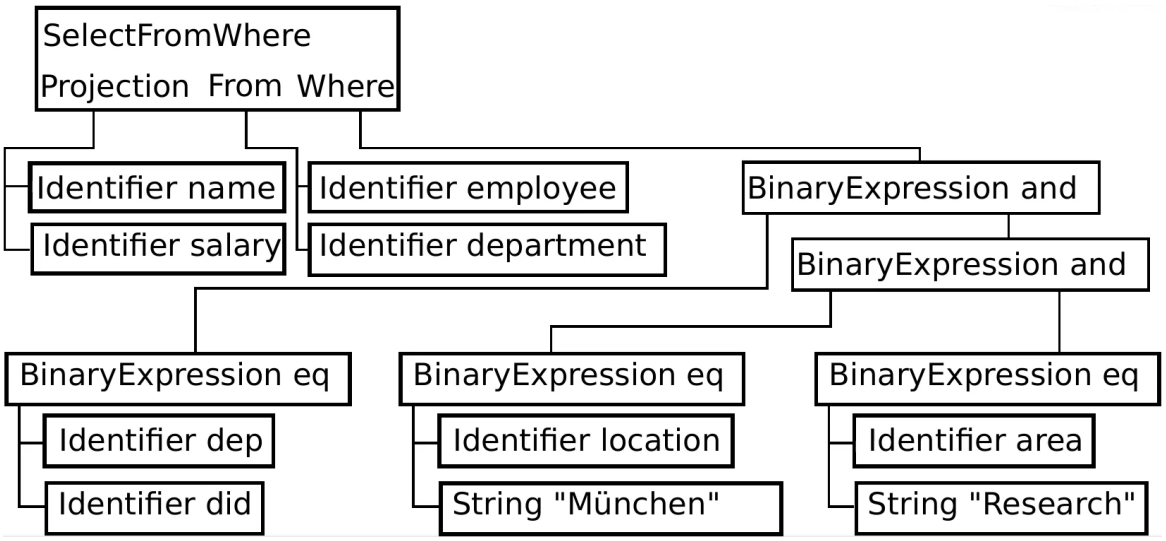
1. Parsing, AST production (abstract syntax to understand the structure);
2. Schema lookup, variable binding, type inference (semantic analysis about relations and columns, syntax check);
3. Normalization, factorization (bringing the query in abstract form, avoiding computing the same thing twice, evaluating expressions);
4. Unnesting, deriving predicates, resolution of views (the plan generator can finally construct a cost-based model);
5. Construction of execution plan;
6. Review, pushing joins and refining the plan in general;

7. Production of imperative plan (code generation).

Rewrite I involves steps 1-3, while 4 and 5 compose rewrite II.

Example (with views):

```
SELECT name, salary
FROM employee, department
WHERE dep = did
AND location = "Munich"
AND area = "Research"
```



Finally, the execution tree is built and polished, with the join operation on top and selects on nodes, reducing the amount of tuples to be joined. In other cases, such as regular expressions which are hard to evaluate, filtering can be done later.

The executable plan is a set of tuples containing variables or constants along and their type, with the main function loading query parameters, operations and allocated resources.

Usually query planners are much more complicated and have practical difficulties: for instance, a long list of AND/OR predicates (machine-generated in the order of hundreds of thousands) can make the binary tree recursion crash due to insufficient space in the stack.

1.2 Query Optimization

Possible goals of query optimization include minimizing response time, resource consumption, time to first tuple (producing the first tuple as quick as possible, for instance with search results) or maximizing throughput. This can be expressed as a cost function: most systems aim to minimize response time, having resources as constraints.

Algebraic optimization is a branch using relational algebra to find the cheapest expression equivalent to the original. However, finding the cheapest is a practically impossible problem: it is hard to test

for equivalence (numerical overflow, undecidable), the set of expressions is potentially huge and some algorithms are NP-hard (actual search space is limited and smaller than the potential one).

There are ways to transform numerical expressions in algebraic ones, yet they might be expensive: calculus is faster to evaluate than algebra.

Optimization approaches can be:

- Transformative, taking an algebraic expression and iteratively making small changes, not efficient in practice;
- Constructing, starting from small expressions and joining them, obtaining larger sets, usually the preferred approach.

1.3 Query Execution

Query execution is the last step, the one directly benefiting from optimization. In reality, operators can perform extremely specialized operations, treating data as bags (sets with duplicates) or streams.

1.3.1 Relational and physical algebra

Relational algebra for query optimization includes the operators of projection, selection and join, plus some additional ones such as mapping and grouping.

However, this does not imply an implementation, which can have a great impact, hence less abstract operators are needed due to stream nature of data (and not sets):

- Sort, according to criteria;
- Temp, which materializes the input stream making further reads cheaper and releasing memory afterwards;
- Ship, sends the input stream to a different host (for distributed computing).

Furthermore, there are different kind of joins having different characteristics:

- Nested Loop Join, the textbook implementation which is very slow (quadratic) but supports all predicates;
- Blockwise Nested Loop Join, reading chunks of the first column in memory and then the second one once (requires memory, can be improved using hashing). The left side is read $\frac{|L|}{|M|}$ times, which asymptotically is the same as above but in practice much faster;
- Sort Merge Join, scanning each column only once, but requiring sorted input ($O(n \log n)$ plus linear runtime assuming no duplicates on at least one side);
- Hybrid Hash Join, partitioning columns and joining them in memory (linear time, but working only for equi-joins).

Hash Join is usually the fastest, but with very large amount of data sorting might be more efficient. Nested Loop Join works well when one of the sides has only one tuple.

Other operators also have different implementations, such as aggregation sorting and hashing, whose speed depends on the algorithm (quick sort, heap sort etc.) and can rely or not on memory.

Physical algebra is hence the output of the query optimizer, since it fixes the exact execution runtime and does not stop to the general approach like relational algebra. It mainly helps to select which operator to use before running the query, so that optimization leads to more efficient plans.

2 Textbook Query Optimization

Textbook query optimization involves techniques to perform a rough first optimization, which however is quite simple. There is a series of steps to translate raw SQL into logical and physical plans, each of them transforming input in a more optimal form.

The output is going to be executable, but still to be improved by non-trivial methods.

2.1 Algebra and tuples

Plain relational algebra is not sufficient itself: it needs to be revisited ensuring correctness (producing the same result) within a formal model. The most relevant problem to tackle is deciding whether two algebraic expressions are the same, but this in difficult in practice.

For instance, performing a selection before a join might be correct (and faster) in case the considered criterion is equality, but can give a different result than selecting after an outer join.

To remedy this issue, it is possible to guarantee that two expressions are equivalent, not accepting false positing yet allowing false negatives.

A formal definition of tuple is an unordered mapping from attribute names to values of a domain. A schema consists in a set of attributes with domain $A(t)$.

Tuple operations are:

- Concatenation, attaching one tuple to another regardless of ordering (union);
- Projection, producing a notation $t.a$ in which it is possible to access single values or multiple $t_{|\{a,b\}}$, getting a subset of the schema.

A set of tuples with the same schema forms a relation. Sets naturally do not comply with real data, since they not allow duplicates, but are used for simplicity.

In most cases, sets and bags can be used interchangeably, but the optimizer considers different semantics: logical algebra operates on bags, physical algebra on streams and sets are only considered after an explicit duplicate elimination.

Set operations are the classic ones of union, intersection and difference, yet are subject to schema constraints. On bags, operations are performed on frequencies.

There are also free variables, which first must be bounded to be evaluated: they are essentials for predicates and algebra expressions, such as dependent joins.

It is important to note that projection removes duplicates within sets, while keeping them in bags.

There are equivalences for selection and projection useful to derive whether a different ordering produces the same output. For instance, applying selection twice is the same as applying it once with two criteria plus an AND. Commutative property also holds.