Summary of Strategies for Query Processing

By S. Xiao Fernández Marín

1 Summary

It is discussed how a query is internally used by a DBMS. It consists of **scanning** for identifying the query tokens, **parsing** to check the query syntax of the language and **validating** to scan if all the attributes and relations names are valid.

Then, a **query tree** (representation of the query) is created. It is also typically a query graph, but after creating one of those, the DBMS do an **execution strategy** and then decide which one fits better by the process of **query optimization**.

From the **execution plan** given by the query optimizer, the **code generator** generates the code and then the **runtime DB processor** runs the code or stores it to execute it whenever it is needed.

SQL queries are normally translated to relational algebra, represented as a query tree data structure and being optimized.

For example, a query block that contains SELECT-FROM-WHERE, would be transformed in $\pi_{select}(\sigma_{where}(from))$.

Some common queries operators are the **JOIN**, where there are semi-joins (EXISTS, IN, ANY) and anti-joins (NOT EXISTS, NOT IN, ALL). The **SELECT**, the search operation, where the algorithms can be linear, binary, using indexes, hash keys, B+tree or a bitmap. If the SELECT is connected with an AND, it is called **conjunctive condition**, and the methods applied are using indexes or pointers. The same happens with the **disjunctive condition**, the SELECT connected with an OR.

To minimize the cost of the query, the system retrieves some information of the useful DB, like the number of rows or the width of a table. This proved to be a problem when using the **JOIN** operation, as it is one of the most time-consuming operations. The methods used are the nested-loop, index-based nested-loop, sort-merge and partition-hash.

The buffer space is a part to take into account the performance of the query, as the algorithm reads one block at a time. One for the inner loop, so the outer-loop blocks in the main memory and an extra buffer is needed to contain the result records.

The fraction of records that will be joined in another file with more records, depends on the equijoin condition, also affecting the performance of the join. The same happens with the hash-join, Where if the partition of both files must be stored on disk, methods become more complex. there are two main techniques *Partition-hash join* and *Hybrid hash-join algorithm.* Being the latter is a variation of the former.

A query when translated into an algebra expression, is very time-consuming if you generate a lot because you have to store the temporary files on the disk. That is why we can combine several operations into one, to have fewer temporary files. This is called **pipelining**

For systems with parallel DB architecture exist different algorithms designed specifically for it. **Sorting:** Each partition is sorted separated in parallel, **Selection:** Selection done in parallel, **Projection and duplicate elimination:** Operations in parallel as data is read from each partition, **Join:** Various techniques: *Equality-based partitioned*, *Inequality join with partitioning and replication* and *Parallel partitioned hash join.* **Aggregation:** Partitioning on the grouping attribute and then aggregating function locally and finally, **Set operations:** Using a hash function in parallel.

These methods have to have an appropriate partitioning or an evaluation of an operator tree. The output of one needs to be generated tuple.by.tuple and fed into another operator to achieve the pipelined parallelism. As the execution of multiple queries in parallel is difficult to achieve, there must be cache coherency.

2 Questions not answered by this text

There is a complex process in the pipeline algorithm. There are the following phases: **Lexing/scanning** detect elements in input stream, **Parsing** converts query string to parse tree, **Resolving** look up identifiers in data dictionary, **Rewriting/transformation** transform query plan and replace views with subqueries, **Planning/optimization** optimize the query and **Execution**.

There are a lot of Join algorithms not mentioned in this paper such as the block nested loop join or merge join.

3 What has changed since this was written

This text is still valid in the year 2022 as just 6 years has passed since it was written.