Summary of Access Path Selection in a Relational Database Management System

By S. Xiao Fernández Marín

1 Summary

The System R is an experimental DBMS based on the relational model of data. This system chooses access paths for each table in the SQL simple and complex queries. It takes the path that has the less total access cost.

For processing a SQL statement there are four phases:

Parsing: Checked for correct syntax, if does not have errors, it pass to the next phase.
Optimization: Verify in System R the names of tables and columns referenced in the query. Each query block is optimized independently and it performs a access path selection and result an execution plan in the ASL, access specification language.
Code generation: Translate the ASL trees into machine language code.
Execution: The machine language code is executed by the plan chosen by the optimizer.

The storage subsystem of System R is called Research Storage System (RSS) and is the responsible of maintaining physical storage and access paths of relations, locking and logging and recovery facilities. It is presented to the users as a tuple-oriented interface (RSI).

Relations are stored in the RSS as a collection of tuples. These tuples are stored pages that are organized into segments. If these tuples are inserted in the index order, the index is clustered. Segments may contain one or more relations, but no relation may span a segment. Tuples from two or more relations may occur on the same page. Each tuple is tagged with the identification of the relation to which it belongs.

For accessing a tuple you can do it via an RSS scan. There are two types, a segment scan to find all the tuples of a relation or using an index scan, with help of B-trees.

The process of choosing a plan for evaluating a query can be made in very different ways. The simplest one is to access a single relation and show how it extends and generalizes to 2-way joins,... (nested queries). Then it examines the access paths and formulates a cost using $COST = PAGEFETCHES + W * (RSICALLS)$
cost = weighted measure of I/O and CPU utilization
PAGE FETCHES = I/O cost
W = adjustable weighting factor between I/O and CPU
RSI = CPU cost

The statistics returned by the optimizer are: **Per each relation:** Cardinality, number of pages and fraction of data pages. **Per index:** number of distinct values (NDV) and number of pages.

Based on this, it can compute the predicates, some of them are: **column=value:** $F = 1/NDV(column)(ifthereisanindexcolumn)$ otherwise: $F = 1/10$ or **column1=column2:** $F = 1/max(NDV(column1), NDV(column2))(ifbothcolumnsareindexed)$ or $F = 1/NDV(columnX)(ifonlycolumnXisindexed)$ or $F = 1/10$.

The access path selection for joins are made in two different ways. The first one is the **Nested loops:** uses scans in any order, on outer and inner relations. **Merging scans:** Only used for equi-joins, merge of inputs in "interesting" order (if there is more than one join predicate, one of them is used as the join predicate and the others are treated as ordinary predicates.) and sort + merge join.

To find the optimal plan for joining n relations, a tree of possible solutions is constructed. Also, to compute the cost, the formulas given previously are used. It involves the cost of retrieving the data using that access path and putting the results unto a temporary list.

We are shown an example of how a tree would be explored. It starts with the access paths for the single relation and from that it creates the search tree for the single relations. After that, it generates the extended search tree for second relation (the nested loop and merged join), it compares which one is the cheapest and the search tree for three relations is created. As before, each costs is compared with the cheapest solution.

A nested query is a query that has a query in the WHERE statement. If the operator is one of the six scalar comparisons, then the subquery returns a single value. If the operator is in the subquery, it return a set of values.

In both cases, the subquery needs to be evaluated only once.

2 Questions not answered by this text

MySQL, Postgress,... inspired by System R. It does not mention outer joins, just inner joins.

It does not mention the index unique key lookup in ways of or accessing a tuple nor the heap table. You can also do a index look up for inner table in the nested loop join method

3 What has changed since this was written

Expanded a lot, like the hash join, in 1984, and System R has a limited number of access methods.

It also use SQL, but it saw standardized in 1986.