Introducing a rewrite rule for optimizing queries with nested CASE expressions in DB2

Supervisor : Professor Jarek Gryz (York University)
Student name: Hossein Torabi

ABSTRACT

A new rewrite rule was created for optimizing queries with nested CASE expressions which are often generated by GUIs in query managers. The algorithm initially developed to handle two-level nested CASE expressions and further improved to flatten multilevel nested CASEs using a recursive method. The condition part of each WHEN clause will be ANDed with the condition part of its parent and the action part of each WHEN clause will replace the action part of its parent provided that the ELSE parts of the parent and the child are identical. If there are multiple WHEN clauses on the same level, each will be merged with a copy of the parent WHEN clause and the result will be added to the right side of the parent WHEN. The algorithm skips nested CASES that do not have the same ELSE value.

BACKGROUND

- There are many plans that a database management system can follow to process a given query and output the result. DB2 SQL compiler however, contains components which can rewrite the queries and generate an optimized plan before executing any query in order to reduce the cost of query processing. The main component for initiating optimized plans is the Query Rewrite (QRW) which transforms queries and rewrites them according to rewrite rules that have been defined for it. As a result the query execution time will be improved and queries will be more declarative.
- Since CASE expressions are not typically indexable, unnecessary use of them can prevent indexing particularly in WHERE clauses. The key point in optimizing a nested CASE expression is how to handle CASE expression tree and manipulate it.
- Each CASE expression consists of a list of WHEN clauses placed under a WHENLIST node. Each WHEN clause in turn contains two parts: 1- Condition 2- Action. The structure of a CASE expression is illustrated below:

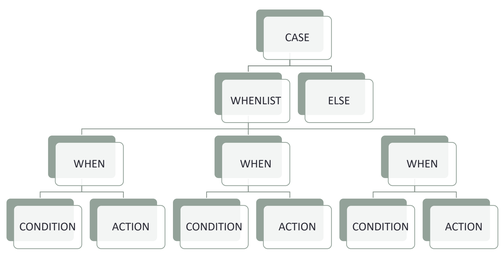

Figure 1. CASE structure in a PID tree

OBJECTIVES

- Flattening nested CASE expressions
- Reducing elapsed time
- Handling multiple WHEN clauses on the same level
- Maintaining the order of WHEN clauses in the flattened CASE expression

RECURSIVE ALGORITHM

The algorithm begins by finding the top most CASE expression and from there it starts a loop over all WHEN clauses that are on the same level while checking if the action part of each WHEN is a CASE. If it is, then a recursive call will be made over the new CASE predicate until the action part of the CASE expression in the lowest level is not a CASE anymore. This is where the base case of recursion is met.

Two scenarios were considered in the base case:

Current WHEN clause is the first one visited in the WHEN LIST

This is identical to the situation covered in the two-level algorithm. The implementation takes place in the following order:

1. ELSE parts of the current CASE will be compared against the parent CASE as a precondition for the next steps
2. Disconnecting the outer condition, inner condition, inner CASE and inner action
3. ANDing outer and inner condition
4. Inserting the new ANDed condition to the parent WHEN
5. Inserting the new action to the parent WHEN
6. Destroying the former parent action (inner CASE PID)

The current WHEN clause has one or more siblings that have already been visited

We need to add a new child for each WHEN to the parent WHENLIST; otherwise they will be missed when the disconnected action part is destroyed. To do that, following actions must be performed:

1. Adding a new WHEN PID to the right side of the parent WHEN
2. Copying the condition of the parent WHEN clause
3. Disconnecting the condition part of the current WHEN clause and ANDing it with the copy of condition in step 2
4. Inserting the new ANDed condition to the newly created WHEN clause
5. Inserting the action of the current WHEN clause as the action part of the new WHEN

The reason for copying instead of disconnecting the condition of the parent WHEN in step 2 is that we still need that condition to be merged with the condition clause of the last child in the WHEN LIST. A pointer is assigned to the position of the parent WHEN after it was merged with the first child WHEN so that that location for adding any new WHEN PID will be tracked.

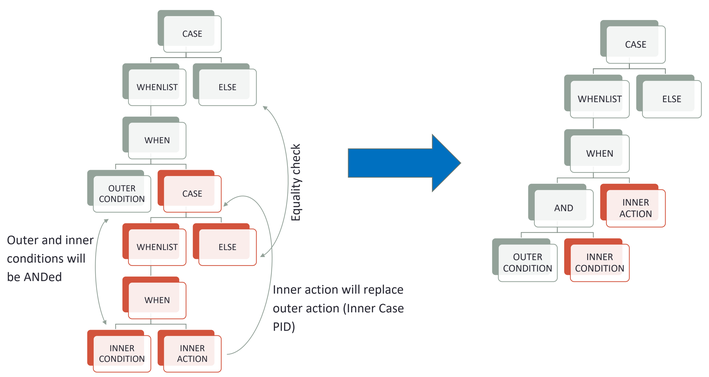

Figure 2. Nested CASE structure before and after merging in a PID tree

SAMPLE QUERY

Original Statement:

```
------------------
SELECT * FROM T WHERE
(CASE WHEN C1=1 THEN (CASE WHEN C2=10 THEN C1+50
                           WHEN C2>10 THEN (CASE WHEN C2<20 THEN
                                                 C1+100 ELSE -1 END)
                           ELSE -1 END)
      WHEN C1=2 THEN (CASE WHEN C2=20 THEN C1+200 ELSE -1 END)
      WHEN C1=3 THEN (CASE WHEN C2=30 THEN C1+300 ELSE -1 END)
      ELSE -1
END) > 0
```

Optimized Statement:

```
-------------------
SELECT * FROM T WHERE
      (CASE WHEN C1=1 AND C2=10 THEN C1+50
            WHEN C1=1 AND (C2>10 AND C2<20) THEN C1+100
            WHEN C1=2 AND C2=20 THEN C1+200
            WHEN C1=3 AND C2=30 THEN C1+300
       ELSE -1 END) > 0
```

RETURN: (Return Result)

Cumulative Total Cost:	10.1548
Cumulative CPU Cost:	83746
Cumulative I/O Cost:	1
Cumulative Re-Total Cost:	1.32612
Cumulative Re-CPU Cost:	33153
Cumulative Re-I/O Cost:	0
Cumulative First Row Cost:	9.85569
Estimated Bufferpool Buffers:	1

TBSCAN: (Table Scan)

Cumulative Total Cost:	10.1548
Cumulative CPU Cost:	83746
Cumulative I/O Cost:	1
Cumulative Re-Total Cost:	1.32612
Cumulative Re-CPU Cost:	33153
Cumulative Re-I/O Cost:	0
Cumulative First Row Cost:	9.85569
Estimated Bufferpool Buffers:	1

```
Access Plan:
-----------
Total Cost:     10.1548
Query Degree:   1

    Rows
   RETURN
   (  1)
    Cost
     I/O
     |
      1
   TBSCAN
   (  2)
   10.1548
     1
     |
     3
 TABLE: HOSSEINT
     T
     Q1
```

EXPERIMENTAL EVALUATION

- Testing on a table with 1000000 records, Two Modes (rule enabled/disabled), repeating execution 10 times
- Test on a query with three WHEN clauses, each with a single nested CASE
  elapsed time: 2.93 seconds (rule disabled) 2.86 (rule enabled) reduction rate : 2.4%
- Test on a query with 50 WHEN clauses each with 10 nested CASE expressions
  elapsed time: 11.88 seconds (rule disabled) 10.15 (rule enabled) reduction rate :14.6%
- The reduction rate in elapsed time improved by 12% as the number of nested case expressions increased