# Database - Bug #3934

## binary logical expressions in a 4GL WHERE clause are executed left to right

03/01/2019 03:29 PM - Eric Faulhaber

| | | | |
|---|---|---|---|
| **Status:** | New | **Start date:** | |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | | **% Done:** | 0% |
| **Category:** | | **Estimated time:** | 0.00 hour |
| **Target version:** | | | |
| **billable:** | No | **case_num:** | |
| **vendor_id:** | GCD | **version:** | |

**Description**

### History

**#1 - 03/01/2019 03:45 PM - Eric Faulhaber**

The 4GL executes WHERE clause sub-expressions like lOp AND rOp and lOp OR rOp from left to right. That is, lOp is always evaluated first and depending on the outcome, the evaluation of rOp may be short-circuited.

SQL, on the other hand, has no such guarantee. Operands may be evaluated in any order a modern RDBMS' query planner decides. The order of evaluation typically is determined by a cost-based analysis of the operands, taking data statistics, indices, etc. into account.

This mismatch can be problematic if the evaluation of either operand has side effects upon which the 4GL developer has encoded an assumption.

For instance, if lOp does some sort of screening test to avoid rOp from executing on a certain condition, and rOp has a side effect (like raising an ERROR if that condition exists), this is likely to cause a problem in converted code, since SQL may well decide that rOp is less expensive to evaluate, and thus evaluates it before lOp. We have seen real examples of this in production code.

One possible solution is to do an early replacement of the problematic binary logical expression with a ternary expression in the Progress AST during WHERE clause conversion. That is:

```
lOp AND rOp
```

becomes:

```
IF lOp THEN rOp ELSE false
```

and

```
lOp OR rOp
```

becomes

```
IF lOp THEN true ELSE rOp
```

These will convert to HQL/SQL CASE syntax:

```
CASE WHEN lOp THEN rOp ELSE false END
```

and

```
CASE WHEN lOp THEN true ELSE rOp END
```

respectively.

This will force the operands to be evaluated in the expected order.

However, we cannot make this change globally, since this would tie the database's cost-based optimizer's hands for the majority of cases where evaluation order does not matter and would otherwise be optimized. This would potentially cause serious performance regressions.