evaluate() PostgreSQL Function for Evaluating Stored Expressions (Part 2)

Reusing existing function for correctness check

Syntactic correctness of stored expressions

The evaluate() function is a useful approach for using stored expressions in `WHERE` clauses of SQL queries, as outlined in Part 1: evaluate() PostgreSQL Function for Evaluating Stored Expressions (Part 1).

In Part 1, in section “Improvement”, I highlighted the fact that expressions are stored in a `varchar` column (since there is no data type like `Expression` available in PostgreSQL for this use case). One implication is that a stored expression might not be syntactically correct: the database would not prevent storing an incorrect expression in a `varchar` column.

This blog describes one approach to checking the syntactic correctness of stored expressions in a `varchar` column at the time of insertion or update. This ensures that expressions, when used in queries, do not cause failures because of syntactic incorrectness during SQL query execution.

Checking correctness of stored expressions

Insert and update trigger

Every time an expression is either stored or updated a check for its syntactic correctness is required. The simplest way to react on each insert or update is to create an insert or update trigger on the table that contains the `varchar` column holding the expressions. This trigger calls a function that performs the syntactic correctness check.

In the example, this would be an insert and update trigger on table `customer` calling a function `check_expression()`. This function (introduced later in this blog) implements the syntactic correctness check of the expression that is being inserted or updated:

```
CREATE TRIGGER check_expression
    BEFORE INSERT OR UPDATE
```

```
    ON customer
    FOR EACH ROW
EXECUTE FUNCTION check_expression();
```

check_expression() function

Reviewing the `evaluate()` function shows that its implementation uses a expression passed as parameter to create a `SELECT` statement and runs the statement to evaluate an expression against a `JSON` object passed in as a another parameter.

This property — the expression being used in a SQL query and executed — can be utilized in `check_expression()`. The function `check_expression()` can apply the new or updated expression and evaluate it against the empty `JSON` object `{}`. If the outcome of this execution is `TRUE` or `FALSE` then it is known that the inserted or updated expression is syntactically valid. If an exception is raised, the expression is syntactically incorrect.

A possible implementation of `check_expression()` is as follows:

```
CREATE OR REPLACE FUNCTION check_expression()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_expression_valid BOOLEAN;
    v_expression       VARCHAR;
    v_empty_object     JSONB;
    v_exception_text   VARCHAR;
    v_exception_hint   VARCHAR;
    v_message          VARCHAR;
    v_hint             VARCHAR;

BEGIN
    v_expression = NEW.interest;
    v_empty_object = '{}';
    BEGIN
        v_expression_valid = evaluate(
                v_empty_object, v_expression);
    EXCEPTION
        WHEN OTHERS THEN
            GET STACKED DIAGNOSTICS
```

```
                v_exception_text = MESSAGE_TEXT,
                v_exception_hint = PG_EXCEPTION_HINT;
            v_message = 'Expression is incorrect: ' ||
                        v_expression;
            v_hint = v_exception_text || '; ' || v_exception_hint;
            RAISE EXCEPTION '%', v_message USING HINT = v_hint;
    END;
    RETURN NEW;
END;
$$;
```

When the inserted or updated expression is syntactically correct, then the transaction succeeds.

If the expression has a syntactic error, an exception is raised indicating the problem. An example is as follows (formatted by me to fit your screen):

```
[P0001] ERROR: Expression is incorrect:
  (object -> 'price')::int < 100000 and object >> 'color' = 'silver'
[2022-12-17 06:25:34] Hint: operator does not exist:
  jsonb >> unknown; No operator matches the given name and
  argument types. You might need to add explicit type casts.
```

Key here is that the original exception information produced by PostgreSQL is passed through without modification so that the original information is available for debugging.

Improvements

The function `check_expression()` refers to the column storing expressions as `interest` since in the example of the `customer` table the column holding the expressions was named this way.

Since it is possible that several tables in a schema are referring to expressions, this would mean that all columns holding expressions would have to be named `interest` in order to reuse the `check_expression()` function for all the tables in the corresponding triggers. While possible, a more neutral name like `expression` might be better suited.

Summary

Using the `evaluate()` function itself for checking the correctness of stored expressions not only reuses existing code but also removes the need for an alternative implementation, especially invoking parsing functionality of arbitrarily complex expressions.