evaluate() PostgreSQL Function for Evaluating Stored Expressions (Part 1)

Standing on the shoulders of database giants

Query predicates as data

I am working on a use case where a SQL query predicate (used in a `WHERE` clause) is stored in a table column as text. This predicate is used in SQL queries when selecting JSON objects from tables.

A good example is customers shopping for cars. For each customer their interest is stored as a predicate. If a new car arrives then the interest of each customer is checked by querying for each user the cars the user is interested in by the users' stored predicate (aka, the interest).

Cars are represented as JSON objects and stored in a table `car`. For example:

```
{
 "make": "Koenigsegg",
 "model": "CC850",
 "color": "silver",
 "horsepower": 1385,
 "price": 3650000
}
```

The `car` table has two columns (and two example rows here):

```
| car_id (int) | new_car (jsonb)         |
+--------------+-------------------------+
| 1            | {                       |
|              |  "make": "Koenigsegg",  |
|              |  "model": "CC850",      |
|              |  "color": "silver",     |
|              |  "horsepower": 1385,    |
|              |  "price": 3650000       |
```

```
|              | }                      |
+--------------+------------------------+
| 2            | {                      |
|              |  "make": "Honda",      |
|              |  "model": "Jazz",      |
|              |  "color": "silver",    |
|              |  "horsepower": 0,      |
|              |  "price": 21394        |
|              | }                      |
+--------------+------------------------+
```

Customers are stored in a `customer` table (with two example rows):

```
| customer_id | name      | interest                          |
| (int)       | (varchar) | (varchar)                         |
+-------------+-----------+-----------------------------------+
| 100         | A         | object -> 'horsepower' > 1000     |
+-------------+-----------+-----------------------------------+
| 101         | B         | object -> 'price' < 100000 and    |
|             |           |  object ->> 'color' = 'silver'    |
+-------------+-----------+-----------------------------------+
```

The goal is to be able to write a query that returns all customer identifiers that have interest in a new car and the corresponding car identifiers. In the example above, if the car as outlined in the JSON object arrives, customer `100` has interest in car `1`, and customer `101` in car `2`.

Note: PostgreSQL has two operators for retrieving properties, `->` and `->>` (https://www.postgresql.org/docs/current/functions-json.html). The former returns `jsonb` and the latter `text`. This means that the expressions must including type casting as required by operators. Above shown predicates would not work, but have to be specified instead as follows:

```
| customer_id | name      | interest                                |
| (int)       | (varchar) | (varchar)                               |
+-------------+-----------+-----------------------------------------+
```

```
| 100         | A         | (object -> 'horsepower')::int > 1000  |
+-------------+-----------+---------------------------------------+
| 101         | B         | (object -> 'price')::int < 100000 and |
|             |           |  object ->> 'color' = 'silver'        |
+-------------+-----------+---------------------------------------+
```

Evaluate() function

When searching for a solution on the Web I came across the paper referenced in [1]. It describes an `evaluate()` operator implementation proposal as part of SQL in context of the Oracle database system. Furthermore, it discusses how a database system can be extended to recognize expression as column type, and to provide indexes into stored expressions for optimization.

In my case I cannot change the implementation of the database system PostgreSQL efficiently, so I asked the question: could I implement an `evaluate()` function instead? Clearly, this approach of implementing a function is not the full featured support as outlined in [1], however, it would be sufficient for my use case.

The functions signature is:

```
evaluate(object jsonb, expression varchar) returns boolean;
```

The function returns `TRUE` if the expression evaluates to `TRUE` for the JSON object, and `FALSE` otherwise.

The query that returns all customers interested in new cars is then specified as follows:

```
select cust.customer_id,
       car.car_id
from customer cust,
     car car
where evaluate(car.new_car, cust.interest);
```

Implementation

The following shows an implementation of the `evaluate()` function:

```
CREATE OR REPLACE FUNCTION evaluate(
    p_object JSONB,
    p_expression VARCHAR
)
    RETURNS BOOLEAN
    LANGUAGE plpgsql
AS
    -- Expression evaluation on object
    -- (a) execute the expression on object of type JSONB
    -- (b) return TRUE if the expression evaluates to true
    -- (c) return FALSE if the expression evaluates to false
$$
DECLARE
    v_result BOOLEAN;
BEGIN
    EXECUTE format('SELECT'
                        || ' CASE'
                        || ' WHEN (SELECT count(*)'
                        || '       FROM (SELECT $1 AS object) temp'
                        || '       WHERE ('
                        || $2
                        || '       )) = 1 THEN TRUE'
                        || ' ELSE FALSE'
        || ' END')
        USING p_object::JSONB, p_expression::TEXT
        INTO v_result;
    RETURN v_result;
END;
$$;

COMMENT ON FUNCTION evaluate(
    p_object JSONB,
    p_expression VARCHAR)
    IS 'Function to evaluate an expression on an JSON object';
```

Parameters start with `p_` and local variables with `v_`. (I'd be interested in any improvement you might be able to suggest, please ping me in that case.)

The execution result of the above query is:

```
| customer_id | car_id |
+-------------+--------+
| 100         | 1      |
+-------------+--------+
| 101         | 2      |
+-------------+--------+
```

Improvements

Above description illustrates a use case using `evaluate()` from a functional perspective. In a production implementation additional work is required to implement a dependable system:

- In the example JSON objects are stored. It is advisable to ensure those being compliant to a JSON schema at least containing the properties referred to by the interests (or those that could be referred to). This ensures that all properties are present as required by the expressions stating the customer's interest. If the interest refers to a property that is not present, `evaluate()` returns `FALSE`.
- The interest is an expression, however, its column is of type `varchar`. This means that a syntactically incorrect expression can be stored in the column representing the interest. An incorrect syntactic expression will result in a runtime error thrown by the database. Therefore it is advisable to check the correctness of expressions before them being inserted or updated.

Summary

The function `evaluate()` is a straight-forward approach for executing stored expressions. It is great to stand on shoulders of giants in [1] and take in their experience during development.

While [1] outlines a general approach to extend a database system implementation's functionality, I had to limit myself to an approach that uses a given system without modifying it. [1] is a general approach, while the one described in this is restricted to JSON objects as input to `evaluate()`.

Given the function `evaluate()` more use cases open up as outlined in [1] and I might find the time to write about at least one of those in an upcoming blog.

Reference

[1] *Managing Expressions as Data in Relational Database Systems*. Aravind Yalamanchi, Jagannathan Srinivasan, Dieter Gawlick. Proceedings of the 2003 CIDR Conference. https://www.cidrdb.org/cidr2003/program/p27.pdf