Row access policies: Performance considerations

Presents two row-access policy definitions for you to compare. Both policy definitions express the same logic, yet yield significantly different performance results.

By Darren Gardner

When creating a row access policy, there are often several ways to express the same logic, and performance can vary dramatically across these variations. Therefore, it is important not only to ensure that the logic is *correct*, it's also worthwhile considering different options and verifying which provides the best performance.

Example:

Here is a row access policy definition that was *not* performing well:

```
 CREATE ROW ACCESS POLICY IF NOT EXISTS salesorg_row_policy AS (src_salesorg
VARCHAR) RETURNS BOOLEAN ->
 (
    EXISTS (
      SELECT 1
        FROM ATLAS_SECURITY.TERRITORY_REPORTING t
            ,LATERAL FLATTEN(input => t.users) AS ur
       WHERE ur.VALUE:"user_email"::STRING = LOWER(CURRENT_USER())
         AND (t.territory_node_sales_org_value IN ('ALL', src_salesorg) )
    )
 )
```

Here is another row access policy definition that yields the same results, but much faster:

```
 CREATE ROW ACCESS POLICY IF NOT EXISTS salesorg_row_policy AS (src_salesorg
VARCHAR) RETURNS BOOLEAN ->
 (
    EXISTS (
      SELECT 1
        FROM ATLAS_SECURITY.TERRITORY_REPORTING t
            ,LATERAL FLATTEN(input => t.users) AS ur
       WHERE ur.VALUE:"user_email"::STRING = LOWER(CURRENT_USER())
         AND t.territory_node_sales_org_value = 'ALL'
```

```
    )
    OR  EXISTS (
       SELECT 1

          FROM ATLAS_SECURITY.TERRITORY_REPORTING t
               ,LATERAL FLATTEN(input => t.users) AS ur
        WHERE ur.VALUE:"user_email"::STRING = LOWER(CURRENT_USER())
           AND  t.territory_node_sales_org_value = src_salesorg

    )
)
```