Row Level Access for L’Oréal

Use Case:

We want to control who gets access to the various employee data in Snowflake.

To do so we can leverage a Snowflake feature called Row Access Policies:

https://docs.snowflake.com/en/user-guide/security-row-intro.html

This solution can be implemented by leveraging the context of who is querying the data to expose only the rows that the centralized policy logic has determined.

Implementation:

According to our business need, we will determine what a user can see by using the CURRENT_USER() function. This function shows the context in which the query in ran, and returns the name of the user.

This can then be joined to the driving table/view to then get the list of employees it has access to.

We will create a driving view/table and a policy for each Subject Domain.

First we create out driving table/view that will dictate what employee has access to which employee data.

This table is created by joining tables that are coming from Success factor, and they union employee to employee access and manager to employee access.

Table/View definition:

SubjectDomain:

ACCESS_EMPLOYEE_ID: ID associated to the person that requests access to data

ACCESS_EMPLOYEE_UPN: Unique Personal Number associated to the person that requests access to data

LIST_EMPLOYEES: List in a array format of the employees that the ACCESS_EMPLOYEE_UPN has access to. If an employee has access to all possible employees, then the column will contain “ALL”.

Logic:

Driving table DDL

```
create or replace view HRDP_CORE_DV_DB.CMN_CORE_SCH.RLS_POC_VW_ANDREA as
select
subjectdomain,
ACCESS_EMPLOYEE_ID,
ACCESS_EMPLOYEE_UPN,
case when max_num_emp=array_size(LIST_EMPLOYEES) then 'ALL'::array else LIST_EMPLOYEES end as EMPLOYEE_ID_LIST
from(
 select distinct
subjectdomain,
ACCESS_EMPLOYEE_ID,
b.DDEP_EMPLOYEE_UPN_DDEP as ACCESS_EMPLOYEE_UPN,
max(max_num_emp) as max_num_emp,
array_agg(distinct EMPLOYEE_ID) as LIST_EMPLOYEES
from
(
select distinct
dpsr.subjectdomain as subjectdomain
,dgu_grt.employee_id as ACCESS_EMPLOYEE_ID
,dgu_tgt.employee_id as EMPLOYEE_ID
,max_num_emp
from
HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_PARAM_SF_ROLES dpsr
JOIN HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_SF_ROLES_GROUP dsrg
on dpsr.SFROLEID=dsrg.SFROLEID
JOIN HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_GROUP_USER dgu_grt
on dgu_grt.GRP_ID=dsrg.GRTGROUP
JOIN HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_GROUP_USER dgu_tgt
on dgu_tgt.GRP_ID=dsrg.TGTGROUP
JOIN (select count(distinct employee_id) as max_num_emp from HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_GROUP_USER)
all_emp
  on 1=1
where subjectdomain = 'PMGM'
union all
select distinct
'PMGM' as subjectdomain ,rem.REEM_MANAGER_ID_DDEP as ACCESS_EMPLOYEE_ID, dep.ddep_employee_id_ddep as
EMPLOYEE_ID, max_num_emp
FROM HRDP_CORE_DV_DB.CMN_CORE_SCH.REL_EMPLOYEE_MANAGERS rem
JOIN HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_EMPLOYEE_PROFILE dep
on dep.DDEP_EMPLOYEE_ID_DDEP=rem.REEM_EMPLOYEE_ID_DDEP
JOIN (select count(distinct employee_id) as max_num_emp from HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_GROUP_USER)
all_emp
 on 1=1
  ) a left join
  HRDP_CORE_DV_DB.CMN_CORE_SCH.DIM_EMPLOYEE_PROFILE b
  on a.ACCESS_EMPLOYEE_ID=b.DDEP_EMPLOYEE_ID_DDEP
  join SNOWFLAKE.ACCOUNT_USAGE.USERS c
  on b.DDEP_EMPLOYEE_UPN_DDEP = c.LOGIN_NAME
   where c.DELETED_ON is null and c.DISABLED = false
 group by 1,2,3);
```

This is currently a view but will be physicalized into a table to increase performance.

Row Access Policy:

The policy will use the aforementioned USER context to determine who is running the query:

ACCESS_EMPLOYEE_UPN = CURRENT_USER()

The row level policy will have 3 possible evaluations to do:

- The first is evaluating if users have access to any of the ADMIN roles, if so then they will have full data access
- The second for when an employee will have access to all the data (“ALL”). This will make sure that it will not need to scan through each employee but will have full access.
- The third will instead need to evaluate what is in the Employee List and use that to join on the table/view we want to apply the policy to.

Logic:

Row Access Policy

```
create or replace row access policy PMGM_RLS_POC_ANDREA as (P_EMPLOYEE_ID varchar)

returns boolean ->

(

EXISTS

(

    select 1 from
table(flatten(input => parse_json(current_available_roles())))
where value in ('HRDP_DOMAIN_ADMIN','HRDP_DV_DOMAIN_ADMIN','HRDP_QA_DOMAIN_ADMIN','SYSADMIN','ACCOUNTADMIN','IT-GLOBAL-HRDP-SNOWFLAKE-DV','IT-GLOBAL-HRDP-SNOWFLAKE-QA','IT-GLOBAL-HRDP-SNOWFLAKE-NP','IT-GLOBAL-HRDP-SNOWFLAKE-PD'
)


)

OR  EXISTS

(


      SELECT 1

    FROM HRDP_CORE_DV_DB.CMN_CORE_SCH.RLS_POC_VW_ANDREA

    WHERE ACCESS_EMPLOYEE_UPN         = CURRENT_USER()

    AND SUBJECTDOMAIN        = 'PMGM'

    AND array_contains('ALL'::variant,EMPLOYEE_ID_LIST)

)

OR  EXISTS

(

    SELECT 1

    FROM HRDP_CORE_DV_DB.CMN_CORE_SCH.RLS_POC_VW_ANDREA

    WHERE ACCESS_EMPLOYEE_UPN         = CURRENT_USER()

    AND SUBJECTDOMAIN        = 'PMGM'

    AND array_contains(P_EMPLOYEE_ID::variant,EMPLOYEE_ID_LIST)

))

;
```

Applying the policy:

This centralized policy can now be applied to all multiple tables as such:

Apply Row Access Policy

```
ALTER TABLE/VIEW t1 add row access policy PMGM_RLS_POC_ANDREA on (COLUMN);
```

T1: table or view name

COLUMN: column to use for policy (employee ID that we want to access to in out case)