Follow

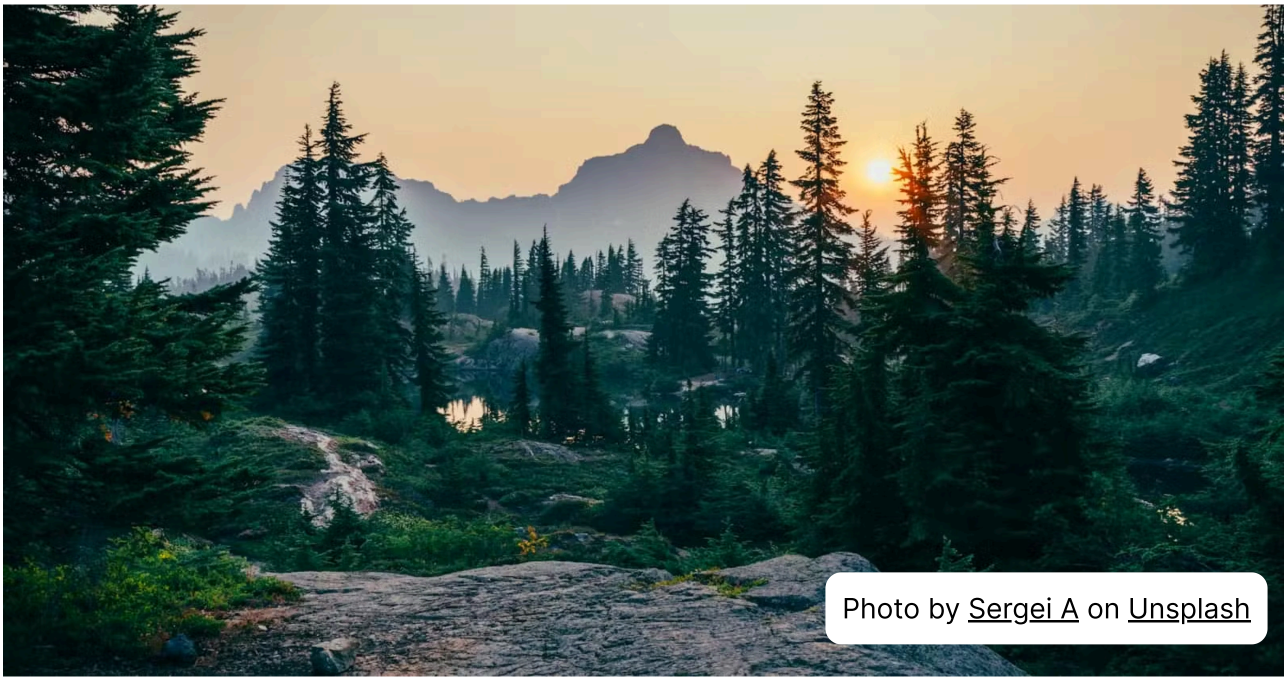
Photo by Sergei A on Unsplash

Building Database Views with Google Drive API

Helping Steven Feuerstein: Extracting Data from Google Drive API for Useful Insights

Wojciech Sowa

May 29, 2024 · 10 min read

Show more

Introduction

Welcome to my next article, which was created based on my collaboration with Steven Feuerstein. I would like to present and describe one of the business requirements that Steven wanted to achieve. The following analysis will be based on an integration using a Service Account. You can read more about this approach in this article.

Motivation

Let's answer the key question - what do we want to achieve? As you probably remember from my previous articles, the Rewild Earth application uses Google Drive to store documentation of rewilding activities in some green areas. Usually, this is photographic documentation that can be paired to visualize the before and after states of restoration.

Each location has its own folder on Google Drive. In these folders, there are subfolders nam and the event date with a short title on third.

COPY

```
root
├── LocationA
│   ├── 2022
│   │   ├── 2022-01-01 Activity1
│   │   ├── 2022-02-15 Activity2
│   │   └── 2022-03-10 Activity3
│   ├── 2023
│   │   ├── 2023-01-10 Activity4
│   │   ├── 2023-04-20 Activity5
│   │   └── 2023-06-25 Activity6
│   └── 2024
│       ├── 2024-02-05 Activity7
│       ├── 2024-05-15 Activity8
│       └── 2024-07-30 Activity9
├── LocationB
│   ├── 2022
│   │   ├── 2022-01-20 Activity10
│   │   ├── 2022-03-05 Activity11
│   │   └── 2022-04-12 Activity12
│   ├── 2023
│   │   ├── 2023-02-25 Activity13
│   │   ├── 2023-03-30 Activity14
│   │   └── 2023-06-10 Activity15
│   └── 2024
│       ├── 2024-01-15 Activity16
│       ├── 2024-04-05 Activity17
│       └── 2024-08-25 Activity18
├── LocationC
│   ├── 2022
│   │   ├── 2022-02-07 Activity19
│   │   ├── 2022-03-21 Activity20
│   │   └── 2022-05-08 Activity21
│   ├── 2023
│   │   ├── 2023
│   │   ├── 2023
│   │   └── 2023-07-22 Activity24
│   └── 2024
│       ├── 2024-03-01 Activity25
│       ├── 2024-06-14 Activity26
│       └── 2024-09-20 Activity27
└── LocationD
    ├── 2022
    │   ├── 2022-01-30 Activity28
    │   ├── 2022-02-18 Activity29
    │   └── 2022-04-25 Activity30
    ├── 2023
    │   ├── 2023-01-14 Activity31
    │   ├── 2023-03-28 Activity32
    │   └── 2023-06-05 Activity33
    └── 2024
        ├── 2024-01-07 Activity34
        ├── 2024-03-15 Activity35
        └── 2024-06-30 Activity36
```

At this point, we are ready to define our goal. We want to be able to generate an activity report with counts for a given location and year, grouped by month. The example below illustrates our plan:

Location	January	February
LocationA	1	2
LocationB	0	4
LocationC	2	1
Rest of locations...		

Of course, the above data layout is just an example, as the integration layer with the Goog ataset. Grouping, aggregations, and analytic functions can be used at the SQL query level - everything depends on the specific business requirements.

Prerequisites

Observations & Assumptions

I decided to focus on a solution using a database view. However, this view must first work with the API to fetch the necessary data about directories located on Drive, and then transform the data using both SQL and PL/SQL.

At the time of developing this solution, using a pipelined function seemed reasonable. An alternative could also be a function returning a collection. All in all, we want a function that works correctly with the TABLE() operator, such as nested tables.

It's worth noting that the Google Drive API has a method that allows you to list the contents of the drive based on various attributes. However, there is no simple method to build a request that returns the data describing the parent - child folder relationships using some notation. Therefore, a certain assumption was made to simplify our work.

The inputs to my function are the URL of the root folder on Google Drive that keeps the folders of all locations plus the year for which we want to make summary. I perform the disk search in three requests. The first request retrieves all directories for which the parent is our root folder. The second gives the subfolders entitled by year. The third request retrieves all directories that are children of the directories found in the secon ourselves to searching three levels down in Google Drive.

Events returned by my function must follow this naming format: 'YYYY-MM-DD some text'. There are also other files and directories on the drive that are not directly related to events, so they can be ignored by filtering them out.

Working with Access Tokens

Each view execution runs first a function that refreshes the access token needed in the request header to the Google Drive API. There are at least three reasons why I chose this approach:

- The table holding the access token may contain an empty row. This happens when we have a fresh installation of the solution and no one has used it in the APEX application yet.
- Depending on the size of the disk and the number of folders to analyze, I assume that I will call the Google API multiple times. To simplify the code, I assumed that I always use a fresh token, which is valid for 3600 seconds. This guarantees that individual calls will not be rejected due to an outdated token.
- The view will most often be run from the database level, so I cannot be sure if the current token stored in the dedicated table is still valid. To be sure, I always fetch a fresh token.

> 💡 If the data on Google Drive changes not really often, consider using a materialized view. This view can be refreshed at a set frequency based on your needs.

Step by Step Setup

Implementation Walkthrough

To start, I define a few complex types so I can build collections on them and use the functionalities of the Collections API, like COUNT.

```
type event_rt is record (
  location     varchar2(2000),
  year         varchar2(2000),
  event_name   varchar2(2000),
  created_time timestamp
);

type events_ntt is table of event_rt;

type folder_rt is record (
  id           varchar2(500),
  name         varchar2(500),
  parent_id    varchar2(500),
  created_time timestamp
);

type folders_ntt is table of folder_rt;

type call_result is record (
  is_success      boolean,
  folders_nt      folders_ntt,
  parent_id_expr  varchar2(32000),
  code_unit       varchar2(500),
  error_message   varchar2(4000)
);
```

I extract the root f .. I fetch a new access token, forcing its refresh.

COPY

```
l_root_folder_id := f_extract_folder_id_from_url(pi_url => pi_root_

l_access_token := f_get_access_token(pi_must_get_new_token => true)
```

I make the calls to the Google Drive API. Each time, I provide an expression that limits the search to folders that have the folders from the previous API call as their parents (`pi_parents_expression` parameter). If my call was successful (status 200), I save the results in the PL/SQL collection (nested table). I am interested in the folder ID, name, ID of parent folder and the time when it was created.

COPY

```
apex_web_service.g_request_headers.delete;
apex_web_service.g_request_headers(1).name  := 'Authorization';
apex_web_service.g_request_headers(1).value := pi_access_token;

l_incomplete_call   := true;
l_next_page_token   := null;
l_return.folders_nt := folders_ntt();

l_list_files_api_url :=
  'https://www.googleapis.com/drive/v3/files?' ||
  chr(38) ||
  'q=(' || pi_parents_expression || ') and trashed=false and mimeTy
  chr(38) ||
  'fields=incompleteSearch,nextPageToken,files(id,name,parents,crea
  chr(38) ||
  'pageSize=1000' ||
  case
    when pi_year_folder_name is not null then
      chr(38) ||                                r_name) || ''''
    else
      null
  end
  ;

while l_incomplete_call loop
  l_list_files_api_target_url :=
    case
      when l_next_page_token is not null then
        l_list_files_api_url || chr(38) || 'pageToken=' || l_next_p
      else
        l_list_files_api_url
    end;

  l_clob := apex_web_service.make_rest_request(
    p_url         => l_list_files_api_target_url,
    p_http_method => 'GET'
  );

  if apex_web_service.g_status_code = 200 then
    apex_json.parse(l_clob);

    l_return.folders_nt.extend(apex_json.get_count(p_path => 'files

    for i in 1..apex_json.get_count(p_path => 'files') loop
      l_parents := apex_json.get_t_varchar2('files[%d].parents', i)

      l_created_time_tmstp := to_timestamp(apex_json.get_varchar2('

      l_return.folders_nt(l_element_counter + i) := folder_rt(
        apex_json.get_varchar2('files[%d].id', i),
        apex_json.get_varchar2('files[%d].name', i),
        l_parents(l_parents.first),
        l_created_time_tmstp
      );

    end loop;

    l_element_counter := l_element_counter + apex_json.get_count(p_
    l_next_page_token := apex_json.get_varchar2('nextPageToken');
    l_incomplete_call :=
      case
        when l_next_page_token is not null then
          true
        else
          false
      end;

  else -- non-200 http status code
    l_return.is_success := false;
    l_return.code_unit   := 'Error when getting event folders from
    l_return.error_message := substr(l_clob, 1, 4000);

    return l_return;
  end if;

end loop;

l_return.is_success := true;

return l_return;
```

💡 When Google Drive has many folders to lookup and their IDs are stored in the variable `pi_parents_expression`, the length of this expression can sometimes be too long for the VARCHAR2 data type. To handle increasing Google Drive usage in the future, it's a good idea to split this expression into smaller parts and pass each part to the API call. This makes the code flexible and avoids the need for changes when the st [illegible]'s control.

What Else?

Even though the API calls are similar, each one has some differences. The results from the API are processed further to keep the function simple and fast as the data increases. The table below shows what actions are taken on the API results in PL/SQL. The following actions help ensure clean data and accurate counts.

Dataset	Tree Folder Level	Action Taken
Locations	1	At this level, there are more folders with different types of data that we don't need to process. We only want to focus on the folders that keep the activity archives.
Years	2	The API call uses a special version of the request URL that includes the folder name, which is just the year number. This helps limit the data on early stage, making the API response smaller and faster.
Activities	3	As those folders can store any kind of crucial data, we stated that activity folder meets the following format 'YYYY-MM-DD some text'.

Returning Results

It's time for the most important part - combining three collections using the TABLE() operator. This will create the following structure, which the function returns:

COPY

```
for rec in (
  select l.name as location,
         y.name as year,
         e.name as event_name,
         e.created_time
    from table(l_location_call_result.folders_nt) l
    join table(l_year_call_result.folders_nt) y
      on l.id = y.parent_id
    join table(l_event_call_result.folders_nt) e
      on y.id = e.parent_id
) loop
  pipe row(
    event_rt(
      rec.location,
      rec.year,
      rec.event_name,
      rec.created_time
    )
  );
end loop;
```

Location	Year	Event_name
LocationA	2024	2024-02-23 Nice Place
LocationB	2024	2024-03-25 Huge Park

Next, this form of data can be properly processed and grouped to obtain a fully functional view presenting the current data retrieved from Google Drive.

Which Date Should Be Chosen to Group Events by Month?

I believe this is an important issue that needs its own paragraph. Let me explain a point that might cause incorrect classification of past events.

As you noticed, my pipelined function returns data about the event date in two columns. One of them is EVENT_NAME, which includes the event title and date. The other is a timestamp obtained from the Google API, which corresponds to the creation date of the resource on Google Drive.

There is a possibility that an event took place, for example, on January 31st, but was uploaded to the Drive some day in February. There could be many reasons for this, such as lack of time :-)

Therefore, I think it's best to use the date in the event folder name. This helps us avoid putting events in the wrong months. In the example above, the event should be assigned to January, not February, as the folder's creation date on Google Drive suggests.

Error Handling

💡 In case of a [illegible] call stage, I throw a defined exception and return a single row with the error details.

COPY

```
when e_getting_counts_error then
  pipe row(
    event_rt(
      l_code_unit,
      l_error_message,
      null,
      null
    )
  );
```

Sample Testing

Below you can find simple calls to my pipelined function. The results returned by this function can be processed on the SQL side as needed.

COPY

```
select s.*
  from table(google_drive_pkg.f_get_file_list_for_counts(
             pi_root_folder_url => 'https://drive.google.com/drive/fo
             pi_year            => '2024')
       ) s
 order by location, event_name;
```

COPY

```
select location, mh, count(*)
  from (
         select
                                                        mh
          from table(google_drive_pkg.f_get_file_list_for_counts(
                    pi_root_folder_url => 'https://drive.google.com
                    pi_year            => '2024')
                ) s
       )
 group by location, mh
 order by location, to_number(mh);
```

Observations

Keeping your drive organized is very important. Arrange your drive resources carefully to reduce API calls, which will improve the performance of the solution.

It's important to check the size of the data sets returned by the API to choose the right communication method. If the results don't fit in one call, make more calls until you get all the results.

Summary

As you can see, with this integration, you can make different requests to work with data easily. Using a pipelined function, we can hide the complexity of the integration and work with the results as if they are from a regular table, fully utilizing Oracle SQL.

💡 As always, if you have any questions, need more details, or have suggestions for improvement, please let me know!

#oracle-apex Google API integration REST API Cloud Google Workspace Google Drive

Written by

Wojciech Sowa

Follow

ARTICLE SERIES

Oracle APEX meets Google API: Let's Dance Together

1

Building Database Views with Google Drive API

Introduction Welcome to my next article, which was created based on my collaboration with Steven Feu...

2

APEX Integration with Google Drive in Pure PL/SQL with Google Service Account

Introduction Welcome to my post about integrating an APEX application with Google Drive (or any Goog...

3

APEX Integration with Google Drive on Javascript Side as an APEX Plugin

Introduction Welco application with Google Drive (or an

4

Using My Skills to Help Heal the Planet: The beginning

Introduction I remember the moment when I started my career as an Oracle Developer. It was back in 2...