Finding Salesforce Duplicate Permission Sets using the Jaccard index

7 min read · 6 days ago

Laurent Kubaski

Or "*why Generative AI should not be seen as the default solution to every problem you find*"

Or "*the surprising relationship between finding duplicate Salesforce permission sets and the study of the Alpine Flora*"

A permission set in the Alps

1. Introduction

During Dreamforce 2025, I co-presented The Future of Setup Powered by Agentforce and I said that one idea on the Agentforce For Setup roadmap was to allow the agent to identify duplicate permission sets in a Salesforce org.

As a product manager I usually don't have to worry about the implementation details but as a former developer, I often ask myself "*if I had to write the code to do that… how would I do it?*" and that's exactly what I started wondering in the plane from Paris to San Francisco.

The immediate solution that came to my mind was to extract all the permission sets from a Salesforce org, save them in a file, upload the file in ChatGPT and ask "*can you help identify the permission sets that are at least 90% similar?".* And then I gave myself a slap on the wrist because as I wrote in Generative AI: the good and the bad,

this is a typical example of the saying: '*when the only tool you have is a hammer, every problem looks like a nail.*'

So is there a better way? Yes there is — spoiler alert: the solution is in the title of this article — but first let's talk about Salesforce permission sets.

2. Extracting the Salesforce Permission Sets

Big thanks to my colleague Cheryl Feldman for educating me on several of the things I'm about to write.

As the name implies, a Permission Set is "*is a collection of permissions that give users access to various tools and functions*". Basically, if you want to allow a Salesforce user "to do something", then you create a permission set that contains the list of permissions needed "to do that thing" and then you assign it to that user.

There are 5 different types of permissions in a permission set:

- User permissions
  - Specify what tasks users can perform and what features users can access
  - There is not a *dedicated* object to retrieve them, but they're stored as individual fields on the PermissionSet object and can be retrieved via describeSObjects().

- Object permissions
  - Specify the base-level access users have to create, read, edit, and delete records for each object
  - Corresponding Object: ObjectPermissions

- Field permissions
  - Specify the access level for each field in an object.
  - Corresponding Object: FieldPermissions

- Tab Setting Permissions
  - Specify a tab's availability in App Launcher
  - Corresponding Object: PermissionSetTabSetting

- Setup Entity Access permissions
  - Grant access to specific Setup objects, like custom objects, Apex classes, or metadata types.
  - Corresponding Object: SetupEntityAccess

The screenshot below shows where you can see those 5 different permission types when looking at a permission set.

So basically, extracting the permission sets from a Salesforce org is as easy as using the Salesforce API to query the 5 objects I've listed above (PermissionSet, ObjectPermissions, FieldPermissions, PermissionSetTabSetting & SetupEntityAccess) and storing everything in a file.

My goal in this article is not to explain in detail how to do this so if you're interested, check the source code on Github.

The output of that program are 2 things:

- A JSON file with a list of all permission sets in the org (see below).
- A list of all the duplicate permission sets, along with their differences.


```
{
  "id": "0PSWs000005gccLOAQ",
  "name": "ActorCASCPermSet",
  "label": "ActorCASCPermSet",
```

```
    "user_license": "Cloud Integration User",
    "permission_set_license": null,
    "system_perms": [
      { "name": "PermissionsChatterInternalUser" },
      { "name": "PermissionsApiUserOnly" }
    ],
    "object_perms": [
      {
        "sobject_type": "Asset",
        "perms": [ "PermissionsCreate", "PermissionsRead", "PermissionsEdit", "Pe
      }
    ],
    "field_perms": [
      {
        "sobject_type": "ACustom__c",
        "sobject_field": "ACustom__c.Custom_Field__c",
        "perms": [ "PermissionsRead" ]
      },
      {
        "sobject_type": "Account",
        "sobject_field": "Account.AccountNumber",
        "perms": [ "PermissionsRead" ]
      }
    ],
    "setup_entity_access": [
      {
        "setup_entity_type": "TabSet",
        "setup_entity_id": "02uWs000003KBp4IAG"
      },
      {
        "setup_entity_type": "FlowDefinition",
        "setup_entity_id": "300Ws00000hxoBJIAY"
      }
    ],
    "permission_set_tab_setting": [
      {
        "name": "ACustom__c",
        "visibility": "DefaultOn"
      }
    ]
  }
```


3. Finding similar Permission Sets using the Jaccard index

OK so let's say that you have a file that contains all the permission sets in your org: now what?

Just for fun, I've tried uploading that file to ChatGPT and ask it to find permission sets that are 90% similar. I got an answer... but it wasn't perfect.

But as you can imagine, there is a better way to find how similar 2 “things” are: it’s called **the Jaccard index.**

> *The Jaccard index measures similarity between finite non-empty sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets:*
>
> *Source: Wikipedia*

According to Wikipedia, it was originally developed Grove Karl Gilbert in 1884 and then independently re-developed in 1901 by Paul Jaccard who was a professor of botany in Zurich. Paul Jaccard calls it “*coefficient de communauté*” (coefficient of community) in his 1901 paper titled Distribution de la Flore Alpine dans le Bassin des Dranses et dans quelques régions voisines.

Jaccard index to measure the proportion of common plants in 2 different locations

Here, Jaccard compares the plants of two specific localities: “Plan la Chaud” and “la Peulaz”.

- Number of different plants in “Plan la Chaud”: 101
- Number of different plants in “La Peulaz”: 107
- Number of common plants (Intersection): 53
- Number of different plants (Union): (101 + 107) — 53 = 155
- Jaccard Index: 53 / 155 = approximately 0.35 or 35%

Hence the Jaccard Index formula:

Jaccard Index = size of the intersection of 2 sets divided by the size of the union of 2 sets

So if we apply this to a list of permission sets, the implementation is very straightforward: no need to use generative AI!

```
class DuplicateFinder:

    SIMILARITY_THRESHOLD = 1

    def __init__(self, permsets: list[PermissionSet]) -> None:
        self.permsets = permsets

    def jaccard(self) -> list[JaccardDifference]:
        logging.debug(">> jaccard")
        duplicates:list[JaccardDifference] = []
        nb_permsets = len(self.permsets)
        for i in range(nb_permsets):
            perms_i = set(self.permsets[i].get_all_perms())
            for j in range(i + 1, nb_permsets):
                perms_j = set(self.permsets[j].get_all_perms())
                intersection = perms_i.intersection(perms_j)
                union = perms_i.union(perms_j)
                if union:  # Avoid division by zero
                    similarity = len(intersection) / len(union)
                    if similarity >= DuplicateFinder.SIMILARITY_THRESHOLD:
                        duplicates.append(JaccardDifference(self.permsets[i], s
                if not union and not intersection:
                    # Both permission sets are empty, consider them identical
                    duplicates.append(JaccardDifference(self.permsets[i], self.
        logging.debug(f"<< jaccard: returning nb records={len(duplicates)}")
        # Sort by similarity descending (most similar first)
        duplicates = sorted(duplicates, key=lambda t: t.similarity, reverse=Tru
        return duplicates
```

Coming back to the sample program on Github: when executed against one of my Salesforce demo orgs, it's telling me 3 things:

- the "DCToDCSharingSetupC2CPermSet" and the "PromptTemplatePermSet" permission sets are similar at 90%.

- They have 74 permissions in common.
- The “DCToDCSharingSetupC2CPermSet” has 6 unique permissions.
- The “PromptTemplatePermSet” has 2 unique permissions.

```
Duplicate found: 'DCToDCSharingSetupC2CPermSet (user_license=Cloud Integration
  No common permissions: 74

  'DCToDCSharingSetupC2CPermSet' unique permissions:
    ObjectPermissions(sobject_type='ExtDataShrSnowflakeConn', perms=['Permissio
    ObjectPermissions(sobject_type='ExtDataShareTargetDef', perms=['Permissions
    ObjectPermissions(sobject_type='ExtDataShrSagemakerConn', perms=['Permissio
    ObjectPermissions(sobject_type='ExtDataShrTrgtConnection', perms=['Permissi
    ObjectPermissions(sobject_type='DataOrgFeatureAllowList', perms=['Permissio
    ObjectPermissions(sobject_type='ExtDataShareTarget', perms=['PermissionsRea

  'PromptTemplatePermSet' unique permissions:
    ObjectPermissions(sobject_type='GenAiPromptTemplate', perms=['PermissionsRe
    SystemPermission(name='PermissionsExecutePromptTemplates')
```

4. The Jaccard Index is not enough!

It’s important to keep in mind that 2 permission sets may look completely different from a Jaccard Index point of view, **but may still give access to the same functionalities**.

4.1. Example: modify all data

Let’s first consider the powerful “modify all data” user permission: amongst other things, it gives implicit Read, Create, Edit, Delete, View All Records, & Modify All Records access on all standard and custom objects **even if those permission are not explicitly enabled in the permission set.**

So if you create a permission set with **only** the “modify all data” user permission and compare it with a permission set that includes all the permissions that are **implicitly** given by this user permission (see Modify All Data Permission in Salesforce) then the Jaccard Index will be 0 even though the 2 permission sets actually give access to the same functionality.

4.2. Example: View All Fields

Another example is the View All Fields object permission. As the name implies, this gives read access to all the object fields without the need to manually give Read

access on each individual field.

From a Jaccard index point of view, a permission set with that object permission will be considered completely different from a permission set that gives explicit Read access on each individual field, but functionally the twopermission sets are still identical.

5. Conclusion

The Jaccard Index is a surprisingly elegant tool born from studying plants in the Alps, yet still useful in the cloud era. It reminds us that not every data problem requires deep learning.

6. Further Reading

- My github project of course :-)
- https://perm-comparator.herokuapp.com (I have not tried it myself though)
- Jaccard Index (wikipedia)
- Jaccard Similarity Made Simple (Medium)

Generative Ai Tools Jaccard Index

Bgerby

What are your thoughts?

More from Laurent Kubaski

Laurent Kubaski

Ollama prompt templates

This article is part my “Ollama Explained” series.

Feb 9 17 1

Laurent Kubaski

Connecting your Python MCP Server to Copilot Chat in Visual Studio Code

This article is part of my MCP Explained series.

Jun 20 5

Laurent Kubaski

Ollama endpoints options parameter

This article is part my “Ollama Explained” series.

Mar 16 37 1

Laurent Kubaski

Connecting your Python MCP Server to Claude Desktop

This article is part of my MCP Explained series.

Jun 20 2

See all from Laurent Kubaski

Recommended from Medium

In InfoSec Write-ups by Rahul Bogar

SQL Injection Leads to dump the Student PII

Hello Everyone myself Rahul Bogar. In this writeup I will tell you how I found the SQLi in the educational website.

Oct 28 32 1

Cloud Security Pro

Understanding GCP Credentials (API Keys, OAuth 2.0

One of the very first GCP cloud security fundamentals I found it helpful to learn was understanding the use cases of various GCP...

Nov 3

Elkassimiabderrahmane

SQL injection — Authentication

The name speaks for itself: this vulnerability appears when it’s possible to inject SQL code into the SQL queries made on a web page. It...

Sep 7

Marduk I Am

Blind SQL Injection with Conditional Errors

A Portswigger Lab

Oct 5 3 2

Azefox innovations

Malware Development 0×1: Process Injection

At Azefox innovations, we don’t just study malware—we dissect how it’s built. Process injection is a foundational malware development...

Aug 22 50

Ramdhan Hidayat

The Limitations of Averages: Why A/B Testing Fails Personalization

Imagine you’re a coffee shop owner trying to decide whether to offer a new dark roast or a light roast.

Aug 23