Analytics Security Implementation Guide
Salesforce, Summer ’19

’19

@salesforcedocs
Last updated: July 18, 2019
**CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURITY FOR SALESFORCE ANALYTICS</td>
<td>1</td>
</tr>
<tr>
<td>SALESFORCE DATA ACCESS IN ANALYTICS</td>
<td>2</td>
</tr>
<tr>
<td>APP-LEVEL SHARING</td>
<td>4</td>
</tr>
<tr>
<td>ROW-LEVEL SECURITY FOR DATASETS</td>
<td>6</td>
</tr>
<tr>
<td>Security Predicates for Datasets</td>
<td>6</td>
</tr>
<tr>
<td>Row-Level Security Example based on Record Ownership</td>
<td>7</td>
</tr>
<tr>
<td>Row-Level Security Example based on Opportunity Teams</td>
<td>12</td>
</tr>
<tr>
<td>Row-Level Security Example based on Role Hierarchy and Record Ownership</td>
<td>20</td>
</tr>
<tr>
<td>Row-Level Security Example Based on Territory Management</td>
<td>29</td>
</tr>
<tr>
<td>Salesforce Sharing Inheritance for Datasets</td>
<td>35</td>
</tr>
<tr>
<td>SECURITY PREDICATE REFERENCE</td>
<td>39</td>
</tr>
<tr>
<td>Predicate Expression Syntax for Datasets</td>
<td>39</td>
</tr>
<tr>
<td>Sample Predicate Expressions for Datasets</td>
<td>44</td>
</tr>
</tbody>
</table>
Analytics has different levels of security that your organization can implement to ensure that the right user has access to the right data.

- The administrator can implement object-level and field-level security to control access to Salesforce data. For example, the administrator can restrict access to prevent the dataflow from loading sensitive Salesforce data into datasets. This document describes how Analytics uses object-level and field-level security on Salesforce data and how to configure permissions on Salesforce objects and fields.

- Dataset owners can implement row-level security on each dataset that they create to restrict access to its records. If a dataset does not have row-level security, users who have access to the dataset can view all records. This document describes how to configure row-level security on datasets and provides some sample implementations based on datasets created from Salesforce data and external data.

  **Note:** Analytics supports security predicates, a robust row-level security feature that enables you to model many different types of access controls on datasets. Also, Analytics supports sharing inheritance, to synchronize with sharing that’s configured in Salesforce, subject to certain limitations. If you use sharing inheritance, you must also set a security predicate to take over in situations when sharing settings can’t be honored.

- App owners, administrators, and users granted manager access to an app control access to datasets, lenses, and dashboards within apps. This document describes the different levels of access for apps and how to share datasets, lenses, dashboards in an app with other users.
Analytics requires access to Salesforce data when extracting the data and also when the data is used as part of row-level security. Analytics gains access to Salesforce data based on permissions of two internal Analytics users: Integration User and Security User.

Analytics uses the permissions of the Integration User to extract data from Salesforce objects and fields when a dataflow job runs. Because the Integration User has View All Data access, consider restricting access to particular objects and fields that contain sensitive data. If the dataflow is configured to extract data from an object or field on which the Integration User does not have permission, the dataflow job fails.

When you query a dataset that has row-level security based on the User object, Analytics uses the permissions of the Security User to access the User object and its fields. The Security User must have at least read permission on each User object field included in a predicate. A predicate is a filter condition that defines row-level security for a dataset. By default, the Security User has read permission on all standard fields of the User object. If the predicate is based on a custom field, then grant the Security User read access on the field. If the Security User does not have read access on all User object fields included in a predicate expression, an error appears when you try to query the dataset using that predicate.

**Important:** Because Analytics requires the Integration User and Security User to access Salesforce data, do not delete either of these users.

### Control Access to Salesforce Objects and Fields

Analytics requires access to Salesforce data when extracting the data and also when the data is used as part of row-level security. Configure the permissions of the Integration User on Salesforce objects and fields to control the dataflow’s access to Salesforce data. Configure the permissions of the Security User to enable row-level security based on custom fields of the User object.

**Control Access to Salesforce Objects and Fields**

Analytics requires access to Salesforce data when extracting the data and also when the data is used as part of row-level security. Configure the permissions of the Integration User on Salesforce objects and fields to control the dataflow’s access to Salesforce data. Configure the permissions of the Security User to enable row-level security based on custom fields of the User object.

When configuring permissions for the Integration User or Security User, make changes to a cloned version of the user profile.

1. From Setup, enter *Profiles* in the Quick Find box, then select *Profiles*, and then select the user profile.
   - For the Integration User, select the Analytics Cloud Integration User profile. For the Security User, select the Analytics Cloud Security User profile.
2. Click *Clone* to clone the user profile.
3. Name and save the cloned user profile.
4. Click *Object Settings*.
5. Click the name of the Salesforce object.
6. Click *Edit*.
   - To enable permission on the object, select *Read* in the Object Permissions section.
b. To enable permission on a field of the object, select **Read** for the field in the Field Permissions section.

- **Note:** You can’t change the permissions on standard fields of the User object.

7. Save the object settings.

8. Assign the cloned user profile to the Integration User or Security User.
   a. From Setup, enter **Users** in the Quick Find box, then select **Users**.
   b. Select the user to which you want to assign the user profile.
   c. Click **Edit**.
   d. In the Profile field, select the user profile.
   e. Click **Save**.

9. Verify that the Integration User or Security User has the right permissions on fields of the objects.
APP-LEVEL SHARING

Analytics apps are like folders, allowing users to organize their own data projects—both private and shared—and control sharing of datasets, lenses, and dashboards.

All Analytics users start off with Viewer access to the default Shared App that’s available out of the box; administrators can change this default setting to restrict or extend access. Each user also has access to a default app out of the box, called My Private App, intended for personal projects in progress. The contents of each user’s My Private App aren’t visible to administrators, but dashboards and lenses in My Private App can be shared.

All other apps created by individual users are private, by default; the app owner and administrators have Manager access and can extend access to other users, groups, or roles.

Here’s a summary of what users can do with Viewer, Editor, and Manager access.

<table>
<thead>
<tr>
<th>Action</th>
<th>Viewer</th>
<th>Editor</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>View dashboards, lenses, and datasets in the app</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><img src="image" alt="Note" /> If the underlying dataset is in a different app than a lens or dashboard, the user must have access to both apps to view the lens or dashboard.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>See who has access to the app</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Save contents of the app to another app that the user has Editor or Manager access to</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Save changes to existing dashboards, lenses, and datasets in the app (saving dashboards requires the appropriate permission set license and permission)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Change the app’s sharing settings</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Rename the app</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Delete the app</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

![Important](image) When users are deactivated, they lose share and delete access to all apps they manage. To avoid "stranding" an app, be sure that manager access is assigned to at least one active user BEFORE deactivating the user who’s the manager of the app.

1. Share an App

   To enable others to see a lens, dashboard, or dataset, one way to share is by sharing the app it’s in.
Share an App

To enable others to see a lens, dashboard, or dataset, one way to share is by sharing the app it's in.

1. On the app page, click the **Share** button.

2. On the Give Access tab:
   a. Choose whether you're sharing the app with a user, group, or role.
   b. Start typing the name and select from the suggested matches.
   c. Choose the level of sharing access: Viewer, Editor, or Manager.
   d. Click **Add**.
   e. Click **Save**, then click **Done**.

**Important:** When users are deactivated, they lose share and delete access to all apps they manage. To avoid "stranding" an app, be sure that manager access is assigned to at least one active user BEFORE deactivating the user who's the manager of the app.
If an Analytics user has access to a dataset, the user has access to all records in the dataset by default. However, you can implement row-level security on a dataset to restrict access to records. Some records contain sensitive data that must not be accessible by everyone.

To implement row-level security for a dataset, either define a security predicate or turn on sharing inheritance. Specify from which objects to migrate the sharing rules. Sharing inheritance works together with security predicates. You can specify a security predicate to take over for those users who fall outside the scope of sharing inheritance.

Security Predicates for Datasets
Applying a predicate to a dataset is more than just defining the predicate expression. You also need to consider how the predicate is dependent on the information in the dataset and where to define the predicate expression.

Row-Level Security Example based on Record Ownership
Let's look at an example where you create a dataset based on a CSV file and then implement row-level security based on record ownership. In this example, you will create a dataset that contains sales targets for account owners. To restrict access on each record in the dataset, you will create a security policy where each user can view only sales targets for accounts that they own. This process requires multiple steps that are described in the sections that follow.

Row-Level Security Example based on Opportunity Teams
Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on an opportunity team. In this example, you will create a dataset that contains only opportunities associated with an opportunity team. To restrict access on each record in the dataset, you will create a security policy where only opportunity members can view their opportunity. This process requires multiple steps that are described in the sections that follow.

Row-Level Security Example based on Role Hierarchy and Record Ownership
Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on the Salesforce role hierarchy and record ownership. In this example, you will create a dataset that contains all opportunities. To restrict access on each record in the dataset, you will create a security policy where each user can view only opportunities that they own or that are owned by their subordinates based on the Salesforce role hierarchy. This process requires multiple steps that are described in the sections that follow.

Row-Level Security Example Based on Territory Management
Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on your defined territories. In this example, you determine what model you use for territory management, so you can later review sample JSON for that dataset. To restrict access on each record in the dataset, you will create a security predicate where each user can view only data appropriate for the territory to which they belong.

Security Sharing Inheritance for Datasets
Use sharing inheritance to let Analytics use the same sharing rules for your datasets as Salesforce uses for your objects.

Security Predicates for Datasets
Applying a predicate to a dataset is more than just defining the predicate expression. You also need to consider how the predicate is dependent on the information in the dataset and where to define the predicate expression.

Define a predicate for each dataset on which you want to restrict access to records. A security predicate is a filter condition that defines row-level access to records in a dataset.
When a user submits a query against a dataset that has a predicate, Analytics checks the predicate to determine which records the user has access to. If the user doesn’t have access to a record, Analytics does not return that record.

**Note:**
- Changes to security settings (rowLevelSharingSource or rowLevelSecurityFilter) in a dataflow have no effect on datasets that already exist. You must change those settings on the edit dataset page.
- When sharing inheritance is enabled, you can set the security predicate to ‘false’ to block all users not covered by sharing. In fact, this predicate is the default when sharing is enabled on existing datasets.

The predicate is flexible and can model different types of security policies. For example, you can create predicates based on:

- Record ownership. Enables each user to view only records that they own.
- Management visibility. Enables each user to view records owned or shared by their subordinates based on a role hierarchy.
- Team or account collaboration. Enables all members of a team, like an opportunity team, to view records shared with the team.
- Combination of different security requirements. For example, you might need to define a predicate based on the Salesforce role hierarchy, teams, and record ownership.

The type of security policy you implement depends on how you want to restrict access to records in the dataset.

**Warning:** If row-level security isn’t applied to a dataset, any user that has access to the dataset can view all records in the dataset.

You can create a predicate expression based on information in the dataset. For example, to enable each user to view only dataset records that they own, you can create a predicate based on a dataset column that contains the owner for each record. If needed, you can load additional data into a dataset required by the predicate.

The location where you define the predicate varies.

- To apply a predicate on a dataset created from a dataflow, add the predicate in the rowLevelSecurityFilter field of the Register transformation. The next time the dataflow runs, Analytics will apply the predicate.
- To apply a predicate on a dataset created from an external data file, define the predicate in the rowLevelSecurityFilter field in the metadata file associated with the external data file. Analytics applies the predicate when you upload the metadata file and external data file. If you already created the dataset from a external data file, you can edit the dataset to apply or change the predicate.

**Row-Level Security Example based on Record Ownership**

Let’s look at an example where you create a dataset based on a CSV file and then implement row-level security based on record ownership. In this example, you will create a dataset that contains sales targets for account owners. To restrict access on each record in the dataset, you will create a security policy where each user can view only sales targets for accounts that they own. This process requires multiple steps that are described in the sections that follow.

**Note:** Although this example is about applying a predicate to a dataset created from a CSV file, this procedure can also be applied to a dataset that is created from Salesforce data.

1. **Determine Which Data to Include in the Dataset**
   First, determine what data you want to include in the dataset. For this example, you will create a Targets dataset that contains all sales targets.

2. **Determine Row-Level Security for Dataset**
   Now it’s time to think about row-level security. How will you restrict access to each record in this dataset?
3. **Add the Predicate to the Metadata File**
   For a dataset created from a CSV file, you can specify the predicate in the metadata file associated with the CSV file or when you edit the dataset.

4. **Create the Dataset**
   Now that you updated the metadata file with the predicate, you can create the dataset.

5. **Test Row-Level Security for the Dataset**
   You must verify that the predicate is applied properly and that each user can see their own sales targets.

---

**Determine Which Data to Include in the Dataset**

First, determine what data you want to include in the dataset. For this example, you will create a Targets dataset that contains all sales targets.

You will obtain sales targets from the CSV file shown below.

<table>
<thead>
<tr>
<th>Account Owner</th>
<th>Region</th>
<th>Target</th>
<th>Target Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tony Santos</td>
<td>Midwest</td>
<td>10000</td>
<td>1/1/2011</td>
</tr>
<tr>
<td>Lucy Timmer</td>
<td>Northeast</td>
<td>50000</td>
<td>1/1/2011</td>
</tr>
<tr>
<td>Lucy Timmer</td>
<td>Northeast</td>
<td>0</td>
<td>12/1/2013</td>
</tr>
<tr>
<td>Bill Rolley</td>
<td>Midwest</td>
<td>15000</td>
<td>1/1/2011</td>
</tr>
<tr>
<td>Keith Laz</td>
<td>Southwest</td>
<td>35000</td>
<td>1/1/2011</td>
</tr>
<tr>
<td>Lucy Timmer</td>
<td>Southeast</td>
<td>40000</td>
<td>1/1/2011</td>
</tr>
</tbody>
</table>

If you were to create the dataset without implementing row-level security, any user that had access to the dataset would be able to see the sales targets for all account owners. For example, as shown below, Keith would be able to view the sales targets for all account owners.
You need to apply row-level security to restrict access to records in this dataset.

**Determine Row-Level Security for Dataset**

Now it's time to think about row-level security. How will you restrict access to each record in this dataset? You decide to implement the following predicate on the dataset.

'AccountOwner' == "$User.Name"

**Note:** All predicate examples in this document escape the double quotes because it's required when you enter the predicate in the Register transformation or metadata file. This predicate implements row-level security based on record ownership. Based on the predicate, Analytics returns a sales target record when the user who submits the query on the dataset is the account owner.

Let's take a deeper look into the predicate expression:

- AccountOwner refers to the dataset column that stores the full name of the account owner for each sales target.
- $User.Name refers to the Name column of the User object that stores the full name of each user. Analytics performs a lookup to get the full name of the user who submits each query.

**Note:** The lookup returns a match when the names in AccountOwner and $User.Name match exactly—they must have the same case.

### Add the Predicate to the Metadata File

For a dataset created from a CSV file, you can specify the predicate in the metadata file associated with the CSV file or when you edit the dataset.

You must escape the double quotes around string values when entering a predicate in the metadata file.

In this example, you add the predicate to the metadata file shown below.

```json
{
  "fileFormat": {
    "charsetName": "UTF-8",
    "fieldsDelimitedBy": ",",
    "fieldsEnclosedBy": "\"",
    "numberOfLinesToIgnore": 1
  },
  "objects": [
    {
      "name": "Targets",
      "fullyQualifiedName": "Targets",
      "label": "Targets",
      "rowLevelSecurityFilter": "'AccountOwner' == "$User.Name\",
      "fields": [
        {
          "name": "AccountOwner",
          "fullyQualifiedName": "Targets.AccountOwner",
          "label": "Account Owner",
          "type": "Text"
        },
        {
          "name": "Region",
          "fullyQualifiedName": "Targets.Region",
```
Create the Dataset

Now that you updated the metadata file with the predicate, you can create the dataset.

⚠️ Warning: If you wish to perform the steps in this sample implementation, perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created.

To create the dataset, perform the following steps.

1. In Analytics, go to the home page.
2. Click Create > Dataset
3. Click CSV.

The following screen appears.
4. Select the CSV file and metadata (schema) file.

5. In the Dataset Name field, enter “SalesTarget” as the name of the dataset.

6. Optionally, choose a different app where you want to store the dataset.

7. Click Create Dataset

Analytics confirms that the upload is successful and then creates a job to create the dataset. You can view the SalesTarget dataset after the job completes successfully.

8. To verify that the job completes successfully, perform the following steps:

a. Click the gear icon (⚙️) and then select Data Monitor to open the data monitor.
   By default, the Jobs View of the data monitor appears. It shows the statuses of dataflow and external data upload jobs.

b. Click the Refresh Jobs button (🔍) to view the latest statuses of the jobs.

Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see their own sales targets.

1. Log in to Analytics as Keith.

2. Open the SalesTargets dataset.
   As shown in the following lens, notice that Keith can see only his sales target.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.
Row-Level Security Example based on Opportunity Teams

Let’s look at an example where you create a dataset based on Salesforce data and then implement row-level security based on an opportunity team. In this example, you will create a dataset that contains only opportunities associated with an opportunity team. To restrict access on each record in the dataset, you will create a security policy where only opportunity members can view their opportunity. This process requires multiple steps that are described in the sections that follow.

1. **Determine Which Data to Include in the Dataset**
   First, determine what data you want to include in the dataset. For this example, you will create an OppTeamMember dataset that contains only opportunities associated with an opportunity team.

2. **Design the Dataflow to Load the Data**
   Now it’s time to figure out how the dataflow will extract the Salesforce data and load it into a dataset. You start by creating this high-level design for the dataflow.

3. **Determine Row-Level Security for the Dataset**
   Now it’s time to think about row-level security. How will you restrict access to each record in this dataset?

4. **Modify the Dataflow Based on Row-Level Security**
   It’s now time to add the predicate in the dataflow definition file.

5. **Create the Dataset**
   Now that you have the final dataflow definition file, you can create the dataset.

6. **Test Row-Level Security for the Dataset**
   You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.
Determine Which Data to Include in the Dataset

First, determine what data you want to include in the dataset. For this example, you will create an OppTeamMember dataset that contains only opportunities associated with an opportunity team.

You will obtain opportunities from the Opportunity object and the opportunity teams from the OpportunityTeamMember object. Both are Salesforce objects.

In this example, your Salesforce organization has the following opportunity team and users.

![Opportunity Team Diagram]

Your organization also contains the following opportunities, most of which are owned by Keith.

![Opportunities Table]

Acc - 1000 Widgets is the only opportunity shared by an opportunity team. Bill is the Sales Manager for this opportunity. Tony is the opportunity owner.
Design the Dataflow to Load the Data

Now it’s time to figure out how the dataflow will extract the Salesforce data and load it into a dataset. You start by creating this high-level design for the dataflow.

The dataflow will extract data from the Opportunity and OpportunityTeamMember objects, join the data, and then load it into the OppTeamMember dataset.

Now let’s implement that design in JSON, which is the format of the dataflow definition file. A dataflow definition file contains transformations that extract, transform, and load data into a dataset.

Based on the design, you create the JSON shown below.

```json
{
  "Extract_OpportunityTeamMember": {
    "action": "sfdcDigest",
    "parameters": {
      "object": "OpportunityTeamMember",
      "fields": [
        { "name": "Name" },
        { "name": "OpportunityId" },
        { "name": "UserId" }
      ]
    }
  },
  "Extract_Opportunity": {
    "action": "sfdcDigest",
    "parameters": {
      "object": "Opportunity",
      "fields": [
        { "name": "Id" },
        { "name": "Name" },
        { "name": "Amount" },
        { "name": "StageName" },
        { "name": "AccountId" },
        { "name": "OwnerId" }
      ]
    }
  },
  "Augment_OpportunityTeamMember_Opportunity": {
    "action": "augment",
    "parameters": {
      "left": "Extract_OpportunityTeamMember",
      "left_key": [ "OpportunityId" ],
      "relationship": "TeamMember",
      "right": "Extract_Opportunity",
      "right_key": [ "Id" ]
    }
  }
}
```
If you were to run this dataflow, Analytics would generate a dataset with no row-level security. As a result, any user that has access to the dataset would be able to see the opportunity shared by the opportunity team.

For example, as shown below, Lucy would be able to view the opportunity that belongs to an opportunity team of which she is not a member.

You need to apply row-level security to restrict access to records in this dataset.
Determine Row-Level Security for the Dataset

Now it’s time to think about row-level security. How will you restrict access to each record in this dataset?

You decide to implement the following predicate on the dataset.

'UserId' == "$User.Id"

This predicate compares the UserId column in the dataset against the ID of the user running a query against the dataset. The UserId column in the dataset contains the user ID of the team member associated with each opportunity. To determine the ID of the user running the query, Analytics looks up the ID of the user making the query in the User object.

For each match, Analytics returns the record to the user.

Modify the Dataflow Based on Row-Level Security

It’s now time to add the predicate in the dataflow definition file.

You add the predicate to the Register transformation that registers the OppTeamMember dataset as shown below.

```json
{
    "Extract_OpportunityTeamMember": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "OpportunityTeamMember",
            "fields": [
                { "name": "Name" },
                { "name": "OpportunityId" },
                { "name": "UserId" }
            ]
        }
    },
    "Extract_Opportunity": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Opportunity",
            "fields": [
                { "name": "Name" },
                { "name": "OpportunityId" },
                { "name": "OpportunityType" },
                { "name": "Stage" },
                { "name": "CloseDate" }
            ]
        }
    }
}
```
"parameters": {
  "object": "Opportunity",
  "fields": [
    { "name": "Id" },
    { "name": "Name" },
    { "name": "Amount" },
    { "name": "StageName" },
    { "name": "AccountId" },
    { "name": "OwnerId" }
  ]
}
,"Augment_OpportunityTeamMember_Opportunity": {
  "action": "augment",
  "parameters": {
    "left": "Extract_OpportunityTeamMember",
    "left_key": [
      "OpportunityId"
    ],
    "relationship": "TeamMember",
    "right": "Extract_Opportunity",
    "right_key": [
      "Id"
    ],
    "right_select": [
      "Name","Amount"
    ]
  }
},
"Register_Dataset": {
  "action": "sfdcRegister",
  "parameters": {
    "alias": "OppTeamMember",
    "name": "OppTeamMember",
    "source": "105_Augment_OpportunityTeamMember_Opportunity",
    "rowLevelSecurityFilter": "'UserId' == "$User.Id""
  }
}
}
Create the Dataset

Now that you have the final dataflow definition file, you can create the dataset.

**Warning:** If you wish to perform the steps in this sample implementation, verify that you have all required Salesforce objects and fields, and perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created. Also, always make a backup of the existing dataflow definition file before you make changes because you cannot retrieve old versions of the file.

To create the dataset, perform the following steps.

1. In Analytics, click the gear icon (⚙️) and then select *Monitor* to open the monitor. The Jobs view of the monitor appears by default.
2. Select *Dataflow View*.
3. Click the actions list (1) for the dataflow and then select *Download* to download the existing dataflow definition file.
4. Open the dataflow definition file in a JSON or text editor.
5. Add the JSON determined in the previous step.
6. Before you save the dataflow definition file, use a JSON validation tool to verify that the JSON is valid. An error occurs if you try to upload the dataflow definition file with invalid JSON. You can find JSON validation tool on the internet.
7. Save and close the dataflow definition file.
8. In the Dataflow View of the monitor, click the actions list for the dataflow and then select *Upload*.
9. Select the updated dataflow definition file and click *Upload*.
10. In the Dataflow View of the monitor, click the actions list for the dataflow and then select *Run* to run the dataflow job.
11. Click the *Refresh Jobs* button (🔄) to view the latest status of the dataflow job. You can view the *OppTeamMember* dataset after the dataflow job completes successfully.

**Note:** If you are adding a predicate to a dataset that was previously created, each user must log out and log back in for the predicate to take effect.
Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

1. Log in to Analytics as Lucy.
2. Open the OppTeamMember opportunity.
   Notice that Lucy can’t view the opportunity associated with the opportunity team anymore because she is not a member of the team.

3. Log out and now log in as Bill.
   Bill can view the opportunity that is shared by the opportunity team of which he is a member.
Row-Level Security Example based on Role Hierarchy and Record Ownership

Let’s look at an example where you create a dataset based on Salesforce data and then implement row-level security based on the Salesforce role hierarchy and record ownership. In this example, you will create a dataset that contains all opportunities. To restrict access on each record in the dataset, you will create a security policy where each user can view only opportunities that they own or that are owned by their subordinates based on the Salesforce role hierarchy. This process requires multiple steps that are described in the sections that follow.

1. **Determine Which Data to Include in the Dataset**
   First, determine what data you want to include in the dataset. For this example, you will create the OppRoles dataset that contains all opportunities as well as user details about each opportunity owner, such as their full name, division, and title.

2. **Design the Dataflow to Load the Data**
   Now it’s time to figure out how the dataflow will extract the data and load it into a dataset. You start by creating this high-level design for the dataflow.

3. **Determine Row-Level Security for the Dataset**
   Now it’s time to think about row-level security. How will you restrict access to each record in this dataset?

4. **Modify the Dataflow Based on Row-Level Security**
   Now it’s time to modify the dataflow definition file to account for the predicate.

5. **Create the Dataset**
   Now that you have the final dataflow definition file, you can create the dataset.

6. **Test Row-Level Security for the Dataset**
   You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

**Determine Which Data to Include in the Dataset**

First, determine what data you want to include in the dataset. For this example, you will create the OppRoles dataset that contains all opportunities as well as user details about each opportunity owner, such as their full name, division, and title.

You will obtain opportunities from the Opportunity object and user details from the User object. Both are objects in Salesforce.

In this example, your Salesforce organization has the following role hierarchy and users:
Also, your organization contains the following opportunities, most of which are owned by Keith.

Design the Dataflow to Load the Data

Now it's time to figure out how the dataflow will extract the data and load it into a dataset. You start by creating this high-level design for the dataflow.

The dataflow will extract data from the Opportunity and User objects, join the data, and then load it into the OppRoles dataset.

Now let's implement that design in JSON, which is the format of the dataflow definition file. A dataflow definition file contains transformations that extract, transform, and load data into a dataset.

Based on the design, you create the JSON shown below.

```json
{
    "Extract_Opportunity": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Opportunity",
            "fields": [
                { "name": "Id" },
                { "name": "Name" },
                { "name": "Amount" },
                { "name": "StageName" },
                { "name": "AccountId" },
                { "name": "OwnerId" }
            ]
        }
    },
    "Extract_User": {
```
If you were to run this dataflow, Analytics would generate a dataset with no row-level security. As a result, any user that has access to the dataset would be able to view all opportunities. For example, as shown below, Bill would be able to view all opportunities, including those owned by his manager Keith.
You need to apply row-level security to restrict access to records in this dataset.

**Determine Row-Level Security for the Dataset**

Now it’s time to think about row-level security. How will you restrict access to each record in this dataset?

You decide to implement the following predicate on the dataset.

'ParentRoleIDs' == "$User.UserRoleId" || 'OwnerId' == "$User.Id"

**Note:** The current dataflow doesn’t contain logic to create a dataset column named “ParentRoleIDs.” ParentRoleIDs is a placeholder for the name of a column that will contain this information. In the next step, you will modify the dataflow to add this column to the dataset. This column name will change based on how you configure the dataflow.

Based on the predicate, Analytics returns an opportunity record if:

- The user who submits the query is a parent of the opportunity owner based on the Salesforce role hierarchy. Analytics determines this based on their role IDs and the role hierarchy.
- Or, the user who submits the query on the dataset is the opportunity owner.

Let’s examine both parts of this predicate.
Modify the Dataflow Based on Row-Level Security

Now it's time to modify the dataflow definition file to account for the predicate. In this scenario, you have to make changes to the dataflow based on the predicate.

- Add a column in the dataset that stores a comma-separated list of the role IDs of all parents for each opportunity owner. When you defined the predicate in the previous step, you temporarily referred to this column as “ParentRoleIDs.” To add the column, you redesign the dataflow as shown in the following diagram:

The new dataflow design contains the following changes:
- Extracts the role IDs from the UserRole object.
- Uses the Flatten transformation to generate a column that stores a comma-separated list of the role IDs of all parents of each user. When you determined the predicate in the previous step, you temporarily referred to this column as “ParentRoleIDs.”
- Link the new column to the OppRoles dataset.
Add the predicate to the Register transformation that registers the OppRoles dataset.

You modify the dataflow as shown below:

```json
{
"Extract_Opportunity": {
"action": "sfdcDigest",
"parameters": {
"object": "Opportunity",
"fields": [
{ "name": "Id" },
{ "name": "Name" },
{ "name": "Amount" },
{ "name": "StageName" },
{ "name": "AccountId" },
{ "name": "OwnerId" }
]
},
"Extract_User": {
"action": "sfdcDigest",
"parameters": {
"object": "User",
"fields": [
{ "name": "Id" },
{ "name": "Username" },
{ "name": "LastName" },
{ "name": "FirstName" },
{ "name": "Name" },
{ "name": "CompanyName" },
{ "name": "Division" },
{ "name": "Department" },
{ "name": "Title" },
{ "name": "Alias" },
{ "name": "CommunityNickname" },
{ "name": "UserType" },
{ "name": "UserRoleId" }
]
},
"Extract_UserRole": {
"action": "sfdcDigest",
"parameters": {
"object": "UserRole",
"fields": [
{ "name": "Id" },
{ "name": "ParentRoleId" },
{ "name": "RollupDescription" },
{ "name": "OpportunityAccessForAccountOwner" },
{ "name": "CaseAccessForAccountOwner" },
{ "name": "ContactAccessForAccountOwner" },
{ "name": "ForecastUserId" },
{ "name": "MayForecastManagerShare" },
{ "name": "LastModifiedDate" },
{ "name": "LastModifiedById" },
```
Modify the Dataflow Based on Row-Level Security

Row-Level Security for Datasets

```json
{
  "name": "SystemModstamp",
  "name": "DeveloperName",
  "name": "PortalAccountId",
  "name": "PortalType",
  "name": "PortalAccountOwnerId"
}

"Flatten_UserRole": {
  "action": "flatten",
  "parameters": {
    "multi_field": "Roles",
    "parent_field": "ParentRoleId",
    "path_field": "RolePath",
    "self_field": "Id",
    "source": "Extract_UserRole"
  }
},

"Augment_User_FlattenUserRole": {
  "action": "augment",
  "parameters": {
    "left": "Extract_User",
    "left_key": [ "UserRoleId"
    ],
    "relationship": "Role",
    "right": "Flatten_UserRole",
    "right_key": [ "Id"
    ],
    "right_select": [ "Roles",
                     "RolePath"
    ]
  }
},

"Augment_Opportunity_UserWithRoles": {
  "action": "augment",
  "parameters": {
    "left": "Extract_Opportunity",
    "left_key": [ "OwnerId"
    ],
    "relationship": "Owner",
    "right": "Augment_User_FlattenUserRole",
    "right_select": [ "Name",
                      "Role.Roles",
                      "Role.RolePath"
    ],
    "right_key": [ "Id"
    ]
  }
}
```
Create the Dataset

Now that you have the final dataflow definition file, you can create the dataset.

**Warning:** If you wish to perform the steps in this sample implementation, verify that you have all required Salesforce objects and fields, and perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created. Also, always make a backup of the existing dataflow definition file before you make changes because you cannot retrieve old versions of the file.

To create the dataset, perform the following steps.

1. In Analytics, click the gear icon ( ) and then select **Data Monitor** to open the data monitor. The Jobs View of the data monitor appears by default.

2. Select **Dataflow View**.

3. Click the actions list (1) for the dataflow and then select **Download** to download the existing dataflow definition file.

4. Open the dataflow definition file in a JSON or text editor.

**Note:** In this example, the dataset has columns Owner.Role.Roles and OwnerId. A user can view the values of these columns for each record to which they have access.
5. Add the JSON determined in the previous step.

6. Before you save the dataflow definition file, use a JSON validation tool to verify that the JSON is valid. An error occurs if you try to upload the dataflow definition file with invalid JSON. You can find JSON validation tool on the internet.

7. Save and close the dataflow definition file.

8. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select Upload.

9. Select the updated dataflow definition file and click Upload.

10. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select Run to run the dataflow job.

11. Click the Refresh Jobs button ( ) to view the latest status of the dataflow job. You can view the OppRoles dataset after the dataflow job completes successfully.

Note: If you are adding a predicate to a dataset that was previously created, each user must log out and log back in for the predicate to take effect.

Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

1. Log in to Analytics as Bill.

2. Open the OppRoles opportunity. Notice that Bill can’t see his manager Keith’s opportunities anymore. Now, he can see only his opportunity and his subordinate Tony’s opportunity.
3. Log out and now log in as Keith.

As expected, Keith can still see all opportunities.

Row-Level Security Example Based on Territory Management

Let’s look at an example where you create a dataset based on Salesforce data and then implement row-level security based on your defined territories. In this example, you determine what model you use for territory management, so you can later review sample JSON for that dataset. To restrict access on each record in the dataset, you will create a security predicate where each user can view only data appropriate for the territory to which they belong.

Territory management is an account sharing system that grants access to accounts based on the characteristics of the accounts. It enables your company to structure your Salesforce data and users the same way you structure your sales territories.

If your organization has a private sharing model, you might have granted users access to accounts based on criteria such as postal code, industry, revenue, or a custom field that is relevant to your business. Perhaps you also need to generate forecasts for these diverse categories of accounts. Territory management solves these business needs and provides a powerful solution for structuring your users, accounts, and their associated contacts, opportunities, and cases.
1. **Determine How You Use Territory Management**
   When working with security related to territory management, it helps to know how your organization implements territory management. Usually, one of 2 methods are used. Either accounts are assigned to regions manually, following some organization-specific precedence, or the organization use’s Salesforce’s territory hierarchy feature.

2. **Create the DataSet**
   Now we look at sample JSON code that describes territory management in a dataset.

3. **Create the Security Predicate**
   Now we can apply a security predicate to filter the dataset.

---

**Determine How You Use Territory Management**

When working with security related to territory management, it helps to know how your organization implements territory management. Usually, one of 2 methods are used. Either accounts are assigned to regions manually, following some organization-specific precedence, or the organization use’s Salesforce’s territory hierarchy feature.

**The manual process**

For this example, any account with a Billing State or Province that is North Pole is manually assigned to the Canada region.

**Territory Management hierarchies**
For this example, we have a user called North America VP who needs access to all accounts in the Canada, Mexico, and US territories. We also have a user called Rep1 Canada who should only have access to the accounts in the Canada territory, not Mexico or US, and nowhere above in the hierarchy.

Create the DataSet

Now we look at sample JSON code that describes territory management in a dataset. In this example, territory management data is stored on the following objects and fields.
Here is an example JSON file for this dataset.

```json
{
    "Extract_AccountShare": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "AccountShare",
            "fields": [
                { "name": "Id" },
                { "name": "RowCause" },
                { "name": "UserOrGroupId" },
                { "name": "AccountId" }
            ]
        }
    },
    "Extract_Group": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Group",
            "fields": [
                { "name": "Name" },
                { "name": "Type" },
                { "name": "Id" },
                { "name": "RelatedId" }
            ]
        }
    },
    "Extract_Territory": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Territory",
            "fields": [
                { "name": "Id" },
                { "name": "Name" },
                { "name": "ParentTerritoryId" }
            ]
        }
    },
    "Extract_User_Territory": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "UserTerritory",
            "fields": [
```
Row-Level Security for Datasets

Create the DataSet

{ "name": "TerritoryId" },
{ "name": "UserId"
}
,
"Extract_User": {
"action": "sfdcDigest",
"parameters": {
"object": "User",
"fields": [
{ "name": "Id" },
{ "name": "Name" }
]
}
,
"Extract_Account": {
"action": "sfdcDigest",
"parameters": {
"object": "Account",
"fields": [
{ "name": "Id" },
{ "name": "Name" },
{ "name": "BillingCountry" }
]
}
,
"Augment_TerritoryUsers": {
"action": "augment",
"parameters": {
"left": "Extract_Territory",
"left_key": [ "Id" ],
"relationship": "TerritoryId",
"right": "Extract_User_Territory",
"right_key": [ "TerritoryId" ],
"right_select": [ "UserId" ],
"operation": "LookupMultiValue"
}
,
"Augment_AccountShare_To_Territory_Groups": {
"action": "augment",
"parameters": {
"left": "Augment_AccountShare_To_Account",
"left_key": [ "UserOrGroupId" ],
"relationship": "UserOrGroupId",
"right": "Extract_Group",
"right_key": [ "TerritoryId" ]
]
When run, this JSON file results in a list of accounts. In this example, a list of 5:

Create the DataSetRow-Level Security for Datasets
Create the Security Predicate

Now we can apply a security predicate to filter the dataset.

Using this example, the following security predicate on the dataset enforces the territory management security rules.

`'Territory.TerritoryId.UserId' == "$User.Id" || 'UserOrGroupId' == "$User.Id"`

**Note:** Update the dataset, and then log out of and back in to the org so you see the changes.

Now you see only 2 accounts - Global Media because it is in the Canada territory, and Santa’s Workshop because of the manual rule.

Salesforce Sharing Inheritance for Datasets

Use sharing inheritance to let Analytics use the same sharing rules for your datasets as Salesforce uses for your objects.

As a Salesforce administrator, you likely use sharing rules so that users have access to data appropriate to their roles. These sharing rules include rules for Apex or custom sharing and manual, hierarchy, role, group, and team-based sharing. For more information, see Sharing Settings.

For supported objects, administrators can enable sharing inheritance in Analytics to use the Salesforce sharing settings in Analytics. When creating your datasets during the ELT (extract, load, and transform) process, or when editing existing datasets, specify the objects that use sharing.

**Limitations**

These limitations could result in leaking information when using sharing inheritance.
• You can inherit sharing settings only from one object, regardless of how many source objects are used in creating a dataset. Because the dataset can be constructed from many objects, each object could be using a different security model.

• The computeRelative and delta dataflow transformations could merge information from records with different security, which can result in leaking information when using sharing inheritance.

• Calculated fields are treated as normal fields. Row-level security applied during the calculation in Salesforce is ignored.

**Important:** If your dataflow doesn’t do a full extraction each time it runs, be sure to evaluate whether security drift is a risk for the datasets you bring into Analytics. Consider whether to use periodic full sync. For more information, see Security Metadata Drift.

Here are some other limitations of the Analytics sharing inheritance feature.

• A dataset using sharing must also have a security predicate defined.

• If a user can see more than 3,000 records on the object in Salesforce but the user does not have the “View All Data” permission, sharing inheritance is not used. The backup security predicate takes effect. This limitation does not apply to the Opportunity object if the critical update “Increase Coverage of Einstein Analytics Sharing Inheritance for Opportunity” is enabled.

• If an opportunity has more than 150 sharing rules, you can’t use the Opportunity object as a sharing source for a dataflow.

• Sharing inheritance might not be used when many sharing settings or rules control access to an opportunity, such as in the following situations:
  – A user has membership in many public groups, each of which grants access to the opportunity.
  – A user is near (or at) the top of role or territory hierarchies in an org that has a complex role or territory configuration.

  In this situation, the backup security predicate takes effect.

• Sharing isn’t automatically applied to datasets. You apply sharing to each dataset manually.

• Changes to the rowLevelSharingSource or rowLevelSecurityFilter security settings in a dataflow have no effect on datasets that exist. Change those settings on the edit dataset page.

• For an object to appear in the security-sharing source list, the primary key of the custom object must be a field in the dataset. A foreign key doesn’t satisfy this requirement. For example, if you have Opportunity.AccountId in your dataset but not Account.Id, you can’t inherit sharing from the Account object.

**Note:** Sharing inheritance can affect the performance of queries and dataflows. If your requirements include best-possible performance, use security predicates instead of sharing inheritance. If not, enjoy the convenience of sharing inheritance.

**Set Up Sharing Inheritance**

To enable sharing inheritance in Analytics, specify which datasets inherit sharing rules, and set a default security predicate.

**Enable or Disable Sharing Inheritance**

Sharing inheritance is turned on by default in new orgs.

1. From Setup, in the Quick Find box, enter *Analytics*, and then click *Settings*.
2. Select *Inherit sharing from Salesforce*, and click *Save*.
Enable Sharing Inheritance When Using Data Sync

If your org has data sync enabled, enable sharing inheritance for each object you want to use as a sharing source.

1. In Analytics Studio, click Data Manager.
2. In Data Manager, click Connect.
3. On the right end of the row for the object you want to enable, click the dropdown list.
4. Click Row Level Sharing.
5. Click Sharing inheritance on.
6. Click Save.

Enhance Sharing Inheritance for the Opportunity Object

Sharing inheritance does have limitations. (See “Limitations” in Salesforce Sharing Inheritance for Datasets.) You can increase availability of sharing inheritance for the Opportunity object by enabling the critical update “Increase Coverage of Einstein Analytics Sharing Inheritance for Opportunity”.

To get the enhanced coverage, the Opportunity object must be the sharing source for both your dataflow and your dataset.

Configure Dataflows

For each dataset that you want to inherit sharing, modify the dataflow.

1. Specify the source object and a default security predicate. For more information, see Configure the Dataflow.
2. Add the rowLevelSharingSource parameter to the sfdcRegister node parameters for the dataset. For more information, see sfdcRegister. The rowLevelSharingSource parameter takes a string, which is the API name for the object from which to inherit sharing. In this example, the parameter specifies that the Salesforce sharing rules on the Opportunity object should be inherited.

```json
"reg": { 
  "action": "sfdcRegister",
  "parameters": { 
    "source": "Opportunity_final",
    "name": "Opportunity w/ Account",
    "alias": "Oppty_w_Acct",
    "rowLevelSharingSource": "Opportunity",
    "rowLevelSecurityFilter": "'OwnerId' == "$User.Id"
  }
},
```

3. When setting rowLevelSharingSource, you must also set the security predicate (rowLevelSecurityFilter). In the example, when sharing limits are exceeded, users see only the opportunities that they own. Set the security predicate to false to block all users not covered by sharing.

Configure Datasets

On datasets that exist, changes to security settings in a dataflow have no effect. You must also change those settings on the edit dataset page.

**Note:** If the settings in the dataset and dataflow don’t match, you see a warning that says, “The sharing source and security predicate in this dataset version must be the same as in the dataflow”.

37
1. Edit the dataset. For more information, see Edit a Dataset.

2. For Sharing Source, select the API name for the object. Only valid objects are displayed in the list. For example, the primary key of the object must be a field in the dataset.

3. For Security Predicate, enter the default security predicate. If you use sharing inheritance, you must also specify a default predicate. Set the security predicate to false to block all users not covered by sharing. This security predicate is the default when sharing is enabled on existing datasets.
Predicate Expression Syntax for Datasets

You must use valid syntax when defining the predicate expression.
The predicate expression must have the following syntax:

\[ \text{dataset column} \ \text{operator} \ \text{value} \]

For example, you can define the following predicate expression for a dataset:

\['UserId' == "$User.Id"\]

You can create more complex predicate expressions such as:

\[ ('Expected_Revenue' > 4000 || 'Stage Name' == "Closed Won") && 'isDeleted' != "False"\]

Consider the following requirements for the predicate expression:

- The expression is case-sensitive.
- The expression cannot exceed 1,000 characters.
- There must be at least one space between the dataset column and the operator, between the operator and the value, and before and after logical operators. This expression is not valid: \`Revenue\’ > 100. It must have spaces like this: \`Revenue\’ > 100.

If you try to apply a predicate to a dataset and the predicate is not valid, an error appears when any user tries to query the dataset.

Dataset Columns in a Predicate Expression
You include at least one dataset column as part of the predicate expression.

Values in a Predicate Expression
The value in the predicate expression can be a string literal or number literal. It can also be a field value from the User object in Salesforce.

Escape Sequences
You can use the backslash character (\) to escape characters in column names and string values in a predicate expression.

Character Set Support
Analytics supports UTF-8 characters in dataset column names and values in a predicate expression. Analytics replaces non-UTF-8 characters with the UTF-8 symbol ( \compatdoc{} ). If Analytics has to replace a non-UTF-8 character in a predicate expression, users may experience unexpected query results.

Special Characters
Certain characters have a special meaning in Analytics.

Operators
You can use comparison operators and logical operators in predicate expressions.
Consider the following requirements for dataset columns in a predicate expression:

- Column names are case-sensitive.
- Column names must be enclosed in single quotes ('). For example, 'Region' == "South"

**Note:** A set of characters in double quotes is treated as a string rather than a column name.

- Single quotes in column names must be escaped. For example, 'Team\'s Name' == "West Region Accounts"

### Values in a Predicate Expression

The value in the predicate expression can be a string literal or number literal. It can also be a field value from the User object in Salesforce. Consider the following requirements for each value type.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Requirements</th>
<th>Predicate Expression Examples</th>
</tr>
</thead>
</table>
| string literal | Enclose in double quotes and escape the double quotes. | • 'Owner' == "Amber"
               |                                                          | • 'Stage Name' == "Closed Won"                                                        |
| number literal | Can be a float or long datatype. Do not enclose in quotes. | • 'Expected_Revenue' >= 2000.00
               |                                                          | • 'NetLoss' < -10000                                                                |
| field value    | When referencing a field from the User object, use the $User.[field] syntax. Use the API name for the field. You can specify standard or custom fields of type string, number, or multi-value picklist. When you define a predicate for a dataset, you must have read access on all User object fields. | • 'Owner.Role' == "$User.UserRoleId"
               |                                                          | • 'GroupID' == "$User.UserGroupId__c" |

**Note:** Supported User object field value types are string, number, and multi-value picklist. Other types (for...
<table>
<thead>
<tr>
<th>Value Type</th>
<th>Requirements</th>
<th>Predicate Expression Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields used to create the predicate expression. However, when a user queries a dataset that has a predicate based on the User object, Analytics uses the access permissions of the Insights Security User to evaluate the predicate expression based on the User object.</td>
<td>example, boolean) are not supported.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** By default, the Security User does not have access permission on custom fields of the User object.

To grant the Security User read access on a field, set field-level security on the field in the user profile of the Security User.

SEE ALSO:

*Salesforce Data Access in Analytics*

## Escape Sequences

You can use the backslash character (\) to escape characters in column names and string values in a predicate expression.

You can use the \ escape sequence to escape a single quote in a column name. For example:

```plaintext
'Team\'s Name' == "West Region Accounts"
```

You can use the following escape sequences for special characters in string values.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>One backspace character</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>\Z</td>
<td>CTRL+Z (ASCII 26)</td>
</tr>
<tr>
<td>&quot;</td>
<td>One double-quote character</td>
</tr>
<tr>
<td>\</td>
<td>One backslash character</td>
</tr>
<tr>
<td>\0</td>
<td>One ASCII null character</td>
</tr>
</tbody>
</table>
Character Set Support

Analytics supports UTF-8 characters in dataset column names and values in a predicate expression. Analytics replaces non-UTF-8 characters with the UTF-8 symbol (メディカル).
If Analytics has to replace a non-UTF-8 character in a predicate expression, users may experience unexpected query results.

Special Characters

Certain characters have a special meaning in Analytics.

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>Single quote</td>
<td>Encloses a dataset column name in a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Expected_Revenue' &gt;= 2000.00</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
<td>Encloses a string value or field value in a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'OpportunityOwner' == &quot;Michael Vesti&quot;</td>
</tr>
<tr>
<td>()</td>
<td>Parentheses</td>
<td>Enforces the order in which to evaluate a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>('Expected_Revenue' &gt; 4000</td>
</tr>
<tr>
<td>$</td>
<td>Dollar sign</td>
<td>Identifies the Salesforce object in a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: You can only use the User object in a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Owner.Role' == &quot;$User.UserRoleId&quot;</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
<td>Separates the object name and field name in a predicate expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
</tbody>
</table>
Operators

You can use comparison operators and logical operators in predicate expressions.

Comparison Operators

Comparison operators return true or false.

Logical Operators

Logical operators return true or false.

Comparison Operators

Comparison operators return true or false.

Analytics supports the following comparison operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equals</td>
<td>True if the operands are equal. String comparisons that use the equals operator are case-sensitive. Example predicate expressions: 'Stage Name' == &quot;Closed Won&quot;</td>
</tr>
<tr>
<td>!=</td>
<td>Not equals</td>
<td>True if the operands are not equal. String comparisons that use the not equals operator are case-sensitive. Example predicate expression: 'isDeleted' != &quot;False&quot;</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>True if the left operand is less than the right operand. Example predicate expression: 'Revenue' &lt; 100</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less or equal</td>
<td>True if the left operand is less than or equal to the right operand.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>True if the left operand is greater than the right operand.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater or equal</td>
<td>True if the left operand is greater than or equal to the right operand.</td>
</tr>
<tr>
<td>in</td>
<td>Multi-value list filter</td>
<td>True if the left operand exists in the list of strings substituted for a multi-value picklist (field value). Example predicate expression: 'Demog' in ['$User.Demographic__c']</td>
</tr>
</tbody>
</table>

In this example, Demographic__c is of type multiPicklistField. During evaluation, the multi-value picklist field is substituted by a list of strings, with 1 string per user-selected item.
You can use the <, <=, >, and >= operators with measure columns only.

**Logical Operators**

Logical operators return true or false.

Analytics supports the following logical operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>True if both operands are true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example predicate expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Stage Name' == &quot;Closed Won&quot; &amp;&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'isDeleted' != &quot;False&quot;</td>
</tr>
</tbody>
</table>

| ||       | Logical OR | True if either operand is true.                  |
|          |            | Example predicate expression:                   |
|          |            | 'Expected_Revenue' > 4000 || 'Stage Name'    |
|          |            | == "Closed Won"                                 |

**Sample Predicate Expressions for Datasets**

Review the samples to see how to structure a predicate expression.

The samples are based on the following Opportunity dataset.

<table>
<thead>
<tr>
<th>Opportunity</th>
<th>Expected_Rev</th>
<th>Owner</th>
<th>OwnerRoleID</th>
<th>Stage_Name</th>
<th>IsDeleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>OppA</td>
<td>2000.00</td>
<td>Bill</td>
<td>20</td>
<td>Prospecting</td>
<td>True</td>
</tr>
<tr>
<td>OppB</td>
<td>3000.00</td>
<td>Joe</td>
<td>22</td>
<td>Closed Won</td>
<td>False</td>
</tr>
<tr>
<td>OppC</td>
<td>1000.00</td>
<td>可爱的花</td>
<td>36</td>
<td>Closed Won</td>
<td>False</td>
</tr>
<tr>
<td>OppD</td>
<td>5000.00</td>
<td>O'Fallon</td>
<td>18</td>
<td>Prospecting</td>
<td>True</td>
</tr>
<tr>
<td>OppE</td>
<td></td>
<td>Joe</td>
<td>22</td>
<td>Closed Won</td>
<td>True</td>
</tr>
</tbody>
</table>

Let’s take a look at some examples to understand how to construct a predicate expression.
<table>
<thead>
<tr>
<th>Predicate Expression</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>'OwnerRoleID' == &quot;$User.UserRoleID&quot;</td>
<td>Checks column values in the User object.</td>
</tr>
<tr>
<td>'Expected_Rev' &gt; 1000 &amp;&amp; 'Expected_Rev' &lt;= 3000</td>
<td></td>
</tr>
<tr>
<td>'Owner' = &quot;Joe&quot;</td>
<td></td>
</tr>
<tr>
<td>('Expected_Rev' &gt; 4000</td>
<td></td>
</tr>
<tr>
<td>'Stage Name' == &quot;Closed Won&quot; &amp;&amp; 'Expected_Rev' &gt; 70000</td>
<td></td>
</tr>
<tr>
<td>'Owner' == &quot;可爱的花&quot;</td>
<td>String contains Unicode characters.</td>
</tr>
<tr>
<td>'Owner' == &quot;O'Fallon&quot;</td>
<td>Single quote in a string requires the escape character.</td>
</tr>
<tr>
<td>'Stage Name' == &quot;&quot;</td>
<td>Checks for an empty string.</td>
</tr>
</tbody>
</table>