

salesforce

Analytics Security Implementation Guide

Salesforce, Winter '19



 @salesforcedocs

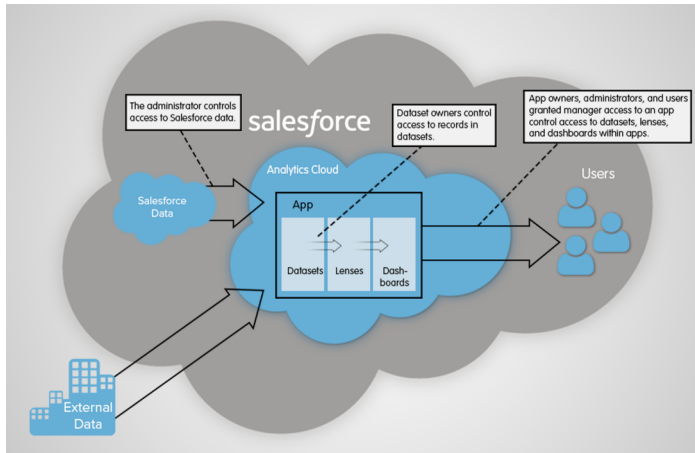
Last updated: December 4, 2018

CONTENTS

SECURITY FOR SALESFORCE ANALYTICS	1
SALESFORCE DATA ACCESS IN ANALYTICS	2
APP-LEVEL SHARING	4
ROW-LEVEL SECURITY FOR DATASETS	6
Security Predicates for Datasets	6
Row-Level Security Example based on Record Ownership	7
Row-Level Security Example based on Opportunity Teams	12
Row-Level Security Example based on Role Hierarchy and Record Ownership	20
Row-Level Security Example Based on Territory Management	31
Salesforce Sharing Inheritance for Datasets	36
SECURITY PREDICATE REFERENCE	40
Predicate Expression Syntax for Datasets	40
Sample Predicate Expressions for Datasets	45

SECURITY FOR SALESFORCE ANALYTICS

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.

SALESFORCE DATA ACCESS IN ANALYTICS

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**.
 - a. To enable permission on the object, select **Read** in the Object Permissions section.

USER PERMISSIONS

To clone a user profile:

- Manage Profiles and Permission Sets

To edit object permissions:

- Manage Profiles and Permission Sets
- AND
- Customize Application


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.

Action	Viewer	Editor	Manager
View dashboards, lenses, and datasets in the app	X	X	X
 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.			
See who has access to the app	X	X	X
Save contents of the app to another app that the user has Editor or Manager access to	X	X	X
Save changes to existing dashboards, lenses, and datasets in the app (saving dashboards requires the appropriate permission set license and permission)		X	X
Change the app's sharing settings			X
Rename the app			X
Delete the app			X

 **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.


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.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance, and Unlimited** Editions. Also available in **Developer Edition**.

USER PERMISSIONS

To share an app:

- Use Analytics and Manager access to the app

ROW-LEVEL SECURITY FOR DATASETS

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.

[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.

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 *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.

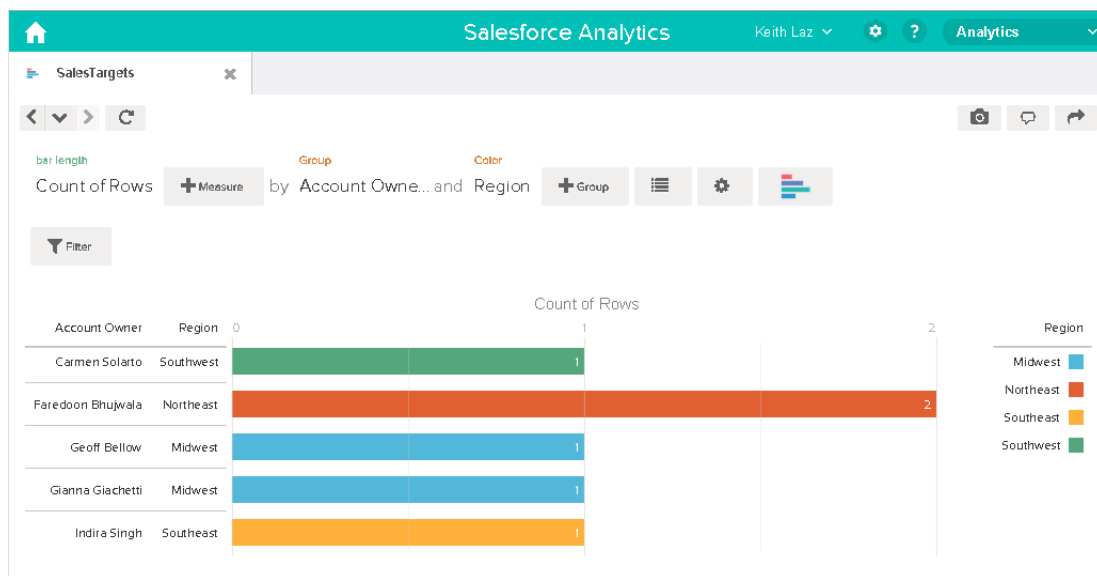
Determine Which Data to Include in the Dataset

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.

AccountOwner	Region	Target	TargetDate
Tony Santos	Midwest	10000	1/1/2011
Lucy Timmer	Northeast	50000	1/1/2011
Lucy Timmer	Northeast	0	12/1/2013
Bill Rolley	Midwest	15000	1/1/2011
Keith Laz	Southwest	35000	1/1/2011
Lucy Timmer	Southeast	40000	1/1/2011

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.

```
{
  "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",
```


```

        "label": "Region",
        "type": "Text"
    },
    {
        "name": "Target",
        "fullyQualifiedName": "Targets.Target",
        "label": "Target",
        "type": "Numeric",
        "precision": 16,
        "scale": 0,
        "defaultValue": "0",
        "format": null
    },
    {
        "name": "TargetDate",
        "fullyQualifiedName": "Targets.TargetDate",
        "label": "TargetDate",
        "description": "",
        "type": "Date",
        "format": "dd/MM/yy HH:mm:ss",
        "isSystemField": false,
        "fiscalMonthOffset": 0
    }
}
]
}

```

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.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.



USER PERMISSIONS

To upload a CSV and metadata file:

- Upload External Data to Analytics

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

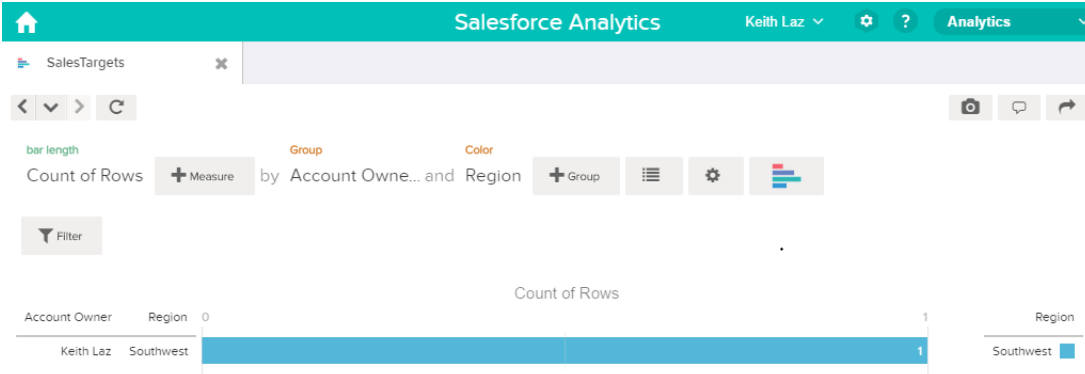
You must verify that the predicate is applied properly and that each user can see their own sales targets.

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](#)

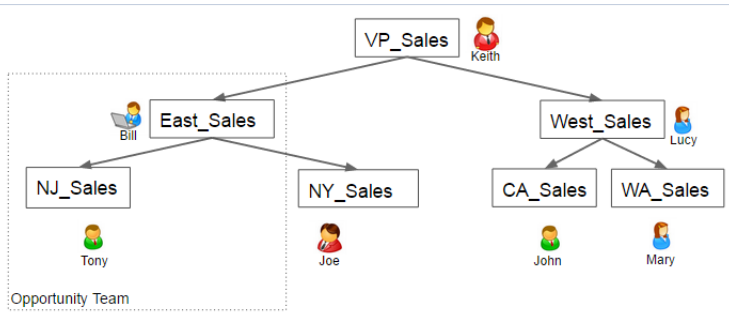
You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

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.



EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.

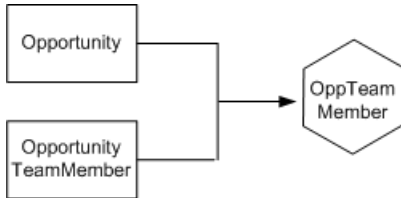
Your organization also contains the following opportunities, most of which are owned by Keith.

Action	Opportunity Name	Account Name	Amount	Close Date	Stage	Opportunity Owner Alias
<input type="checkbox"/> Edit Del +	Acc - 1000 Widgets	Acc_salesrep		9/4/2014	Prospecting	Tony
<input type="checkbox"/> Edit Del +	Acme - 1,200 Widgets	Acme	\$140,000.00	6/14/2012	Value Proposition	Keith
<input type="checkbox"/> Edit Del +	Acme - 200 Widgets	Acme	\$20,000.00	10/13/2012	Prospecting	Keith
<input type="checkbox"/> Edit Del +	Acme - 600 Widgets	Acme	\$70,000.00	8/10/2012	Needs Analysis	Keith
<input type="checkbox"/> Edit Del +	ESales_01	East_Sales_acc_01		9/4/2014	Prospecting	Bill
<input type="checkbox"/> Edit Del +	Global Media - 400...	Global Media	\$40,000.00	7/13/2012	Id. Decision Makers	Keith
<input type="checkbox"/> Edit Del +	salesforce.com - 1...	salesforce.com	\$100,000.00	6/14/2012	Negotiation/Review	Keith
<input type="checkbox"/> Edit Del +	salesforce.com - 2...	salesforce.com	\$20,000.00	8/12/2012	Value Proposition	Keith
<input type="checkbox"/> Edit Del +	salesforce.com - 50...	Global Media	\$50,000.00	5/12/2012	Closed Won	Keith
<input type="checkbox"/> Edit Del +	salesforce.com - 50...	Global Media	\$500,000.00	5/12/2012	Closed Won	Keith
<input type="checkbox"/> Edit Del +	West_Sales_01	West_Sales_Acc_01		9/4/2014	Prospecting	Lucy

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.

```

{
  "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",
    }
  }
}

```

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise**, **Performance**, and **Unlimited** Editions. Also available in **Developer Edition**.

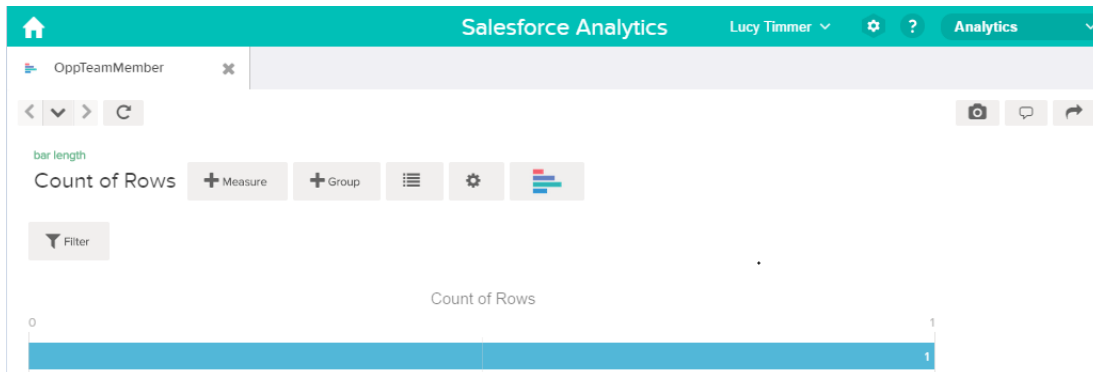
```

    "right": "Extract_Opportunity",
    "right_key": [
      "Id"
    ],
    "right_select": [
      "Name", "Amount"
    ]
  }
},
"Register_Dataset": {
  "action": "sfdcRegister",
  "parameters": {
    "alias": "OppTeamMember",
    "name": "OppTeamMember",
    "source": "Augment_OpportunityTeamMember_Opportunity",
    "rowLevelSecurityFilter": ""
  }
}
}

```

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.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.

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.

```
{
  "Extract_OppportunityTeamMember": {
    "action": "sfdcDigest",
    "parameters": {
      "object": "OppportunityTeamMember",
      "fields": [
        { "name": "Name" },
        { "name": "OppportunityId" },
        { "name": "UserId" }
      ]
    }
  },
  "Extract_Oppportunity": {
    "action": "sfdcDigest",
```

EDITIONS


Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.


```
"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 **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.

EDITIONS

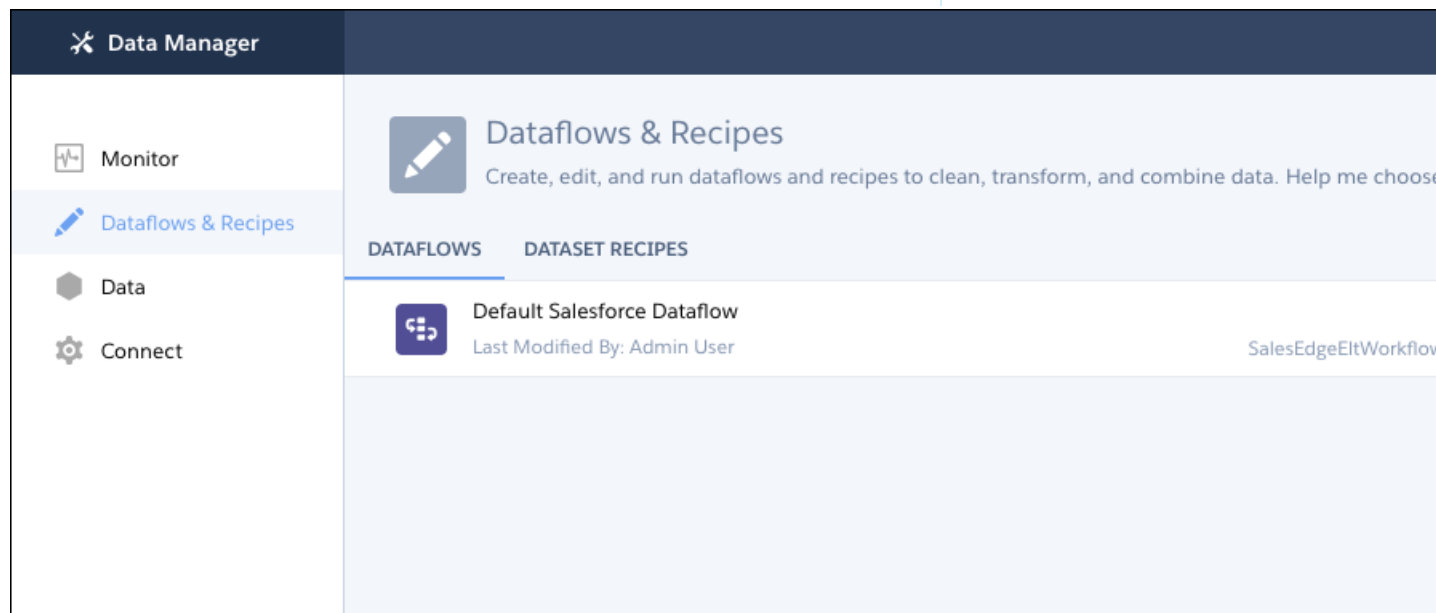
Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.

USER PERMISSIONS

To download, upload, run, and monitor a dataflow:

- Edit Analytics Dataflows



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 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 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

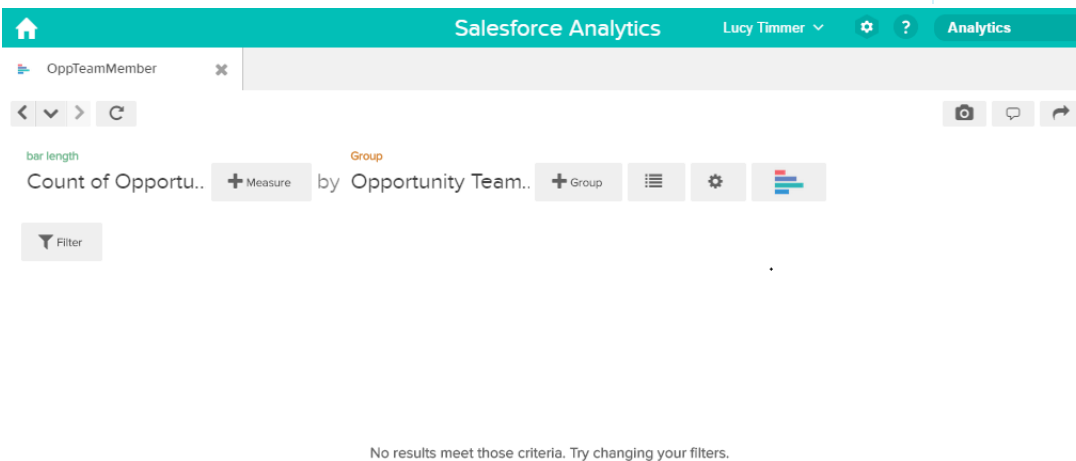
You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

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.

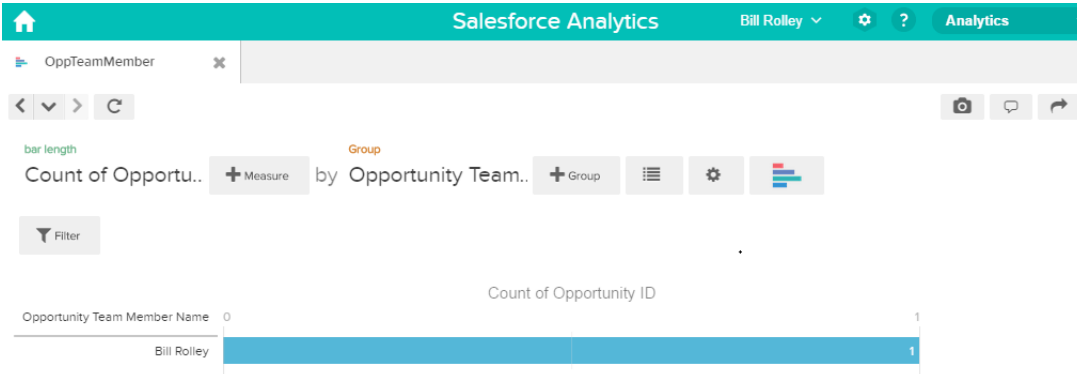
EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise**, **Performance**, and **Unlimited** Editions. Also available in **Developer Edition**.



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](#)

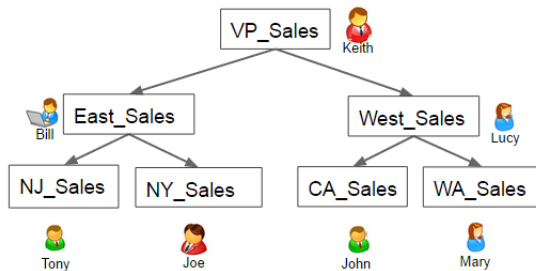
You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

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.

Action	Opportunity Name	Account Name	Amount	Close Date	Stage	Opportunity Owner Alias
Edit Del +	Acc - 1000 Widgets	Acc_salesrep		9/4/2014	Prospecting	Tony
Edit Del +	Acme - 1,200 Widgets	Acme	\$140,000.00	6/14/2012	Value Proposition	Keith
Edit Del +	Acme - 200 Widgets	Acme	\$20,000.00	10/13/2012	Prospecting	Keith
Edit Del +	Acme - 600 Widgets	Acme	\$70,000.00	8/10/2012	Needs Analysis	Keith
Edit Del +	ESales_01	East_Sales_acc_01		9/4/2014	Prospecting	Bill
Edit Del +	Global Media - 400...	Global Media	\$40,000.00	7/13/2012	Id. Decision Makers	Keith
Edit Del +	salesforce.com - 1...	salesforce.com	\$100,000.00	6/14/2012	Negotiation/Review	Keith
Edit Del +	salesforce.com - 2...	salesforce.com	\$20,000.00	8/12/2012	Value Proposition	Keith
Edit Del +	salesforce.com - 50...	Global Media	\$50,000.00	5/12/2012	Closed Won	Keith
Edit Del +	salesforce.com - 50...	Global Media	\$500,000.00	5/12/2012	Closed Won	Keith
Edit Del +	West_Sales_01	West_Sales_Acc_01		9/4/2014	Prospecting	Lucy

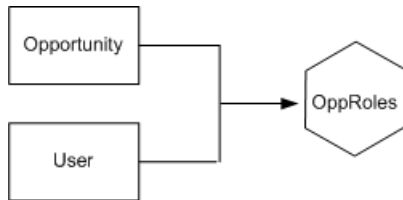
EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.

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.

```

{
  "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" }
      ]
    }
  }
}

```

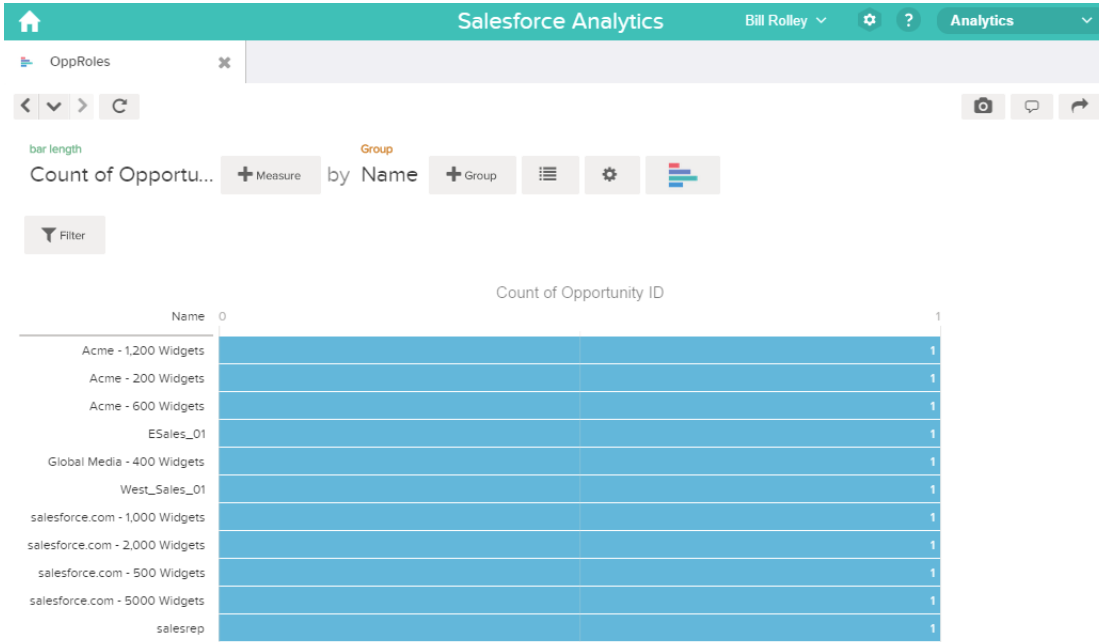
EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise**, **Performance**, and **Unlimited** Editions. Also available in **Developer Edition**.

```
    }
  },
  "Augment_Opportunity_User": {
    "action": "augment",
    "parameters": {
      "left": "Extract_Opportunity",
      "left_key": [
        "OwnerId"
      ],
      "right": "Extract_User",
      "relationship": "Owner",
      "right_select": [
        "Name"
      ],
      "right_key": [
        "Id"
      ]
    }
  },
  "Register": {
    "action": "sfdcRegister",
    "parameters": {
      "alias": "OppRoles",
      "name": "OppRoles",
      "source": "Augment_Opportunity_User",
      "rowLevelSecurityFilter": ""
    }
  }
}
```

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.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise**, **Performance**, and **Unlimited** Editions. Also available in **Developer Edition**.

```
'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.

Predicate Part	Description
'ParentRoleIDs' == "\$User.UserRoleId"	<ul style="list-style-type: none"> ParentRoleIDs refers to a dataset column that contains a comma-separated list of role IDs of all users above the opportunity owner based on the role hierarchy. You will create this dataset column in the next section. \$User.UserRoleId refers to the UserRoleId column of the User object. Analytics looks up the user role ID of the user who submits the query from the User object.
'OwnerId' == "\$User.Id"	<ul style="list-style-type: none"> OwnerId refers to the dataset column that contains the user ID of the owner of each opportunity. \$User.Id refers to the Id column of the User object. Analytics looks up the user ID of the user who submits the query from the User object.

Modify the Dataflow Based on Row-Level Security

Now it's time to modify the dataflow definition file to account for the predicate.

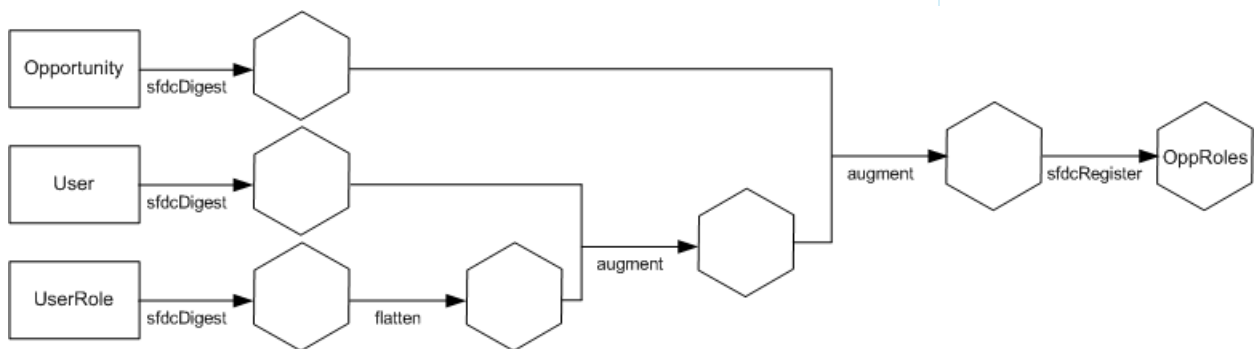
In this scenario, you have to make changes to the dataflow based on the predicate.

- 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:

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in **Enterprise, Performance, and Unlimited** Editions. Also available in **Developer Edition**.



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.

```
{
  "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" },
      ]
    }
  }
}
```

```

        { "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"
        ],
        "right": "Augment_User_FlattenUserRole",
        "relationship": "Owner",
        "right_select": [
            "Name",
            "Role.Roles",
            "Role.RolePath"
        ],
        "right_key": [
            "Id"
        ]
    }
}
}

```

```


    },
    "Register": {
      "action": "sfdcRegister",
      "parameters": {
        "alias": "OppRoles",
        "name": "OppRoles",
        "source": "Augment_Opportunity_UserWithRoles",
        "rowLevelSecurityFilter": "'Owner.Role.Roles' == \"\$User.UserRoleId\" || 'OwnerId'
== \"\$User.Id\""
      }
    }
  }
}

```


 **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.

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.

EDITIONS

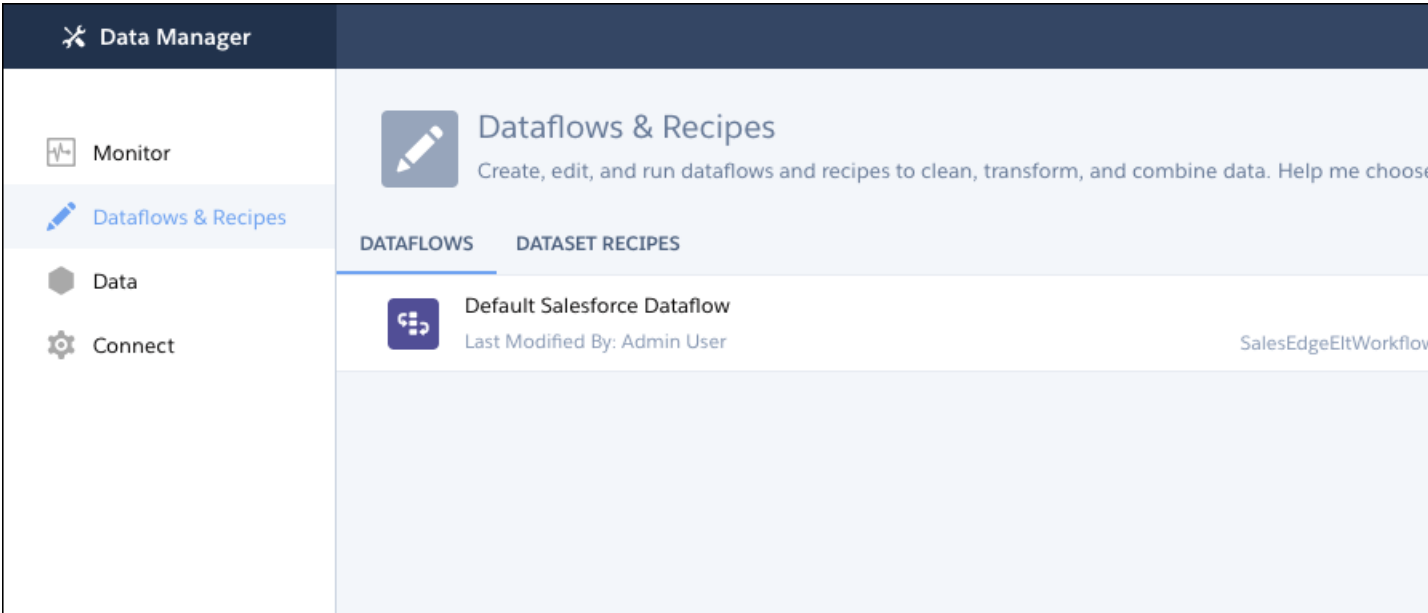
Available in Salesforce Classic and Lightning Experience.



Available for an extra cost in **Enterprise, Performance,** and **Unlimited** Editions. Also available in **Developer Edition**.

USER PERMISSIONS

To download, upload, run, and monitor a dataflow:

- Edit Analytics Dataflows



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 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.

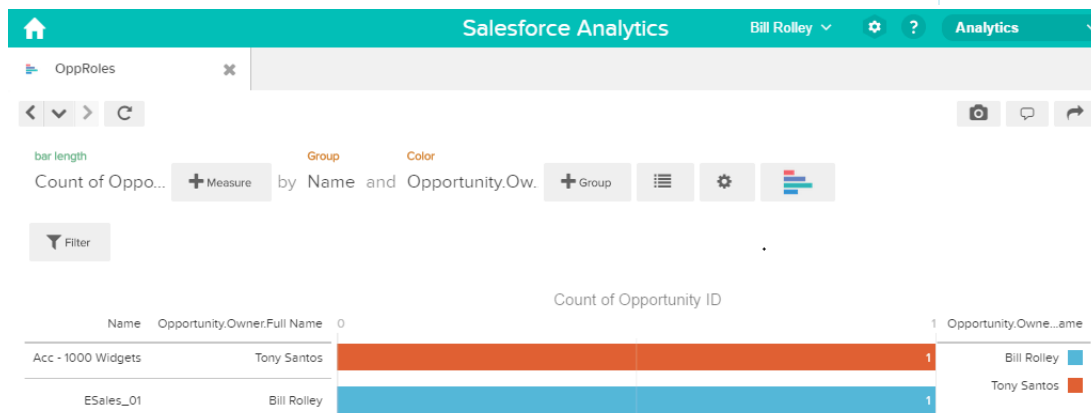
EDITIONS

Available in Salesforce Classic and Lightning Experience.

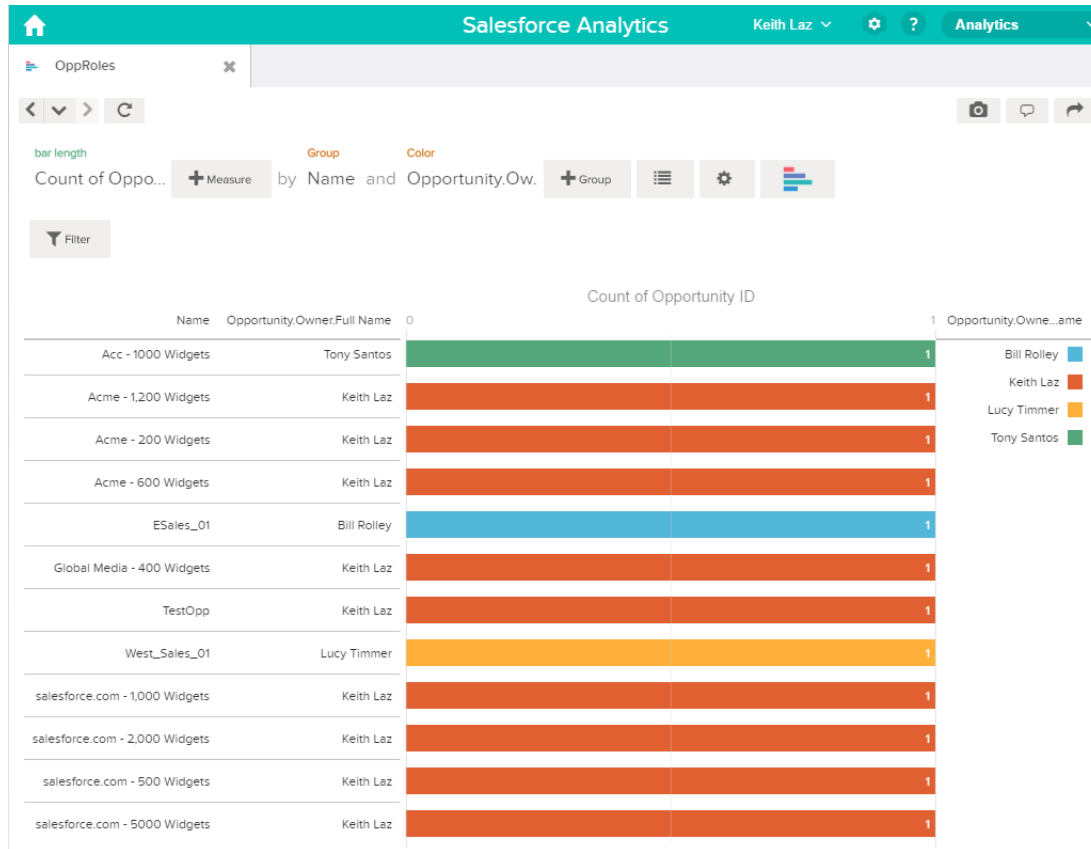
Available for an extra cost in **Enterprise**, **Performance**, and **Unlimited** Editions. Also available in **Developer Edition**.

USER PERMISSIONS

-
-



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

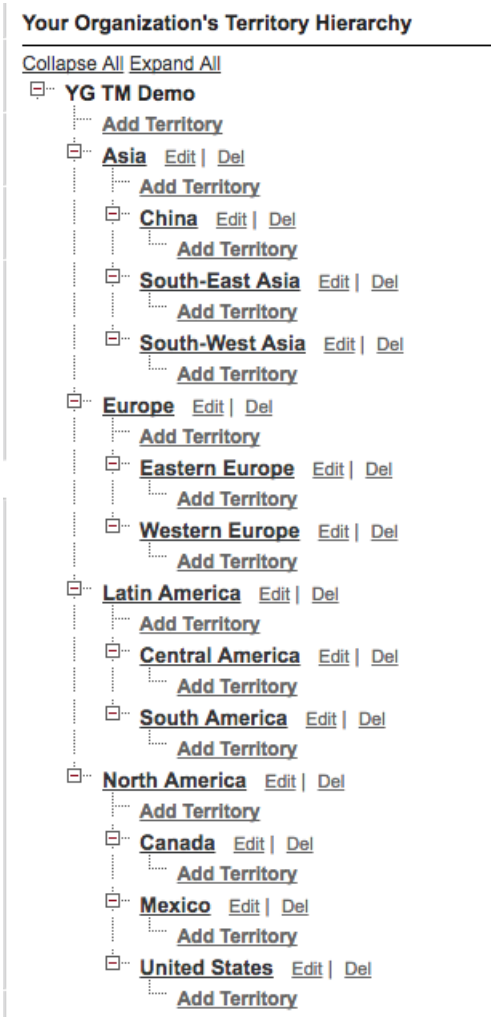
When working with security related to territory management, it helps to know how your organization implements territory management. Usually, one of 2 methods are used. Either accounts are assigned to regions manually, following some organization-specific precedence, or the organization use's Salesforce's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

The manual process

Manually Assigned Accounts						
Action	Account Name	Billing State/Province	Phone	Type	Account Owner Alias	Owner Alias
Remove	Santa's Workshop	North Pole			AUser	

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

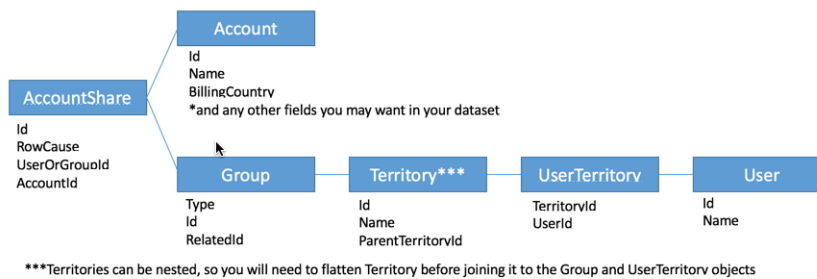


For this example, we have a user called North America VP who needs access to all accounts in the Canada, Mexico, and US territories. We also have a user called Rep1 Canada who should only have access to the accounts in the Canada territory, not Mexico or US, and nowhere above in the hierarchy.

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.

```

{
  "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": [
        { "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": [
            "Id"
        ],
        "right_select": [
            "Name",
            "RelatedId"
        ]
    }
},
"Augment_AccountShare_To_Territory": {
    "action": "augment",
    "parameters": {
        "left": "Augment_AccountShare_To_Territory_Groups",
        "left_key": [
            "UserOrGroupId.RelatedId"
        ],
        "relationship": "Territory",
        "right": "Augment_TerritoryUsers",
        "right_key": [
            "Id"
        ],
        "right_select": [
            "TerritoryId.UserId"
        ],
        "operation": "LookupMultiValue"
    }
},
"Augment_AccountShare_To_Account": {
    "action": "augment",
    "parameters": {
        "left": "Extract_AccountShare",
        "left_key": [
            "AccountId"
        ],
        "relationship": "AccountId",
        "right": "Extract_Account",
        "right_key": [
            "Id"
        ],
        "right_select": [
            "Name"
        ]
    }
}

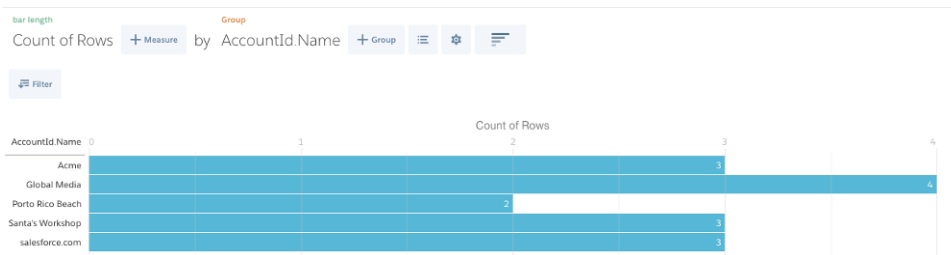
```

```

    }
  },
  "Register_Territory_GroupUsers": {
    "action": "sfdcRegister",
    "parameters": {
      "alias": "Register_Territory_GroupUsers",
      "name": "Register_Territory_GroupUsers",
      "source": "Augment_AccountShare_To_Territory"
    }
  }
}

```

When run, this JSON file results in a list of accounts. In this example, a list of 5:




Create the Security Predicate

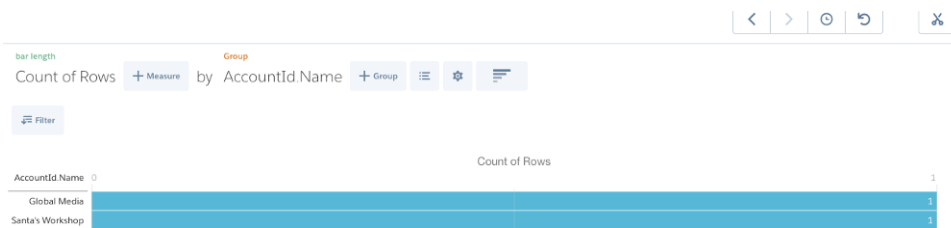
Now we can apply a security predicate to filter the dataset.

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.


- Sharing inheritance supports Account, Campaign, Case, Contact, Opportunity, Lead, Order, User, and custom objects. If you use other objects, such as Idea or Site, you must use security predicates for those objects.
 - 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.
 - 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. You must 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.

Set Up Sharing Inheritance

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

 **Note:** We recommend testing in a sandbox environment before rolling out sharing inheritance to production. Test your particular use cases against your org's security model and data to make sure that sharing inheritance works for you.

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 select **Settings**.
2. Select **Inherit sharing from Salesforce**, and click **Save**.

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.

```
"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".

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.

Security

Sharing Source

Apply Salesforce sharing settings from **No sharing source** 

Security Predicate

None 

SECURITY PREDICATE REFERENCE

Predicate Expression Syntax for Datasets

You must use valid syntax when defining the predicate expression.

The predicate expression must have the following syntax:

```
<dataset column> <operator> <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 (). 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.

Dataset Columns in a Predicate Expression

You include at least one dataset column as part of the predicate expression.


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.

Value Type	Requirements	Predicate Expression Examples
string literal	Enclose in double quotes and escape the double quotes.	<ul style="list-style-type: none"> • 'Owner' == "Amber" • 'Stage Name' == "Closed Won"
number literal	Can be a float or long datatype. Do not enclose in quotes.	<ul style="list-style-type: none"> • 'Expected_Revenue' >= 2000.00 • 'NetLoss' < -10000
field value	<p>When referencing a field from the User object, use the \$User.[field] syntax. Use the API name for the field.</p> <p>You can specify standard or custom fields of type string, number, or multi-value picklist.</p> <p>When you define a predicate for a dataset, you must have read access on all User object</p>	<ul style="list-style-type: none"> • 'Owner.Role' == "\$User.UserRoleId" • 'GroupID' == "\$User.UserGroupId__c" <p> Note: Supported User object field value types are string, number, and multi-value picklist. Other types (for</p>

Value Type	Requirements	Predicate Expression Examples
	<p>fields used to create the predicate expression.</p> <p>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.</p> <p> Note: By default, the Security User does not have access permission on custom fields of the User object.</p> <p>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.</p>	<p>example, boolean) are not supported.</p>

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:

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

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


Sequence	Meaning
\b	One backspace character
\n	New line
\r	Carriage return
\t	Tab
\Z	CTRL+Z (ASCII 26)
\"	One double-quote character
\\	One backslash character
\0	One ASCII null character

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 (). 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.

Character	Name	Description
'	Single quote	Encloses a dataset column name in a predicate expression. Example predicate expression: <code>'Expected_Revenue' >= 2000.00</code>
"	Double quote	Encloses a string value or field value in a predicate expression. Example predicate expression: <code>'OpportunityOwner' == "Michael Vesti"</code>
()	Parentheses	Enforces the order in which to evaluate a predicate expression. Example predicate expression: <code>('Expected_Revenue' > 4000 'Stage Name' == "Closed Won") && 'isDeleted' != "False"</code>
\$	Dollar sign	Identifies the Salesforce object in a predicate expression.  Note: You can only use the User object in a predicate expression. Example predicate expression: <code>'Owner.Role' == "\$User.UserRoleId"</code>
.	Period	Separates the object name and field name in a predicate expression. Example predicate expression:

Character	Name	Description
		'Owner' == "\$User.UserId"

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.

Operator	Name	Description
==	Equals	True if the operands are equal. String comparisons that use the equals operator are case-sensitive. Example predicate expressions: <code>'Stage Name' == "Closed Won"</code>
!=	Not equals	True if the operands are not equal. String comparisons that use the not equals operator are case-sensitive. Example predicate expression: <code>'isDeleted' != "False"</code>
<	Less than	True if the left operand is less than the right operand. Example predicate expression: <code>'Revenue' < 100</code>
<=	Less or equal	True if the left operand is less than or equal to the right operand.
>	Greater than	True if the left operand is greater than the right operand.
>=	Greater or equal	True if the left operand is greater than or equal to the right operand.
in	Multi-value list filter	True if the left operand exists in the list of strings substituted for a multi-value picklist (field value). Example predicate expression: <code>'Demog' in ["\$User.Demographic__c"]</code> In this example, <code>Demographic__c</code> is of type <code>multiPicklistField</code> . During evaluation, the multi-value picklist field is substituted by a list of strings, with 1 string per user-selected item.

Operator	Name	Description
		 Note: Comma-separated lists are not supported within the square-bracket construct.

You can use the <, <=, >, and >= operators with measure columns only.

Logical Operators

Logical operators return true or false.

Analytics supports the following logical operators.

Operator	Name	Description
&&	Logical AND	True if both operands are true. Example predicate expression: 'Stage Name' == "Closed Won" && 'isDeleted' != "False"
	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.

Opportunity	Expected_Rev	Owner	OwnerRoleID	Stage_Name	IsDeleted
OppA	2000.00	Bill	20	Prospecting	True
OppB	3000.00	Joe	22	Closed Won	False
OppC	1000.00	可爱的花	36	Closed Won	False
OppD	5000.00	O'Fallon	18	Prospecting	True
OppE		Joe	22	Closed Won	True

Let's take a look at some examples to understand how to construct a predicate expression.

Predicate Expression	Details
'OwnerRoleId' == "\$User.UserRoleId"	Checks column values in the User object.
'Expected_Rev' > 1000 && 'Expected_Rev' <= 3000	
'Owner' = "Joe" 'Owner' = "Bill"	
('Expected_Rev' > 4000 'Stage Name' == "Closed Won") && 'isDeleted' != "False"	Parentheses specify the order of operations.
'Stage Name' == "Closed Won" && 'Expected_Rev' > 70000	
'Owner' == "可爱的花"	String contains Unicode characters.
'Owner' == "O\'Fallon"	Single quote in a string requires the escape character.
'Stage Name' == ""	Checks for an empty string.