
Scoping Rules Developer Guide

Version 60.0, Spring '24



CONTENTS

Chapter 1: About This Guide	1
Scoping Rules	2
Quick Start: Display Records by Branch Location	3
Before You Start	4
Use the SOQL Operator in Scoping Rule Record Criteria	4
Create a Branch Management Scoping Rule Using the Tooling API	6
Create a Branch Management Scoping Rule Using the Metadata API	9
Create a Wealth Management Scoping Rule Using the Tooling API	11
Create a Wealth Management Scoping Rule Using the Metadata API	12
Considerations for Scoping Rules	13
Example Scenarios	15
Display a Branch Location's Records by Default	16
Display a Department's Records by Default	17
Display a Division's Tasks by Default	17
Scope Records Using Multiple String or ID Values in Record Criteria	18
Tooling API Reference	20
RestrictionRule	20
Metadata API Reference	24
RestrictionRule	24

CHAPTER 1 About This Guide

In this chapter ...

- [Scoping Rules](#)
- [Quick Start: Display Records by Branch Location](#)
- [Considerations for Scoping Rules](#)
- [Example Scenarios](#)
- [Tooling API Reference](#)
- [Metadata API Reference](#)

Based on criteria that you select, you can set rules to help your users see only records that are relevant to them. Scoping rules don't restrict the record access that your users already have. Your users can still open and report on all records that they have access to per your org's sharing settings.

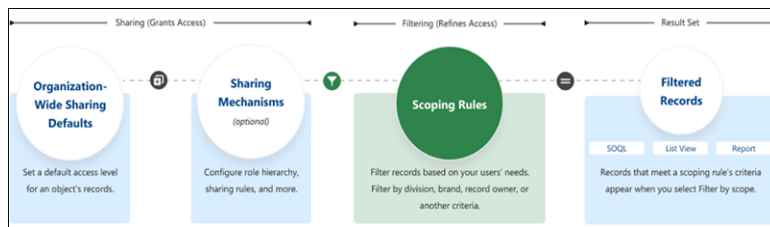
Scoping Rules

Scoping rules let you control the records that your users see based on criteria that you select. You can set up scoping rules for different users in your Salesforce org so that they can focus on the records that matter to them. Scoping rules are available for custom objects and the account, case, contact, event, lead, opportunity, and task standard objects. Create, edit, or delete scoping rules using the Tooling or Metadata API, or in Salesforce Setup.

You can provide feedback and suggestions for scoping rules in [Scoping Rules](#) group in the Trailblazer Community.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.



When Do I Use Scoping Rules?

Use scoping rules when you want to let users control the record set that they see. A scoping rule doesn't restrict users' access to other records that they sometimes need. Instead, scoping rules let your users focus on one set of records, then change their focus or search to find a record that's not in the scoped record set when they need to.

For example, you have users who support multiple agencies in your org. Each user is assigned to a specific agency. You can set up scoping rules so that they filter the records that your users see in list views and reports. Users don't have to spend time looking for the correct records, but they still have access to the other agencies' records if they need them.

You can also use scoping rules with Flow Builder to set scope according to a choice your user makes. For example, you have users who work on account records that belong to different divisions in your organization. You want to scope the account records that users see by division, giving your users an easy way to switch between different divisions' record sets. You can set up a flow that your users access using the Lightning Utility Bar to set the scope of records that the user sees in list views, reports, and other features.

How Do Scoping Rules Affect User Access?

Scoping rules are flexible. You can enable and disable them on a query-by-query basis. Plus, they don't restrict the access that your users have to records. Your users can still open and report on all the records that they can access according to your org's sharing settings.

Where Are Scoping Rules Applied?

This table shows how scoping rules work with other Salesforce features.

Feature	Description
List Views	Applied in Lightning Experience if Filter by scope is selected
Reports	Applied in Lightning Experience if Filter by scope is selected
SOQL	Applied, unless a scope other than <code>scopingrule</code> is specified

How Do I Configure Scoping Rules?

Create and manage scoping rules by navigating to a supported object in the Object Manager. Or use the RestrictionRule Tooling API object or RestrictionRule Metadata API type.

 **Note:** The RestrictionRule API object is also used to manage restriction rules. For information on restriction rules, see the Restriction Rule Developer Guide.

When creating more than one scoping or restriction rule, configure the rules so that only one active rule applies to a given user. Salesforce doesn't validate that only one active rule applies for a given user. If you create two active rules, and both rules apply to a given user, only one of the active rules is observed.

After creating rules, you can use a change set or unlocked package to move scoping rules from one org to another.

SEE ALSO:

[Salesforce Help: Create a Scoping Rule](#)

[Restriction Rule Developer Guide](#)

[Metadata API Guide: RestrictionRule](#)

[Tooling API Guide: RestrictionRule](#)

Quick Start: Display Records by Branch Location

In this Quick Start, we create a scoping rule using the SOQL operator. In the first example, a scoping rule created via Tooling API shows a banker only the accounts that match their current branch. The second example creates the same scoping rule via Metadata API.

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

Before You Start

Before you create a scoping rule, make sure you that have the needed permissions and tools.

[Use the SOQL Operator in Scoping Rule Record Criteria](#)

When a rule's record criteria must refer to a field on an object other than the target object, such as a junction object, use the SOQL operator in record criteria. To create or edit scoping rules with the SOQL operator in the record criteria, use the Tooling or Metadata API.

[Create a Branch Management Scoping Rule Using the Tooling API](#)

Create a scoping rule that filters account records based on a banker's branch location. This example uses the branch management data model included in Financial Services Cloud and the RestrictionRule Tooling API object.

[Create a Branch Management Scoping Rule Using the Metadata API](#)

Create a scoping rule that filters account records based on a banker's branch location. This example uses the branch management data model included in Financial Services Cloud and the RestrictionRule Metadata API type.

[Create a Wealth Management Scoping Rule Using the Tooling API](#)

Create a scoping rule that shows a sales support associate who supports multiple financial advisors only the record set that corresponds to the financial advisor that the associate is working with. Use the RestrictionRule Tooling API object.

[Create a Wealth Management Scoping Rule Using the Metadata API](#)

Create a scoping rule that shows a sales support associate who supports multiple financial advisors only the record set that corresponds to the financial advisor that the associate is working with. Use the RestrictionRule Metadata API type.

Before You Start

Before you create a scoping rule, make sure you that have the needed permissions and tools.

Creating scoping rules requires you to be comfortable using an API development application, such as [Postman](#). You can use either the [Tooling](#) or [Metadata](#) API to create, retrieve, update, and delete scoping rules. In this Quick Start, we show you how to use both.

For help with creating scoping rules in Setup, see [Create a Scoping Rule](#).

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules

Use the SOQL Operator in Scoping Rule Record Criteria

When a rule’s record criteria must refer to a field on an object other than the target object, such as a junction object, use the SOQL operator in record criteria. To create or edit scoping rules with the SOQL operator in the record criteria, use the Tooling or Metadata API.

This scoping rule scopes the account records that a banker sees by filtering according to the branch where the banker is working.

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules

```

1 {
2   "FullName": "BranchRule",
3   "Metadata": {
4     "active": true,
5     "description": "Scoping rule where users can scope records by their current
6 branch",
7     "enforcementType": "Scoping",
8     "masterLabel": "BranchRule",
9     "recordFilter": "SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE
10 EVERYTHING WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE
11 UserOnContactId = $User.Id))",
12     "targetEntity": "Account",
13     "userCriteria": "$User.IsActive = true",
14     "version": 1
15   }
16 }

```


1	SOQL Operator	The operator that tells Salesforce to run a SELECT statement on the junction object.
2	Record Filter	The record criteria in this expression is used to filter records when the users specified by the user criteria use a list view, report, or SOQL query.

3	Target Entity	The standard or custom object whose records are scoped by the scoping rule. Scoping rules support custom objects and the account, case, contact, event, lead, opportunity, and task standard objects only.
---	---------------	--

Detail of Record Filter:

```
<recordFilter>SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE EVERYTHING
WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE UserOrContactId =
$User.Id) )</recordFilter>
```

1. In your rule's `recordFilter`, start a query between double quotes: `"SOQL () "`.
2. In the query, specify a left operand that is an ID or reference field from the target entity object that you want to retrieve. The left operand must query a single ID (primary key) or reference (foreign key) field.
3. Write a `SELECT` statement that specifies the field and the object that stores the field.

 **Important:** The `SELECT` statement, including nested `SELECT` statements, must include `USING SCOPE EVERYTHING`. `USING SCOPE EVERYTHING` is the only valid scope clause syntax for scoping rules.

Performance Considerations

Before you activate a scoping rule that includes the SOQL operator, test the record filter's performance impact by running the `SELECT` statement separately in your API client of choice.

- Take the `SELECT` statement and run it.
- Evaluate whether the results that are returned make sense. Does the `SELECT` statement return the expected results rapidly? If it's fast for a given user, the rule is likely to run efficiently.
- If the `SELECT` statement isn't performant, isolate the field that is slowing performance. Work with Salesforce customer support to find out if the field can be indexed.

Other Considerations

- In a SOQL operator's `SELECT` statement, the query's junction object and the scoping rule's `targetEntity` can't be the same object.
- The SOQL operator doesn't support `$User` syntax except for `$User.Id`. Dynamic queries within the SOQL operator aren't supported, including on other user object fields.
- Don't use objects that aren't supported in subqueries in your record filter's `SELECT` statement. See [Comparison Operators](#) for a list of valid operators that you can use in the field expression of a `WHERE` clause, which you use in a `SELECT` statement.

SEE ALSO:

[Knowledge Article: Improve Performance of SOQL Queries using a Custom Index](#)


Create a Branch Management Scoping Rule Using the Tooling API

Create a scoping rule that filters account records based on a banker's branch location. This example uses the branch management data model included in Financial Services Cloud and the `RestrictionRule` Tooling API object.

This example uses the SOQL operator in its `recordFilter`. Check out the [Branch Management data model](#) to understand the objects used in this example and how they relate to each other.

1. Use the `RestrictionRule` object to create and manage both restriction rules and scoping rules. Include the `FullName` value and all required fields. For more information, see the reference topic [RestrictionRule](#).

In this example, we used these values.

 **Note:** The `userCriteria` in this example applies this rule to any active user in your org. Adjust the `userCriteria` if the rule must apply to a different subset of your users.

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules

```
{
  "FullName": "BranchRuleOnAccount",
  "Metadata": {
    "active": true,
    "description": "Scoping rule where users can scope account records by the user's
current
branch",
    "enforcementType": "Scoping",
    "masterLabel": "BranchRuleOnAccount",
    "recordFilter": "SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE
EVERYTHING WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE
UserOrContactId = $User.Id))",
    "targetEntity": "Account",
    "userCriteria": "$User.IsActive = true",
    "version": 1
  }
}
```

This example also uses objects from the [Branch Management data model](#) and sets a scoping rule that shows users lead records related to the user's current branch location.

```
{
  "FullName": "BranchRuleOnLead",
  "Metadata": {
    "active": true,
    "description": "Scoping rule where users can scope lead records by the user's
current
branch",
    "enforcementType": "Scoping",
    "masterLabel": "BranchRuleOnLead",
    "recordFilter": "SOQL(Id, SELECT RelatedRecordId FROM BranchUnitRelatedRecord
USING SCOPE
EVERYTHING WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE
```

```

UserOrContactId = $User.Id))",
    "targetEntity": "Lead",
    "userCriteria": "$User.IsActive = true",
    "version": 1
  }
}

```

To create a similar scoping rule for a different object, adjust the `targetEntity` field to include case, contact, or another supported standard or custom object.

- To create the scoping rule, use a POST request in the tooling API.
POST `/services/data/v60.0/tooling/subjects/RestrictionRule`
- Copy your scoping rule definition into the request body.
- Execute your request. Copy the ID returned for the scoping rule for later reference.

Let's take a closer look at the Branch Rule On Account example's SOQL operator.

```

SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE EVERYTHING WHERE
BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE UserOrContactId = $User.Id) )

```

- The SOQL statement takes the `Id` from the account object, which is the target entity whose records the scoping rule filters, and selects `AccountId` from the `BranchUnitCustomer` object.
- The where clause gets the `BranchUnitId`, which is a unique identifier of each branch, from a nested query. The nested query finds each banker's current branch from the `Banker` object by matching the `UserOrContactId` to the currently logged-in user.

When writing scoping rules using SOQL, follow these guidelines.

- In SOQL operators, the SOQL query object and the scoping rule target entity can't be the same object. In this example, the SOQL query object is `BranchUnitCustomer` and the scoping rule object, called the `targetEntity`, is `account`.
- In the SOQL type `RecordCriteria`, the left operand must query a single ID (primary key) or reference (foreign key) field. In this example, the left operand is a field on the target entity called `Id`.

For more tips about writing scoping rules using a performant SOQL operator, see [Scoping Rules Considerations](#). The SOQL operator is supported for scoping rules only.

Retrieve and Update Information

Use the GET, PATCH, and DELETE methods to retrieve, update, and delete scoping rules.

SEE ALSO:

- [Considerations for Scoping Rules](#)
- [Restriction Rule Developer Guide](#)

Retrieve and Update Information

Use the GET, PATCH, and DELETE methods to retrieve, update, and delete scoping rules.

Retrieve

To retrieve information about a scoping rule, use the GET method.

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

Example HTTP Method and URI:

GET

`/services/data/v60.0/tooling/query/?sObject=Id,targetEntity,enforcementType,recordFilter,userCriteria#FROMRestrictionRule#WHEREenforcementType='Scoping'`

Update

To update a scoping rule, use the PATCH method.

We recommend that you don't update the value of `targetEntity` after a scoping rule is created. Instead, delete the scoping rule and create another one with the correct values.

Example HTTP Method and URI:

PATCH `/services/data/v60.0/tooling/subjects/RestrictionRule/0eYxxxxxxxxxxxxx2AY`

Replace `0eYxxxxxxxxxxxxx2AY` with the ID returned when the scoping rule was created.

Example Request Body:

Include all Metadata fields, even if you aren't updating them. Specify the `FullName` value only if you're changing this field.

This example deactivates the scoping rule by setting `active` to `false`.

```
{
  "Metadata": {
    "active": false,
    "description": "sales support associate sees only account records of specified
advisor",
    "enforcementType": "Scoping",
    "masterLabel": "Advisor1 Record Set",
    "recordFilter": "SOQL(id, SELECT Account__c FROM Client_Entitlement__c USING SCOPE
EVERYTHING
WHERE Team_Entitlement__c IN (
SELECT Team_Entitlement__c
FROM User_Entitlement__c
USING SCOPE EVERYTHING
WHERE User__c = $User.id)
)",
    "targetEntity": "Account",
    "userCriteria": "$User.ProfileId = '00xxxxxxxxxxxx'",
    "version": 1
  }
}
```


Delete

To delete a scoping rule, use the DELETE method.

Example HTTP Method and URI:

DELETE `/services/data/v60.0/tooling/subjects/RestrictionRule/0eYxxxxxxxxxxxxx2AY`

Replace `0eYxxxxxxxxxxxxx2AY` with the ID returned when creating the scoping rule.

 **Note:** If the `userCriteria` or `recordCriteria` field contains a Salesforce org ID and you're deploying to a different org than the org you retrieved them from, modify the Salesforce ID first.

Create a Branch Management Scoping Rule Using the Metadata API

Create a scoping rule that filters account records based on a banker's branch location. This example uses the branch management data model included in Financial Services Cloud and the `RestrictionRule` Metadata API type.

This example uses the SOQL operator in its `recordFilter` to identify the accounts that match a retail banker's branch location. Check out the [Branch Management data model](#) to understand the objects used in this example and how they relate to each other.

1. Use the [RestrictionRule](#) type to create and manage both restriction rules and scoping rules. Set up the `package.xml` manifest file and your directory.

Example `package.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>*</members>
    <name>RestrictionRule</name>
  </types>
  <version>60.0</version>
</Package>
```

Example directory:

```
myPackage/package.xml
myPackage/restrictionRules
myPackage/restrictionRules/Rule1.rule
myPackage/restrictionRules/Rule2.rule
```

2. Include all required fields.

In this example, we used these values.

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Scoping rule where users can scope account records by the user's current
branch</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>BranchRuleOnAccount</masterLabel>
  <recordFilter>SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE
EVERYTHING WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE
UserOrContactId = $User.Id)</recordFilter>
  <targetEntity>Account</targetEntity>
  <userCriteria>$User.IsActive = true</userCriteria>
  <version>1</version>
</RestrictionRule>
```

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules

This example also uses objects from the [Branch Management data model](#) and sets a scoping rule that shows users lead records related to the user's current branch location.

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Scoping rule where users can scope lead records by their current
branch</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>BranchRuleOnLead</masterLabel>
  <recordFilter>SOQL(Id, SELECT RelatedRecordId FROM BranchUnitRelatedRecord USING SCOPE
EVERYTHING WHERE BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE
UserOrContactId = $User.Id))</recordFilter>
  <targetEntity>Lead</targetEntity>
  <userCriteria>$User.IsActive = true</userCriteria>
  <version>1</version>
</RestrictionRule>
```

To create a similar scoping rule for a different object, adjust the `targetEntity` field to include case, contact, or another supported standard or custom object.

3. Zip your directory, and deploy your changes. For more information, see [Deploying and Retrieving Metadata](#) in the Metadata API Developer Guide.

Let's take a closer look at the Branch Rule On Account example's SOQL operator.

```
SOQL(Id, SELECT AccountId FROM BranchUnitCustomer USING SCOPE EVERYTHING WHERE
BranchUnitId IN(SELECT CurrentBranchId From Banker WHERE UserOrContactId = $User.Id))
```

- The SOQL statement takes the `Id` from the account object, which is the target entity whose records the scoping rule filters, and selects `AccountId` from the `BranchUnitCustomer` object.
- The where clause gets the `BranchUnitId`, which is a unique identifier of each branch, from a nested query. The nested query finds each banker's current branch from the `Banker` object by matching the `UserOrContactId` to the currently logged-in user.

When writing scoping rules using SOQL, follow these guidelines.

- In SOQL operators, the SOQL query object and the scoping rule target entity can't be the same object. In this example, the SOQL query object is `BranchUnitCustomer` and the scoping rule object, called the `targetEntity`, is `account`.
- In the SOQL type `RecordCriteria`, the left operand must query a single ID (primary key) or reference (foreign key) field. In this example, the left operand is a field on the target entity called `Id`.

For more tips about writing scoping rules using a performant SOQL operator, see [Scoping Rules Considerations](#). The SOQL operator is supported for scoping rules only.

Retrieve and Update Information

Use the `deploy()` and `retrieve()` calls to move metadata (XML files) between Salesforce and a local file system. You can delete scoping rules by using the same procedure used to deploy components and including a delete manifest file.

SEE ALSO:

- [Considerations for Scoping Rules](#)
- [Restriction Rule Developer Guide](#)

Retrieve and Update Information

Use the `deploy()` and `retrieve()` calls to move metadata (XML files) between Salesforce and a local file system. You can delete scoping rules by using the same procedure used to deploy components and including a delete manifest file.

For more information, see [Deploying and Retrieving Metadata](#) in the Metadata API Developer Guide.

If the `userCriteria` or `recordCriteria` field contains a Salesforce org ID and you're deploying to a different org than the org you retrieved them from, modify the Salesforce ID first.

Note: We recommend that you don't update the value of `targetEntity` after a scoping rule is created. Instead, delete the scoping rule and create another one with the correct values.

To delete components, use the same procedure as with deploying components, but include a delete manifest file that's named `destructiveChanges.xml` and lists the components to delete. To learn more, see [Deleting Components from an Organization](#).

SEE ALSO:

[Metadata API Guide: Deleting Components from an Organization](#)

EDITIONS

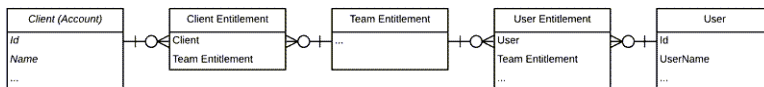
Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

Create a Wealth Management Scoping Rule Using the Tooling API

Create a scoping rule that shows a sales support associate who supports multiple financial advisors only the record set that corresponds to the financial advisor that the associate is working with. Use the `RestrictionRule` Tooling API object.

Note: You can use the `RestrictionRule` object to create and manage both restriction rules and scoping rules. For information on restriction rules, see the [Restriction Rule Developer Guide](#).

This example uses the SOQL operator in the `recordFilter` field to determine which accounts to display to the user based on the account, team, and user entitlements.



1. Set a value for the `FullName` value (the full name of the associated metadata object in Metadata API).
2. Include all other required fields. For more information, see the reference topic [RestrictionRule](#).

In this example, we used these values.

```

{
  "FullName": "SalesSupportAssociateScopingRule",
  "Metadata": {
    "active": true,
    "description": "Sales support associate sees only account records of of Advisor1",

    "enforcementType": "Scoping",
    "masterLabel": "Advisor1 Record Set",
    "recordFilter": "SOQL(id, SELECT Account__c FROM Client_Entitlement__c USING
SCOPE EVERYTHING
  
```

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules

```


WHERE Team_Entitlement__c IN (
  SELECT Team_Entitlement__c
  FROM User_Entitlement__c
  USING SCOPE EVERYTHING
  WHERE User__c = $User.id)
)",
  "targetEntity": "Account",
  "userCriteria": "$User.ProfileId = '00exxxxxxxxxxxxx'",
  "version": 1
}
}

```

3. Use a POST request to create the scoping rule.
POST /services/data/60.0/tooling/subjects/RestrictionRule
4. Copy your scoping rule definition into the request body.
5. Execute your request. Copy the ID returned for the scoping rule for later reference.

Create a Wealth Management Scoping Rule Using the Metadata API

Create a scoping rule that shows a sales support associate who supports multiple financial advisors only the record set that corresponds to the financial advisor that the associate is working with. Use the RestrictionRule Metadata API type.

 **Note:** You can use the RestrictionRule object to create and manage both restriction rules and scoping rules. For information on restriction rules, see the Restriction Rule Developer Guide.

1. Set up the package.xml manifest file and your directory.

Example package.xml file:

```

<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>*</members>
    <name>RestrictionRule</name>
  </types>
  <version>60.0</version>
</Package>

```

Example directory:

```

myPackage/package.xml
myPackage/restrictionRules

```

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

USER PERMISSIONS

To create and manage scoping rules:

- Manage Sharing

To view scoping rules:

- View Setup & Configuration AND View Restriction and Scoping Rules


```
myPackage/restrictionRules/Rule1.rule
myPackage/restrictionRules/Rule2.rule
```

2. Include all required fields. For more information, see the reference topic [RestrictionRule](#).

In this example, we used these values.

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Sales support associate sees only account records of specified
advisor</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>Advisor1 Record Type</masterLabel>
  <recordFilter>recordTypeId = '012xxxxxxxxxxxx'</recordFilter>
  <targetEntity>Account</targetEntity>
  <userCriteria>$User.ProfileId = '00xxxxxxxxxxxx'</userCriteria>
  <version>1</version>
</RestrictionRule>
```

3. Zip your directory, and deploy your changes. For more information, see [Deploying and Retrieving Metadata](#) in the Metadata API Developer Guide.

Considerations for Scoping Rules

Familiarize yourself with these considerations for using scoping rules.

EDITIONS

Available in: Lightning Experience in **Performance** and **Unlimited** Editions

Creating Scoping Rules

Your edition affects how many active rules you can have.

- Create up to two active scoping rules per object in Developer editions.
- Create up to five active scoping rules per object in Performance and Unlimited editions.
- Create only one scoping or restriction rule per object per user. For a given object, only one scoping or restriction rule's `userCriteria` field can evaluate to `true` for a given user.
- Creating a scoping rule for an object impacts only that object and doesn't affect child objects.
- When you reference the `Owner` field, you must specify the object type in your syntax. For example, the `Owner` field on an event object can contain a user or a queue, but queues aren't supported in scoping rules. So it's necessary to specify `Owner:User` in the `recordFilter` syntax when the filter allows only users.
- You can reference another object's field using dot notation in the `recordFilter` field. You can use only one "dot" (one lookup level from the `targetEntity`). For example, `Owner.UserRoleId`.
- These data types are supported in the `recordFilter` and `userCriteria` fields.
 - boolean
 - date (yyyy-MM-dd)
 - dateTime (yyyy-MM-dd HH:mm:ss)
 - double
 - int
 - reference

- string
- time
- single picklist

 **Note:** Comma-separated ID or string values are supported in the Record Criteria field.

- Including a null or blank value in record criteria isn't supported and can result in unexpected behavior.
- Don't create rules on `Event.IsGroupEvent`, which indicates whether the event has invitees.
- Use the Activity Timeline instead of Open Activities or Activity History. If you use Open Activities and Activity History related lists, create rules on task or event objects using fields that are only available on the `OpenActivity` and `ActivityHistory` objects.
- For list views and reports, you can apply the scope through Metadata API (using the `filterScope` field on the `ListView` type and the `scope` field on the `Report` type "scope").
- If you include an ID in your `recordFilter` or `userCriteria` field that is specific to your Salesforce org (such as a role, record type, or profile ID), you must modify the ID in the target org if it's different from the org where the scoping rule was originally created. Keep this consideration in mind when deploying rules between sandboxes or to a production org.

Using SOQL

- You can use a SOQL operator in record criteria only when creating scoping rules via API.
- Unless you use SOQL, scoping rules support only the EQUALS operator. The AND and OR operators aren't supported.
- When using the SOQL operator in the record filter, the SELECT statement, including nested SELECT statements, must include USING SCOPE EVERYTHING. USING SCOPE EVERYTHING is the only valid scope clause syntax for scoping rules.
- The SOQL operator doesn't support \$User syntax except for \$User.Id. Dynamic queries within the SOQL operator aren't supported, including on other user object fields.


 **Example:** Supported SOQL Syntax

```
SOQL(Id, SELECT Account.id FROM AccountAdvisors USING SCOPE EVERYTHING WHERE userid
= $User.Id)
```

Unsupported SOQL Syntax

```
SOQL(Id, SELECT Account.id FROM AccountAdvisors USING SCOPE EVERYTHING WHERE userid
= $User.Current_Advisor__c)
```

- Using the same object as the SOQL Query object and the Scoping Rule object isn't supported.
- The left operand in the SOQL type `RecordCriteria` must query a single ID (primary key) or reference (foreign key) field. See Comparison Operators for a list of valid operators that you can use in the field expression of a `WHERE` clause, which you use in a `SELECT` statement.

 **Example:** `"recordFilter": "SOQL(OwnerId, Select Id from User USING SCOPE Everything LIMIT 2) "`

The left operand is `OwnerId` in this example.

Modifying Scoping Rules

- We recommend not editing the `targetEntity` field after a scoping rule is created. Instead, delete the rule and create another one with the correct values.

- To disable a scoping rule, first delete the list views and reports that have **Filter by scope** selected. After a scoping rule is disabled, the list views and reports aren't functional nor modifiable.
- The scoping rule `userCriteria` field supports custom permissions. If you delete the custom permission, the scoping rules that use the custom permissions don't work.
- Scoping rules support custom picklist values in record filter and user criteria. If you delete a custom picklist value used in a scoping rule, the rule no longer works as intended.

Accounts, Contacts, and Person Accounts

- Scoping rules don't support `IsPersonAccount` fields on the account object. When setting a scoping rule, don't use `IsPersonAccount` fields such as `PersonDepartment` or `PersonLeadSource` in record filter criteria. Find a list of `IsPersonAccount` fields on the [Account](#) page.
- An error can result if you navigate to a person account detail page from a Contacts list view. To navigate to a person account detail page when there's a scoping rule on the account object, use an Accounts list view such as All Accounts.
- In related lists, all associated records that a user can access are visible, regardless of scope, except in the contact role related list. When a scoping rule is applied on the contact object, scope is applied to the [contact role](#) related list that appears on account, opportunity, case, and contract records. So it's possible that users, such as members of a sales team, see a filtered set of contact roles without knowing that the list is filtered.
- When an org uses duplicate rules to prevent creating duplicate records, scoping rules limit the potential duplicates that are shown, even when **Bypass sharing rules** is turned on. Duplicate records are limited by the scope set in the scoping rule.

Performance Considerations

Scoping rules were built to support sharing needs in a performant way. Your data volume and architecture are factors in rule performance. Salesforce reserves the right to disable a scoping rule if a rule you create is inefficient or if your data model has so much data that scoping rules cause slowness when applied. To prevent throttling or deactivation, test the scoping rules that you plan to apply in a sandbox environment before enabling them in production.

- To test the performance impact of a rule that uses a SOQL operator, take the SOQL statement and run it in your API client of choice. If it's fast for a given user, the rule is likely to run efficiently.
- If a rule isn't performant, isolate the field that is slowing performance. Work with Salesforce customer support to find out if the field can be indexed.

SEE ALSO:

[Knowledge Article: Improve Performance of SOQL Queries using a Custom Index](#)

[SOQL and SOSL Reference: Comparison Operators](#)

Example Scenarios

These sample scoping rules provide relevant records to users.

[Display a Branch Location's Records by Default](#)

This scoping rule displays task records associated with a particular bank branch location by default. A custom field called `Branch__c` stores the bank's branch locations.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

[Display a Department's Records by Default](#)

This scoping rule displays contact records associated with a particular department by default for a user who works on them. The rule dynamically matches the contact owner's department with the current user's department.

[Display a Division's Tasks by Default](#)

This scoping rule displays records associated with a particular division by default for a user.

[Scope Records Using Multiple String or ID Values in Record Criteria](#)

This scoping rule allows active users to scope the records they see to records whose Name__c field matches the rule's record criteria value. The record criteria contains strings separated by a comma. ID values are also supported. Double-quotes specify that the value inside the quotes isn't considered a delimiter.

Display a Branch Location's Records by Default

This scoping rule displays task records associated with a particular bank branch location by default. A custom field called Branch__c stores the bank's branch locations.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

Tooling API

```
{
  "FullName": "Task scoping rule on Bank Branch 1",
  "Metadata": {
    "active": true,
    "description": "View tasks for Bank Branch 1.",
    "enforcementType": "Scoping",
    "masterLabel": "SR for Bank Branch 1",
    "recordFilter": "Branch__c = $User.Branch__c",
    "targetEntity": "Task",
    "userCriteria": "$User.UserRoleId = '00Exxxxxxxxxxxx'",
    "version": 1
  }
}
```

Metadata API

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>View tasks for Bank Branch 1.</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>SR for Bank Branch 1</masterLabel>
  <recordFilter>Branch__c = $User.Branch__c</recordFilter>
  <targetEntity>Task</targetEntity>
  <userCriteria>$User.UserRoleId = '00Exxxxxxxxxxxx'</userCriteria>
  <version>1</version>
</RestrictionRule>
```

Display a Department's Records by Default

This scoping rule displays contact records associated with a particular department by default for a user who works on them. The rule dynamically matches the contact owner's department with the current user's department.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

Tooling API

```
{
  "FullName": "Department A contact scoping rule",
  "Metadata": {
    "active": true,
    "description": "View contacts from Department A.",
    "enforcementType": "Scoping",
    "masterLabel": "SR for Department A",
    "recordFilter": "Department=$User.Department",
    "targetEntity": "Contact",
    "userCriteria": "$User.UserRoleId = '00Exxxxxxxxxxxx'",
    "version": 1
  }
}
```

Metadata API

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>View tasks contacts from Department A.</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>SR for Department A contacts</masterLabel>
  <recordFilter>Department=$User.Department</recordFilter>
  <targetEntity>Contact</targetEntity>
  <userCriteria>$User.UserRoleId = '00Exxxxxxxxxxxx'</userCriteria>
  <version>1</version>
</RestrictionRule>
```

Display a Division's Tasks by Default

This scoping rule displays records associated with a particular division by default for a user.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

Tooling API

```
{
  "FullName": "Task scoping rule on current user's Division",
  "Metadata": {
```

```

    "active":true,
    "description":"View tasks in the current user's Division.",
    "enforcementType":"Scoping",
    "masterLabel":"SR for Divisions",
    "recordFilter":"Division=$User.Division",
    "targetEntity":"Task",
    "userCriteria":"$User.ProfileId = '00xxxxxxxxxxxx'",
    "version":1
  }
}

```

Metadata API

```

<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>View tasks in the current user's Division.</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>SR for Divisions</masterLabel>
  <recordFilter>Division=$User.Division</recordFilter>
  <targetEntity>Task</targetEntity>
  <userCriteria>$User.ProfileId = '00xxxxxxxxxxxx'</userCriteria>
  <version>1</version>
</RestrictionRule>

```

Scope Records Using Multiple String or ID Values in Record Criteria

This scoping rule allows active users to scope the records they see to records whose Name__c field matches the rule's record criteria value. The record criteria contains strings separated by a comma. ID values are also supported. Double-quotes specify that the value inside the quotes isn't considered a delimiter.

This rule uses a custom object called Agent__c with a text field called Name__c.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

Tooling API

```

{
  "FullName":"Agent records matching name field",
  "Metadata": {
    "active":true,
    "description":"Show Records Matching Name__c field",
    "enforcementType":"Scoping",
    "masterLabel":"Records Matching Name__c field",
    "recordFilter":"Name__c='Tom, Anita, \"Torres, Jia\"'",
    "targetEntity":"Agent__c",
    "userCriteria":"$User.IsActive=true",
    "version":1
  }
}

```

Metadata API

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Show Records Matching Name__c field</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>Records Matching Name__c field</masterLabel>
  <recordFilter>Name__c='Tom, Anita, "Torres, Jia"</recordFilter>
  <targetEntity>Agent__c</targetEntity>
  <userCriteria>$User.IsActive=true</userCriteria>
  <version>1</version>
</RestrictionRule>
```

This scoping rule allows active users to see records owned by two different managers. In this example, the rule's record criteria contains IDs separated by a comma.

Tooling API

```
{
  "FullName": "Records Owned By Managers",
  "Metadata": {
    "active": true,
    "description": "Displays records owned by two department managers",
    "enforcementType": "Scoping",
    "masterLabel": "RR for manager records",
    "recordFilter": "Agent__c.Owner:User.ManagerId=001xx000003HNy7, 001xx000003HNut",
    "targetEntity": "Agent__c",
    "userCriteria": "$User.IsActive=true",
    "version": 1
  }
}
```

Metadata API

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Displays records owned by two department managers</description>
  <enforcementType>Scoping</enforcementType>
  <masterLabel>RR for manager records</masterLabel>
  <recordFilter>Agent__c.Owner:User.ManagerId=001xx000003HNy7,
001xx000003HNut</recordFilter>
  <targetEntity>Agent__c</targetEntity>
  <userCriteria>$User.IsActive=true</userCriteria>
  <version>1</version>
</RestrictionRule>
```

Tooling API Reference

This section provides more information on the `RestrictionRule` Tooling API object used to create scoping rules.

`RestrictionRule`

Represents a restriction rule or a scoping rule. A restriction rule has `EnforcementType` set to `Restrict` and controls the access that specified users have to designated records. A scoping rule has `EnforcementType` set to `Scoping` and controls the default records that your users see without restricting access.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

RestrictionRule

Represents a restriction rule or a scoping rule. A restriction rule has `EnforcementType` set to `Restrict` and controls the access that specified users have to designated records. A scoping rule has `EnforcementType` set to `Scoping` and controls the default records that your users see without restricting access.

This object is available in API version 52.0 and later.

Supported SOAP API Calls

`create()`, `delete()`, `describeSObjects()`, `query()`, `retrieve()`, `update()`, `upsert()`

Supported REST API Methods


`DELETE`, `GET`, `HEAD`, `PATCH`, `POST`, `Query`

Special Access Rules

Only users with the View Restriction and Scoping Rules permission can view restriction rules and scoping rules via the API. Only users with the Manage Sharing permission can view, create, update, and delete restriction rules and scoping rules.

Fields

Field	Details
Description	<p>Type textarea</p> <p>Properties Filter, Group, Nillable, Sort</p> <p>Description Required. The description of the rule.</p>
DeveloperName	<p>Type string</p>

Field	Details
	<p>Properties Filter, Group, Sort</p> <p>Description The unique name for the RestrictionRule object.</p> <p>This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores. This field is automatically generated, but you can supply your own value if you create the record using the API.</p> <p> Note: Only users with View DeveloperName OR View Setup and Configuration permission can view, group, sort, and filter this field.</p>
EnforcementType	<p>Type picklist</p> <p>Properties Defaulted on create, Filter, Group, Restricted picklist, Sort</p> <p>Description Required. The type of rule.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • <code>FieldRestrict</code>—Don't use. • <code>Restrict</code>—Restriction rule. • <code>Scoping</code>—Scoping rule.
FullName	<p>Type string</p> <p>Properties Create, Group, Nillable</p> <p>Description Required. The full name of the associated RestrictionRule in Metadata API. The full name can include a namespaceprefix.</p> <p>Query this field only if the query result contains no more than one record. Otherwise, an error is returned. If more than one record exists, use multiple queries to retrieve the records. This limit protects performance.</p>
IsActive	<p>Type boolean</p> <p>Properties Defaulted on create, Filter, Group, Sort</p> <p>Description Indicates whether the rule is active (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code>.</p>

Field	Details
Language	<p>Type picklist</p> <p>Properties Defaulted on create, Filter, Group, Nillable, Restricted picklist, Sort</p> <p>Description The language of the rule. The value for this field is the language value of the org.</p>
MasterLabel	<p>Type string</p> <p>Properties Filter, Group, Sort</p> <p>Description Label for the rule.</p>
Metadata	<p>Type mns : RestrictionRule</p> <p>Properties Create, Nillable, Update</p> <p>Description The restriction rule's metadata. Query this field only if the query result contains no more than one record. Otherwise, an error is returned. If more than one record exists, use multiple queries to retrieve the records. This limit protects performance.</p>
RecordFilter	<p>Type textarea</p> <p>Properties Create, Filter, Group, Sort, Update</p> <p>Description Required. The criteria that determine which records are accessible via the rule.</p>
TargetEntity	<p>Type picklist</p> <p>Properties Filter, Group, Restricted picklist, Sort</p> <p>Description Required. The object for which you're creating the rule. We recommend that you don't edit this field after the rule is created. If <code>EnforcementType</code> is set to <code>Restrict</code>, custom objects, external objects, and these objects are supported:</p> <ul style="list-style-type: none"> • Contract

Field	Details
	<ul style="list-style-type: none"> • Event • Task • TimeSheet • TimeSheetEntry <p>If <code>EnforcementType</code> is set to <code>Scoping</code>, custom objects and these objects are supported:</p> <ul style="list-style-type: none"> • Account • Case • Contact • Event • Lead • Opportunity • Task
UserCriteria	<p>Type textarea</p> <p>Properties Create, Filter, Group, Sort, Update</p> <p>Description Required. The users that this rule applies to, such as all active users or users with a specified role or profile.</p>
Version	<p>Type int</p> <p>Properties Filter, Group, Sort</p> <p>Description Required. The rule's version number.</p>

Usage

For more information on restriction rules, see the Restriction Rules Developer Guide.

Metadata API Reference

This section provides more information on the `RestrictionRule` Metadata API type used to create scoping rules.

`RestrictionRule`


Represents a restriction rule or a scoping rule. A restriction rule has `enforcementType` set to `Restrict` and controls the access that specified users have to designated records. A scoping rule has `enforcementType` set to `Scoping` and controls the default records that your users see without restricting access. This type extends the `Metadata` metadata type and inherits its `fullName` field.

EDITIONS

Available in: Lightning Experience in **Performance, Unlimited,** and **Developer** editions.

RestrictionRule

Represents a restriction rule or a scoping rule. A restriction rule has `enforcementType` set to `Restrict` and controls the access that specified users have to designated records. A scoping rule has `enforcementType` set to `Scoping` and controls the default records that your users see without restricting access. This type extends the `Metadata` metadata type and inherits its `fullName` field.

 **Important:** Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

File Suffix and Directory Location

`RestrictionRule` components have the suffix `.rule` and are stored in the `restrictionRules` folder.

Version

`RestrictionRule` components are available in API version 52.0 and later.

Special Access Rules

Only users with the View Restriction and Scoping Rules permission can view restriction rules and scoping rules via the API. Only users with the Manage Sharing permission can view, create, update, and delete restriction rules and scoping rules.

Fields

Field Name	Field Type	Description
<code>active</code>	boolean	Indicates whether the rule is active (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>description</code>	string	Required. The description of the rule.
<code>enforcementType</code>	EnforcementType (enumeration of type string)	Required. The type of rule. Valid values are: <ul style="list-style-type: none"> <code>FieldRestrict</code>—Don't use. <code>Restrict</code>—Restriction rule. <code>Scoping</code>—Scoping rule.

Field Name	Field Type	Description
masterLabel	string	Required. The name of the rule.
recordFilter	string	Required. The criteria that determine which records are accessible via the rule.
targetEntity	string	<p>Required. The object for which you're creating the rule. We recommend that you don't edit this field after the rule is created.</p> <p>If <code>enforcementType</code> is set to <code>Restrict</code>, custom objects, external objects, and these objects are supported:</p> <ul style="list-style-type: none"> • Contract • Event • Task • TimeSheet • TimeSheetEntry <p>If <code>enforcementType</code> is set to <code>Scoping</code>, custom objects and these objects are supported:</p> <ul style="list-style-type: none"> • Account • Case • Contact • Event • Lead • Opportunity • Task
userCriteria	string	Required. The users that this rule applies to, such as all active users or users with a specified role or profile.
version	int	Required. The rule's version number.

Declarative Metadata Sample Definition

The following is an example of a `RestrictionRule` component.

```
<?xml version="1.0" encoding="UTF-8"?>
<RestrictionRule xmlns="http://soap.sforce.com/2006/04/metadata">
  <active>true</active>
  <description>Allows users with a specific profile to see only tasks that they
own.</description>
  <enforcementType>Restrict</enforcementType>
  <masterLabel>Tasks You Own</masterLabel>
  <recordFilter>OwnerId = $User.Id</recordFilter>
  <targetEntity>Task</targetEntity>
  <userCriteria>$User.ProfileId = '005xxxxxxxxxxxxx'</userCriteria>
  <version>1</version>
</RestrictionRule>
```

The following is an example `package.xml` that references the previous definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>*</members>
    <name>RestrictionRule</name>
  </types>
  <version>55.0</version>
</Package>
```

Usage

For more information on restriction rules, see the [Restriction Rules Developer Guide](#).