On this page

Apache Ranger: Row-Level Filtering & Column-Masking Enterprise

Dremio offers both Apache Ranger security policy support and built-in SQL functions for applying row-level filtering and column-masking.

Column-Masking Overview

Column-masking is a secure and flexible resource-based solution to hiding sensitive information rapidly on a Hive source. Via Apache-Ranger-based security policies or using Dremio's built-in masking, you may mask or scramble private data at the column-level in a dynamic fashion for Hive query outputs. Utilizing masking methods, you may set a column to only display the year of a data, the first or last four digits of a value, and more.

Utilizing services like Apache Ranger allow you to apply access policies to a Hive source so that filters may be based upon specific users, groups, and conditions. Thus, sensitive information never leaves the source and no changes are required by the source. This likewise removes the need to produce a secondary set of data with protected information manually removed.

The following conditions apply to column-masking:

- Multiple masking types are available
- Masks may be applied to users, groups, and conditions
- Each column must have its own masking policy
- Masks are evaluated in the order they are presented in a query or on a security policy
- Wildcard matching is not supported

For Apache Ranger implementations, additional use cases may be found at 3. Use cases: data-masking.

Row-Level Filtering Overview

Row-level filtering both simplifies queries and adds a layer of security to the data returned for user/role queries. Either SQL functions or Apache-Ranger-based security policies limit access down to the dataset layer, which then affects how queries are handled upon execution. Row-level security on supported tables helps reduce exposure of sensitive data to specific users or groups.

context of the query are displayed from Dremio's SQL Editor.

Row-level restrictions may be set by user, group/role, and other conditions (conditions only available for Ranger implementations, as described further under Row Filter Conditions).

The following examples serve as use cases where row-level filtering would prove beneficial:

- Hospitals may create security policies enabling 1) doctors to view only the rows containing their patients, 2) insurance claims adjusters to view only rows pertaining to their site/facility, and 3) medical billing coders to only view rows pertaining to specific medical disciplines.
- Financial institutes may create policies restricting access to rows pertaining to a user's specific division, geographic location or site, or role, meaning only employees in Collections would only be allowed to see outstanding unpaid claims, collection payment plans, and so on.
- Organizations utilizing multi-tenant applications may use row-level filters to set logical separations of each tenant's data, thus ensuring a tenant only has access to their own data rows.

For Apache Ranger implementations, additional use cases may be found at 2. Use cases: row-level filters.

Using Apache Ranger Security Policies

For organizations configured to use Apache Ranger and Hive sources, support automatically exists in Dremio to handle security policies set from Ranger. Based on the user, group/role, and conditions set externally, Dremio automatically applies restrictions to a user's query and applies row-level filtering and column-masking in the background. Upon query completion, you will then only see the results for rows and columns you have access to, without any visual indication that rows have been removed from view.

Requirements

- Dremio 20.0+
- Apache Ranger configured
  - Admin privileges to add access control policies
- Hive source

How It Works

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

dremio Documentation

Dremio Software

Ranger-based row filtering and column-masking functions as an “implicit view,” replacing a table/view reference in an SQL statement prior to processing the statement. This implicit view is created through an examination of user permissions. For example, consider a user with access to `table_1`, while also having a mask applied on `table_1.column_1`, effectively translating the column to “xxx.” Simultaneously, a row filter exists for `table_1.column_2`.

The original query would appear as:

Original query

```
SELECT column_1 FROM table_1 WHERE column_3
```

With both column-masking and row-level filtering policies applied from Ranger, the query above is rewritten to the following:

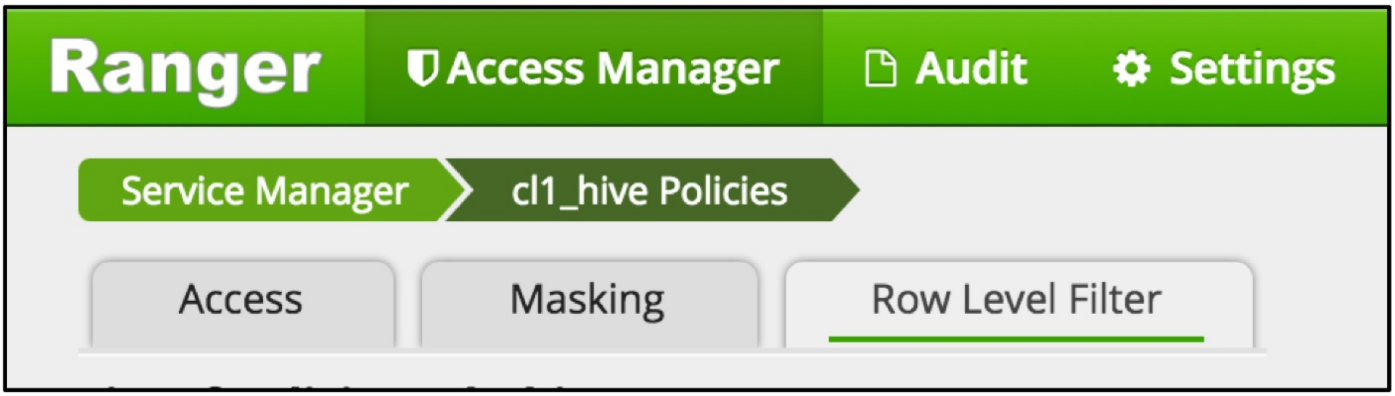
Query with column-masking and row-level filtering policies

```
WITH filtered_and_masked_table_1 AS ( SELECT 'xxx' AS column_1, column_2, col
```

For organizations currently utilizing Apache Ranger and configured to apply policies to Dremio, the application of row-level filtering and column-masking is automatic. However, in order to apply these security measures, you must also create security policies from Ranger, which will then propagate down to Dremio when the affected users perform a query.

To create a security policy in Apache Ranger:

1. Navigate to the *Service Manager* page, and then select the desired **Hive Service**.
2. Click the **Column Masking** or **Row Level Filter** tab.



3. Click **Add New Policy**.

Now you are at the **Add Policy** screen. The sections below describe the elements contained on that page.

Policy Details

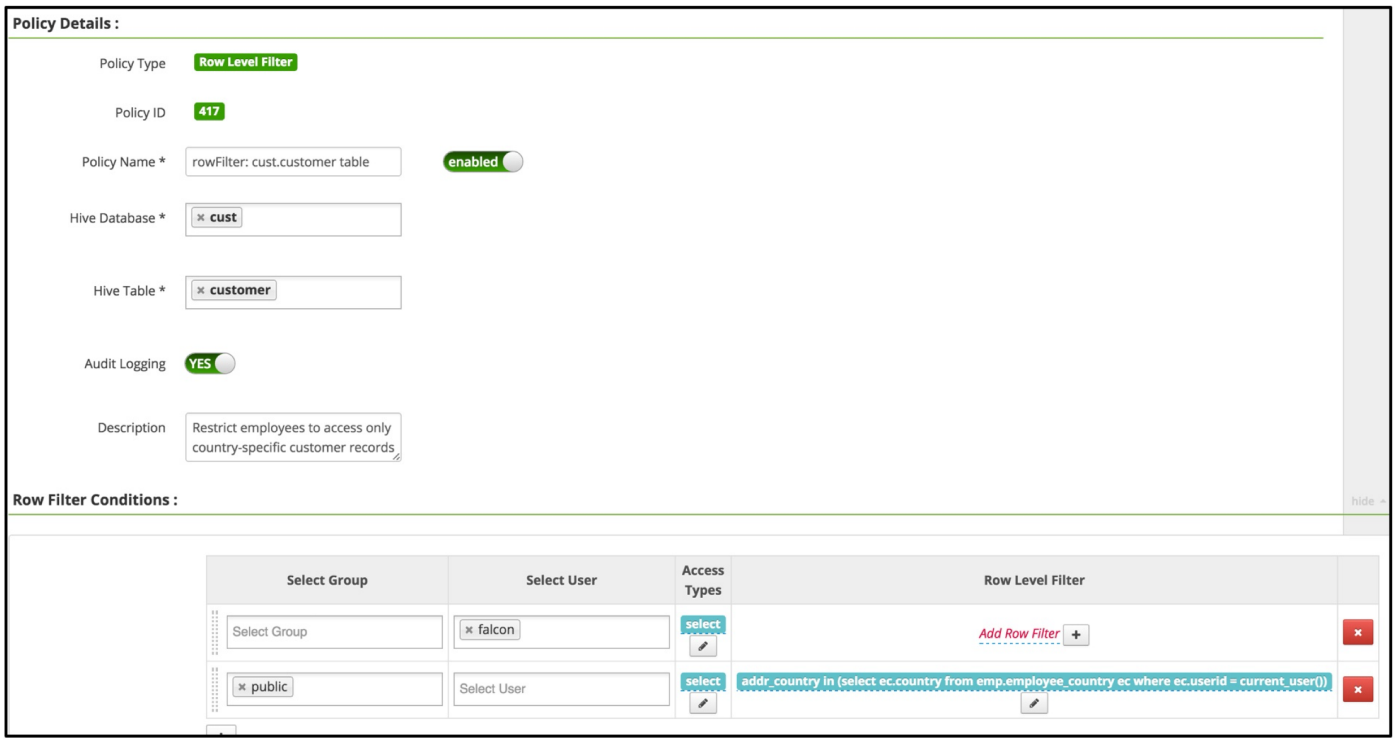
The following table describes the **Policy Details** section of the *Create Policy* screen.

Field	Required	Description
Policy Name	YES	The name of the policy. This value cannot be duplicated in another policy.
Policy Label		Tags to help categorize and make the policy more searchable.
Hive Database	YES	The name of the database(s) to which this policy applies. The field will display auto-complete options based on what matches the current entered value. The database must be a parent to any specified table(s) below, otherwise it will fail to apply.
Hive Table	YES	The name of the table(s) to apply the policy toward. Please ensure the tables are associated with the database(s) specified

Description		A description of the policy to explain its intended purpose, its audience, and any other relevant details.
enabled/disabled	YES	Determines whether the specific policy apply to the specified users, groups/roles, and conditions. If disabled, the security policy will not affect user queries.
normal/override	YES	Controls how the policy is prioritized against other existing security policies. If set to **override**, this policy will ignore other policies that may restrict or grant access beyond the scope specified here.
Audit Logging	YES	Controls whether auditing is enabled and is set to **YES** by default. Auditing tracks all user actions impacted by this policy.

Row Filter Conditions

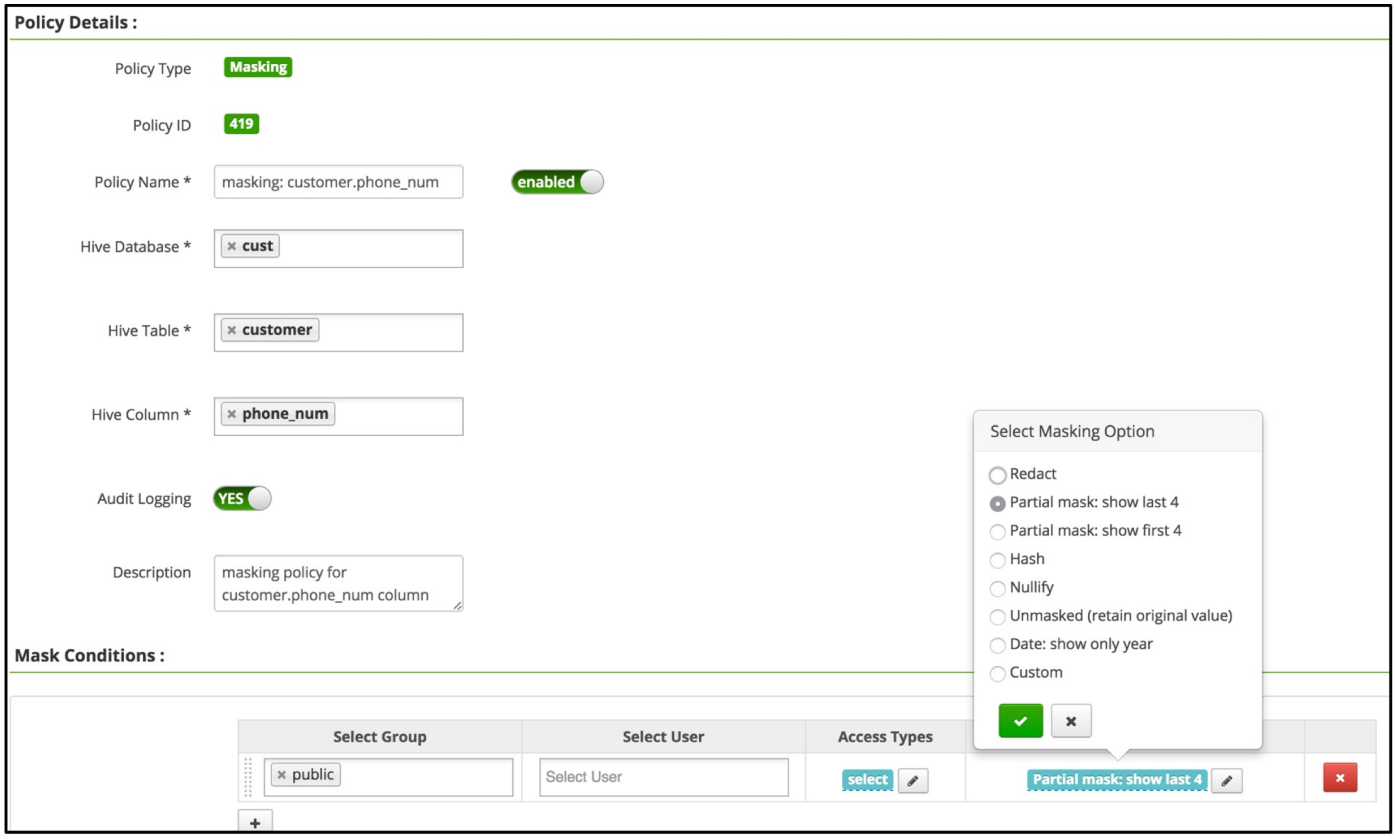
The following table describes the **Row Filter Conditions** section of the *Create Policy* screen.



Field	Description
Select Group	The group(s) of users to which this policy applies. The `public` group will apply to all users. If no group is specified, a user must be provided.

Access Types	The action which the specified group(s) or user(s) may utilize from the Dremio SQL Editor. Currently, the only type available is `select`. This is used in tandem with the `WHERE` clause as specified in the **Row Level Filter** field.
Row Level Filter	A valid `WHERE` clause as entered in the **Enter filter expression** pop-up upon clicking the **Add Row Filter** button. To allow full `SELECT` access to users without row-level filtering, do not click this button. Filters are applied based on top-down order, meaning the filter at the top is applied first, then the second filter, and so on.

Mask Conditions



Field	Description
Select Group	The group(s) of users to which this policy applies. The `public` group will apply to all users. If no group is specified, a user must be provided.
Select User	The individual user(s) to which this policy applies. If no user is specified, a group must be provided.

Types	with the `WHERE` clause as specified in the **Row Level Filter** field.
Select Masking Type	The type of column-masking behavior to apply to the associated users/groups when they query the table specified on this policy. • **Redact -** Replaces all alphabetic characters with `x` and all numeric characters with `n`. • **Partial mask: show last 4 -** Displays only the last four characters of the full column value's. • **Partial mask: show first 4 -** Displays only the first four characters of the full column value's. • **Hash -** Replaces all characters with a hash of the entire cell's value. • **Nullify -** Replaces all characters in the cell with a `NULL` value. • **Unmasked (retain original value) -** No masking is applied to the cell. • **Date: show only year -** Displays the year portion of a date string, defaulting the month and day to `01/01`. • **Custom -** Specifies a custom column masked value or valid Dremio expression. Custom masking may not use Hive UDFs. Masks are applied based on top-down order, meaning the mask at the top is applied first, then the second mask, and so on.

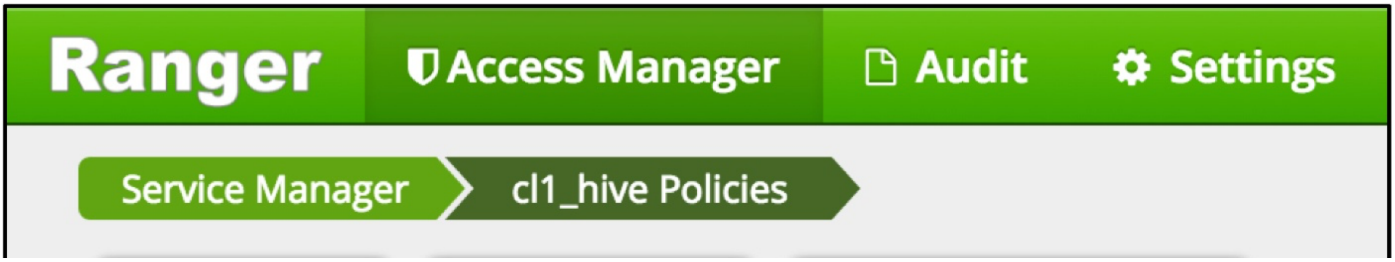
Adding a Row-Level Filter Policy

This section outlines how to create a row-level filter policy from the Apache Ranger console.

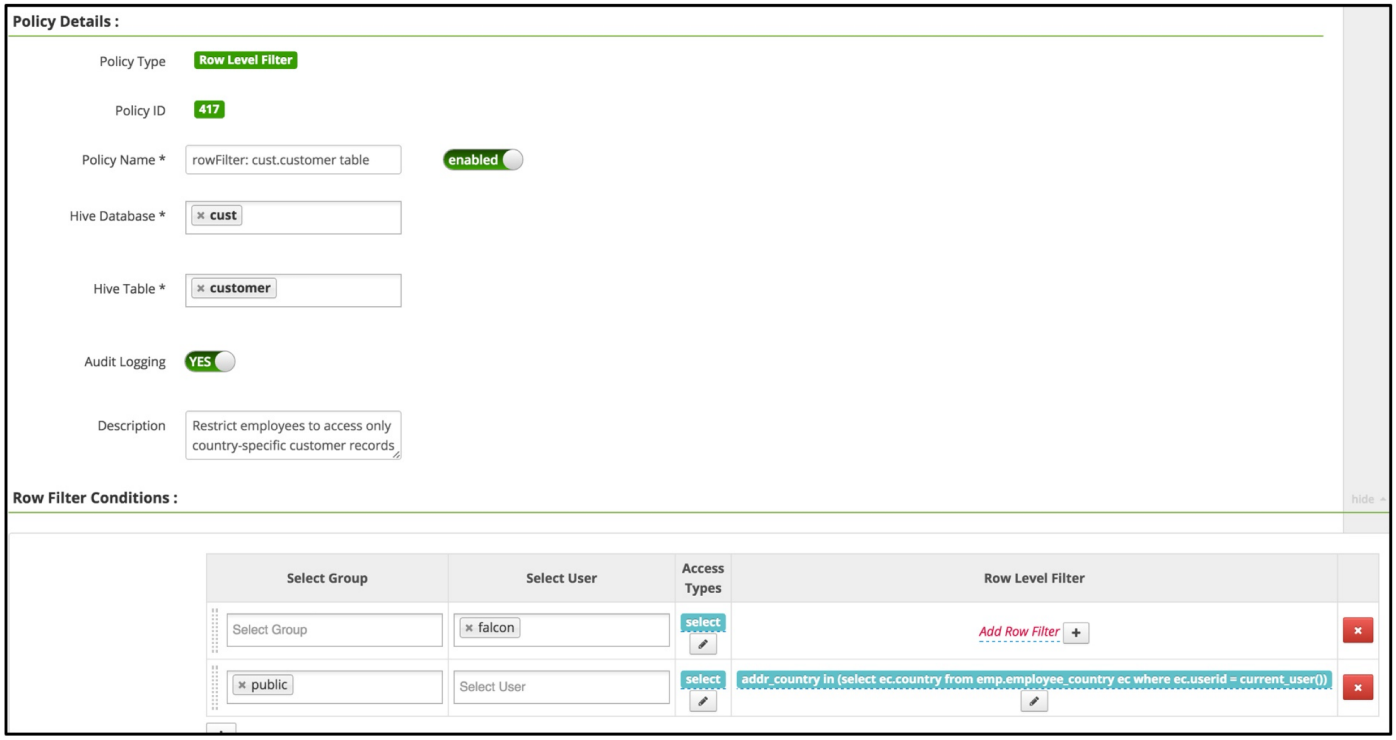
For additional instructions and information about row-level filtering, see Row-level filtering and column-masking using Apache Ranger policies in Apache Hive.

To create a policy that enforces row-level access control, perform the following steps:

1. From the Apache Ranger console, navigate to the *Serivce Manager* page, and then select the desired **Hive Service**.
2. Click the **Row Level Filter** tab.



3. Click **Add New Policy**.
4. From the *Create Policy* page, provide values for the **Policy Details** and **Row Filter Conditions** sections.
5. Add any desired conditions, or else leave the **Row Filter Conditions** section blank to apply no filtering.



6. To move a condition under the **Row Filter Conditions** section, click the dotted icon on the left-hand side of the row, and then drag it to the desired new location,
7. Click **Add** to save the new policy.

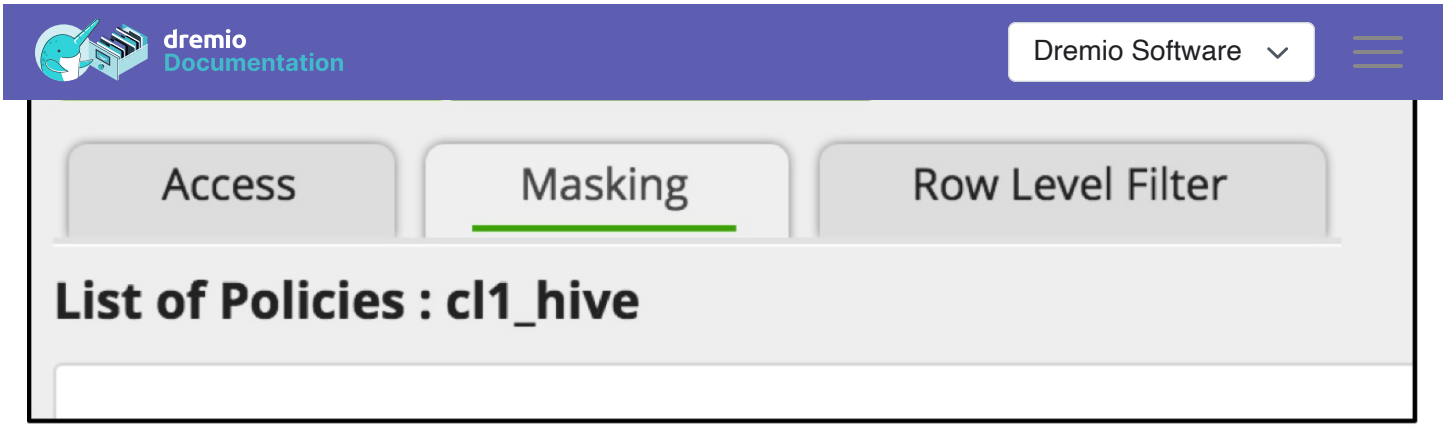
Adding a Column-Masking Policy

This section outlines how to create a column-masking policy from the Apache Ranger console.

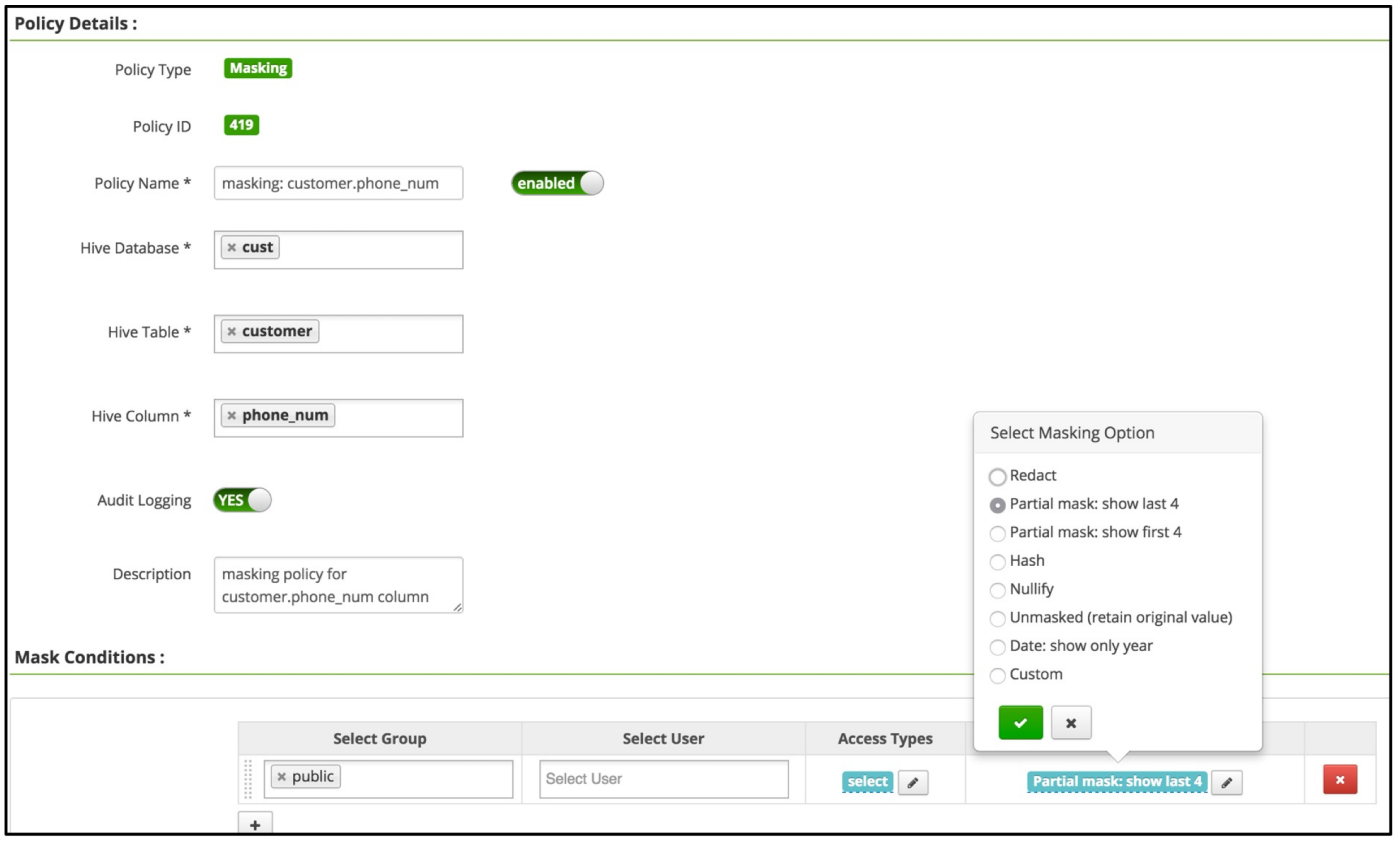
For additional instructions and information about column-masking, see Row-level filtering and column-masking using Apache Ranger policies in Apache Hive.

To create a policy that enforces row-level access control, perform the following steps:

1. From the Apache Ranger console, navigate to the *Serivce Manager* page, and then select the desired **Hive Service**.
2. Click the **Row Level Filter** tab.



3. Click **Add New Policy**.
4. From the *Create Policy* page, provide values for the **Policy Details** and **Mask Conditions** sections.
5. Create any desired masking conditions under the **Mask Conditions** section, or else select **Unmasked (retain original value)** to not apply masking for a user or group.



5. To move a condition under the **Mask Conditions** section, click the dotted icon on the left-hand side of the row, and then drag it to the desired new location,
6. Click **Add** to save the new policy.

Using Dremio's Built-In Filtering/Masking

comparison to the security policies possible with Ranger implementations. Where possible, utilize this service to enforce row-level permissions and column-masking as described above.

> **Note**) We recommend using Dremio 20.0+ in tandem with Apache Ranger to apply user/role-based security policies across all datasets while querying a table/view. Otherwise, you may utilize Dremio's built-in SQL functions (as describe below) to manually enforce filtering and masking.

Creating a Virtual Dataset with Column-Masking

By using the query_user() or `is_member()` SQL functions, a virtual dataset can be configured manually to allow selective masking of columns for different users/roles without the need to create multiple datasets.

The following is a sample SQL command for a virtual dataset (VDS) using column-masking syntax:

Example for virtual dataset (VDS) using column-masking

```
SELECT CASE WHEN query_user() IN ('dave','mike') OR is_member('Accounting') T
```

The SQL function `is_member()` is case-insensitive by default. This may be circumvented by adding a boolean `is_member(groupname, <case-sensitivity boolean>)` to control case-sensitivity. Simply set it to `true` to enable case-sensitivity or `false` to disable. If omitted from the SQL command, the boolean defaults to `false`.

Creating a Virtual Dataset with Row-Level Permissions

By using the query_user() or `is_member()` SQL functions, a virtual dataset can be configured to allow manual selective filtering of rows for different users/roles without the need to create multiple datasets.

The following is a sample SQL command for a virtual dataset (VDS) using row-level filtering syntax:

Example for virtual dataset (VDS) using row-level filtering

```
SELECT * FROM mongo.vds.business WHERE (state = 'NV' AND query_user() IN ('da
```

Simply set it to `true` to enable case-sensitivity or `false` to disable. If omitted from the SQL command, the boolean defaults to `false`.

Limitations

- Column access policies are not supported yet
- Reflections are removed/invalidated from consideration if the policy alters column data output on runtime