Introduction

Apache Ranger provides centralized security for Enterprise Hadoop ecosystem, including fine-grained access control and centralized auditing. In ranger-0.5 Version, Apache Ranger introduced stack-model to make it easier for new components to use Apache Ranger authorization and auditing. Further, to enable extending/adapting Apache Ranger for new or deployment-specific authorization requirements, the stack model provides hooks like context-enrichers and policy-conditions.

In this document, we will see the details of extending Apache Ranger to support authorization of a Hive request based on all Hive resources requested by it. In particular, a Ranger Policy may be written which prohibits certain users/groups from accessing specified Hive Resources together in the Hive query.

Here is the outline of the tasks to be done:

- Enable Ranger to support deny policies for Hive.
- Register a policy-condition to enforce mutual exclusion/ prohibition of Hive objects specified in the policy.
- Create/update Apache Ranger policies to specify the locations to allow/deny the access

Enable Ranger to support deny policies for Hive

To enable Ranger GUI to support specification of deny policies for the Hive component please update Hive's service-definition by including the following:

```
"options":  {
     "enableDenyAndExceptionsInPolicies":"true"
}
```

Register Policy Conditions

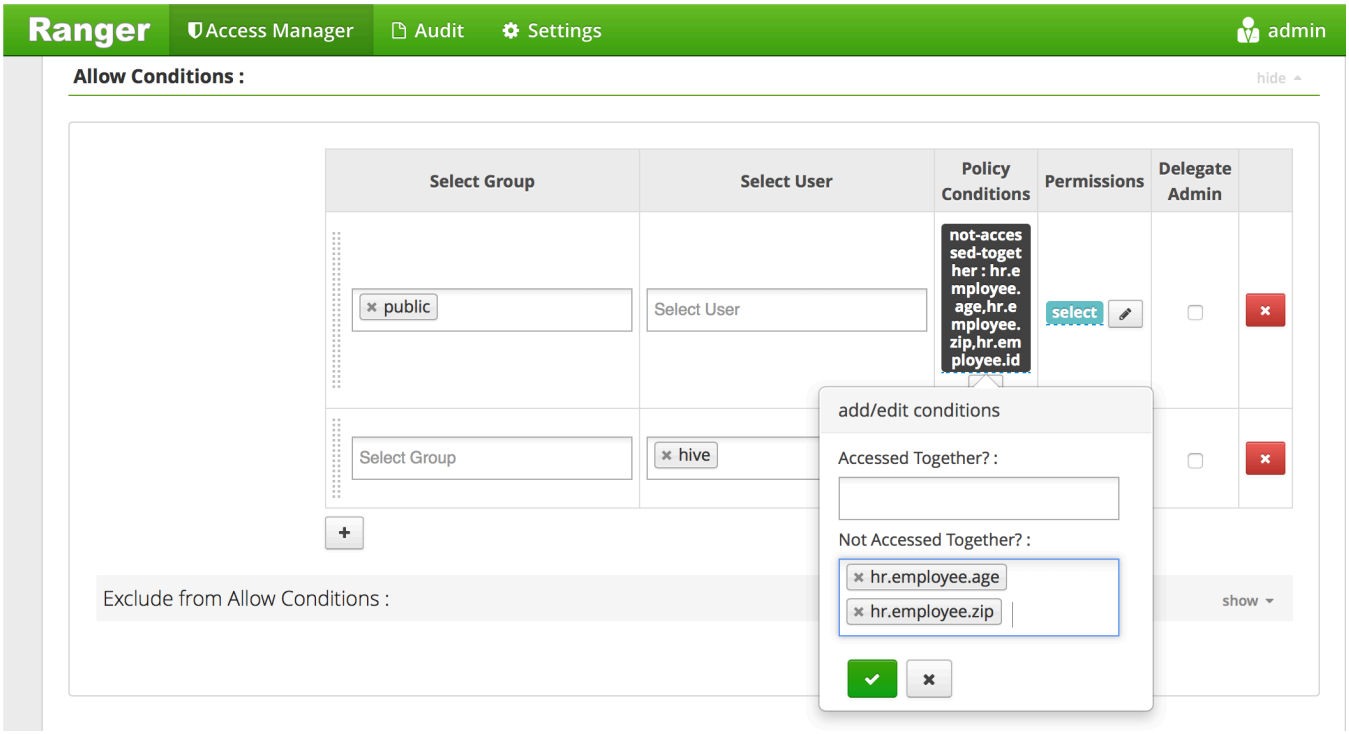
Apache Ranger provides policy-condition hooks to execute custom conditions while evaluating authorization requests. To determine the authorization result, Apache Ranger policy engine evaluates the policies that are applicable to the accessed resource. Only when various criteria like user/group, access-type and policy-conditions specified in the policy match the request, the policy engine will use the policy to determine the result.

Policy conditions **`RangerHiveResourcesAccessedTogether`** (and **`RangerHiveResourcesNotAccessedTogether`**), available in Ranger 0.6, evaluate to true only when the specified request contains (does not contain) minimum of two resources specified in the policy-condition. This can be used to deny Hive requests that include Hive resources that should not be accessed together.

To register policy-condition for the Hive component, please update the Hive component's service-definition by including the following:

```
      "policyConditions": [{
            "itemId": 1,
            "name": "accessed-together",
            "evaluator":
"org.apache.ranger.plugin.conditionevaluator.RangerHiveResourcesAcces
sedTogetherCondition",
            "evaluatorOptions": {},
            "label": "Accessed Together ?",
            "description": "List of Hive resources"
      }, {
            "itemId": 2,
            "name": "not-accessed-together",
            "evaluator":
"org.apache.ranger.plugin.conditionevaluator.RangerHiveResourcesNotAc
cessedTogetherCondition",
            "evaluatorOptions": {},
            "label": "Not Accessed Together?",
            "description": "List of Hive resources"
      }]
```

Once this policy condition is registered with Ranger, the policy editing UI will prompt for condition values to be used during evaluation – as shown below:



Example

In this section, we will see the details of an Apache Ranger policy that denies access to a specific Hive Resources when they are accessed together with other Hive Resources using the policy-conditions described above. The steps are :

- Create a sample Hive table.
- Determine the access policy for this table.
- Create a Ranger policy to implement access policy.
- Test Ranger policy by accessing Hive resources using a command tool (such as beeline).

Create a sample Hive table

For a Postgres database, create an employee table in hr database with the following commands:

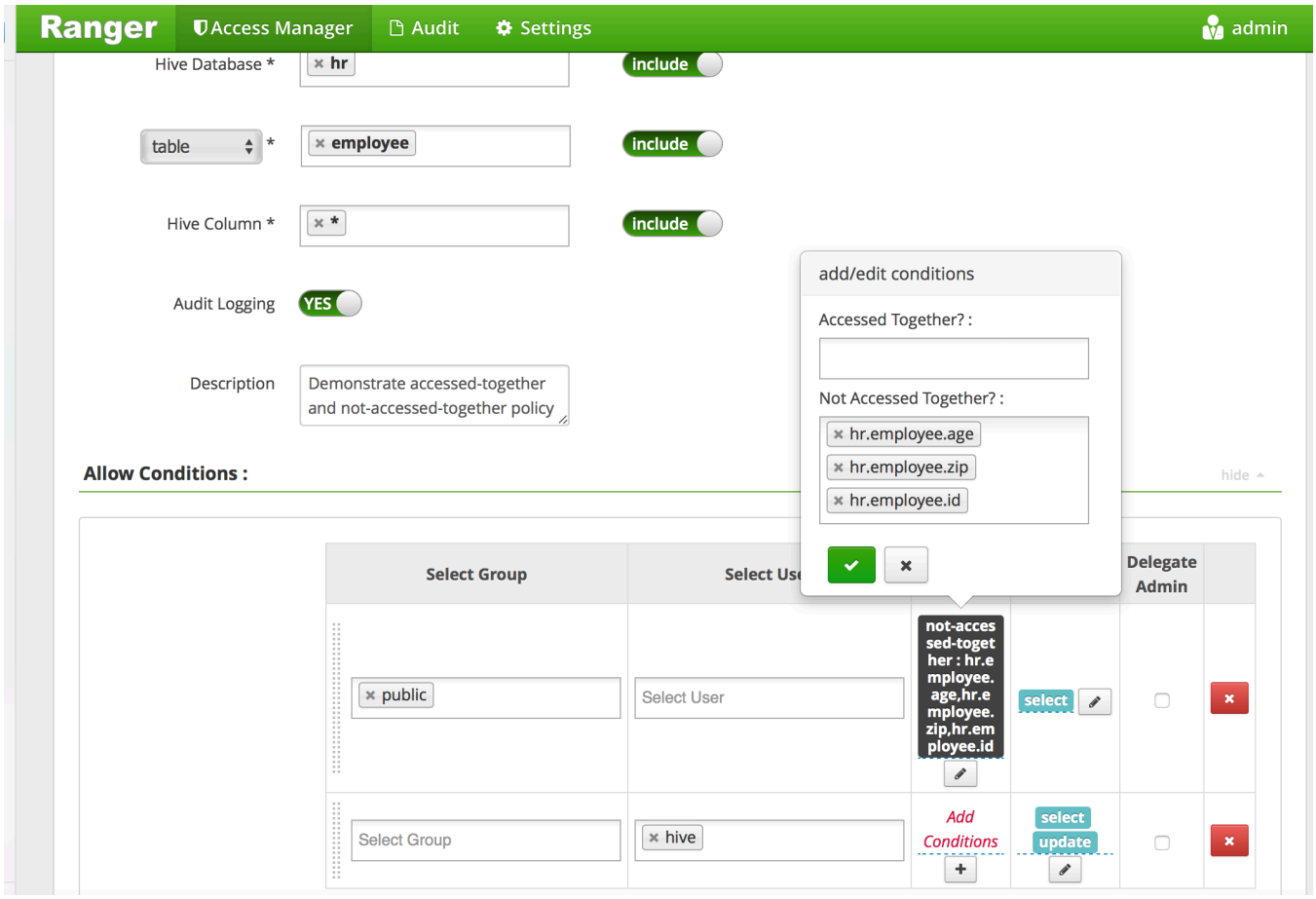
```
==> CREATE DATABASE hr;
==> CREATE TABLE hr.employee(id INT, name STRING, salary INT, age
INT, zip STRING);
```

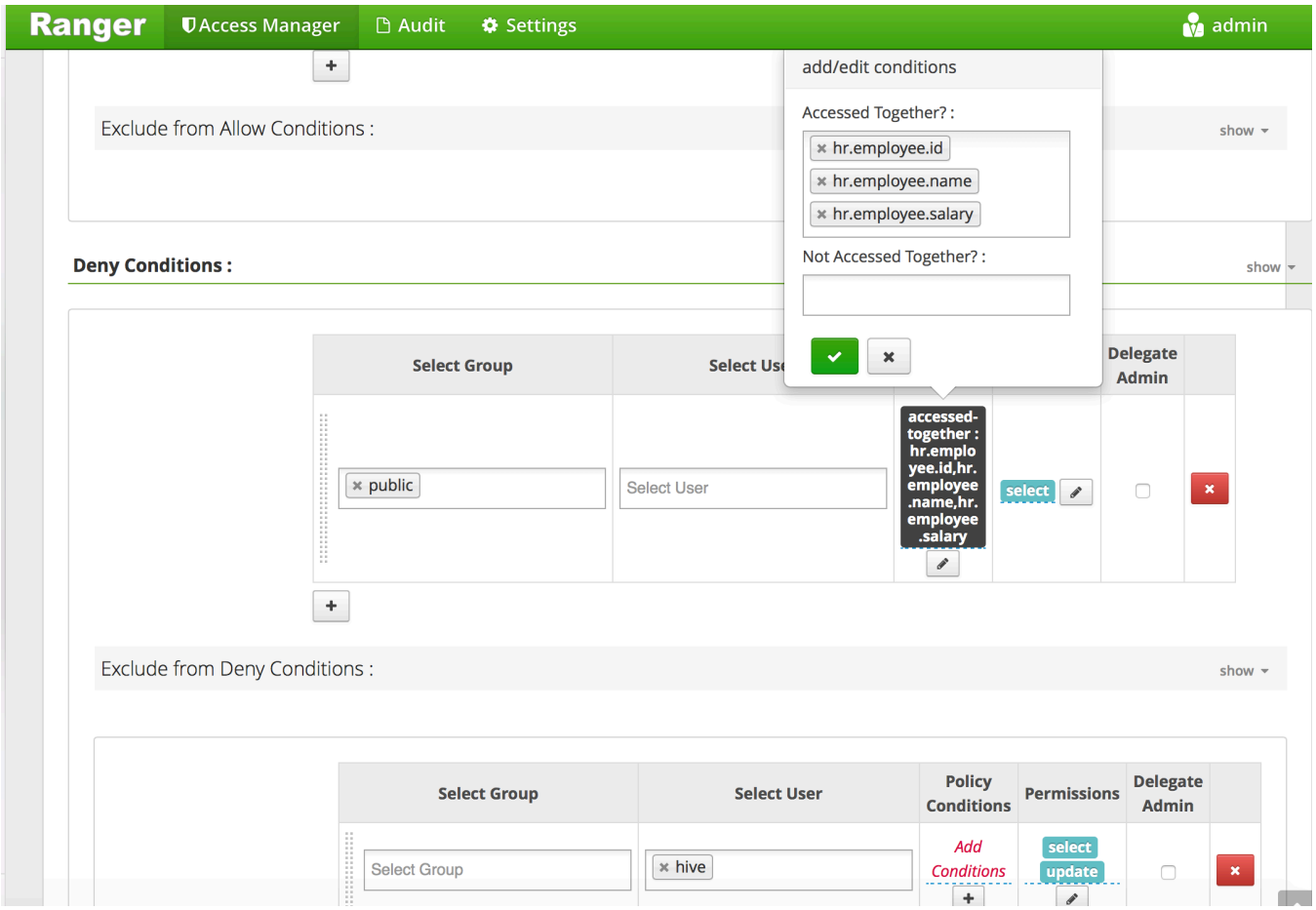
Determine the access policy for this table

- User ‘hive’ may read/update all columns of employee table in hr database individually or collectively.
- All other users cannot read
    - id column with either age, zip, name or salary column
    - age column with zip column
    - name column with salary column

Create a Ranger policy to implement access policy

Ranger policy for implementing the access policy above is as follows:





This policy applies to all columns of employee table. Allow policy-items specify that user ‘hive’ can read/update any Hive resource in the request, but for all other users (i.e. members of group ‘public’), allow read access only if any two columns from id, age and zip do not appear together in the request. Similarly, deny policy-items specifies that all users, except ‘hive’, is denied access if any two columns from id, name and salary appear together in the request.

Testing Ranger policy by accessing Hive resources using beeline

When logged in as ‘hive’:

```
% ./beeline -n hive -p hive -u jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 1.2.1000.2.5.0.0-1245)
Driver: Hive JDBC (version 1.2.1000.2.5.0.0-1245)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.2.1000.2.5.0.0-1245 by Apache Hive
```

```
0: jdbc:hive2://localhost:10000> select id, name, salary, age, zip from
hr.employee;
+-----+-------+---------+------+------+--+
| id  | name  | salary  | age  | zip  |
+-----+-------+---------+------+------+--+
+-----+-------+---------+------+------+--+
No rows selected (0.171 seconds)
```

When logged in as 'testuser':

```
% ./beeline -n testuser -p testuser -u jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 1.2.1000.2.5.0.0-1245)
Driver: Hive JDBC (version 1.2.1000.2.5.0.0-1245)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.2.1000.2.5.0.0-1245 by Apache Hive
0: jdbc:hive2://localhost:10000> select id, name, salary, age, zip from
hr.employee;
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied:
user [testuser] does not have [SELECT] privilege on [hr/employee/age,id,name,salary,zip]
(state=42000,code=40000)
0: jdbc:hive2://localhost:10000> select id, name from hr.employee;
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied:
user [testuser] does not have [SELECT] privilege on [hr/employee/id,name]
(state=42000,code=40000)
0: jdbc:hive2://localhost:10000> select age, zip from hr.employee;
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied:
user [testuser] does not have [SELECT] privilege on [hr/employee/age,zip]
(state=42000,code=40000)
0: jdbc:hive2://localhost:10000> select name, salary from hr.employee;
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied:
user [testuser] does not have [SELECT] privilege on [hr/employee/name,salary]
(state=42000,code=40000)
0: jdbc:hive2://localhost:10000> select name, zip from hr.employee;
+-------+------+--+
| name  | zip  |
+-------+------+--+
+-------+------+--+
No rows selected (0.211 seconds)
0: jdbc:hive2://localhost:10000> select salary, age from hr.employee;
+---------+------+--+
| salary  | age  |
+---------+------+--+
+---------+------+--+
```

Syntax of specifying Hive Objects in policy conditions

As shown above, a dot-separated string, such as ‘hr.employee.id’, specifies column ‘id’ in table ‘employee’ of database ‘hr’. It is possible to omit some but not all parts of this specification. The following shows how such specification is interpreted.

- ‘hr.employee’ - interpreted as ‘hr.employee.*’
- ‘hr.employee.’ – interpreted as ‘hr.employee.*’
- ‘hr..id’ – interpreted as ‘hr.*.employee’
- ‘employee’ – interpreted as ‘employee.*.*’

If no value is specified for policy-condition, then such policy-condition is considered as not specified at all for that policy-item.