Authorization Rules

Setup of Authorization Rules based on Groups of Users

Rolf Krahl

ICAT Meeting, Dublin, Mar 2014

Why?

Why using groups to setup the authorization rules?

- Authorization based on groups is very flexible.
- Easy to grant or to revoke permissions: simply add the user to or remove him from the corresponding group.
- Access policies may be individually defined for each investigation.
- Users may manage permissions themselves. They only need CRUD permission on UserGroup related to the corresponding group.
- InvestigationUser is also used for other purposes (e.g. TopCAT). One might wish to setup permissions independently from this.

Access Groups

For each investigation, create three access groups:

`investigation_<name>_writer`: Shall get CRUD permission on objects related to the investigation, such as Datafiles, Datasets, Samples, Keywords, Parameters and so on.

`investigation_<name>_reader`: Shall get R permission on objects related to the investigation.

`investigation_<name>_owner`: Shall get permission to manage access permissions on the investigation.

Here `<name>` is replaced by the investigation name.

Rules: Variant 1, per Investigation Rules

Simple way to setup access rules: create a set of rules for each investigation.

Rule

crudFlags: CRUD

what: `Datafile <-> Dataset <-> Investigation[name='<name>']`

grouping: `investigation_<name>_writer`

Rule

crudFlags: CRUD

what: `UserGroup <-> Grouping[name='investigation_<name>_writer']`

grouping: `investigation_<name>_owner`

Rules: Variant 1, per Investigation Rules

- Simple.
- Works fine in test installation.
- Drawback: excessive number of rules, 28 rules per investigation, more then a half of all objects in the ICAT are rules.
- Will this scale reasonably in a production size deployment?

Rules: Variant 2, Static Rules Based on “Magic Names”

Try to achieve the same result with a limited number of generic rules:

Rule

crudFlags: CRUD

what:
```
SELECT df FROM Datafile df
JOIN df.dataset ds JOIN ds.investigation i
JOIN Grouping g JOIN g.userGroups ug
JOIN ug.user u
WHERE g.name =
CONCAT('investigation_',i.name,'_writer')
AND u.name = :user
```

grouping: NULL

Rules: Variant 2, Static Rules Based on “Magic Names”

Rule

crudFlags:	CRUD
what:	see below
grouping:	NULL

```
SELECT aug FROM UserGroup aug
JOIN aug.grouping ag
JOIN Investigation i
JOIN Grouping g JOIN g.userGroups ug
JOIN ug.user u
WHERE ag.name =
CONCAT('investigation_',i.name,'_writer)
AND g.name =
CONCAT('investigation_',i.name,'_owner')
AND u.name = :user
```

Rules: Variant 2, Static Rules Based on “Magic Names”

- Works in principle in test installation.
- Only fixed set of static rules.
- Drawback: incredible slow! Seven minutes to answer a simple query on a test ICAT having about 700 investigations.

Rules: Variant 2, Static Rules Based on “Magic Names”

Why is it so slow?

Query

```
SELECT df FROM Datafile df
JOIN df.dataset ds JOIN ds.investigation i
JOIN Grouping g JOIN g.userGroups ug
JOIN ug.user u
WHERE g.name = CONCAT('investigation_',i.name,'_writer')
AND u.name = :user
```

Missing relation between Investigation and Grouping
$\Rightarrow$ Need to evaluate string expression on full cartesian product.

Complexity: $\mathcal{O}(n^2)$ in the number of investigations.

InvestigationGroup

Possible solution: Add the missing relation. Add to ICAT schema:

InvestigationGroup

Many to many relationship between investigation and grouping
Uniqueness constraint: grouping, investigation

Relationships:

Card	Class	Field	Cascaded
1,1	Investigation	investigation	No
1,1	Grouping	grouping	No

Other fields:

Field	Type
role	String [255]

Rules: Variant 3, InvestigationGroup

Add relations between Investigation and Groups:

InvestigationGroup

investigation:	Investigation `<name>`
grouping:	Grouping `investigation_<name>_writer`
role:	writer

and accordingly for reader and owner.
Add rules:

Rule

crudFlags: CRUD

what:
```
Datafile <-> Dataset <-> Investigation <->
InvestigationGroup [role='writer'] <->
Grouping <-> UserGroup <-> User [name=:user]
```

grouping: NULL

Rules: Variant 3, InvestigationGroup

Rule

crudFlags: CRUD

what:
```
SELECT tug FROM UserGroup tug
JOIN tug.grouping tg
JOIN tg.investigationGroups tig
JOIN tig.investigation i
JOIN i.investigationGroups uig
JOIN uig.grouping ug
JOIN ug.userGroups uug JOIN uug.user u
WHERE (tig.role = 'writer'
OR tig.role = 'reader')
AND uig.role = 'owner' AND u.name = :user
```

grouping: NULL

Rules: Variant 3, InvestigationGroup

- Solves the issue: only one fixed set of static rules. Only three Grouping and three InvestigationGroup per Investigation.
- Provides all the flexibility.
- Should scale reasonably, at least no obvious reason why it should not.
- Requires a change in the ICAT schema.
- But: this change is limited to the addition of the new type. Already existing types are not altered ⇒ no compatibility issues. Sites not using it should not be affected in any way.

Discussion

Thank you for your attention!

Comments? Discussion?

Which Objects to Setup Rules for?

Writers get CRUD permission on:

- `Sample <-> Investigation`,
- `Dataset <-> Investigation`,
- `Datafile <-> Dataset <-> Investigation`,
- `InvestigationParameter <-> Investigation`,
- `SampleParameter <-> Sample <-> Investigation`,
- `DatasetParameter <-> Dataset <-> Investigation`,
- `DatafileParameter <-> Datafile <-> Dataset <-> Investigation`,
- `Shift <-> Investigation`,
- `Keyword <-> Investigation`,
- `Publication <-> Investigation`,
- `InvestigationInstrument <-> Investigation`,

they get RU permission on `Investigation`, and R permission on `InvestigationUser <-> Investigation`.