Arvados - Bug #16007

Permission graph update is slow with large numbers of groups

01/13/2020 03:43 PM - Peter Amstutz

Status:	Resolved	**Start date:**	05/26/2020
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:	API	**Estimated time:**	0.00 hour
Target version:	2020-07-01 Sprint		

Description

400 users
5k groups
3.5k permission links

May be due to the large number of permission links (one project is described as having 20 sharing links).

Investigate, try to recreate

Propose an update strategy that is more efficient that current one (which recomputes all permissions any time any permission changes).

Subtasks:

Task # 16185: Review 16007-permission-table-rb	**Resolved**
Task # 16481: Review 16007-validate-group-class	**Resolved**

Related issues:

Related to Arvados Epics - Story #16443: Redesign permission table updates	**Resolved**	**04/01/2020**	**06/17/2020**
Related to Arvados - Bug #16811: Ensure that "public favorites" still work	**Resolved**	**09/15/2020**	
Related to Arvados - Bug #12994: Can't create user group with the same name a...	**Resolved**		

Associated revisions

Revision 6989ee05 - 06/16/2020 09:25 PM - Peter Amstutz

Merge branch '16007-permission-table-rb' refs #16007

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <peter.amstutz@curii.com>

Revision de11b137 - 06/19/2020 03:38 PM - Peter Amstutz

Merge branch '16007-validate-group-class' refs #16007

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <peter.amstutz@curii.com>

Revision e3722334 - 06/22/2020 08:06 PM - Peter Amstutz

Restore link to fix test refs #16007

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <peter.amstutz@curii.com>

Revision 72bac44a - 06/25/2020 08:04 PM - Peter Amstutz

require 'update_permissions' fix for migration refs #16007

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <peter.amstutz@curii.com>

Revision cd9f489c - 06/26/2020 02:28 PM - Peter Amstutz

Don't validate links in 'fix roles' migration refs #16007

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <peter.amstutz@curii.com>

History

#1 - 01/13/2020 03:56 PM - Peter Amstutz

- Description updated

- Subject changed from Permission graph update is slow with large numbers of groups to Permission graph update is slow

#2 - 01/13/2020 06:25 PM - Peter Amstutz

- Category set to API

- Subject changed from Permission graph update is slow to Permission graph update is slow with large numbers of groups

#3 - 02/19/2020 05:05 PM - Peter Amstutz

- Story points set to 2.0

- Description updated

#4 - 02/19/2020 05:09 PM - Peter Amstutz

- Target version set to 2020-03-11 Sprint

#5 - 02/19/2020 05:10 PM - Tom Clegg

If building the materialized permission view is unavoidably slow, perhaps we can compute a per-user subset of it while it's being updated, instead of blocking on the full update.

#6 - 02/20/2020 07:18 AM - Stanislaw Adaszewski

```
@Just thinking loud here - maybe more indirection levels could help. For example:

links:
tail    permission    head
Group_1 --can_read--> Group_2
Group_1 --can_read--> Project_3
Group_2 --can_read--> User_4
Group_2 --can_read--> User_5

materialized_group_permissions (no users here, hence the big speed win)
tail    permission    head
Group_1 --can_read--> Group_2
Group_1 --can_read--> Project_3
Group_2 --can_read--> Project_3 # this is a derived link

then to query user-accessible collections one could

SELECT DISTINCT c.* FROM collections AS c
  LEFT JOIN materialized_group_permissions AS gp ON (gp.head=c.owner_uuid)
  LEFT JOIN links AS ug ON(ug.tail=gp.tail AND ug.head=:current_user)
  WHERE c.owner_uuid=:current_user OR ug.permission IS NOT NULL

If it gets properly planned by the SQL engine it shouldn't be a very heavy query. Am I missing something here?
@
```

#7 - 02/20/2020 07:35 AM - Stanislaw Adaszewski

One could even take it a step further:

```
links:
tail    permission    head
Group_1 --can_read--> Group_2
Group_2 --can_read--> Group_3
Group_1 --can_read--> Project_3
Group_3 --can_read--> User_4
Group_3 --can_read--> User_5

Project_3 --can_read--> Project_4
Project_4 --can_read--> Project_5

materialized_group_permissions (no users OR projects here!)
tail    permission    head
Group_1 --can_read--> Group_2
Group_2 --can_read--> Group_3
Group_1 --can_read--> Group_3 # derived

materialized_project_permissions (no users here)
tail      permission    head
Project_3 --can_read--> Project_4
Project_4 --can_read--> Project_5
Project_3 --can_read--> Project_5 # derived
```

```
SELECT DISTINCT c.* FROM collections AS c
  LEFT JOIN materialized_project_permissions AS pp ON(pp.head=c.owner_uuid)
  LEFT JOIN links AS ll ON(ll.head=pp.tail)
  LEFT JOIN materialized_group_permissions AS gp ON (gp.head=ll.tail)
  LEFT JOIN links AS ug ON(ug.tail=gp.tail AND ug.head=:current_user)
  LEFT JOIN links AS dp ON(dp.tail=:current_user AND db.head=pp.tail)
  WHERE c.owner_uuid=:current_user # direct ownership
    OR ug.permission IS NOT NULL   # permission via group
    OR dp.permission IS NOT NULL   # direct permission for user
```

#8 - 02/20/2020 07:38 AM - Stanislaw Adaszewski

IF I am not mistaken you would have to refresh the materialized tables only on some particular changes, e.g. adding a Group->User link wouldn't entail a refresh at all. And those refreshes should be much much faster. Unless I missed something, the only question would be whether it is detrimental to SELECT queries.

#9 - 02/20/2020 07:57 AM - Stanislaw Adaszewski

And on top of everything you could also follow the idea in #16155 and instead of refreshes just insert/remove the relevant rows manually as it would be probably easier to conceptualize in the above setup. I think this could be done only for materialized_project_permissions because projects are for sure the most numerous among groups and they are constantly created, it is a fundamental operation in organizing data. Much more frequent than Group/User operations.

#10 - 02/26/2020 05:01 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

#11 - 03/10/2020 01:56 PM - Peter Amstutz

- Target version changed from 2020-03-11 Sprint to 2020-03-25 Sprint

#12 - 03/23/2020 10:17 PM - Peter Amstutz

Current strategy under consideration:

Materialized view becomes a regular table. Perform incremental updates. Added permissions can be inserted immediately, removing permissions requires removing affected subgraph from the permission table and recomputing from remaining permissions.

Scaling will generally be (number of users that can see a group) * (1 + number of subprojects under a group).

Enforce the following constraints:

- group_class must be one of 'role' or 'project' and cannot change group_class after creation
- permission links cannot be changed after creation
- valid permission links are
  - from 'user' to 'role'
  - from 'role' to 'project'
  - from 'role' to 'user'
  - from 'user' to 'project'
  - from 'user' to 'collection' (??? traditionally allowed but adds complexity)
- owner_uuid for most objects only 'user' or 'project'
- 'role' can only be owned by 'user'

When a new project is created

1. Select where target_uuid = owner_uuid, get a list of (user_uuid, perm_level)
2. Insert new rows for each (user_uuid, new project uuid, perm_level from parent project row)

When a project is trashed

1. Starting from project_uuid, traverse down projects owned by the starting project
2. Update trashed = 1

When a project is untrashed

1. Starting from project_uuid, traverse down projects owned by the starting project
2. Update trashed = 0

When a project is deleted

1. Starting from project_uuid, traverse down projects owned by the starting project
2. Delete all rows for target_uuid of project or child project

When a link to a group (role or project) is added

1. Select where target_uuid = tail_uuid, get a list of (user_uuid, perm_level)
2. Starting from head_uuid, traverse permission links and subprojects to get list of targets, compute perm_level
3. Insert rows for each (user_uuid, target uuid, min(link perm, target perm_level))

When a link to a group (role or project) is removed

1. Starting from head_uuid, traverse permission links and subprojects to get list of targets
2. Delete all rows for target_uuid of targets
3. Recompute ownership
    1. Select where target_uuid = owner_uuid, get a list of (user_uuid, perm_level)
    2. Insert new rows for each (user_uuid, target_uuid, perm_level)
4. Recompute links
    1. Find links with head_uuid in list of targets
    2. Re-add permissions associated with links to targets

When project is moved (owner_uuid changes)

Treat like link remove behavior + link add behavior.

#13 - 03/25/2020 01:43 PM - Peter Amstutz

- Target version changed from 2020-03-25 Sprint to 2020-04-08 Sprint

#14 - 03/25/2020 01:43 PM - Peter Amstutz

- Story points changed from 2.0 to 5.0

#15 - 03/25/2020 01:45 PM - Peter Amstutz

- Status changed from New to In Progress

#16 - 03/28/2020 10:16 AM - Stanislaw Adaszewski

Sounds great, very much looking forward to it.

If this turns out not to be sufficient I was doing some reading and developed a conviction that https://debezium.io/ + Apache Kafka + (Neo4j or Dgraph) would surely do the trick. It could complicate the architecture of Arvados a bit but an in-memory graph database would be able to query the permissions in real time, thus we would eliminate all delays for good (as long as we don't absolutely need joins, which from a brief look at the code we don't).

Hope the simpler enhancement is enough.

#18 - 04/08/2020 12:39 PM - Peter Amstutz

- Target version changed from 2020-04-08 Sprint to 2020-04-22

#19 - 04/22/2020 02:44 PM - Peter Amstutz

- Target version changed from 2020-04-22 to 2020-05-06 Sprint

#20 - 05/01/2020 02:55 PM - Peter Amstutz

- File select_subtree.sql added

notes

#21 - 05/01/2020 02:55 PM - Peter Amstutz

- File populate.py added

script to create a lot of users, groups and projects

#22 - 05/06/2020 02:24 PM - Peter Amstutz

- Target version changed from 2020-05-06 Sprint to 2020-05-20 Sprint

#23 - 05/18/2020 06:29 PM - Peter Amstutz

- Related to Story #16443: Redesign permission table updates added

#25 - 05/20/2020 02:17 PM - Peter Amstutz

- Target version changed from 2020-05-20 Sprint to 2020-06-03 Sprint

#26 - 05/20/2020 06:57 PM - Peter Amstutz

Some very rough benchmarks.

Test: creating a group.

Doing an incremental update followed by a full database refresh ("checking" mode)

real 0m12.052s
real 0m12.978s
real 0m12.671s

Doing just incremental update:

real 0m0.743s
real 0m0.726s
real 0m0.725s

The most expensive query in incremental update (obviously, because this the query that does all the work).

update_permissions.select (149.4ms)

Haven't done any analysis on the query to see if it can be streamlined with indexes, or if the construction of the query interferes with the ability of the query planner to optimize.

#27 - 05/21/2020 03:21 PM - Peter Amstutz

More benchmarks.

Create a database with

- 500 users
- 100 groups
- 5000 projects
- each user belongs to 10 groups
- each group can read 10 projects
- there are 100 top level projects
- each project has 10 subprojects
- each subproject has 5 sub-sub-projects

timing taken with

```
time arv group create --group '{"name": "foobar*", "group_class": "project"}'
```

owner_uuid	master	16007
	0m10.818s	0m2.881s
	0m9.507s	0m2.668s
	0m10.781s	0m2.976s
x5ya6-j7d0g-l9y46sifreth480	0m9.495s	0m2.578s
x5ya6-j7d0g-l9y46sifreth480	0m11.080s	0m2.518s
x5ya6-j7d0g-l9y46sifreth480	0m9.681s	0m2.545s

So it is a measurable improvement, however the "compute permissions" part is still taking ~2s which seems like a lot. I also noticed that a full permission recompute on master takes about 10s while a full permission recompute on 16007 takes about 40s. So there certainly seems to be an opportunity for query optimization here.

#28 - 05/22/2020 01:16 PM - Peter Amstutz

Did some query optimization.

owner_uuid	master	16007	16007 with optimization work
	0m10.818s	0m2.881s	0m0.593s
	0m9.507s	0m2.668s	0m0.609s
	0m10.781s	0m2.976s	0m0.693s

x5ya6-j7d0g-l9y46sifreth480	0m9.495s	0m2.578s	0m0.589s
x5ya6-j7d0g-l9y46sifreth480	0m11.080s	0m2.518s	0m0.570s
x5ya6-j7d0g-l9y46sifreth480	0m9.681s	0m2.545s	0m0.586s

#29 - 05/26/2020 06:25 PM - Peter Amstutz

16007-permission-table-rb @ a5c857c5ab354d3b0e6a51653d0f1f21c108e131

https://ci.arvados.org/view/Developer/job/developer-run-tests/1872/

Branch was rebased to minimize the noise from development churn.

Extensive code comments on the new method in the following files:

services/api/app/models/arvados_model.rb

services/api/db/migrate/20200501150153_permission_table.rb

services/api/lib/refresh_permission_view.rb

A couple things left to do, to in this branch yet:

- Adding constraints/validating/migrating groups to avoid undefined states, as mentioned at the top of #note-12 and discussed at https://gitter.im/arvados/development?at=5eb5a315a9de3d01b1f86455
- Adding an upgrade note.

#30 - 05/26/2020 09:35 PM - Lucas Di Pentima

Reviewing c36d5cb

First of all, thanks for all the comments and explanations, they'll be of enormous help both at review time and later on the lifecycle of the code. Some comments & questions from my 1st review pass. I haven't tried the code yet nor dug into the SQL stuff.

- Commit 2353613 has lots of bulletpoints. Don't know if it's the result of several commits being squashed into one, but as a reviewer I would prefer to have one commit per bullepoint as it's easier to follow the author's thought process.
- File services/api/app/models/link.rb - Line 89: Commented code?
- File services/api/lib/refresh_permission_view.rb
  - Maybe the file should be renamed as we don't have a permission view anymore.
  - Should queries from update_permissions() called from within a transaction?
  - Line 147: Should check_permissions_against_full_refresh() only run when RAILS_ENV == development?
- API.AsyncPermissionsUpdateInterval should be marked as No-Op on the default config comments from the Go code.
- File services/api/test/unit/user_test.rb
  - Typo at line 171: destory
  - Commented code at line 173
- At dc67102, the migration file services/api/db/migrate/20200501150153_permission_table.rb got a code change on line 103 (s/STABLE/IMMUTABLE) but on the structure.sql file, line 159 the should_traverse_owned() function definition still has the STABLE keyword (not sure what's that about yet). thought would be worth mentioning.

#31 - 05/28/2020 08:24 PM - Peter Amstutz

Lucas Di Pentima wrote:

> Reviewing c36d5cb
>
> First of all, thanks for all the comments and explanations, they'll be of enormous help both at review time and later on the lifecycle of the code. Some comments & questions from my 1st review pass. I haven't tried the code yet nor dug into the SQL stuff.
>
> - Commit 2353613 has lots of bulletpoints. Don't know if it's the result of several commits being squashed into one, but as a reviewer I would prefer to have one commit per bullepoint as it's easier to follow the author's thought process.

That's right, it is a bunch of commit comments squashed together.

The original branch is still available 16007-permission-table. But the history has a lot of commits that are experiments, dead ends, debugging, WIPs and other noise. I'm not sure I could even reconstruct my own thought process from it, that's why I squash/rebased it.

> - File services/api/app/models/link.rb - Line 89: Commented code?

Enabled.

- File services/api/lib/refresh_permission_view.rb
  - Maybe the file should be renamed as we don't have a permission view anymore.

Now update_permissions.rb

- Should queries from update_permissions() called from within a transaction?

I added a transaction. However, calling LOCK TABLE outside a transaction is an error, so in all cases where it was being called it was already in a transaction.

- Line 147: Should check_permissions_against_full_refresh() only run when RAILS_ENV == development?

That's a good idea -- now it runs in 'test' mode and disabled for the others.

- API.AsyncPermissionsUpdateInterval should be marked as No-Op on the default config comments from the Go code.
- File services/api/test/unit/user_test.rb
  - Typo at line 171: destory
  - Commented code at line 173

Took that out, it wasn't doing anything.

- At dc67102, the migration file services/api/db/migrate/20200501150153_permission_table.rb got a code change on line 103 (s/STABLE/IMMUTABLE) but on the structure.sql file, line 159 the should_traverse_owned() function definition still has the STABLE keyword (not sure what's that about yet). thought would be worth mentioning.

I re-ran the migration and updated structure.sql

now 16007-permission-table-rb @ a4ade714a38980e239e9bc01244cba6b33575206

https://ci.arvados.org/view/Developer/job/developer-run-tests/1874/

--> I checked this and there's a failure with group-sync tests, there may be a bug.

#32 - 05/29/2020 02:43 AM - Peter Amstutz

Fixed a bug (& added a test so it would be caught by the API tests and not just the group-sync tool tests). See added code comments for details.

I also rebased because git started complaining about a missing license header.

Now 16007-permission-table-rb @ b879b9cd18ddba6ba87b65f81eba676114478a06

https://ci.arvados.org/view/Developer/job/developer-run-tests/1876/

#33 - 06/03/2020 03:50 PM - Peter Amstutz

- Target version changed from 2020-06-03 Sprint to 2020-06-17 Sprint

#34 - 06/05/2020 09:12 PM - Peter Amstutz

16007-permission-table-rb @ 22e96d42f3c1d2414a52f266096b74011deabbf2

Changed approach a little bit from #note-32. The permission computation no longer traverses users except in the special case of starting from the user to compute their complete permissions.

This avoids the disastrous case where we traverse a group with many other users group and end up inefficiently traversing every user on the system.

This simplifies and speeds up the primary compute_permission_subgraph query. However, it means transitive permissions through users (user has can_manage on another user and can see their stuff) has to be handled separately.

My solution is to determine the set of users that the current user has can_manage to, and add them to the readable_by query.

This works for one level of indirection -- user A can_manage user B means A can see B's stuff. However if B -> C, it won't find A -> C. I could make it use a recursive query but I am reluctant to increase the complexity of a query on the critical path, so I have left it for now.

The ability for users to "manage" other users is a relatively obscure and undocumented feature. Given the complexity involved in maintaining it (without exaggeration, it probably added weeks of development time to the branch), we should reevaluate this feature in the future.

The other major change from #note-32 is backing off on the use of postgres functions and going with inline subqueries via string templates. This is less elegant but it is necessary for the query optimizer to do its job, the optimization barrier penalty of lateral joins and postgres functions was just too much.

https://ci.arvados.org/view/Developer/job/developer-run-tests/1898/

#35 - 06/08/2020 03:27 PM - Peter Amstutz

16007-validate-group-class @ 1f1cec38f109a93513fab7f2a2d0c774290ac8fa

Rebased on 22e96d42f (review can start from the commit after this one).

Adds the following new constraints, and migrates existing databases to conform:

1. group_class must be either "role" or "project". Invalid group_class are migrated to "role"
2. "role" cannot own things. Anything owned by a role is migrate to a can_manage link and reassigned to the system user.
3. only "role" and "user" can have outgoing permission links. Permission links originating from projects are deleted by the migration.
4. "role" is always owned by the system_user. When a group is created, it creates a can_manage link for whoever would have been the owner_uuid. Migration adds can_manage links and reassigns roles to the system user.
5. A permission link can have the permission level ('name') updated but not head_uuid, tail_uuid or link_class.

The rationale for these constraints is that these are already de-facto imposed by the Workbench UI. A user or admin who used the command line or SDK to create an invalid group or use a group in an invalid way would become confused as it wouldn't be displayed properly by the UI. Adding these constraints now also potentially makes it easier to change the permission system in the future.

Adds a migration, and a test for the migration. Manually tested migration of a large database (used for testing the earlier 16007 branch).

https://ci.arvados.org/view/Developer/job/developer-run-tests/1901/

#36 - 06/09/2020 02:57 PM - Lucas Di Pentima

Peter Amstutz wrote:

> 16007-permission-table-rb @ 22e96d42f3c1d2414a52f266096b74011deabbf2

I don't want to block this any further, so these are my questions after reading all the changes several times. I found the changes to be difficult to follow for me, but that's surely because I also didn't worked on the previous permission view code. As we have extensive testing coverage my questions are more related to performance/readability than to code correctness.

As I mentioned previously, I've made a back box type test, checking the time a new arvbox instance takes to run the populate.py script:

- Master version: 77 minutes 8 segs
- 16007 version: 22 minutes 35 segs

Questions/comments:

- Would it be a good idea in terms of code readability to refer to perm_level values as constants where possible?
- On the permission_graph_edges view creation, would it be useful to add DISTINCT on links' @SELECT@s to avoid duplicate work if multiple equal links exist on the database?
- Regarding temporary tables usage: I've read psql documentation and it says that the autovacuum doesn't see those tables, and thus we may need to submit a manual ANALYZE command before doing queries on it to avoid the optimizer making wrong assumptions. On update_trash do you think it would be useful to add an index on the trash_at column?

Other than that, it LGTM.

#37 - 06/09/2020 07:50 PM - Tom Clegg

16007-validate-group-class @ 1f1cec38f

It seems possible to have a hierarchy of groups with no group_class (and permission links from projects) that work just fine up until now because nobody bothers to look at them in Workbench. This migration would be messy in such a case. Seems like we should have an upgrade note with a hint about how to check for non-standard groups/links that will be affected/deleted.

Updating the database directly ("update groups set group_class=...") is fast, but skips creation of audit logs that could be useful for someone trying to understand/reverse change history. Since we expect very few rows to be affected, perhaps it's better to optimize for auditing rather than speed. ActiveRecord update would also give us an opportunity to log the changes, like we do in the "projects can't have outgoing permission links" part. (Is there a chicken-and-egg problem here where we need to combine 1) and 2) to avoid saving a partly-fixed but still-invalid-according-to-new-rules version?)

If I'm following, @role_creator is used to track the value that would normally be owner_uuid when creating a "role" group so we can add a "manage role" permission link later, and it's either an "owner_uuid" value provided by the caller, or current_user by default. If the first case, @role_creator is a bit misleading. Maybe @requested_owner_uuid or @implicit_manager_uuid?

In app/models/link.rb, error "must be a role, was #{tail_obj.group_class}" -- how about "must be a user or role, was group with group_class #{...}"

Log message "...will be removed" should be "removing ..." -- and may as well mention the uuid of the link itself in case someone actually follows up on these messages and wants to find the deleted links in the logs table.

https://doc.arvados.org/api/permission-model.html needs a few minor changes to sync up with this change.

What's the rationale for removing the empty_collection seed? It looks like we added it in #3072 to make things easier for Workbench and other clients

which were assuming that if a PDH works at all, it works everywhere, even "list collections where ...". Removing from database_seeds.rb seems to mean any side effects (anticipated or not) will only appear on new installations.

#38 - 06/11/2020 03:29 PM - Peter Amstutz

Lucas Di Pentima wrote:

> Questions/comments:
>
> - Would it be a good idea in terms of code readability to refer to perm_level values as constants where possible?

Yes. I added constants REVOKE_PERM=0 and CAN_MANAGE_PERM=3 so the the calls to update_permissions are clearer.

> - On the permission_graph_edges view creation, would it be useful to add DISTINCT on links' @SELECT@s to avoid duplicate work if multiple equal links exist on the database?

I don't think so. In practice this seems force the query to do a full scan of all the edges in order to sort them and eliminate duplicates. The current approach allows the optimizer to use indexes and avoid retrieving permission links it isn't interested in. Redundant traversals get eliminated later.

> - Regarding temporary tables usage: I've read psql documentation and it says that the autovacuum doesn't see those tables, and thus we may need to submit a manual ANALYZE command before doing queries on it to avoid the optimizer making wrong assumptions. On update_trash do you think it would be useful to add an index on the trash_at column?

You mean, create a temporary index for the temporary table? I've noticed that the query plans will sometimes sort intermediate results to get similar benefits to an index. I don't think it makes sense to add indexes without spending time with EXPLAIN ANALYZE to identify places where the query planner needs some help.

#39 - 06/11/2020 05:17 PM - Peter Amstutz

Now @ 6f514b3e6aa21afddaa527bf852cff3a5801aa19

https://ci.arvados.org/view/Developer/job/developer-run-tests/1908/

#40 - 06/11/2020 05:22 PM - Lucas Di Pentima

Updates at 6f514b3e6aa21afddaa527bf852cff3a5801aa19 LGTM, thanks!

#41 - 06/16/2020 07:59 PM - Peter Amstutz

After merging master, it turned up another edge case. When there are redundant edges (for example, two permission links granting X permission on Y where except one is can_read and the other is can_write) updating or removing one permission link would ignore the other, leading to incorrect permissions. I solved this by making the concept of "override the edge being updated" more specific and passing in the target or link uuid, so that other edges are still considered in the additional_perms clause.

98ec1f0093bb097f9ccb78ac43a9858f20084ad6

https://ci.arvados.org/view/Developer/job/developer-run-tests/1920/

#42 - 06/16/2020 09:23 PM - Lucas Di Pentima

98ec1f0093bb097f9ccb78ac43a9858f20084ad6 LGTM.

#43 - 06/17/2020 03:42 PM - Peter Amstutz

- Target version changed from 2020-06-17 Sprint to 2020-07-01 Sprint

#44 - 06/18/2020 12:02 AM - Peter Amstutz

Tom Clegg wrote:

> 16007-validate-group-class @ 1f1cec38f
>
> It seems possible to have a hierarchy of groups with no group_class (and permission links from projects) that work just fine up until now because nobody bothers to look at them in Workbench. This migration would be messy in such a case. Seems like we should have an upgrade note with a hint about how to check for non-standard groups/links that will be affected/deleted.

I added an upgrade note with all the information and commands.

> Updating the database directly ("update groups set group_class=...") is fast, but skips creation of audit logs that could be useful for someone trying to understand/reverse change history. Since we expect very few rows to be affected, perhaps it's better to optimize for auditing rather than speed. ActiveRecord update would also give us an opportunity to log the changes, like we do in the "projects can't have outgoing permission

> links" part. (Is there a chicken-and-egg problem here where we need to combine 1) and 2) to avoid saving a partly-fixed but still-invalid-according-to-new-rules version?)

A little bit, you can't set group_class to 'role' unless the owner_uuid is also the system user. I rearranged it so it sets both at the same time.

> If I'm following, @role_creator is used to track the value that would normally be owner_uuid when creating a "role" group so we can add a "manage role" permission link later, and it's either an "owner_uuid" value provided by the caller, or current_user by default. If the first case, @role_creator is a bit misleading. Maybe @requested_owner_uuid or @implicit_manager_uuid?

Split the difference and renamed it to @requested_manager_uuid

> In app/models/link.rb, error "must be a role, was #{tail_obj.group_class}" -- how about "must be a user or role, was group with group_class #{...}"

Done.

> Log message "...will be removed" should be "removing ..." -- and may as well mention the uuid of the link itself in case someone actually follows up on these messages and wants to find the deleted links in the logs table.

Done.

> https://doc.arvados.org/api/permission-model.html needs a few minor changes to sync up with this change.

Updated. Ended up doing a bit more rewriting than a few minor changes.

> What's the rationale for removing the empty_collection seed? It looks like we added it in #3072 to make things easier for Workbench and other clients which were assuming that if a PDH works at all, it works everywhere, even "list collections where ...". Removing from database_seeds.rb seems to mean any side effects (anticipated or not) will only appear on new installations.

The immediate issue was that the empty collection was owned by the anonymous_group, but the anonymous group needs to be a role (so you can share things with it) which means it can't directly own things any more. I thought it was mainly something tests rely on, but couldn't find any tests that really required it either, so I thought I could take it out.

I put the empty_collection seed back. Now it is owned by the system user and there is a can_read link from anonymous_group.

5502559ac286dcf807261cec86b983f061788908

https://ci.arvados.org/view/Developer/job/developer-run-tests/1924/

re-run of workbench integration

https://ci.arvados.org/job/developer-run-tests-apps-workbench-integration/2033

#45 - 06/18/2020 02:56 PM - Tom Clegg

16007-validate-group-class @ 5502559ac

Upgrade notes:

"The arvados-sync-groups tool has been updated to reflect these constraints." -- I think if this is a hint that a site using the group-sync tool needs to do/check/plan something during the upgrade, it should be more explicit about what they should do/check/plan, otherwise it should be removed.

"To determine which groups have invalid group_class with this command (...):" -- sentence error, maybe remove "with this command" to match the other examples

"To list which project groups have outgoing permission links. Such links are now invalid and will be deleted by the migration:" -- ditto, maybe use parens like the other examples

Suggest adding a sentence before those examples like "Before upgrading, use the following commands to find out which groups and permissions in your database will be automatically modified or deleted during the upgrade."

"Permissions can be obtained indirectly by following multiple permission links or nested ownership." -- the following list doesn't seem to mention nested ownership -- is it worth defining that, or is it obvious enough? Should the bullet points here say "role Group" specifically?

"Permission links where tail_uuid is a User allow can_read on the link record by that user. (User can discover her own permission grants.)" -- Punctuation/parens should match the style in the other nearby sentences (like this).

"A role can be both the target (head_uuid) and origin (tail_uuid) of a permission link." -- since the permission link is singular, this seems to say it's OK to make a useless permission link from a role to itself. Is the intent to say a permission link's tail_uuid and head_uuid can be two different roles?

(I'm assuming a permission link from a role to itself is safely ignored, but it doesn't seem like a super valuable thing to mention here.)

"To make objects visible to the public, they can be shared with the "anonymous" role." -- This isn't new material so it's scope creep but maybe this should say "...visible to the public as well as all logged-in users"? IIRC that is the sneaky difference between sharing with the anon user (which only shares when not logged in) and sharing with the anon role (which is what most people probably want/expect).

Maybe mention that role groups might be renamed during the migration in order to avoid collisions.

Rest LGTM, thanks!

#46 - 06/18/2020 04:17 PM - Peter Amstutz

Tom Clegg wrote:

> 16007-validate-group-class @ 5502559ac

Incorporated your notes into here:

95e79c507c74ee2364a01b82c771495b91a6de0d

#47 - 06/19/2020 01:36 PM - Tom Clegg

Noticed doc/api/permission-model.html.textile.liquid is missing a word, "roles will [be] renamed to ensure they are unique"

Rest LGTM, thanks!

#48 - 06/23/2020 01:59 PM - Peter Amstutz

- Status changed from In Progress to Resolved

#49 - 09/14/2020 06:36 PM - Tom Clegg

- Related to Bug #16811: Ensure that "public favorites" still work added

#50 - 10/07/2020 02:11 AM - Peter Amstutz

- Release set to 25

#51 - 04/13/2021 05:42 PM - Tom Clegg

- Related to Bug #12994: Can't create user group with the same name as a top level project added

Files

select_subtree.sql	10.7 KB	05/01/2020	Peter Amstutz
populate.py	2.58 KB	05/01/2020	Peter Amstutz