Arvados - Story #12032

[API] Allow projects to be deleted (ie placed in the trash can)

07/25/2017 06:39 PM - Tom Morris

Status:	Resolved	**Start date:**	08/08/2017
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:	API	**Estimated time:**	0.00 hour
Target version:	2017-09-27 Sprint		

Description

As a user, I would like deleted projects to be placed in the Trash rather than cluttering my Home project and when the Trash is emptied, the project and all of its contents, recursively, get deleted.

Details about desired behavior after the Workbench "trash" button is hit on a project:

- The project and everything below it (including other subprojects) stop showing up in the usual places in Workbench, arv-mount, etc., even if there are explicit permission links (like collection-sharing links) to the project or its contents.
- The project and its contents can still be retrieved using "get" and "list" API calls with the include_trash flag.
- The project appears in its parent project's "trashed items" tab in Workbench. Clicking the project shows the trashed project with its implicitly-trashed contents: there is an obvious indication that the project itself is trashed, and (TBD?) all of its contents appear in the "trashed items" tab instead of their usual places.
- If a user follows an old link/bookmark to the project, the page is not found. Except: (TBD?) The "not found" error page should acknowledge that the project is still available in the trash, and offer a link to the "trashed project" view described above.
- The project and all of its previously-untrashed contents can be un-trashed by calling the "untrash" method on the project. However, if the project already contained items which were trash before the project was trashed, untrashing the project does not untrash those items.
- Any individual item contained in the trashed project can be untrashed by changing its owner_uuid field to a project/user that is not trashed.
- Data integrity: There is no sequence of API calls that could result in a collection being deleted (or otherwise made invisible to keep-balance) less than blobSignatureTTL seconds after the last time a client read or updated that collection's manifest.

The trash/untrash APIs should be similar to the collection trash/untrash APIs. Specifically:

- Each project should have a delete_at timestamp that can be set to a time ≥blobSignatureTTL in the future

The trash/untrash APIs should work quickly and atomically, even when a large hierarchy of items is affected. This implies that the permission graph gains a "trashed?" flag, which is taken into account when retrieving results for get/list APIs, even for admin users.

Components to be updated:

- API server
- API docs
- Workbench
- arv-mount

Subtasks:

Task # 12131: Review 12032-project-trash	**Resolved**
Task # 12126: API server support for trash	**Resolved**

Related issues:

Related to Arvados - Story #10816: [AP] [spike] Implement permission lookups ...	**Resolved**	**01/04/2017**
Related to Arvados - Story #12125: Client support for deleting projects	**Resolved**	**08/08/2017**

Associated revisions

Revision 68bdf4cb - 09/25/2017 07:39 PM - Peter Amstutz

Merge branch '12032-project-trash' refs #12032

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <pamstutz@veritasgenetics.com>

Revision 0017c5ba - 09/26/2017 07:41 PM - Tom Clegg

12032: Fix compatibility with PostgreSQL 9.4.

9.4 doesn't have idle_in_transaction_session_timeout or row_security
configs, so couldn't execute structure.sql.

9.4 doesn't write "Dumped from/by version X", so 9.4 and 9.6 dev boxes
want an edit war.

refs #12032

Arvados-DCO-1.1-Signed-off-by: Tom Clegg <tclegg@veritasgenetics.com>

History

#1 - 07/25/2017 06:48 PM - Tom Morris

- Description updated

#2 - 07/25/2017 06:51 PM - Tom Clegg

Implementation thoughts:

- Add is_trashed flag to groups (projects)
- Permission graph (recursive view) needs an is_trashed flag indicating "this or an ancestor is trashed"
- Ensure untrashed project B inside trashed project A is not readable via permission link from user U → project B

#3 - 08/02/2017 07:14 PM - Tom Morris

- Target version changed from 2017-08-16 sprint to 2017-08-30 Sprint

#4 - 08/08/2017 07:08 PM - Tom Morris

- Story points set to 3.0

#5 - 08/08/2017 07:33 PM - Tom Clegg

- Description updated

#6 - 08/16/2017 07:51 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

#7 - 08/30/2017 06:47 PM - Peter Amstutz

- Status changed from New to In Progress

#8 - 08/30/2017 06:49 PM - Peter Amstutz

- Target version changed from 2017-08-30 Sprint to 2017-09-13 Sprint

#9 - 08/31/2017 05:44 PM - Peter Amstutz

12032-project-trash @ dfcbdd5823bf85a462305a58df79a0112d3020d8

- Updates permission_view to propagate "trashed" status
- Fix bug in permission_view where permissions would "follow" through users as if they were groups
- Refactor "trashable" model and controller behavior into modules, shared by Collection and Group
- readable_by in ArvadosModel, Log and Container now use subquery on permission_view to apply permissions and filter for "trashed"
- Unit & controller tests for new behavior
- Nothing actually gets deleted yet, even when delete_at has passed.
- Documentation updated, but Workbench & arv-mount changes will go in their own branch.

Some notes:

> Any individual item contained in the trashed project can be untrashed by changing its owner_uuid field to a project/user that is not trashed.

Currently doesn't work because the generic update operation fails before getting to the point where it changes the owner_uuid.

> The project appears in its parent project's "trashed items" tab in Workbench. Clicking the project shows the trashed project with its implicitly-trashed contents: there is an obvious indication that the project itself is trashed, and (TBD?) all of its contents appear in the "trashed items" tab instead of their usual places.

Depending on exactly how we want this to behave, we may need a "only_trashed" query mode; presently with include_trash: true there is no way to distinguish items contained in a trashed project from regular not-trashed items. (Currently not implemented in this branch.)

> Data integrity: There is no sequence of API calls that could result in a collection being deleted (or otherwise made invisible to keep-balance) less

> than blobSignatureTTL seconds after the last time a client read or updated that collection's manifest.

The rules for setting delete_at are the same for both Group and Collection.

Currently the collection object is not aware if it is part of a trashed collection, so if you request a trashed collection record with include_trash it may have signatures that are later than the parent group's delete_at.

#10 - 08/31/2017 07:50 PM - Tom Clegg

Asking for a few clarifications before looking at the code:

Peter Amstutz wrote:

> - Fix bug in permission_view where permissions would "follow" through users as if they were groups

Following through users is the desired behavior when there's a group→user "manage" link. Was this happening for read/write, too?

>> Any individual item contained in the trashed project can be untrashed by changing its owner_uuid field to a project/user that is not trashed.
>
> Currently doesn't work because the generic update operation fails before getting to the point where it changes the owner_uuid.

Is there a failing test for this to make sure we don't forget?

>> The project appears in its parent project's "trashed items" tab in Workbench. Clicking the project shows the trashed project with its implicitly-trashed contents: there is an obvious indication that the project itself is trashed, and (TBD?) all of its contents appear in the "trashed items" tab instead of their usual places.
>
> Depending on exactly how we want this to behave, we may need a "only_trashed" query mode; presently with include_trash: true there is no way to distinguish items contained in a trashed project from regular not-trashed items. (Currently not implemented in this branch.)

I don't quite follow this. Isn't that what the is_trashed flag tells you? (If you had to use include_trash to get it, but it has is_trashed=false, then it's trash only by virtue of an ancestor being trash; if it has is_trashed=true, then it will still be trash even when its parent is untrashed.)

>> Data integrity: There is no sequence of API calls that could result in a collection being deleted (or otherwise made invisible to keep-balance) less than blobSignatureTTL seconds after the last time a client read or updated that collection's manifest.
>
> Currently the collection object is not aware if it is part of a trashed collection, so if you request a trashed collection record with include_trash it may have signatures that are later than the parent group's delete_at.

Hm, I think we should fix all data integrity bugs before merging. :) Perhaps we can skip manifest-signing when accessing via include_trash?

#11 - 08/31/2017 08:03 PM - Peter Amstutz

Tom Clegg wrote:

> Following through users is the desired behavior when there's a group→user "manage" link. Was this happening for read/write, too?

Yes, because the initial set of users were marked "follow" which is the "follow permissions through groups" propagation behavior. The initial set of users are now marked as "starting nodes" instead. A "can_manage" link to a user still sets the "follow" flag.

>> Currently doesn't work because the generic update operation fails before getting to the point where it changes the owner_uuid.
>
> Is there a failing test for this to make sure we don't forget?

There's a passing test that asserts the 404 with a note we might change it later. We can make it a failing test with a skip if you'd prefer.

>> Depending on exactly how we want this to behave, we may need a "only_trashed" query mode; presently with include_trash: true there is no way to distinguish items contained in a trashed project from regular not-trashed items. (Currently not implemented in this branch.)
>
> I don't quite follow this. Isn't that what the is_trashed flag tells you? (If you had to use include_trash to get it, but it has is_trashed=false, then it's trash only by virtue of an ancestor being trash; if it has is_trashed=true, then it will still be trash even when its parent is untrashed.)

The current "Trash" page on workbench does an index query on collections with is_trashed = true (or something to that effect). Users can search the trashed list.

If we want to be able to search collections which are trashed indirectly by being in a trashed project, that won't work, because include_trash lists both trashed and untrashed objects and is_trashed isn't set on those objects for filtering.

On the other hand, if we just want to browse trashed collections using the "project contents" API, that will work fine.

>> Currently the collection object is not aware if it is part of a trashed collection, so if you request a trashed collection record with include_trash it may have signatures that are later than the parent group's delete_at.

> Hm, I think we should fix all data integrity bugs before merging. :) Perhaps we can skip manifest-signing when accessing via include_trash?

That seems like a good policy.

#12 - 08/31/2017 08:40 PM - Tom Clegg

Peter Amstutz wrote:

> There's a passing test that asserts the 404 with a note we might change it later. We can make it a failing test with a skip if you'd prefer.

Ah, I was thinking of a real test that fails until the thing works. But it sounds like you're planning to do that part in a separate branch. I suppose it's not very visible anyway until the Workbench part happens, so, sure.

> If we want to be able to search collections which are trashed indirectly by being in a trashed project, that won't work, because include_trash lists both trashed and untrashed objects and is_trashed isn't set on those objects for filtering.

Sorry, still not seeing the problem...

"there is an obvious indication that the project itself is trashed" → "get project" with include_trash=false, now you know whether the project is trashed (whether due to an ancestor or not), and therefore whether its contents are implicitly trashed

Then, you can

- get all stuff: include_trash=true
- get stuff for non-trash tab: include_trash=true, filters=[is_trashed=false]
- get stuff for trash tab: include_trash=true, filters=[is_trashed=true]

And the way you render the "non-trash" stuff just depends on what you learned in step 1 about the project itself being trash.

#13 - 09/06/2017 02:03 PM - Tom Clegg

I don't think the permission bug existed, and the fix should be reverted. Here's my reasoning:

- If we were following through users this way, "can see the other user in sharing tab" would be equivalent to "can read all of the other user's stuff", which would surely have been failing tests and alarming users. The test for this specific bug is "PermissionTest#test_users_with_bidirectional_read_permission_in_group_can_see_each_other,_but_cannot_see_each_other's_private_articles" and it fails if you create the bug by changing pv.val=3 to pv.val>0 in the "select from links" section.
- You changed the condition from "follow" to "follow or startnode" and set startnode=true for the user rows anyway, so you're still following the same edges.
- I reverted the fix (follow=true for the perm_edges taken from the users table, and remove the startnode column) and all tests still pass.

It looks like the only thing the "split follow into follow+startnode" change accomplished was to omit the user->user row from the permission_view rows, which required you to add that as a special case in the Ruby logic. But one of the advantages of the permission view is that it keeps the Ruby logic to a minimum -- so I think we should un-special-case this.

```
@@ -41,24 +41,22 @@ perm_edges (tail_uuid, head_uuid, val, follow, trashed) AS (
               CASE WHEN trash_at IS NOT NULL and trash_at < clock_timestamp() THEN 1 ELSE 0 END
               FROM groups
        ),
-perm (val, follow, user_uuid, target_uuid, trashed, startnode) AS (
+perm (val, follow, user_uuid, target_uuid, trashed) AS (
      SELECT 3::smallint             AS val,
-             false                  AS follow,
+             true                   AS follow,
             users.uuid::varchar(32) AS user_uuid,
             users.uuid::varchar(32) AS target_uuid,
-             0::smallint            AS trashed,
-             true                   AS startnode
+             0::smallint            AS trashed
             FROM users
      UNION
      SELECT LEAST(perm.val, edges.val)::smallint  AS val,
             edges.follow                          AS follow,
             perm.user_uuid::varchar(32)           AS user_uuid,
             edges.head_uuid::varchar(32)          AS target_uuid,
-            GREATEST(perm.trashed, edges.trashed)::smallint AS trashed,
-            false                                 AS startnode
+            GREATEST(perm.trashed, edges.trashed)::smallint AS trashed
             FROM perm
             INNER JOIN perm_edges edges
-            ON (perm.startnode or perm.follow) AND edges.tail_uuid = perm.target_uuid
+            ON perm.follow AND edges.tail_uuid = perm.target_uuid
 )
 SELECT user_uuid,
        target_uuid,
```

The new Ruby logic does uncacheable things with dynamic SQL queries and looks like the sort of slow/hairy beast we just replaced by implementing permission_view. The new "trashed" column in permission_view seems to tell us whether the object is implicitly-or-implicitly-trashed, which is how I imagined this would work. But if we have that, couldn't we just load the "trashed" flag into our permission cache alongside the read/write/manage flags, and pass along the "include trash" flag to groups_i_can() so it can omit trashed items when appropriate, and go back to the simple version of readable_by?

#14 - 09/06/2017 05:01 PM - Peter Amstutz

master branch:

```
PermissionPerfTest#test_groups_index = Time spent creating records:
  0.240000   0.050000   0.290000 (  0.856856)
created 6561
Time spent getting group index:
  6.720000   0.090000   6.810000 (  8.134053)
  6.100000   0.050000   6.150000 (  7.410151)
  6.810000   0.060000   6.870000 (  8.165069)
  7.200000   0.070000   7.270000 (  8.590978)
  8.040000   0.120000   8.160000 (  9.466958)
43.40 s = .
```

Using 12032 using subqueries:

```
PermissionPerfTest#test_groups_index = Time spent creating records:
  0.330000   0.020000   0.350000 (  0.958686)
created 6561
Time spent getting group index:
  6.020000   0.120000   6.140000 ( 30.231668)
  6.050000   0.070000   6.120000 ( 31.016187)
  7.330000   0.070000   7.400000 ( 32.833348)
  6.680000   0.070000   6.750000 ( 32.175968)
  7.440000   0.090000   7.530000 ( 33.304580)
161.40 s = .
```

So that's not going to work.

#15 - 09/06/2017 06:37 PM - Peter Amstutz

Tom Clegg wrote:

> I don't think the permission bug existed, and the fix should be reverted. Here's my reasoning:
>
> - If we were following through users this way, "can see the other user in sharing tab" would be equivalent to "can read all of the other user's stuff", which would surely have been failing tests and alarming users. The test for this specific bug is "PermissionTest#test_users_with_bidirectional_read_permission_in_group_can_see_each_other,_but_cannot_see_each_other's_private_articles" and it fails if you create the bug by changing pv.val=3 to pv.val>0 in the "select from links" section.
> - You changed the condition from "follow" to "follow or startnode" and set startnode=true for the user rows anyway, so you're still following the same edges.
> - I reverted the fix (follow=true for the perm_edges taken from the users table, and remove the startnode column) and all tests still pass.
>
> It looks like the only thing the "split follow into follow+startnode" change accomplished was to omit the user->user row from the permission_view rows, which required you to add that as a special case in the Ruby logic. But one of the advantages of the permission view is that it keeps the Ruby logic to a minimum -- so I think we should un-special-case this.

I took it out and it works for me as well. An earlier version of the permission logic relied on it, but it looks like the current version doesn't.

> The new Ruby logic does uncacheable things with dynamic SQL queries and looks like the sort of slow/hairy beast we just replaced by implementing permission_view. The new "trashed" column in permission_view seems to tell us whether the object is

> implicitly-or-implicitly-trashed, which is how I imagined this would work. But if we have that, couldn't we just load the "trashed" flag into our permission cache alongside the read/write/manage flags, and pass along the "include trash" flag to groups_i_can() so it can omit trashed items when appropriate, and go back to the simple version of readable_by?

It's not as simple as just omitting certain groups from groups_i_can(), because "readable" and "not trashed" are distinct features. The is an expression like:

(A path exists with read permission between the user and target **or** target owner) **and** (the target is not trashed **and** the owner is not trashed)

In particular, there's the case where a child group is trashed but the parent group is not trashed. The naive approach "check if uuid or owner_uuid is in this list of owner uuids" will result in the trashed child considered readable due to the fact that the parent is readable and not trashed.

Similarly, there can be permission links to individual objects. Because this bypasses the normal ownership chain, we can't infer that the the uuid or owner_uuid missing from the list of owner uuids means the object is trashed.

I think the extra complexity is coming from trying to handle inferred permissions for both Groups and non-groups in the same query. I think we actually want the following separate cases:

- Query on groups: can assume the group is listed directly as a target permission_view, can filter directly on "perm_level" and "trashed" field
- Query on non-groups: need to check for permission on both uuid and owner_uuid, but can assume that the value of "trashed" for permission rows will be the same for both "uuid" and "owner_uuid", also allowing us to filter directly on "perm_level" and "trashed" field

If we split these cases the queries become much simpler and then in my benchmark, this query clause has equal or better performance than the version in master that uses the owner_uuid literal set:

```
EXISTS(SELECT 1 FROM permission_view WHERE user_uuid IN (:user_uuids) AND trashed = 0 AND #{perm_check} AND ta
rget_uuid = #{sql_table}.uuid)
```

#16 - 09/06/2017 07:01 PM - Tom Clegg

OK, it sounds like you're saying the "trashed" column in permission_view is not actually what we want it to be, despite the encouraging-looking max(trashed) stuff, because the recursive query doesn't handle the case where the explicitly-trashed parent is not encountered anywhere on the permission path from a given user to a descendant project.

But before we give up... can we fix it? My sense is that we won't get good performance from anything that requires another query on permission_view every time someone calls readable_by(). (Are you benchmarking on a su92l-like database?)

I think it would help to state exactly what "trashed" does mean as implemented so far. It seems like we should be able to make it mean "user A can read this project/group/user by following an ownership chain from an explicitly trashed group." (Is that what it means now, or is it something different?)

It would be most convenient if it meant "any user can ..." (not just the user whose permission this row is establishing) but that might be too finicky to bother with.

Even with the former definition, it seems like we could get what we need in the permission cache by doing a self-join on permission_view instead of a straight query: "select a.*, max(b.trashed) from permission_view a where ... join permission_view b on a.owner_uuid=b.owner_uuid".

#17 - 09/06/2017 07:03 PM - Peter Amstutz

Special casing groups, and doing a simpler subquery on permission_view:

```
PermissionPerfTest#test_groups_index = Time spent creating records:
  0.310000   0.040000   0.350000 (  0.959721)
created 6561
Time spent getting group index:
  6.400000   0.100000   6.500000 (  6.766211)
  5.910000   0.060000   5.970000 (  6.159582)
  7.760000   0.090000   7.850000 (  8.053858)
  6.070000   0.080000   6.150000 (  6.338377)
  7.020000   0.090000   7.110000 (  7.329790)

Group Load (52.1ms)
Group Load (40.7ms)
Group Load (41.1ms)
Group Load (38.8ms)
Group Load (40.0ms)
```

Compare to master:

```
PermissionPerfTest#test_groups_index = Time spent creating records:
  0.280000   0.030000   0.310000 (  0.864029)
created 6561
Time spent getting group index:
  6.810000   0.130000   6.940000 (  8.244489)
  6.320000   0.050000   6.370000 (  7.635715)
  8.080000   0.060000   8.140000 (  9.400862)
  7.420000   0.080000   7.500000 (  8.753632)
  8.160000   0.060000   8.220000 (  9.563759)
45.26 s = .
```

```
Group Load (570.8ms)
Group Load (568.1ms)
Group Load (563.4ms)
Group Load (568.9ms)
```

#18 - 09/06/2017 07:16 PM - Tom Clegg

Hm. My note-16 doesn't account for items that are accessible through direct permission links (i.e., where there would be no row in permission_view to join). So... yeah, in order to get the filtering in one query, we need either a separate memory cache of the set of trashed/untrashed projects, or we need to do a join every time. I think previous lessons have been that the memory cache is significantly faster.

#19 - 09/06/2017 08:14 PM - Peter Amstutz

So the "startnode" workaround is needed to distinguish between "iteratively explore edges (startnode=true)" and "propagate permissions across edges (follow=true)". This is indicated by the "target_owner_uuid" column. If this column is NULL, then the permissions don't propagate, otherwise they do.

User A → (can read) → Group B → (can read) → User C → (can read and follow=true) → Object D

User A → can read Object D

Vs.

User A → (can read) → Group B → (can read) → User C → (can read and follow=false) → Object D

User A → (does not propagate permission to read because target_owner_uuid is NULL) Object D

Due to the design of the query, we can't avoid having the user→user rows, but we can prevent them being used to propagate group-like permissions via the "follow" and "target_owner_uuid" column, which is what the "startnode" flag does.

#20 - 09/07/2017 03:32 AM - Peter Amstutz

Using materialized permission & better indexes:

```
time arv collection list --limit 1000 --count=none
real    0m1.373s
user    0m0.244s
sys    0m0.044s
```

```
time arv collection list --limit 1000 --count=exact
real    0m8.188s
user    0m0.264s
sys    0m0.028s
```

master + index improvements, but using groups_i_can() and gigantic queries:

```
time arv collection list --limit 1000 --count=none
real    0m2.691s
user    0m0.256s
sys    0m0.036s
```

```
 time arv collection list --limit 1000 --count=exact
real    0m24.344s
user    0m0.240s
sys    0m0.060s
```

#21 - 09/07/2017 03:58 AM - Peter Amstutz

12032-project-trash @ 893b9849ee1f3d407778a666702dfa300e0cebbc

Make permission_view a materialized view, with indexes. It is refreshed as needed based on the existing permission cache invalidation logic.

Observed that our default list behavior doesn't use an index (!!!) so added appropriate ones to the migration.

Ran some test queries against copy of su92l database. To retrieve 1000 items, query time is 2-4 times better compared to master.

Using a materialized view is a similar strategy to the existing permission cache, but has some significant advantages:

- Can be use explain / analyze to understand what its doing
- Can add indexes based on what explain / analyze tells us

- Avoid constructing gigantic query strings, which make SQL statement logging unreadable (I assume we suppress these in production, because otherwise the volume would be overwhelming)
- Avoid doing redundant work like loading/deserializing permission cache object on every request
- Much more likely to scale when we need to support 10x more groups/projects/collections than we have now

#22 - 09/07/2017 01:47 PM - Peter Amstutz

	Database query time for 1000 items	For 1 item
12032	Collection Load (491.5ms)	Collection Load (6.0ms)
master	Collection Load (2041.7ms)	Collection Load (69.9ms)

#23 - 09/07/2017 02:27 PM - Tom Clegg

Postgresql docs say "only one REFRESH at a time may run against any one materialized view" but we have no locking here. So... what happens when we try to do concurrent refreshes -- do some of the API requests crash? I can't tell whether the docs are trying to say "concurrent refresh attempts fail" or "concurrent refresh attempts get queued".

Do all of the necessary features work on Postgresql 9.2 (centos7), 9.3 (ubuntu1404)?

This doesn't seem right in read_permissions:

```
((links.head_uuid)::text ~~ 'su92l-j7d0g-%'::text)) AS follow
```

structure.sql is no longer runnable because it tries to create permission_view before the users table exists. I suppose the easy way to fix this is renaming permission_view and read_permissions so they lexically follow users.

In the "exclude rows that are explicitly trashed" code, the "else" case seems to cover objects that have no owner_uuid column, but are listed in permission_view as being trashed. Is that possible?

Code maintainability

- move the permission view sql to lib/*.sql and get rid of the old unused one
- add a comment to the modified_at,uuid index part of the migration just mentioning that it's not really related to the permission view stuff
- what's the procedure for modifying the views in a migration while, say, crunch-dispatch is running? (this wasn't an issue with the temporary view, but now we have to make sure we're not painting ourselves into a corner)

#24 - 09/07/2017 03:01 PM - Tom Clegg

There's a bit of a refresh race bug here too: the view can become stale between "refresh materialized view" and "set everything-is-up-to-date flag in Rails cache".

#25 - 09/07/2017 03:07 PM - Peter Amstutz

Tom Clegg wrote:

> Peter Amstutz wrote:
>
> > There's a passing test that asserts the 404 with a note we might change it later. We can make it a failing test with a skip if you'd prefer.
>
> Ah, I was thinking of a real test that fails until the thing works. But it sounds like you're planning to do that part in a separate branch. I suppose it's not very visible anyway until the Workbench part happens, so, sure.

So, there was a bug in the test (duh). Changing the owner actually does work as specified.

> > If we want to be able to search collections which are trashed indirectly by being in a trashed project, that won't work, because include_trash lists both trashed and untrashed objects and is_trashed isn't set on those objects for filtering.
>
> Sorry, still not seeing the problem...
>
> "there is an obvious indication that the project itself is trashed" → "get project" with include_trash=false, now you know whether the project is trashed (whether due to an ancestor or not), and therefore whether its contents are implicitly trashed
>
> Then, you can
>
> - get all stuff: include_trash=true
> - get stuff for non-trash tab: include_trash=true, filters=[is_trashed=false]
> - get stuff for trash tab: include_trash=true, filters=[is_trashed=true]
>
> And the way you render the "non-trash" stuff just depends on what you learned in step 1 about the project itself being trash.

I think we might starting from different assumptions here.

The current trash tab shows all trashed collections globally. It would be easy to extend that to also show the root trashed projects.

The complicated part is showing the *contents* (nested subprojects/collections) of trashed groups.

If we just want to browse an individual trashed group it is sufficient to provide include_trash=true.

On the other hand, if we want to search across all trashed items (for example, for a collection in a trashed project with a particular name or PDH) we can't easily distinguish between trashed and untrashed collections with a flat query. Include_trash gives us *everything* so we'd need to query with include_trash=true and issue a second query with include_trash=false in order to infer which collections were trash (in order to display them on the "trash can" page).

#26 - 09/07/2017 05:38 PM - Peter Amstutz

Tom Clegg wrote:

> Postgresql docs say "only one REFRESH at a time may run against any one materialized view" but we have no locking here. So... what happens when we try to do concurrent refreshes -- do some of the API requests crash? I can't tell whether the docs are trying to say "concurrent refresh attempts fail" or "concurrent refresh attempts get queued".

I'm pretty sure it is locked with an exclusive lock, which means both readers and writers block. Postgres 9.4 has a "concurrent" refresh mode where readers can see the old view contents until the new one is ready, but that would probably have the same drawbacks as previous experiments with asynchronous update (e.g. read-after-write inconsistency). We could also do something similar to the current cache strategy and create a separate materialized view for each user, but that seems inelegant.

> Do all of the necessary features work on Postgresql 9.2 (centos7), 9.3 (ubuntu1404)?

It requires Postgres 9.3. Nico reports that the clusters we maintain are all on at least Postgres 9.4. Other features that are on the roadmap (e.g. JSONB) also require Postgres 9.4.

> This doesn't seem right in read_permissions: [...]

So I think the story here is that I've been testing with a dump of the database from su92l, and these must be holdovers from experiments in https://dev.arvados.org/issues/10816 ! They don't belong and have been dropped.

> structure.sql is no longer runnable because it tries to create permission_view before the users table exists. I suppose the easy way to fix this is renaming permission_view and read_permissions so they lexically follow users.

I had reverted some hunks from the staged file before committing it because the diff showed it the user and groups definitions being moved around in the file. Now I know why it did that. Fixed.

> In the "exclude rows that are explicitly trashed" code, the "else" case seems to cover objects that have no owner_uuid column, but are listed in permission_view as being trashed. Is that possible?

I think there may be only one table that has uuid without owner_uuid, which is api_client_authorizations. That doesn't make sense to check trash, so I can remove that clause.

> Code maintainability
>
> - move the permission view sql to lib/*.sql and get rid of the old unused one

Do you mean "remove lib/create_permission_view.sql" ? The text of the permission view SQL was copied into the migration. Unless you think the migration should read from lib/create_permission_view.sql, but that seems likely to cause problems if a future commit changes the contents of create_permission_view.sql.

> - add a comment to the modified_at,uuid index part of the migration just mentioning that it's not really related to the permission view stuff

They are related, though, from looking at query plans the indexes on the other tables seem are essential to constructing an efficient join with permission_view.

> - what's the procedure for modifying the views in a migration while, say, crunch-dispatch is running? (this wasn't an issue with the temporary view, but now we have to make sure we're not painting ourselves into a corner)

It can be updated with "ALTER MATERIALIZED VIEW" (which presumably acquires an exclusive lock while its being altered).   Otherwise I believe the same procedures apply as updating a table. Provided we don't change the format of the rows that are returned (or only add new columns to the end) it seems like it should be possible to tweak the logic on the fly, should that be necessary.

#27 - 09/07/2017 07:01 PM - Tom Clegg

Peter Amstutz wrote:

> Tom Clegg wrote:
>
> > Postgresql docs say "only one REFRESH at a time may run against any one materialized view" but we have no locking here. So... what happens when we try to do concurrent refreshes -- do some of the API requests crash? I can't tell whether the docs are trying to say "concurrent refresh attempts fail" or "concurrent refresh attempts get queued".
>
> I'm pretty sure it is locked with an exclusive lock, which means both readers and writers block. Postgres 9.4 has a "concurrent" refresh mode where readers can see the old view contents until the new one is ready, but that would probably have the same drawbacks as previous experiments with asynchronous update (e.g. read-after-write inconsistency). We could also do something similar to the current cache strategy and create a separate materialized view for each user, but that seems inelegant.

Yeah, I don't think view-per-user would solve anything -- many concurrent requests often affect the same user(s). "Concurrent" sounds promising for the future but we still support ≤9.3 so it's an easy decision for now.

If refreshing the materialized view follows the usual rules of consistency/locking/transactions, can we do it in after_* hooks, drop all the Rails cache stuff entirely, and magically fix the bug in note-24...?

> > Do all of the necessary features work on Postgresql 9.2 (centos7), 9.3 (ubuntu1404)?
>
> It requires Postgres 9.3. Nico reports that the clusters we maintain are all on at least Postgres 9.4. Other features that are on the roadmap (e.g. JSONB) also require Postgres 9.4.

Our install docs claim centos7 is supported. It says to install the centos7-provided Postgresql. But doing that, and installing/updating to this branch, would fail or result in a non-working system, wouldn't it? I don't think "our own clusters are fine" or "we need 9.3 in the future anyway" affect anything. Our install/upgrade docs must work on centos7 -- either that, or we stop claiming to support centos7.

(This is probably why I didn't bother going down the materialized view road in #10816...)

> Do you mean "remove lib/create_permission_view.sql" ? The text of the permission view SQL was copied into the migration. Unless you think the migration should read from lib/create_permission_view.sql, but that seems likely to cause problems if a future commit changes the contents of create_permission_view.sql.

Indeed, I was only thinking of the normal upgrade path, but if you upgrade from A to D, then roll back to C, the migrations from B are considered done, so it's good if migration B used SQL from B, not D. So it's good as is.

On that node, I suppose you should revert create_permission_view.sql to its previous state. During an upgrade, dispatchers (and possibly other scripts) still run the old code, and could load/install the temporary view at any moment.

Also, the new view can't be called permission_view, because an old dispatcher version would see it when testing for presence of its temporary view. (I guess you have to rename it for structure.sql's sake anyway.)

> > - add a comment to the modified_at,uuid index part of the migration just mentioning that it's not really related to the permission view stuff
>
> They are related, though, from looking at query plans the indexes on the other tables seem are essential to constructing an efficient join with permission_view.

Yeah, that would be good as a comment.

> > - what's the procedure for modifying the views in a migration while, say, crunch-dispatch is running? (this wasn't an issue with the temporary view, but now we have to make sure we're not painting ourselves into a corner)
>
> It can be updated with "ALTER MATERIALIZED VIEW" (which presumably acquires an exclusive lock while its being altered). Otherwise I believe the same procedures apply as updating a table. Provided we don't change the format of the rows that are returned (or only add new columns to the end) it seems like it should be possible to tweak the logic on the fly, should that be necessary.

I just mean sooner or later we'll need to change the format again, kind of like we're doing right now. I guess we could add a new view with a versioned name, if it comes to that.

#28 - 09/07/2017 08:19 PM - Peter Amstutz

Tom Clegg wrote:

> If refreshing the materialized view follows the usual rules of consistency/locking/transactions, can we do it in after_* hooks, drop all the Rails cache stuff entirely, and magically fix the bug in note-24...?

Do you mean running refresh view proactively when something changes instead of clearing the "fresh" flag and refreshing it the next time it is needed? That should be fine, although might be a drag on batch creation of users / groups.

> Our install docs claim centos7 is supported. It says to install the centos7-provided Postgresql. But doing that, and installing/updating to this branch, would fail or result in a non-working system, wouldn't it? I don't think "our own clusters are fine" or "we need 9.3 in the future anyway" affect anything. Our install/upgrade docs must work on centos7 -- either that, or we stop claiming to support centos7.

You're right, we'll need to update the generic install docs to mention installing a Postgres backport package.

I also talked to Tom and Nico, they will check in with major users to see if there are concerns about requiring Postgres 9.3+.

> On that node, I suppose you should revert create_permission_view.sql to its previous state. During an upgrade, dispatchers (and possibly other scripts) still run the old code, and could load/install the temporary view at any moment.

Sure.

> Also, the new view can't be called permission_view, because an old dispatcher version would see it when testing for presence of its temporary view. (I guess you have to rename it for structure.sql's sake anyway.)

Do you prefer "materialized_permission_view" or "permission_view_v2"

> > > - add a comment to the modified_at,uuid index part of the migration just mentioning that it's not really related to the permission view stuff
> >
> > They are related, though, from looking at query plans the indexes on the other tables seem are essential to constructing an efficient join with permission_view.
>
> Yeah, that would be good as a comment.

Added a more detailed comment in the migration.

By the way, have you tried running the services/api tests on this branch? I'm seeing failures on jenkins I can't reproduce.

#29 - 09/07/2017 08:59 PM - Tom Clegg

I only have one of the failures Jenkins had.

```
  1) Failure:
Arvados::V1::RepositoriesControllerTest#test_get_all_permissions_does_not_give_any_access_to_user_without_perm
ission [/home/tom/src/arvados/services/api/test/functional/arvados/v1/repositories_controller_test.rb:79]:
project_viewer should only have permissions on public repos.
Expected: ["arvados"]
  Actual: []
```

That's at d03fcecb39335ac364e3f6f11cdfdc668fbda559 with

```
Checking dependencies:
virtualenv: 15.1.0
ruby: ruby 2.3.4p301 (2017-03-30 revision 58214) [x86_64-linux]
go: go version go1.8.3 linux/amd64
gcc: gcc (Debian 6.3.0-18) 6.3.0 20170516
fuse.h: /usr/include/fuse/fuse.h
gnutls.h: /usr/include/gnutls/gnutls.h
Python2 pyconfig.h: /usr/include/x86_64-linux-gnu/python2.7/pyconfig.h
Python3 pyconfig.h: /usr/include/x86_64-linux-gnu/python3.5m/pyconfig.h
nginx: nginx version: nginx/1.10.3
perl: This is perl 5, version 24, subversion 1 (v5.24.1) built for x86_64-linux-gnu-thread-multi
perl ExtUtils::MakeMaker: 7.1002
perl JSON: 2.90
perl LWP: 6.15
perl Net::SSL: 2.88
gitolite: /usr/bin/gitolite
/usr/bin/npm
WORKSPACE=/home/tom/src/arvados
```

#30 - 09/08/2017 03:47 PM - Peter Amstutz

Remaining TODOs:

- Resolve why some tests fail on jenkins
- ~~Rename materialized view so it doesn't conflict with existing temporary view~~
- ~~Restore temporary view code to master version to be available during migrations (?)~~
- ~~Update docs to specify Postgres 9.3 & provide instructions to install Postgres 9.5 from software collections on centos7~~

#31 - 09/08/2017 05:27 PM - Tom Clegg

Also

- Add entry to Upgrading to master: upgrade to Postgres ≥9.3, with link to docs for migrating db to the new (postgres)cluster.
- Rename views so structure.sql still works after "rake db:migrate" re-generates it with lexical ordering.

#32 - 09/08/2017 06:37 PM - Peter Amstutz

Tom Clegg wrote:

> Also
>
> - Add entry to Upgrading to master: upgrade to Postgres ≥9.3, with link to docs for migrating db to the new (postgres)cluster.

Done, will need to be touched up post-merge.

> - Rename views so structure.sql still works after "rake db:migrate" re-generates it with lexical ordering.

I'm not sure exactly what you mean? The structure.sql that is committed in the branch is the one I currently get from by postgres (unedited), and the view is now called "materialized_permission_view" to avoid conflicting with the previous "permission_view".

Does it work for you?

#33 - 09/08/2017 06:44 PM - Peter Amstutz

Argh workbench tests are failing on jenkins (but API server's own tests pass). Might be failing to start up API server for workbench integration tests. Will need to investigate.

#34 - 09/08/2017 07:02 PM - Tom Clegg

Peter Amstutz wrote:

> I'm not sure exactly what you mean? The structure.sql that is committed in the branch is the one I currently get from by postgres (unedited), and the view is now called "materialized_permission_view" to avoid conflicting with the previous "permission_view".
>
> Does it work for you?

Nearly. "RAILS_ENV=test bundle exec rake db:migrate" still changes the order of things in structure.sql. The resulting rearrangement does work now, though.

The migration file itself has quoting problems, though. If I run it (drop database, install a fresh one from master, upgrade, rake db:migrate), I get:

```
rake aborted!
SyntaxError: /home/tom/src/arvados/services/api/db/migrate/20170906224040_materialized_permission_view.rb:22: 
syntax error, unexpected tIDENTIFIER, expecting ')'
...means permission should "follow through" 
...                              ^
/home/tom/src/arvados/services/api/db/migrate/20170906224040_materialized_permission_view.rb:30: syntax error,
 unexpected tIDENTIFIER, expecting keyword_end
... in the working set and "follow" is true
...                              ^
/home/tom/src/arvados/services/api/db/migrate/20170906224040_materialized_permission_view.rb:85: syntax error,
 unexpected ')', expecting keyword_end
```

#35 - 09/11/2017 12:15 PM - Peter Amstutz

jenkins run @ https://ci.curoverse.com/job/developer-run-tests/441/

#36 - 09/11/2017 08:59 PM - Peter Amstutz

Another try, I think this will be the one: https://ci.curoverse.com/job/developer-run-tests/448/

#37 - 09/13/2017 03:10 PM - Tom Clegg

Why do we need to explicitly invalidate the permission cache in test teardown, instead of relying on transactions like we do at runtime?

Why is it necessary to refresh the view on every server/worker startup? Couldn't we seed it during the migration, and then let the post-commit hooks keep it up to date? It seems to me that if it's possible for the permission view to be out of date, we have to fix that -- we can't rely on correcting it at the next server startup event. (Surely this doesn't cause the same crashing bug (#11917) that running Rails.cache.clear at server startup caused, but it still seems like the wrong place to do an equivalent thing.)

Migrating my dev cluster to 9.6 made Rails stop trying to edit-war the changes in b5a56c9e3820b738459238d18dc3e1ffc726a5a6. However, it seems like master + migrations ≠ db/structure.sql. If I run db:setup with this branch, then down-and-up migrate the last migration to force Rails to rewrite structure.sql, it agrees with git. But if I checkout master, run db:setup, checkout 12032, and run db:migrate, I get something different. See 12032-sync-structure-sql @ 5230dcc75bf9fe8addc321b79c828cc370eee07b. Do you get different results?

In notes 13 and 15 it seemed like we agreed that "startnode" didn't fix a bug and merely required adding back special cases to the Ruby code for "user permission on self" -- then in note-19 you said startnode is needed after all. But my reasoning and experiment from note-13 still hold. You changed the follow condition to "follow or startnode", and on every row where you changed follow to false, you also set startnode to true; therefore every row that followed before still follows. All tests pass if you revert. If my reasoning is wrong and "startnode" is actually doing something, can you please a test that fails without it?

#38 - 09/13/2017 05:53 PM - Peter Amstutz

Tom Clegg wrote:

> Why do we need to explicitly invalidate the permission cache in test teardown, instead of relying on transactions like we do at runtime?

It seems that rolling back the transaction rolls does not roll back refreshes of the view (i.e. tests fail mysteriously if we don't do this.)

> Why is it necessary to refresh the view on every server/worker startup? Couldn't we seed it during the migration, and then let the post-commit hooks keep it up to date? It seems to me that if it's possible for the permission view to be out of date, we have to fix that -- we can't rely on correcting it at the next server startup event. (Surely this doesn't cause the same crashing bug (#11917) that running Rails.cache.clear at server startup caused, but it still seems like the wrong place to do an equivalent thing.)

I didn't think of doing the refresh in the migration. Done. I guess we can assume that a newly initialized database doesn't have any users or groups, so the permission graph will be empty as well.

> Migrating my dev cluster to 9.6 made Rails stop trying to edit-war the changes in b5a56c9e3820b738459238d18dc3e1ffc726a5a6. However, it seems like master + migrations ≠ db/structure.sql. If I run db:setup with this branch, then down-and-up migrate the last migration to force Rails to rewrite structure.sql, it agrees with git. But if I checkout master, run db:setup, checkout 12032, and run db:migrate, I get something different. See 12032-sync-structure-sql @ 5230dcc75bf9fe8addc321b79c828cc370eee07b. Do you get different results?

I followed the same procedure and got the same result, as you. I committed an update structure.sql which should match up now.

> In notes 13 and 15 it seemed like we agreed that "startnode" didn't fix a bug and merely required adding back special cases to the Ruby code for "user permission on self" -- then in note-19 you said startnode is needed after all. But my reasoning and experiment from note-13 still hold. You changed the follow condition to "follow or startnode", and on every row where you changed follow to false, you also set startnode to true; therefore every row that followed before still follows. All tests pass if you revert. If my reasoning is wrong and "startnode" is actually doing something, can you please a test that fails without it?

The permission checking query has gone through a bunch of iterations, some versions of which were confused with the "follow" flag true on the user→user permissions. However you are right, the current version (which applies different policies to queries of groups vs everything else) doesn't need the startnode workaround, so I removed it (again).

Rebased, ran tests:

https://ci.curoverse.com/job/developer-run-tests/452/

#39 - 09/13/2017 06:52 PM - Tom Clegg

Peter Amstutz wrote:

> Tom Clegg wrote:
>
> > Why do we need to explicitly invalidate the permission cache in test teardown, instead of relying on transactions like we do at runtime?
>
> It seems that rolling back the transaction rolls does not roll back refreshes of the view (i.e. tests fail mysteriously if we don't do this.)

Hm, I was afraid of that.

If the materialized view isn't protected by transactions like everything else, how can we convince ourselves that it will behave predictably (i.e., have the correct content) after races between permission-modifying API calls, and after failed transactions?

> I didn't think of doing the refresh in the migration. Done. I guess we can assume that a newly initialized database doesn't have any users or groups, so the permission graph will be empty as well.

Yes, "rake db:seed" should invoke the appropriate hook when adding the initial superuser etc., right?

> I followed the same procedure and got the same result, as you. I committed an update structure.sql which should match up now.

Phew. Thanks.

> The permission checking query has gone through a bunch of iterations, some versions of which were confused with the "follow" flag true on the user→user permissions. However you are right, the current version (which applies different policies to queries of groups vs everything else) doesn't need the startnode workaround, so I removed it (again).

I didn't verify, but I think that means these special cases aren't needed any more:

```
    else
      # Can read object (evidently a group or user) whose UUID is listed
      # explicitly in user_uuids.
      sql_conds.push "#{sql_table}.uuid IN (:user_uuids)"
```

```
        sql_conds.push "#{sql_table}.owner_uuid IN (:user_uuids)"
```

```
-       all_perms[user_uuid] ||= {}
+       all_perms[user_uuid] ||= {user_uuid => {:read => true, :write => true, :manage => true}}
```

...and the Ruby code is looking much more understandable now, which is great. Thanks.

#40 - 09/13/2017 08:27 PM - Peter Amstutz

Tom Clegg wrote:

> If the materialized view isn't protected by transactions like everything else, how can we convince ourselves that it will behave predictably (i.e., have the correct content) after races between permission-modifying API calls, and after failed transactions?

I had added them in because I was getting test failures that seemed like they were due to stale permissions. However, I just took out all of the calls to User.invalidate_permissions_cache in the tests and tests are passing, so I guess transactions actually work just as expected?

> > I didn't think of doing the refresh in the migration. Done. I guess we can assume that a newly initialized database doesn't have any users or groups, so the permission graph will be empty as well.
>
> Yes, "rake db:seed" should invoke the appropriate hook when adding the initial superuser etc., right?

Yes, the after_create hooks for users and groups refresh the view.

> > The permission checking query has gone through a bunch of iterations, some versions of which were confused with the "follow" flag true on the user→user permissions. However you are right, the current version (which applies different policies to queries of groups vs everything else) doesn't need the startnode workaround, so I removed it (again).
>
> I didn't verify, but I think that means these special cases aren't needed any more:
> [...]
> ...and the Ruby code is looking much more understandable now, which is great. Thanks.

As suggested, I removed all that stuff, and tests still seem to be passing. So that's nice. Latest jenkins run:

https://ci.curoverse.com/job/developer-run-tests/454/

#41 - 09/14/2017 02:35 PM - Tom Clegg

This seems important, so I tried using the postgres console to find out how races are handled.

Where "check" means

```
select greatest(trashed) from materialized_permission_view where user_uuid = 'zzzzz-tpzed-xurymjxw79nv3jz' and
 target_uuid = 'zzzzz-j7d0g-mhtfesvgmkolpyf';
```

...and the experiment involves changing the owner of a group such that it's no longer readable by the "active user" test fixture, which means greatest(trashed) should change from 0 (1 row) to nothing (0 rows)...

It seems that readers will wait for an uncommitted refresh to commit before using either the old or new version, which is good:

pg client 1	pg client 2
begin;	
update groups set owner_uuid = 'zzzzz-tpzed-000000000000000' where uuid = 'zzzzz-j7d0g-mhtfesvgmkolpyf';	
	check → 0 (1 row)
refresh materialized view materialized_permission_view;	
	check → ... (blocking)
rollback;	
	... → 0
begin;	
update groups set owner_uuid = 'zzzzz-tpzed-000000000000000' where uuid = 'zzzzz-j7d0g-mhtfesvgmkolpyf';	
	check → 0 (1 row)
refresh materialized view materialized_permission_view;	
	check → ... (blocking)
commit;	
	... → (0 rows)

However, if there are two concurrent updates, one of them is lost:

pg client 1	pg client 2	
begin;		
	begin;	
update groups set owner_uuid = 'zzzzz-tpzed-000000000000000' where uuid = 'zzzzz-j7d0g-mhtfesvgmkolpyf';		
	update groups set owner_uuid = 'zzzzz-tpzed-000000000000000' where uuid = 'zzzzz-j7d0g-zhxawtyetzwc5f0';	
refresh materialized view materialized_permission_view;		
	refresh materialized view materialized_permission_view; → ... (blocks)	
commit;		
	... → refresh OK	
	commit;	
check → 0 (1 row)		(bug!)
	check → 0 (1 row)	(bug!)
refresh materialized view materialized_permission_view; check → (0 rows)		(refreshi

Did I make a mistake here, or does this mean the current implementation is broken? And if the latter, how should we fix it? Hopefully it's something simple like adding "for update" to the selects in the view definition...

#42 - 09/14/2017 03:55 PM - Peter Amstutz

Thanks for looking at this.

Unfortunately, "FOR SHARE" and "FOR UPDATE" can't be used with "UNION" and "WITH RECURSIVE".

I think we need to go back to a lazy update strategy. This is the only way to guarantee that the permission view matches the other table contents in the transaction. However I need to give some thought about what is the right protocol for managing a "dirty" flag.

#43 - 09/14/2017 04:10 PM - Peter Amstutz

pg client1	pg client 2

dirty = 1	dirty = 1
refresh	
dirty = 0	
update blah	
dirty = 1	refresh
commit	dirty = 0
	commit

dirty = ?

pg client1	pg client 2	pg client 3	notes
begin	begin		
dirty is (ts1, 1)	dirty is (ts1, 1)		
refresh			
update dirty = (ts2, 0) if dirty = (ts1, 1)			
update groups			
update dirty = (ts2, 1)			within a transaction, now() returns the same value
	refresh		within transaction, refreshed view will be consistent with snapshot at ts1
	update dirty = (ts3, 0) if dirty = (ts1, 1)		pg2 blocks until transaction pg1 commits, no update
commit			
		begin	
		dirty is (ts2, 1)	
		refresh	
		update dirty = (ts4, 0) if dirty = (ts2, 1)	pg3 blocks until transaction pg2 commits, does update
	commit		
		commit	

dirty is (ts4, 0)

#44 - 09/14/2017 04:51 PM - Tom Clegg

The problem in note-41 seems to be that the race-losing "refresh materialized view" has already decided to use the stale source data by the time it starts blocking on the race-winning transaction.

So perhaps we just need a 1-row table "permission_updates", where we do "update permission_updates set updates=updates+1" before "refresh materialized view"? That should force the race-losing transaction to wait *before* doing the recursive query, thus ensuring that in any given sequence of commits, the last commit saves a permission set that incorporates all changes from all previous commits.

I'm wary of any solution that involves timestamps...

#45 - 09/14/2017 05:29 PM - Peter Amstutz

Tom Clegg wrote:

> The problem in note-41 seems to be that the race-losing "refresh materialized view" has already decided to use the stale source data by the time it starts blocking on the race-winning transaction.
>
> So perhaps we just need a 1-row table "permission_updates", where we do "update permission_updates set updates=updates+1" before "refresh materialized view"? That should force the race-losing transaction to wait *before* doing the recursive query, thus ensuring that in any given sequence of commits, the last commit saves a permission set that incorporates all changes from all previous commits.

I'm not sure if this works. The 2nd race-losing transaction still may not be able to see the outcome of the 1st transaction if the 1st transaction commits after the 2nd transaction begins. (Need to investigate).

> I'm wary of any solution that involves timestamps...

It doesn't have to be a timestamp, it could use a transaction id, or just a counter, as the purpose is just to assert that before clearing the dirty flag, the row has not been touched since the beginning of the transaction.

The nice thing about doing the refresh on-demand at the start of the transaction is that it ensures that the permission view is consistent with the current transaction.

#46 - 09/14/2017 06:28 PM - Tom Clegg

Instead of "update ... set x=x+1", we should just do "lock table permission_update_lock" before refreshing the permission view.

#47 - 09/14/2017 07:47 PM - Peter Amstutz

- Target version changed from 2017-09-13 Sprint to 2017-09-27 Sprint

#48 - 09/14/2017 07:55 PM - Tom Clegg

I'd say it's worth adding a comment in the transaction block assuring the reader that "lock table" really is necessary because the materialized view doesn't/can't use "select for update". We don't have a test case for this, and even a well informed developer could erroneously conclude it's superfluous.

LGTM @ 1a97222f80188c878cd1a57aa8c6620748b1db16

#49 - 09/27/2017 03:18 PM - Peter Amstutz

- Status changed from In Progress to Resolved

#50 - 09/27/2017 04:46 PM - Ward Vandewege

- Subject changed from Allow projects to be deleted (ie placed in the trash can) to [API] Allow projects to be deleted (ie placed in the trash can)

- Status changed from Resolved to In Progress

#51 - 09/27/2017 06:34 PM - Ward Vandewege

- Status changed from In Progress to Resolved