Grouper - Loader

These topics are discussed in the "Grouper Loader" training series.

Grouper Loader

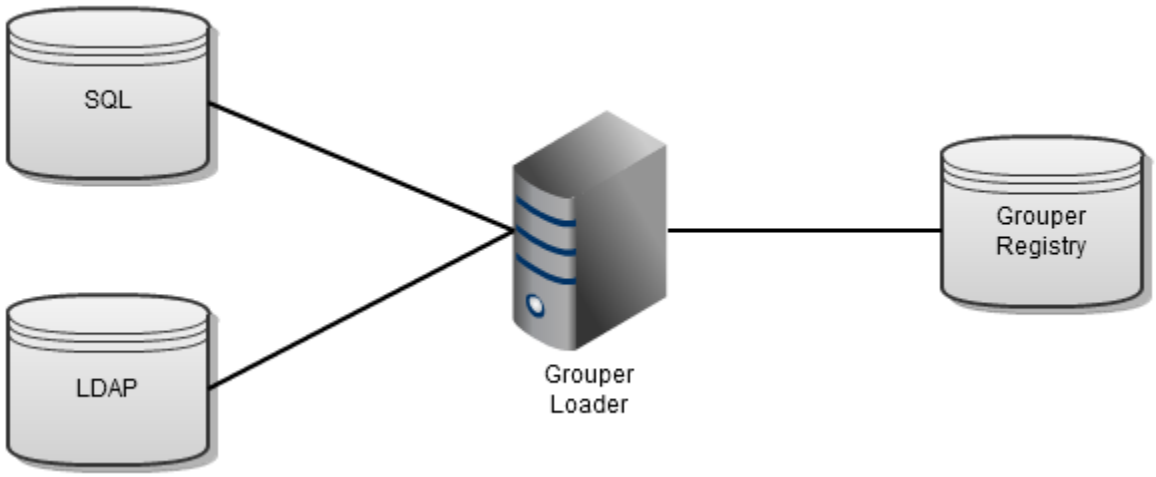
The Grouper Loader allows you to automatically manage Grouper memberships based on a data source. See architectural diagram.

To find out if Grouper Loader jobs are executing successfully, use Grouper Diagnostics.

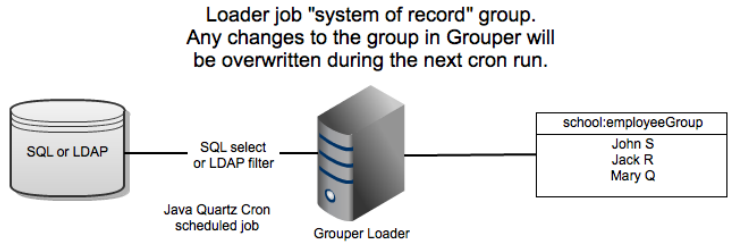
Here is an example of Grouper Loader being used to automatically manage Grouper memberships based on a data source.

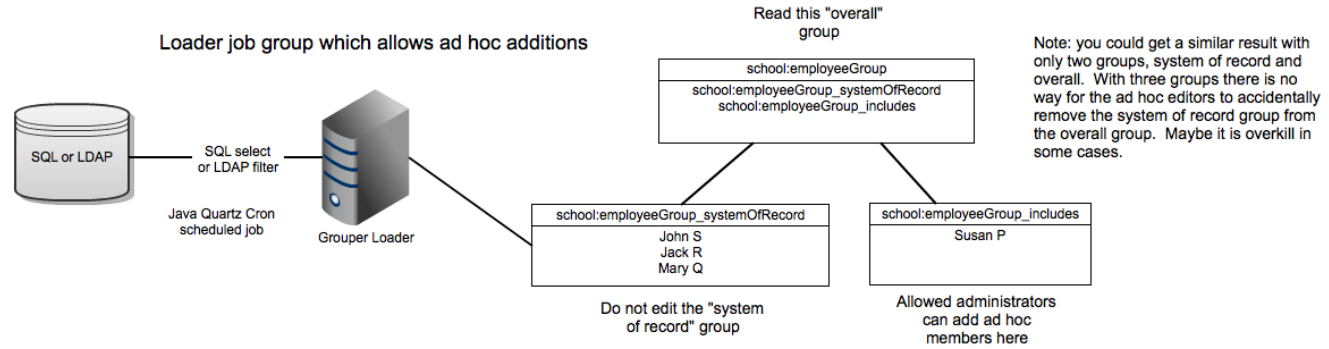
You should run the grouper loader all the time for any Grouper deployment. It also runs background tasks even if you aren't automatically loading groups.

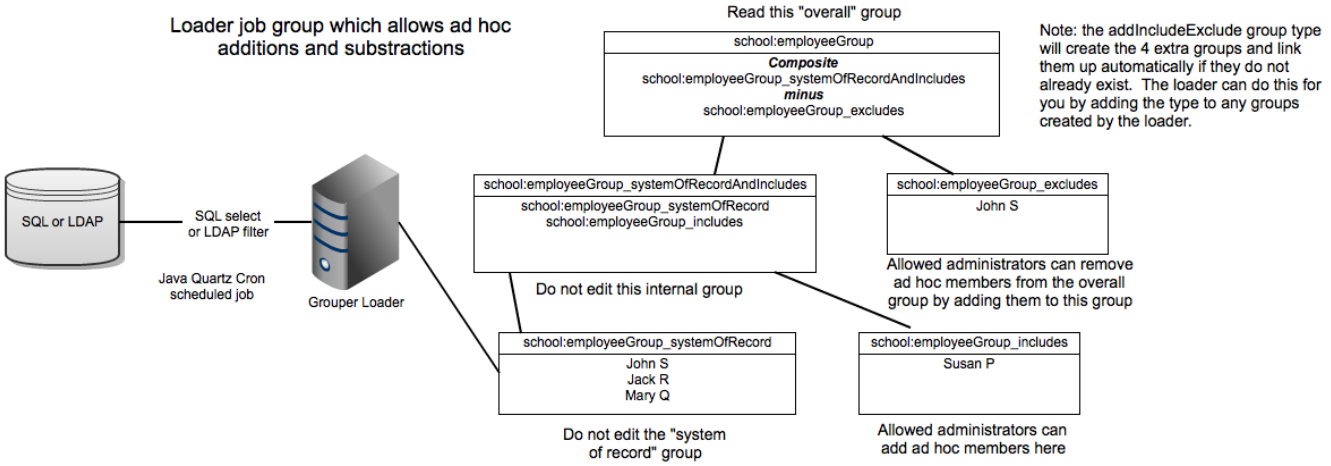
Penn is using it in production to load membership for groups, and for groups of groups (in Penn's case, org lists).



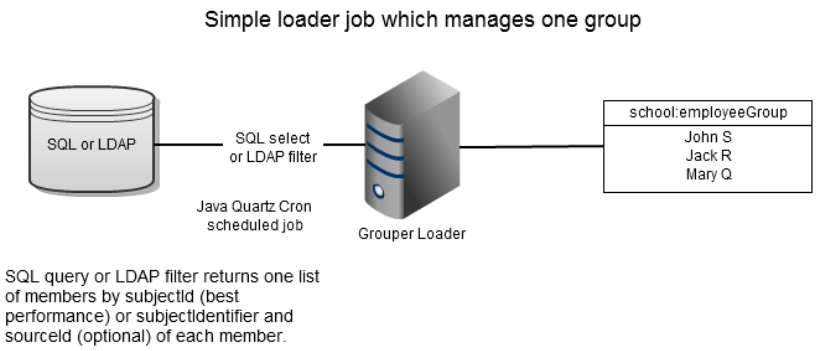
Ad hoc groups with the Grouper Loader

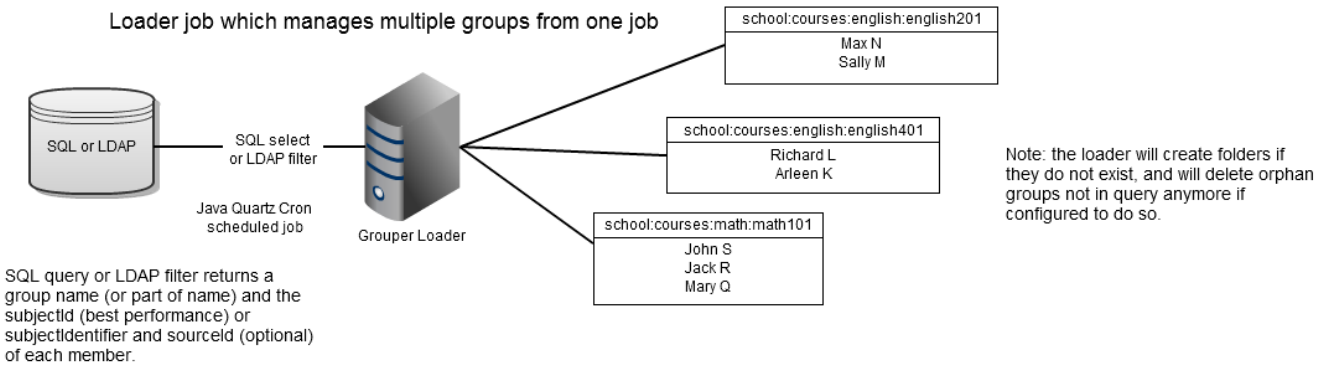






Loader manages simple group or a list of groups in one job





One-time setup in your Grouper database

To make a dynamic (loadable) group, first you need the correct metadata in Grouper. The easiest way is to set the grouper-loader.properties key loader.autoadd.typesAttributes to true. If you don't want to do that, then here is the setup in GSH:

```
subj=SubjectFinder.findById("GrouperSystem")
sess=GrouperSession.start(subj)
type=GroupType.createType(sess, "grouperLoader")
type.addAttribute(sess, "grouperLoaderType", false)
type.addAttribute(sess, "grouperLoaderDbName", false)
type.addAttribute(sess, "grouperLoaderScheduleType", false)
type.addAttribute(sess, "grouperLoaderQuery", false)
type.addAttribute(sess, "grouperLoaderQuartzCron", false)
type.addAttribute(sess, "grouperLoaderIntervalSeconds", false)
type.addAttribute(sess, "grouperLoaderPriority", false)
type.addAttribute(sess, "grouperLoaderAndGroups", false)
type.addAttribute(sess, "grouperLoaderGroupTypes", false)
type.addAttribute(sess, "grouperLoaderGroupsLike", false)
type.addAttribute(sess, "grouperLoaderGroupQuery", false)
```

Note that a loadable group has the type "grouperLoader", and there are some attributes that you can set about the group:

- **grouperLoaderType**: there will be various types to choose from, currently only these are available :
  - SQL_SIMPLE is a group whose membership is fed from a query, and the whole query's results will be the groups results (not incremental).
  - SQL_GROUP_LIST requires a group_name column in query, so one query can control multiple group memberships. This will default to SQL_SIMPLE if no group_name before the FROM in the query. Else it will be SQL_GROUP_LIST.
- **grouperLoaderDbName**: if it is a sql based type, this is the name in the grouper-loader.properties of the db connection properties. If this is set to the reserve value "grouper" it will use the grouper db (in grouper.hibernate.properties). Here is a snippet from grouper-loader.properties where grouperLoaderDbName=warehouse :

```
# specify the db connection with user, pass, url, and driver class
# the string after "db." is the name of the connection, and it should not have
# spaces or other special chars in it
db.warehouse.user = mylogin
db.warehouse.pass = secret
db.warehouse.url = jdbc:mysql://localhost:3306/grouper
db.warehouse.driver = com.mysql.jdbc.Driver
```

- **grouperLoaderScheduleType**: Grouper-loader uses the quartz open source job scheduler, and currently supports two schedule types:
    1. CRON: This is a cron-like syntax that I think is quartz specific
    2. START_TO_START_INTERVAL: This is a repeated schedule that runs based on a delay from the start of one run to the start of another run

This defaults to CRON if there is a value for grouperLoaderQuartzCron (see below), else it defaults to START_TO_START_INTERVAL. *Note that a job will not start if a previous run has not finished*.

- **grouperLoaderQuery**: This is the query to run in the DB, which must have certain columns required or optional based on the grouperLoaderType. e.g. for SQL_SIMPLE, the SUBJECT_ID is required, and the SUBJECT_SOURCE_ID is optional. If your DB supports views, might not be a bad idea to link query up to a view so you can easily see what it will return and change it without affecting the group attribute. But will work with any select query. This is required. Note: in Grouper v2.1 you can try having SUBJECT_IDENTIFIER or SUBJECT_ID_OR_IDENTIFIER instead of SUBJECT_ID, though each is less efficient than the next since they require an extra subject lookup or multiple subject lookups per row. SQL_GROUP_LIST requires a group_name column in query,
- **grouperLoaderQuartzCron**: If a CRON schedule type, this is the cron setting string from the quartz product to run a job daily, hourly, weekly, etc: http://www.opensymphony.com/quartz/wikidocs/TutorialLesson6.html
- **grouperLoaderIntervalSeconds**: If a START_TO_START_INTERVAL schedule type, this is the number of seconds between the start of one run to the start of another run. This defaults to daily if not filled in. Note, for daily jobs, it is probably better to use cron so it won't fire up each time the loader is restarted.
- **grouperLoaderPriority**: Quartz has a fixed threadpool (max configured in the grouper-loader.properties), and when the max is reached, then jobs are prioritized by this integer. The higher the better, and the default if not set is 5.
- **grouperLoaderAndGroups**: If you want to restrict membership in the dynamic group based on other group(s), put the list of group names here comma-separated. The require groups means if you put a group names in there (e.g. school:community:employee) then it will "and" that group with the member list from the loader. So only members of the group from the loader query who are also employees will be in the resulting group
- **grouperLoaderGroupTypes**: whatever you put in the value should be comma separated GroupTypes which will be applied to the loaded groups. The reason this enhancement exists is so we can do a SQL_GROUP_LIST query and attach addIncludeExclude to the groups. Note, if you do this (or use some requireGroups), the group name in the loader query should end in the system of record suffix, which by default is _systemOfRecord.
    - To use the "addIncludeExclude" GroupType you need to set `grouperIncludeExclude.use = true` in grouper.properties

- **grouperLoaderGroupsLike**: this should be a sql like string (e.g. school:orgs:%org%_systemOfRecord), and the loader should be able to query group names to see which names are managed by this loader job. So if a group falls off the loader resultset (or is moved), this will help the loader remove the members from this group. Note, if the group is used anywhere as a member or composite member, it won't be removed. All include /exclude/requireGroups will be removed. Though the two groups, include and exclude, will not be removed if they have members. There is a grouper-loader.properties setting to note remove loader groups if empty and not used:

```
#if using a sql table, and specifying the name like string, then should the group (in addition to memberships)
# be removed if not used anywhere else?
loader.sqlTable.likeString.removeGroupIfNotUsed = true
```

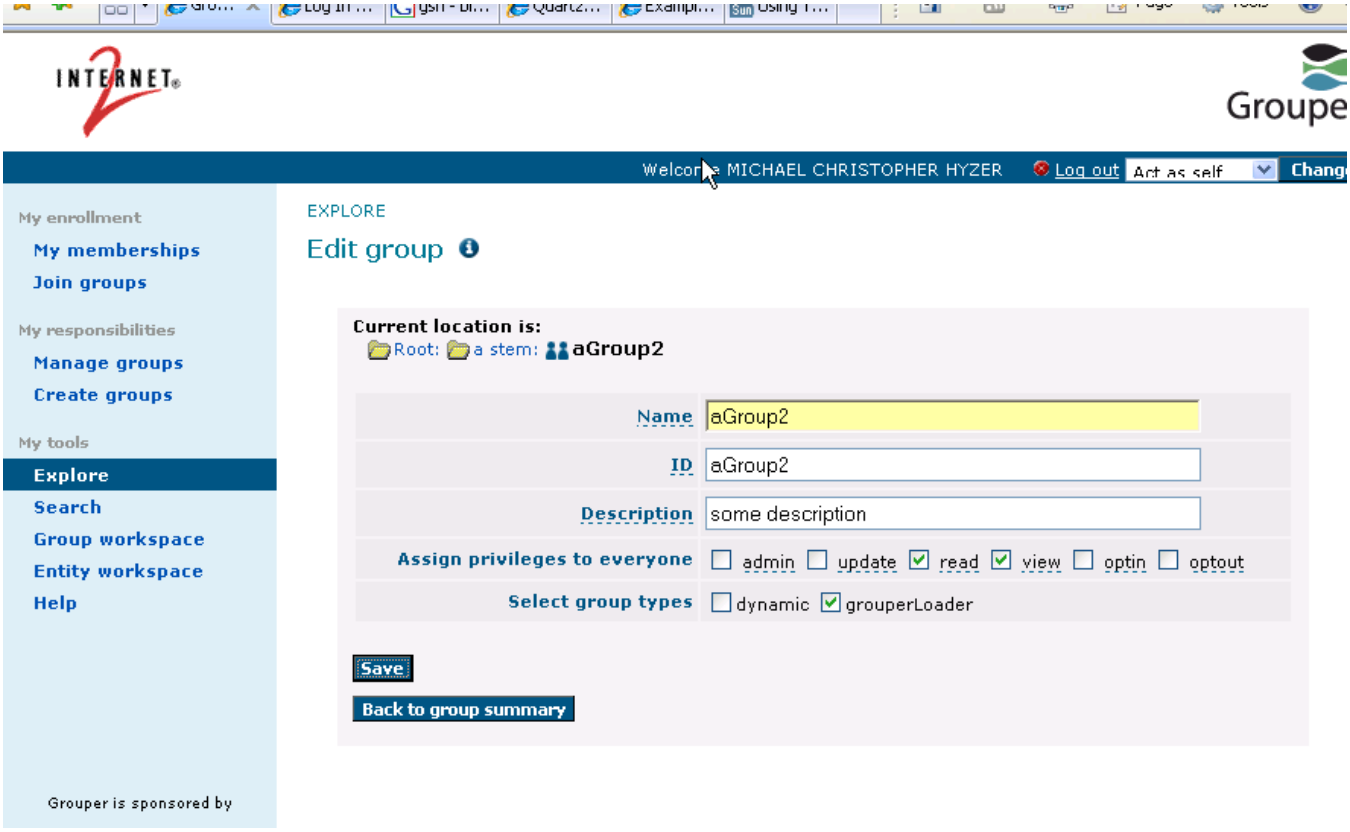
- **grouperLoaderGroupQuery**: query (optional) for SQL_GROUP_LIST which should return cols: group_name (full path of the group), group_display_name (optional, will be used for group display name), group_description (optional, will default to "<extension> auto-created by grouperLoader" unless loader.allowBlankGroupDescriptions is set to true in grouper-loader.properties). Note: the parent stem display names are only changed when creating them. This should return all groups in the membership list, and if not there, its ok, the extension will be used as display extension, and a generated description. Note: the display name is the display path, with the display extension of each parent stem. If there is a column named any of the following: readers, viewers, admins, updaters, optins, optouts, group_attr_readers, group_attr_updaters, then the data in the column (comma separated subjectId's or subjectIdentifers (which can include group names) will be assigned to that privilege list. Note, existing assignments to that privilege list will not be removed, so if you remove an item from the query, you will need to manually remove it from the groups. This is a way to have a loaderJob-wide list of readers or viewers.

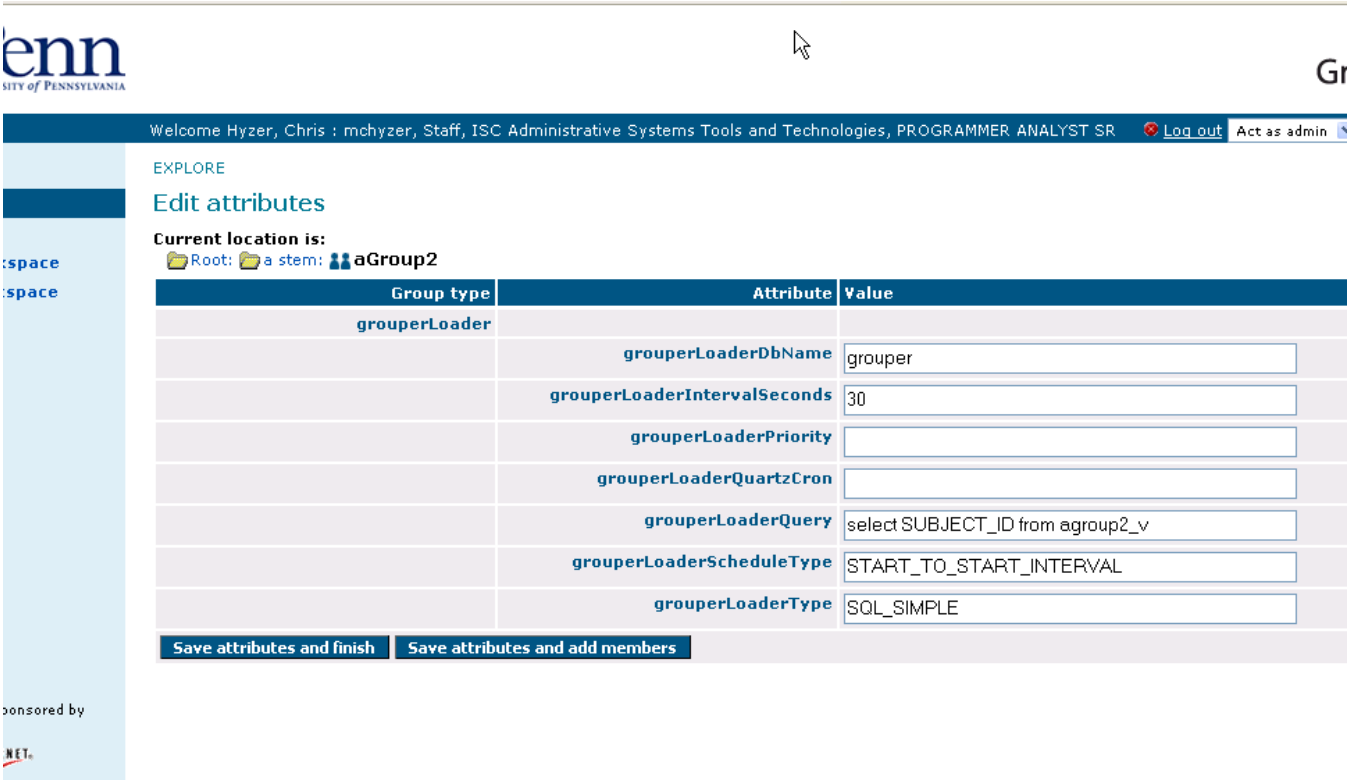
Configure a loadable group (obviously any number of dynamic loadable groups can exist at once)

With GSH, it would look like this:

```
group=getGroups("aStem:aGroup2")
groupAddType("aStem:aGroup2", "grouperLoader")
setGroupAttr("aStem:aGroup2", "grouperLoaderDbName", "grouper")
setGroupAttr("aStem:aGroup2", "grouperLoaderType", "SQL_SIMPLE")
setGroupAttr("aStem:aGroup2", "grouperLoaderScheduleType", "START_TO_START_INTERVAL")
setGroupAttr("aStem:aGroup2", "grouperLoaderQuery", "select SUBJECT_ID, SUBJECT_SOURCE_ID from agroup2_v")
setGroupAttr("aStem:aGroup2", "grouperLoaderIntervalSeconds", "30")
```

You can also use the UI, here are screenshots (obviously these need some work).





Run grouper daemon

The first time you run, it will probably fail, and give you DDL in the logs to run in your database (to add a couple of tables). Run the scripts and you should be all set.

Run with:

```
GROUPER_HOME/bin/gsh.sh -loader
```

This will kick off as a command line program that you will want to run as a service. This process will be always running, and the scheduler will schedule the jobs. You should monitor the process with a monitoring tool like nagios or whatever you use at your institution so that you know when it is not up.

You can also run a one-timer via gsh. This is useful to run once at the beginning, and not have to wait for the schedule. Or to troubleshoot e.g.

```
loaderGroup = GroupFinder.findByName(GrouperSession.startRootSession(), "school:orgs:orgGroup");
loaderRunOneJob(loaderGroup);
loaderRunOneJob("MAINTENANCE_cleanLogs");
```

Note that as of Grouper 2.3, if you create a new loader job, you can have it scheduled without restarting the daemon by using an option in the New UI. If you're an admin of the group, under "More actions", there's an option named "Schedule loader process". This option can also be used if you change the loader's schedule.

Groups as members

If you are using the loader to load hierarchies where groups are members, then you need to have a query which has the column SUBJECT_IDENTIFIER with the group system name, or SUBJECT_ID with the group uuid. Useful for org charts. See this example

Logging of jobs in DB

Each job (and subjob if the job manages multiple things) will have an entry in the grouper_loader_log table. This will show the following information. This can be used to tune performance problems, see which jobs have unresolvable subjects, verify that jobs are running, etc.

```
COLUMN_NAME                                DATA_TYPE

ID                                                 VARCHAR2
JOB_NAME                                      VARCHAR2
STATUS                                         VARCHAR2
STARTED_TIME                                TIMESTAMP(6)
ENDED_TIME                                         TIMESTAMP(6)
MILLIS                                             NUMBER
MILLIS_GET_DATA                               NUMBER
MILLIS_LOAD_DATA                              NUMBER
JOB_TYPE                                           VARCHAR2
JOB_SCHEDULE_TYPE                             VARCHAR2
JOB_DESCRIPTION                               VARCHAR2
JOB_MESSAGE                                        VARCHAR2
HOST                                              VARCHAR2
GROUP_UUID                                        VARCHAR2
JOB_SCHEDULE_QUARTZ_CRON                 VARCHAR2
JOB_SCHEDULE_INTERVAL_SECONDS        NUMBER
JOB_SCHEDULE_PRIORITY                    NUMBER
LAST_UPDATED                                   TIMESTAMP(6)
UNRESOLVABLE_SUBJECT_COUNT               NUMBER
INSERT_COUNT                                   NUMBER
UPDATE_COUNT                                  NUMBER
DELETE_COUNT                                   NUMBER
TOTAL_COUNT                                           NUMBER
PARENT_JOB_NAME                               VARCHAR2
PARENT_JOB_ID                                 VARCHAR2
```

You can also look at log4j debug log messages, and info log messages (less frequent). to see these, set log level in log4j.properties

Log debug info on loader to see progress etc
log4j.logger.edu.internet2.middleware.grouper.app.loader = INFO
-or-
log4j.logger.edu.internet2.middleware.grouper.app.loader = DEBUG

Misc

- You can set transaction level in the grouper-loader.properties. It defaults to not use transaction since for huge groups, you might have memory or db problems
- By default only wheel group members can edit the grouperLoader type or attributes. You can edit these settings in the grouper.properties in the type security part.

Example test proving that only certain members can edit loader attributes

grouper.properties:

```
security.types.grouperLoader.wheelOnly = false

security.types.grouperLoader.allowOnlyGroup = etc:someAdminGroup

wheel configured:

groups.wheel.use                     = true

# Set to the name of the group you want to treat as the wheel group.
# The members of this group will be treated as root-like users.
groups.wheel.group                   = etc:sysadmingroup
```

create the security group in gsh

```
grouperSession = GrouperSession.startRootSession();

someSysAdminGroup = new GroupSave(grouperSession).assignName("etc:someAdminGroup").assignGroupNameToEdit("etc:
someAdminGroup").save();
```

make sure subject not wheel

```
gsh 9% hasMember("etc:sysadmingroup", "test.subject.0");
false
```

add a group for that user to be an admin of (GSH)

```
someGroup = new GroupSave(grouperSession).assignName("a:b").assignGroupNameToEdit("a:b").
assignCreateParentStemsIfNotExist(true).save();

grantPriv("a:b", "test.subject.0", AccessPrivilege.ADMIN);
```

auto add attributes in grouper-loader.properties

```
# auto-add grouper loader types and attributes when grouper starts up if they are not there
loader.autoadd.typesAttributes = true
```

start the loader to init the attributes (command line)

```
gsh -loader
```

login to UI as test.subject.0, Try to add grouperLoader type to group, and get this error:

```
Error: This operation is not allowed: Not allowed to edit type: grouperLoader, adding type since the user
Subject id: test.subject.0, sourceId: jdbc is not in group: etc:someAdminGroup.
```

In the UI see that the type is not assigned. Check the DB if you don't believe:

```
SELECT * FROM grouper_groups_types_v WHERE group_name = 'a:b'
```

Verify the SQL, when I look in grouperSql.log, I see these SQLs:

```
update grouper_groups set hibernate_version_number=1, parent_stem='088956f34e064116b68b97198ea422f7',
creator_id='233cdc87a0654c03b37e59ea0bc7b52c', create_time=1273197358184,
modifier_id='9221f2ae35d44d23b7bdb469e9e96278', modify_time=1273198909003, name='a:b', display_name='a:b',
extension='b', display_extension='b', description=null, context_id='29523c8d2dac4ddd8294c40fadfd0f7f',
alternate_name=null, type_of_group='group' where id='6e70a1c3a2d246669244051e44439374' and
hibernate_version_number=0
2010/05/06 22:21:49:003, 0ms, statement: ByObjectStatic.java.saveOrUpdate() line 327, Hib3AuditEntryDAO.java.
saveOrUpdate() line 26, AuditEntry.java.saveOrUpdate() line 307, Group.java.callback() line 3900, Group.java.
store() line 3826, SaveGroupAction.java.grouperExecute() line 237, GrouperCapableAction.java.callback() line
217, Hib3TransactionDAO.java.callback() line 51, Hib3TransactionDAO.java.transactionCallback() line 41,
GrouperCapableAction.java.grouperTransactionExecute() line 214, GrouperCapableAction.java.execute() line 279,
LoginCheckFilter.java.callback() line 173, GrouperSession.java.callbackGrouperSession() line 644,
LoginCheckFilter.java.doFilter() line 168, ErrorFilter.java.doFilter() line 132, GrouperUiFilter.java.doFilter()
line 398
   insert into grouper_audit_entry (hibernate_version_number, act_as_member_id, audit_type_id, context_id,
created_on, description, env_name, grouper_engine, grouper_version, int01, int02, int03, int04, int05,
last_updated, logged_in_member_id, server_host, string01, string02, string03, string04, string05, string06,
string07, string08, duration_microseconds, query_count, user_ip_address, server_user_name, id) values (0, null,
'd087478a3a334c41a104a9a0b47e2b3e', '29523c8d2dac4ddd8294c40fadfd0f7f', 1273198909003, 'Updated group: a:b,
Fields changed: none', '', 'grouperUI', '1.5.3', null, null, null, null, null, 1273198909003,
'9221f2ae35d44d23b7bdb469e9e96278', 'mchyzer-PC', '6e70a1c3a2d246669244051e44439374', 'a:b',
'088956f34e064116b68b97198ea422f7', 'a:b', '', null, null, null, 4444, 1, '0:0:0:0:0:0:0:1', 'mchyzer',
'691681cbbb7243caa3e4852ced79b3bc')
2010/05/06 22:21:49:003, 0ms, statement: ByObject.java.save() line 197, Hib3GroupDAO.java.callback() line 119,
Hib3GroupDAO.java.addType() line 108, Group.java.callback() line 916, Group.java.addType() line 890, Group.java.
addType() line 860, SaveGroupAction.java.doTypes() line 349, SaveGroupAction.java.grouperExecute() line 240,
GrouperCapableAction.java.callback() line 217, Hib3TransactionDAO.java.callback() line 51, Hib3TransactionDAO.
java.transactionCallback() line 41, GrouperCapableAction.java.grouperTransactionExecute() line 214,
GrouperCapableAction.java.execute() line 279, LoginCheckFilter.java.callback() line 173, GrouperSession.java.
callbackGrouperSession() line 644, LoginCheckFilter.java.doFilter() line 168, ErrorFilter.java.doFilter() line
132, GrouperUiFilter.java.doFilter() line 398
   insert into grouper_groups_types (hibernate_version_number, group_uuid, type_uuid, context_id, id) values
(0, '6e70a1c3a2d246669244051e44439374', '57433a6dbdf14008a371ef18cd5c9c8d', 'b296ae3dd6b448d28c9ba2e643903087',
'402881822870817d01287091964b0002')
GrouperLoaderArchitectureDiagram.tiff2010/05/06 22:22:08:489, 0ms, statement: ByHqlStatic.java.uniqueResult()
line 297, Hib3GroupDAO.java.findByName() line 907, GroupFinder.java.findByName() line 225, GroupFinder.java.
findByName() line 198, GroupTypeSecurityHook.java.vetoIfNecessary() line 194, GroupTypeSecurityHook.java.
groupTypeTupleHelper() line 300, GroupTypeSecurityHook.java.groupTypeTuplePostInsert() line 289, GrouperUtil.
java.invokeMethod() line 3422, GrouperHooksUtils.java.executeHook() line 476, GrouperHooksUtils.java.
callHooksIfRegistered() line 276, GrouperHooksUtils.java.callHooksIfRegistered() line 215, GrouperHooksUtils.
java.callHooksIfRegistered() line 141, GroupTypeTuple.java.onPostSave() line 265, Hib3GroupDAO.java.callback()
line 119, Hib3GroupDAO.java.addType() line 108, Group.java.callback() line 916, Group.java.addType() line 890,
Group.java.addType() line 860, SaveGroupAction.java.doTypes() line 349, SaveGroupAction.java.grouperExecute()
line 240, GrouperCapableAction.java.callback() line 217, Hib3TransactionDAO.java.callback() line 51,
Hib3TransactionDAO.java.transactionCallback() line 41, GrouperCapableAction.java.grouperTransactionExecute()
line 214, GrouperCapableAction.java.execute() line 279, LoginCheckFilter.java.callback() line 173,
GrouperSession.java.callbackGrouperSession() line 644, LoginCheckFilter.java.doFilter() line 168, ErrorFilter.
java.doFilter() line 132, GrouperUiFilter.java.doFilter() line 398
```

The next commit or rollback in the file is a rollback (after it checks to see if the user is the member of the allowed group or wheel or GrouperSysadmin)

Failsafe to not remove too many members by mistake

You can configure the loader to not make changes if too many members are to be removed. The use case is if the source for the loader groups gets blanked out accidentally, it shouldn't remove everyone. However, if groups are supposed to drastically change, it means a user needs to manually change this flag, run the sync, and change it back.

```
# if the loader should check to see too many users were removed, if so, then error out and
# wait for manual intervention
loader.failsafe.use = false

# if a group has a size less than this (default 200), then make changes including blanking it out
loader.failsafe.minGroupSize = 200

# if a group with more members than the loader.failsafe.minGroupSize have more than this percent (default 30)
# removed, then log it as error, fail the job, and don't actually remove the members
# In order to run the job, an admin would need to change this param in the config,
# and run the job manually, then change this config back
loader.failsafe.maxPercentRemove = 30
```

Unschedule a database quartz job

Note, you can look in the table grouper_qz_triggers and grouper_qz_job_details for more info

```
select * from grouper_loader_log where lower(job_name) like '%abc%' order by started_time desc;
select * from GROUPER_QZ_TRIGGERS where lower(trigger_name) like '%abc%';
select * from GROUPER_QZ_CRON_TRIGGERS where lower(trigger_name) like '%abc%';
select * from GROUPER_QZ_FIRED_TRIGGERS where lower(trigger_name) like '%abc%';
select * from grouper_qz_job_details gqjd where lower(GQJD.JOB_NAME ) like '%abc%';
```

If you want to unschedule BLOCKED jobs (the code part is one line of GSH)

```
GrouperSession.startRootSession();
for (String triggerName : HibernateSession.bySqlStatic().listSelect(String.class, "select trigger_name from
GROUPER_QZ_TRIGGERS where trigger_state = 'BLOCKED'", null)) {GrouperLoader.schedulerFactory().getScheduler().
unscheduleJob(org.quartz.TriggerKey.triggerKey(triggerName));}
```

If you get errors about a database quartz job, you can unschedule with (or whatever the trigger name is):

```
select trigger_name from GROUPER_QZ_TRIGGERS where lower(trigger_name) like '%abc%';
gsh 2% GrouperLoader.schedulerFactory().getScheduler().unscheduleJob(org.quartz.TriggerKey.triggerKey
("triggerMessaging_MESSAGING_listener_messagingListener"))
```

Performance Settings

These are the default settings:

```
# if should use threads in the loader for add/remove member
# {valueType: "boolean", required: true}
loader.use.membershipThreads=true

# number of threads to use for each group job (not shared among jobs)
# {valueType: "integer", required: true}
loader.membershipThreadPoolSize=10

# if should use threads in the loader for each group in a group of groups
# {valueType: "boolean", required: true}
loader.use.groupThreads=true

# number of threads to use for each list of groups job (not shared among jobs)
# {valueType: "integer", required: true}
loader.groupThreadPoolSize=20
```

So for a simple job (SQL_SIMPLE, LDAP_SIMPLE), you can have up to 10 threads. However for a single groupList job (SQL_GROUP_LIST, LDAP_GROUP_LIST, LDAP_GROUPS_FROM_ATTRIBUTES), you can have up to 200 threads since each of the 20 group threads can have 10 membership threads.

Possible to do's

- add subject source to group attribute (or default at least) so it doesn't have to be a sql column if all are the same
- make specific blackout times (runtime via config file?)
- make jobs based on person trigger. If there is a query that says when people change, then update that person's memberships in the groups that are dynamic based on that person-change-query
- make full refresh jobs for the incremental jobs (e.g. weekly)
- make an RMI server so that interactions can happen at runtime (to see status, stop/start jobs, etc). Maybe this would happen from gsh
- try the name pattern after loader is done, and if the number of groups is less than the number of groups in this round of loader, set the job status to WARNING and add a descriptive message

See Also

Grouper Loader Class List Example from University of Pennsylvania