Coding new DDL

This document helps developers work with Grouper DDL. If you work with Grouper DDL, please keep this document up to date.

Make sure to:

- Use lower case for views
- Test on case sensitive mysql container

The first thing to know about is the grouper_ddl table. This has one entry for each ddl type. A ddl type means database objects with a certain prefix (e.g. the grouper one starts with grouper_, the default subject one starts with subject, and the organization management one starts with grouperorgs_). On startup, grouper will see if the version in the DB matches the version in the jar (an enum). If not, an error will be logged, and optionally grouper will try to auto-upgrade in 2.5 or print out a script to run if not auto-ddl.

Each version of grouper that changes DDL should have its own class. See GrouperDdl2_5.java for an example. The name of the class should correspond to the version that will be released with the new DDL. e.g. GrouperDdl2_5_38

Each object that is changed should be in its own method. Check to make sure if it has been done before. See if it should even run. There are a few cases

Object type	Check	Description	Example
New table	Check if going to current version or above	This is not that important since it is only called from current version	`GrouperDdl.V32.getVersion() <= ddlVersionBean.getBuildingToVersion();`
New column,view, comment, index, foreign key, etc	Check if going to current version or above	This is important since the method is called from two places, 1. when the table/object is created 2. when this version is upgraded	`GrouperDdl.V32.getVersion() <= ddlVersionBean.getBuildingToVersion();`

Changed view	This is complicated and requires multiple steps (example in 2.5)
	1. In previous version, if building to that version, create some view with same name.
	2. In current version, if not building from scratch, drop the view
	3. in current version, if building to current version of above, create the view
		If you dont create the view with temp name in previous version, then ddlutils wont detect that it needs to drop it	previous version:
			`GrouperDdl.V32.getVersion() > ddlVersionBean.getBuildingToVersion();`
			building from scratch:
			`ddlVersionBean.getBuildingFromVersion() <= 0`

Update statement	Check to see if needs to update	You can check by version number or see if you can find the table in the object model (isTableNew)	`Table groupTable = GrouperDdlUtils.ddlutilsFindTable(database, Group.TABLE_GROUPER_GROUPS, true);`
			`boolean enabledColumnIsNew = false;`
			`if (groupTable != null) { enabledColumnIsNew = null == GrouperDdlUtils.ddlutilsFindColumn (groupTable, Group.COLUMN_ENABLED, false); }`

Example of adding new column

Step	Summary	Description

1. Add the fields/getters	In the bean, add getters and setters and fields	`/**`
		` * when this group was removed from grouper`
		` */`
		`private Timestamp inGrouperEnd;`
		
		`/**`
		` * when this group was removed from grouper`
		` * @return`
		` */`
		`public Timestamp getInGrouperEnd() {`
		`  return inGrouperEnd;`
		`}`
		
		`/**`
		` * when this group was removed from grouper`
		` * @param inGrouperEnd`
		` */`
		`public void setInGrouperEnd(Timestamp inGrouperEnd) {`
		`  this.inGrouperEnd = inGrouperEnd;`
		`}`

2. Scan class for more work	If all bean properties are used elsewhere in class,
	add those too
		`clone() {`
		`  gcGrouperSyncGroup.id = this.id;`
		`  gcGrouperSyncGroup.inGrouperDb = this.inGrouperDb;`
		`  gcGrouperSyncGroup.inGrouperEnd = this.inGrouperEnd;`

3. Edit the hbm mapping file	Look at HBM and edit it accordingly	`  <property name="inGrouperEnd" column="in_grouper_end" type="` `timestamp"/>`

4. Add column if upgrade	If this is an upgrade, add the column to the table	`  if (!buildingToThisVersionAtLeast(ddlVersionBean)) {`
		`  return;`
		`  }`
		
		`  // if building from scratch its already got it`
		`  if(buildingFromScratch(ddlVersionBean)) {`
		`  return;`
		`  }`
		
		`  if (ddlVersionBean.didWeDoThis`
		`("v2_5_38_addGrouperSyncStartColumns", true)) {`
		`  return;`
		`  }`
		`  GrouperDdlUtils.ddlutilsFindOrCreateColumn`
		`(grouperSyncGroupTable, "in_grouper_end",`
		`  Types.TIMESTAMP, "1", false, false);`

5. Add in table if starting anew	If loading DDL from scratch, the column should
	be added when the table is created, not as an
	alter table. But only do this if building passed
	the version where it is added
		`  if (GrouperDdl2_5_38.buildingToThisVersionAtLeast`
		`(ddlVersionBean)) {`
		
		`  GrouperDdlUtils.ddlutilsFindOrCreateColumn`
		`(grouperSyncGroupTable, "in_grouper_end",`
		`  Types.TIMESTAMP, "10", false, false);`
		
		`  }`

6. Similarly add comments	Add comments to all objects for postgres/oracle	`GrouperDdlUtils.ddlutilsColumnComment(ddlVersionBean, "grouper_sync_group", "in_grouper_end", "when this was taken out of grouper");`
7. For each DB, install ddl utils	Droponly all objects, then do a -registry -check -useDdlUtils	This will generate the full generated DDL script
7a. Note, a unit test makes this a lot faster		

8. Merge changes to static dll	Merge changes from that full script to the conf/ddl /GrouperDdl_Grouper_install_<db>.sql
	
	For each DB
		Dont copy the whole script in, just the thing that changed
		`in_target_start TIMESTAMP,`
		`in_target_end TIMESTAMP,`
		`provisionable_start TIMESTAMP,`

8a. Add an upgrade script for each database	conf/ddl /GrouperDdl_Grouper_XX_upgradeTo_XX_<dbVendor>.sql	See others for example, and pay attention to bottom version numbers

9. Correct bottom version	At the bottom of the file is an insert statement,
	make sure the version is correct
	
	For each DB
		e.g. for hsql:
		`insert into grouper_ddl (id, object_name, db_version, last_updated, history) values ('c08d3e076fdb4c41acdafe5992e5dc4d', 'Grouper', 35, to_char (CURRENT_TIMESTAMP, 'YYYY/MM/DD HH24:mi:DD'), to_char(CURRENT_TIMESTAMP, 'YYYY/MM/DD HH24:mi:DD') || ': upgrade Grouper from V0 to V35, ');`
		`commit;`

10. Try an install using the static file	just start grouper shell (with auto ddl) For each DB	Make sure the new column / comment is there
11. Get the DB to the previous	-droponly. Then you can use the previous full SQL from git, make sure the grouper ddl version at bottom of file is correct (previous version) For each DB	Run that SQL in your DB client (e.g. dbeaver). See the column isnt there

12. Upgrade using ddl utils (dont run script)	-registry -check -useDdlUtils
	
	For each DB. Note, you can do this twice,
	once before the changes, once after, and compare
		The output of that script is a mess. We want the diff to be minimal. Use what you can and craft an upgrade script. Dont worry about the order of columns. If ddlutils says to add a column after another column, ignore that part
		`ALTER TABLE GROUPER_SYNC_GROUP`
		`ADD COLUMN in_grouper_end TIMESTAMP;`

15. Deep ddl check	-registry -deep should report a missing column For each DB	
14. Upgrade with static DDL	just start grouper shell (with auto ddl) For each DB	Make sure the new column / comment is there

16. Deep ddl check	-registry -deep should report the DB is up to date For each DB	