DDL Management

See DDL in Grouper v2.5+

This wiki is largely out of date and exists for development purposes

Using Grouper database structure DDL (database definition language)

To run the Grouper database DDL, e.g. if you have an error about mismatched versions.
Note, unique indexes and not constraints are added, so ignore foreign key errors when running in Oracle or database that care (for now).

1. configure DDL properties in grouper.properties:

```
# ddlutils db name will be set by default, you can override it here, it must be one of:
# axion, cloudscape, db2, db2v8, derby, firebird, hsqldb, interbase, maxdb, mckoi,
# mssql, mysql, mysql5, oracle, oracle10, oracle9, postgresql, sapdb, sybase, sybasease15,
#
#ddlutils.dbname.override = oracle10

# if you want to not create the subject tables (grouper examples for unit testing),
# then set this to true
ddlutils.exclude.subject.tables = false

# set the path where ddl scripts are generated (they will be uniquely named in this directory).
# if blank, the directory used will be the current directory
ddlutils.directory.for.scripts =

# during schema export, should it drop db objects then create?  Or just try to create on top of what is there.
default true
ddlutils.schemaexport.dropThenCreate = true

# during schema export, should it write and run the script, or just write the script without running it,
default false.  recommended to false especially in prod
ddlutils.schemaexport.writeAndRunScript = false

# during schema export, should it install grouper data also or not.  e.g. insert the root stem, default true
ddlutils.schemaexport.installGrouperData = true

# when grouper starts, should it shut down if not right version?
ddlutils.failIfNotRightVersion = true
```

2. run: ant schemaexport

3. Another option is to manually edit the grouper_ddl table to make the versions match (or set it back), or you can set the grouper.properties config param: ddlutils.failIfNotRightVersion to false

New features / requirements:

- The output script name is unique, it will not clobber existing scripts. No need to configure schemaexport dir in build config
- Schema export can init the registry (true by default). No need for a registry init ant task
- DDL manages foreign keys (no need for existing foreign key script or ant task)
- You dont have to add in the subject tables anymore (grouper.properties config option)
- All grouper tables and views must start with "grouper_"
- All subject tables and views must start with "subject"
- No tables/views not managed from grouper can start with "grouper_" or "subject"
- Can more easily unit test with junit (and it is unit tested)
- Supports managing DDL for extensions of apps (e.g. WS or UI). The extension or app needs to register the ddl app name (its a column in the DB) using the LifecycleHook.ddlInit() hook. If the hook is named "Foo" (not Class-like camel case), then make an enum: edu.internet2.middleware.grouper.ddl.FooDdl, and make it implement the interface: DdlVersionable. In fact, just copy the GrouperDdl or SubjectDll and modify it.
- ant schemaexport has always removed user defined indexes and foreign keys (on tables in question e.g. grouper_*, not other tables), and it still does. If you have an index on a grouper table, either make a ddl hook (see above), or manually add this after each schemaexport

Documentation

The existing ddl management is to describe the schema in hibernate mapping files and use the hibernate tools schemaupdate to change the DB. If there are db specific workarounds we put those in the upgrade instructions, and if there is something drastic, the user can export the db to xml and import into the new schema. The deficiencies are: hibernate mappings arent rich enough (lacking complex indexes, foreign keys, views, etc), exporting large db's is cumbersome, might be difficult to know what scripts to run if db is partway upgraded or skipping versions, hard to know the diff ddl, hard to know if out of sync, etc. Finally it is not possible to mix ddl and sql (e.g. to remove the ID cols if we wanted a script to do this)

The design for the grouper 1.4 ddl management could change to use ddlutils (jakarta package). The concept would be to keep a db table that would hold the current version of the grouper ddl (and any extensions), and on startup of grouper this table will be queried to see if out of date. If so, then the ddl needed to get up to date will be logged as error. This can also be an ant task. Users can manually tweak version numbers to affect the script if they like (e.g. set all to -1 to get a full ddl check, or move to the current version to remove logging of ddl updates if they dont want them). The java version is logged as info, but it will also be displayed when the versions to not match.

Here is an example of a versioned db space . Note that the schema is maintained in Java, and not XML. This is so we can run diagnostics to see if something is there (if column is there, dont add), and can intersperse DDL and SQL. We can make utility methods to make working with ddlutils easier. Here is an example of adding a column in a version:

```
   /** second version of grouper loader */
  V1 {
    /**
     * detect and add column for priority of job
     * @see edu.internet2.middleware.grouper.ddl.GrouperLoaderDdl#updateVersionFromPrevious(org.apache.ddlutils.
model.Database)
     */
    @Override
    public void updateVersionFromPrevious(Database database) {

      //see if the grouper_ext_loader_log table is there
      Table grouploaderLogTable = GrouperDdlUtils.ddlutilsFindOrCreateTable(database,"grouploader_log",
          "log table with a row for each grouper loader job run");

      GrouperDdlUtils.ddlutilsFindOrCreateColumn(grouploaderLogTable, "job_schedule_priority",
          "Priority of this job (5 is unprioritized, higher the better)", Types.INTEGER, null, false, false);
    }
  },
```

To update the db to the current ddl, the user will have to manually run the DDL against the DB. The ddlutils ddl is not 100% foolproof, so the user should inspect and make sure it is not dropping tables or whatever.

ddlutils generally does a good job, but sometimes not. If there are things that we want to improve upon (e.g. when adding a non-null column, something maybe we shouldnt do), if there is a default value, it will make a temp table, copy all data to it, drop and recreate the old table, copy data back). If there wasnt a default value, then the table is just dropped. Anyways, to make specific scripts (on specific db platforms), we can put scripts in a certain dir, named a certain thing, and that script will be found and used instead of ddlutils script. This is all or nothing when updating versions, so specific pieces cannot be replaced, only the whole version. So we can make as many versions as we want, and we probably should segment appropriately.

Here is an example generated script (moving to grouper v0, and grouperLoader v1):

```
 2008/04/30 04:59:24.486 [main] ERROR GrouperDdlUtils.bootstrap(193) - Database requires updates:

/* upgrade Grouper from V-1 to V0 */
CREATE TABLE grouper_ddl
(
    id VARCHAR2(128) NOT NULL,
    object_name VARCHAR2(128),
    db_version INTEGER,
    java_version INTEGER,
    PRIMARY KEY (id)
);



insert into grouper_ddl values ('87408b12-4ae7-483b-8454-12ba3193d772', 'Grouper', 0, 0);
commit;

/* upgrade GrouperLoader from V-1 to V0 */
CREATE TABLE grouploader_log
(
    id VARCHAR2(128) NOT NULL,
    job_name VARCHAR2(512),
    status VARCHAR2(10),
    started_time DATE,
    ended_time DATE,
    millis INTEGER,
    millis_get_data INTEGER,
    millis_load_data INTEGER,
    job_type VARCHAR2(128),
    job_schedule_type VARCHAR2(128),
    job_description VARCHAR2(4000),
    job_message VARCHAR2(4000),
    host VARCHAR2(128),
    group_uuid VARCHAR2(128),
    job_schedule_quartz_cron VARCHAR2(128),
    job_schedule_interval_seconds INTEGER,
    PRIMARY KEY (id)
);



insert into grouper_ddl values ('99aed2ea-acb3-4f18-8032-0d9152e512b8', 'GrouperLoader', 0, 1);
commit;

/* upgrade GrouperLoader from V0 to V1 */
ALTER TABLE grouploader_log
    ADD job_schedule_priority INTEGER;



update grouper_ddl set db_version = 1 where object_name = 'GrouperLoader';
commit;
```

Note, if we are going from scratch, it would also be possible to get the one-time script which isnt a create and a bunch of alters... not sure if this is ideal or not since custom scripts would be out the door, but we could make it an option...