Database Conversion v1.2.1 - v1.3.0

Database Conversion

To help the performance of database reads and to help the integrity of the data, we have added additional indexes and constraints on the Grouper database tables. There are two options available to convert a Grouper 1.2.1 database to a Grouper 1.3.0 database.

1. With the first option, you can execute SQL statements that will update your database to the new version. In order to do this, you will have to review the sample SQL statements provided and possibly adjust them depending on your database system. This is a recommended approach for production databases that have large amounts of data and cannot have a downtime.
2. With the second option, you can use Grouper's Export and Import features to export all of your data to an XML file and then import that into a Grouper 1.3.0 database.

Option 1: Database Conversion using SQL

Modifications have been made to the indexes and constraints on the Grouper tables for the 1.3.0 release of Grouper. Below you will find a description of each change and an example SQL statement for Oracle that you may execute to make the modification in your database.

Database Conversion v1.2.0 - v1.2.1

If you originally installed Grouper 1.2.0, you will need to first make some index modifications that were part of the Grouper 1.2.1 release. Note that even if you upgraded from 1.2.0 to 1.2.1 in the past, you likely did not update your indexes. Please see the following document for more information.

For your convenience, we have also included SQL scripts that contain all of these index and constraint modifications described below. Please review the file and pay close attention to any comments in the file. Also, if you're not using the Subject tables, you can remove them from the script.

1. Oracle SQL Script
2. MySQL SQL Script
3. Postgres SQL Script

Modifications to indexes:

1. Drop the index on grouper_attributes.field_name.
```
drop index attribute_field_idx;
```
2. Create a concatenated index on grouper_attributes using the columns field_name and value.
```
create index attribute_field_value_idx on grouper_attributes (field_name, value);
```
3. Drop the concatenated index on grouper_composites with columns left_factor and right_factor.
```
drop index composite_factor_idx;
```
4. Create an index on grouper_composites.left_factor.
```
create index composite_left_factor_idx on grouper_composites (left_factor);
```
5. Create an index on grouper_composites.right_factor.
```
create index composite_right_factor_idx on grouper_composites (right_factor);
```
6. Drop the index on grouper_memberships.depth.
```
drop index membership_depth_idx;
```
7. Create a concatenated index on grouper_memberships using the columns member_id and via_id.
```
create index membership_member_via_idx on grouper_memberships (member_id, via_id);
```
8. Create an index on grouper_sessions.member_id.
```
create index session_member_idx on grouper_sessions (member_id);
```

9. Drop the index on grouper_memberships.via_id.

```
drop index membership_via_idx;
```

10. Create an index on SubjectAttribute.value.

```
create index subjectattribute_value_idx on SubjectAttribute (value);
```

11. Drop the concatenated index on Subject with columns subjectId and subjectTypeId.

```
drop index subject_idx;
```

12. Drop the index on SubjectAttribute.subjectId.

```
drop index subjectattribute_id_idx;
```

13. Drop the index on SubjectAttribute.name.

```
drop index subjectattribute_key_idx;
```

Modifications to check constraints:

1. Prevent null values in grouper_attributes.group_id.

```
alter table grouper_attributes modify(group_id not null);
```

2. Prevent null values in grouper_memberships.depth.

```
alter table grouper_memberships modify(depth not null);
```

3. Prevent null values in grouper_memberships.membership_uuid.

```
alter table grouper_memberships modify(membership_uuid not null);
```

4. Prevent null values in grouper_memberships.create_time.

```
alter table grouper_memberships modify(create_time not null);
```

5. Prevent null values in the grouper_sessions.session_uuid.

```
alter table grouper_sessions modify(session_uuid not null);
```

Modifications to unique keys:

1. Drop the concatenated key on grouper_attributes with columns group_id, field_name, and value. **The constraint name was originally generated by the database so substitute constraint_name with the actual name.**

```
alter table grouper_attributes drop constraint constraint_name;
```

2. Create a concatenated key on grouper_attributes with columns group_id and field_name.

```
alter table grouper_attributes add (unique (group_id, field_name));
```

3. Create a key on grouper_composites.uuid.

```
alter table grouper_composites add (unique (uuid));
```

4. Create a key on group_fields.field_uuid.

```
alter table grouper_fields add (unique (field_uuid));
```

5. Create a concatenated key on grouper_fields with columns name and type.

```
alter table grouper_fields add (unique (name, type));
```

6. Create a key on grouper_groups.uuid.
```
alter table grouper_groups add (unique (uuid));
```
7. Create a key on grouper_members.member_uuid.
```
alter table grouper_members add (unique (member_uuid));
```
8. Create a key on grouper_memberships.membership_uuid.
```
alter table grouper_memberships add (unique (membership_uuid));
```
9. Create a key on grouper_sessions.session_uuid.
```
alter table grouper_sessions add (unique (session_uuid));
```
10. Create a key on grouper_stems.uuid.
```
alter table grouper_stems add (unique (uuid));
```
11. Create a key on grouper_types.type_uuid.
```
alter table grouper_types add (unique (type_uuid));
```

Modifications to foreign keys:

The sample SQL scripts provided above contain the foreign key modifications. However, if you are not using one of the sample SQL scripts, use ant and the build process for the Grouper 1.3.0 release to accomplish this task: **ant addforeignkeys**

Note that you may receive an error indicating that the Subject or the SubjectAttribute table does not exist. This is expected if you modified or deleted one of those tables. The Subject tables are not required for Grouper. They are available as example tables to store Subject data.

Option 2: Database Conversion using Grouper Export and Import

A database conversion using Grouper Export and Import can take a substantial amount of time, depending of course on how large it is. One on demo HSQLDB database containing 628 stems, 640 groups, and 14,447 memberships and non-default privilege assignments, on a laptop it took 1 minute 18 seconds to export, and 3 minutes 57 seconds to re-import, taking on average about 6ms to create each direct or indirect membership and privilege. Your mileage will vary, but it will take time to convert a substantial database.

Conversion steps

This process assumes that you have the Grouper API v1.3.0 RC1 or later installed and configured to work with your Grouper database, and that you have a shell open on the root of the Grouper API distribution directory.

1. **ant xml-export -Dcmd="GrouperSystem aFileName"**

The default xml-export properties are correct for doing a complete dump of the database. The filename can refer to a file in the current directory, or it can be a relative or absolute pathname.

2. **Create a new database container**

Do whatever you have to do to setup a fresh database with your RDBMS terchnology.

3. **Setup the SA account used by the Grouper API**

Establish the credential used by the Grouper API for database access in your new database.

4. **Review conf/grouper.hibernate.properties**

Ensure that properties declaring how the Grouper API will connect to your database are correctly declared in the conf/grouper.hibernate.properties file.

4. **ant schemaexport**

5. **ant db-init**

These two steps create the Grouper v1.3.0 schema in your new database.

6. **ant xml-import -Dcmd="GrouperSystem theSameFileName"**

This re-loads everything into the new database.