Grouper loader real time updates

This feature allows you to have incremental loading of memberships for SQL and LDAP based loader jobs. For SQL jobs, you can add database triggers onto your existing loader tables to populate a new table to indicate which users have been updated for which jobs. And for both SQL and LDAP jobs, you can add messages to a message queue and have Grouper look for changes there. Then the incremental loader can process updates quickly just for those users.

If you use this with SQL_GROUP_LIST or LDAP_GROUPS_FROM_ATTRIBUTES, the loader job must have grouperLoaderGroupsLike configured. Also, for SQL_GROUP_LIST and LDAP_GROUPS_FROM_ATTRIBUTES, if a new group doesn't exist in Grouper yet, the incremental loader will trigger a full sync.

Support for LDAP_SIMPLE and LDAP_GROUP_LIST jobs is minimal right now. If a message is added to the message queue for those 2 types of jobs, a full sync is simply performed. If you plan to use this feature and would like to see this improved, please contact the Grouper Team.

For LDAP_GROUPS_FROM_ATTRIBUTES, the specified loaderGroupName can be ALL_LDAP_GROUPS and that will sync all groups of type LDAP_GROUPS_FROM_ATTRIBUTES for the given user.

Configuration

By default, multiple threads will be used to improve performance. This is the default configuration but can be adjusted in grouper-loader.properties.

```
loader.incrementalThreads=true
loader.incrementalThreadPoolSize=10
```

In grouper-loader.properties, configure your incremental jobs. You can have multiple jobs, which would mainly be relevant if you use multiple databases for your loader jobs.

```
otherJob.incrementalLoader1.class = edu.internet2.middleware.grouper.app.loader.GrouperLoaderIncrementalJob
otherJob.incrementalLoader1.quartzCron = 0 * * * * ?
otherJob.incrementalLoader1.databaseName=warehouse
otherJob.incrementalLoader1.tableName=myincrementaltable
otherJob.incrementalLoader1.skipIfFullSyncDisabled=true

otherJob.incrementalLoader2.class = edu.internet2.middleware.grouper.app.loader.GrouperLoaderIncrementalJob
otherJob.incrementalLoader2.quartzCron = 0 * * * * ?
otherJob.incrementalLoader2.databaseName=warehouse2
otherJob.incrementalLoader2.tableName=myincrementaltable
otherJob.incrementalLoader2.skipIfFullSyncDisabled=true
```

Note that skipIfFullSyncDisabled defaults to true. This means that updates to groups will be skipped if the overall loader job associated with the group is disabled. This status is available in the "All daemon jobs" page.

Database setup

Your incremental table should have the following columns:

- id - primary key (e.g. auto incremental or a sequence)
- subject_id - mutually exclusive with subject_identifier/subject_id_or_identifier
- subject_identifier - mutually exclusive with subject_id/subject_id_or_identifier
- subject_id_or_identifier - mutually exclusive with subject_id/subject_identifier
- subject_source_id - optional
- loader_group_name - Grouper group name of the loader group that stores the definition.
- timestamp - milliseconds. Your trigger should populate this.
- completed_timestamp - milliseconds. The incremental loader will populate this when the row is processed. After one day, the row will be deleted.

```
HSQLDB:

CREATE TABLE myincrementaltable
(
    id INTEGER NOT NULL,
    subject_id VARCHAR(255),
    subject_identifier VARCHAR(255),
    subject_id_or_identifier VARCHAR(255),
    subject_source_id VARCHAR(255),
    loader_group_name VARCHAR(1024) NOT NULL,
    timestamp BIGINT NOT NULL,
    completed_timestamp BIGINT,
    PRIMARY KEY (id)
);
```

```
Postgresql:  Note that the trigger (see examples below) does not need to fill in the timestamp as it
defaults in the table DDL.  The indexes are suggested if you are putting more than a 1000 or so rows
in here per day.  Most sites run the incremental_loader very often, so the query to find zero rows
should complete very quickly.

CREATE TABLE IF NOT EXISTS myincrementaltable(
    id SERIAL NOT NULL UNIQUE,
    subject_id VARCHAR( 256 ),
    subject_identifier VARCHAR( 256 ),
    subject_id_or_identifier VARCHAR( 256 ),
    subject_source_id VARCHAR( 256 ),
    loader_group_name VARCHAR( 1024 ) NOT NULL,
    timestamp BIGINT NOT NULL DEFAULT( EXTRACT(EPOCH FROM CLOCK_TIMESTAMP())*1000::BIGINT ),
    completed_timestamp BIGINT,
    PRIMARY KEY( id )
);

CREATE INDEX IF NOT EXISTS myincrementaltable_completed_timestamp_index ON
    myincrementaltable( completed_timestamp );
CREATE INDEX IF NOT EXISTS myincrementaltable_loader_group_name_index ON
    myincrementaltable( loader_group_name );

GRANT ALL ON myincrementaltable TO grouperROLE;
GRANT ALL ON myincrementaltable_id_seq TO grouperROLE;

--
-- This might be useful as well.
--

CREATE OR REPLACE VIEW view_myincrementaltable AS
    SELECT subject_id, TO_TIMESTAMP( timestamp/1000 ) AS queued_timestamp,
        TO_TIMESTAMP( completed_timestamp/1000 ) AS completed_timestamp,
        TO_TIMESTAMP( completed_timestamp/1000 ) -
            TO_TIMESTAMP( timestamp/1000 ) AS elapsed

        FROM myincrementaltable ORDER BY id DESC;

GRANT SELECT ON myincrementaltable TO grouperROLE;
```

Messaging

If you are triggering updates via messaging instead of database triggers, the table above is still needed. A builtin message listener will simply monitor your queue and add rows to the table. Note that the "id" column has to be populated automatically on insert (i.e. use an auto increment type column or use a trigger to populate based on a sequence before the insert).

In your grouper-loader.properties file, add the following configuration for a new message listener:

```
messaging.listener.myCustomMessagingListener.class = edu.internet2.middleware.grouper.app.loader.
GrouperLoaderIncrementalMessagingListener
messaging.listener.myCustomMessagingListener.quartzCron = 0 * * * * ?
messaging.listener.myCustomMessagingListener.messagingSystemName = grouperBuiltinMessaging
messaging.listener.myCustomMessagingListener.queueName = abc
messaging.listener.myCustomMessagingListener.numberOfTriesPerIteration = 3
messaging.listener.myCustomMessagingListener.pollingTimeoutSeconds = 18
messaging.listener.myCustomMessagingListener.sleepSecondsInBetweenIterations = 0
messaging.listener.myCustomMessagingListener.maxMessagesToReceiveAtOnce = 20
# if there are 20 messages to receive at once, then do this 50 times per call max
messaging.listener.myCustomMessagingListener.maxOuterLoops = 50
messaging.listener.myCustomMessagingListener.incrementalLoaderJobName = incrementalLoader1
messaging.listener.myCustomMessagingListener.messageQueueType = queue
```

- Update the messagingSystemName to point to your messaging system (Grouper supports a built in messaging system along with RabbitMQ, AWS, etc).
- Update the queueName
- Update incrementalLoaderJobName based on what was configured earlier in the Configuration section above.

Format of messages:

```
{'subjectId':'test.subject.0', 'loaderGroupName':'test:owner', 'subjectSourceId':'jdbc'}
```

- Must have subjectId, subjectIdentifier or subjectIdOrIdentifier.
- Must have loaderGroupName
- subjectSourceId is optional

Fail safe

If there are more than a certain number of changes for a loader job, the incremental loader will avoid processing those changes and instead immediately trigger a full sync. This helps with performance but also outsources fail safe to the full sync, which has its own configuration options. The number by default is 100 but can be adjusted in grouper-loader.properties:

```
otherJob.incrementalLoader1.fullSyncThreshold=100
```

Examples

Example of SQL_SIMPLE using Oracle (loader table has a group name field to allow multiple SQL_SIMPLE jobs):

Say you have a loader table that looks like the following:

```
CREATE TABLE myloadertable
(
    subject_id VARCHAR(255),
    group_name VARCHAR(1024)
);
```

With the following incremental table:

```
CREATE TABLE myincrementaltable
(
    id NUMBER NOT NULL,
    subject_id VARCHAR(255),
    subject_identifier VARCHAR(255),
    subject_id_or_identifier VARCHAR(255),
    subject_source_id VARCHAR(255),
    loader_group_name VARCHAR(1024) NOT NULL,
    timestamp NUMBER NOT NULL,
    completed_timestamp NUMBER,
    PRIMARY KEY (id)
);
```

And a sequence for the primary key on the incremental table:

```
CREATE SEQUENCE myincrementaltable_seq;
```

And the following loader job:

```
addRootStem("test", "test")
addGroup("test", "loader1", "loader1")
groupAddType("test:loader1", "grouperLoader")
setGroupAttr("test:loader1", "grouperLoaderDbName", "grouper")
setGroupAttr("test:loader1", "grouperLoaderType", "SQL_SIMPLE")
setGroupAttr("test:loader1", "grouperLoaderScheduleType", "START_TO_START_INTERVAL")
setGroupAttr("test:loader1", "grouperLoaderQuery", "select subject_id from myloadertable")
setGroupAttr("test:loader1", "grouperLoaderIntervalSeconds", "86400")
```

Assuming this is on the same database as Grouper, you could add the following configuration in grouper-loader.properties (run every 5 seconds):

```
otherJob.incrementalLoader1.class = edu.internet2.middleware.grouper.app.loader.GrouperLoaderIncrementalJob
otherJob.incrementalLoader1.quartzCron = 0/5 * * * * ?
otherJob.incrementalLoader1.databaseName=grouper
otherJob.incrementalLoader1.tableName=myincrementaltable
```

And the following trigger:

```
CREATE OR REPLACE TRIGGER mytrigger
AFTER INSERT OR DELETE OR UPDATE ON myloadertable
FOR EACH ROW
DECLARE
  timemillis NUMBER;
BEGIN
  select extract(day from(sys_extract_utc(systimestamp) - to_timestamp('1970-01-01', 'YYYY-MM-DD'))) * 86400000
     + to_number(to_char(sys_extract_utc(systimestamp), 'SSSSSFF3')) into timemillis from dual;
  IF (:new.subject_id is not null) THEN
    INSERT INTO myincrementaltable (id, subject_id, loader_group_name, timestamp) values
(myincrementaltable_seq.nextval, :new.subject_id, :new.group_name, timemillis);
  END IF;
  IF (:old.subject_id is not null) THEN
    INSERT INTO myincrementaltable (id, subject_id, loader_group_name, timestamp) values
(myincrementaltable_seq.nextval, :old.subject_id, :old.group_name, timemillis);
   END IF;
END;
```

Example of SQL_GROUP_LIST using Oracle (assumes a single loader job):

Say you have a loader table that looks like the following:

```
CREATE TABLE myloadertable
(
    subject_id VARCHAR(255),
    group_name VARCHAR(1024)
);
```

With the following incremental table:

```
CREATE TABLE myincrementaltable
(
    id NUMBER NOT NULL,
    subject_id VARCHAR(255),
    subject_identifier VARCHAR(255),
    subject_id_or_identifier VARCHAR(255),
    subject_source_id VARCHAR(255),
    loader_group_name VARCHAR(1024) NOT NULL,
    timestamp NUMBER NOT NULL,
    completed_timestamp NUMBER,
    PRIMARY KEY (id)
);
```

And a sequence for the primary key on the incremental table:

```
CREATE SEQUENCE myincrementaltable_seq;
```

And the following loader job:

```
addRootStem("test", "test")
addGroup("test", "owner", "owner")
groupAddType("test:owner", "grouperLoader")
setGroupAttr("test:owner", "grouperLoaderDbName", "grouper")
setGroupAttr("test:owner", "grouperLoaderType", "SQL_GROUP_LIST")
setGroupAttr("test:owner", "grouperLoaderScheduleType", "START_TO_START_INTERVAL")
setGroupAttr("test:owner", "grouperLoaderQuery", "select group_name, subject_id from myloadertable")
setGroupAttr("test:owner", "grouperLoaderIntervalSeconds", "86400")
setGroupAttr("test:owner", "grouperLoaderGroupsLike", "test:loader%")
```

Assuming this is on the same database as Grouper, you could add the following configuration in grouper-loader.properties (run every 5 seconds):

```
otherJob.incrementalLoader1.class = edu.internet2.middleware.grouper.app.loader.GrouperLoaderIncrementalJob
otherJob.incrementalLoader1.quartzCron = 0/5 * * * * ?
otherJob.incrementalLoader1.databaseName=grouper
otherJob.incrementalLoader1.tableName=myincrementaltable
```

And the following trigger:

```
CREATE OR REPLACE TRIGGER mytrigger
AFTER INSERT OR DELETE OR UPDATE ON myloadertable
FOR EACH ROW
DECLARE
  timemillis NUMBER;
BEGIN
  select extract(day from(sys_extract_utc(systimestamp) - to_timestamp('1970-01-01', 'YYYY-MM-DD'))) * 86400000
     + to_number(to_char(sys_extract_utc(systimestamp), 'SSSSSFF3')) into timemillis from dual;
  IF (:new.subject_id is not null) THEN
    INSERT INTO myincrementaltable (id, subject_id, loader_group_name, timestamp) values
(myincrementaltable_seq.nextval, :new.subject_id, 'test:owner', timemillis);
  END IF;
  IF (:old.subject_id is not null) THEN
    INSERT INTO myincrementaltable (id, subject_id, loader_group_name, timestamp) values
(myincrementaltable_seq.nextval, :old.subject_id, 'test:owner', timemillis);
   END IF;
END;
```

Example of SQL_SIMPLE using MySQL (loader table has a group name field to allow multiple SQL_SIMPLE jobs):

Say you have a loader table that looks like the following:

```
CREATE TABLE myloadertable
(
    subject_id VARCHAR(255),
    group_name VARCHAR(1024)
);
```

With the following incremental table:

```
CREATE TABLE myincrementaltable
(
    id BIGINT NOT NULL AUTO_INCREMENT,
    subject_id VARCHAR(255),
    subject_identifier VARCHAR(255),
    subject_id_or_identifier VARCHAR(255),
    subject_source_id VARCHAR(255),
    loader_group_name VARCHAR(1024) NOT NULL,
    timestamp BIGINT NOT NULL,
    completed_timestamp BIGINT,
    PRIMARY KEY (id)
);
```

And the following loader job:

```
addRootStem("test", "test")
addGroup("test", "loader1", "loader1")
groupAddType("test:loader1", "grouperLoader")
setGroupAttr("test:loader1", "grouperLoaderDbName", "grouper")
setGroupAttr("test:loader1", "grouperLoaderType", "SQL_SIMPLE")
setGroupAttr("test:loader1", "grouperLoaderScheduleType", "START_TO_START_INTERVAL")
setGroupAttr("test:loader1", "grouperLoaderQuery", "select subject_id from myloadertable")
setGroupAttr("test:loader1", "grouperLoaderIntervalSeconds", "86400")
```

Assuming this is on the same database as Grouper, you could add the following configuration in grouper-loader.properties (run every 5 seconds):

```
otherJob.incrementalLoader1.class = edu.internet2.middleware.grouper.app.loader.GrouperLoaderIncrementalJob
otherJob.incrementalLoader1.quartzCron = 0/5 * * * * ?
otherJob.incrementalLoader1.databaseName=grouper
otherJob.incrementalLoader1.tableName=myincrementaltable
```

And the following trigger:

```
delimiter |
CREATE TRIGGER mytrigger_insert
AFTER INSERT ON myloadertable
FOR EACH ROW
BEGIN
  INSERT INTO myincrementaltable (subject_id, loader_group_name, timestamp) values (NEW.subject_id, NEW.
group_name, ROUND(UNIX_TIMESTAMP(CURTIME(4)) * 1000));
END;

|
CREATE TRIGGER mytrigger_update
AFTER UPDATE ON myloadertable
FOR EACH ROW
BEGIN
  INSERT INTO myincrementaltable (subject_id, loader_group_name, timestamp) values (NEW.subject_id, NEW.
group_name, ROUND(UNIX_TIMESTAMP(CURTIME(4)) * 1000));
  INSERT INTO myincrementaltable (subject_id, loader_group_name, timestamp) values (OLD.subject_id, OLD.
group_name, ROUND(UNIX_TIMESTAMP(CURTIME(4)) * 1000));
END;
|
CREATE TRIGGER mytrigger_delete
AFTER DELETE ON myloadertable
FOR EACH ROW
BEGIN
  INSERT INTO myincrementaltable (subject_id, loader_group_name, timestamp) values (OLD.subject_id, OLD.
group_name, ROUND(UNIX_TIMESTAMP(CURTIME(4)) * 1000));
END;
|
```

```
delimiter ;
```

Next steps / dev notes

- Better support for LDAP loader jobs - currently if there's an update for an LDAP job, it will just run the full sync, which obviously isn't efficient. Proposed updates:
    - The basic idea is to adjust the search filters when invoked via the real time loader to limit the results.
        - LDAP_SIMPLE - Have a config option that converts the subject id/identifier to the DN (grouperLoaderLdapSubjectReverseExpression). Basically, this is the opposite of grouperLoaderLdapSubjectExpression. Then the real-time loader can quickly see if the user is supposed to be in the group by adding the result of this expression to the search filter, e.g. (&(existingFilter)(subjectAttribute=result_of_expression))
        - LDAP_GROUP_LIST - Similar to above with the use of grouperLoaderLdapSubjectReverseExpression. Also, the group name expression code should be moved to allow it to be reused by the real-time loader as well.
        - ~~LDAP_GROUPS_FROM_ATTRIBUTES - Perhaps only support the real-time loader if grouperLoaderLdapSubjectAttribute is specified and grouperLoaderLdapSubjectExpression is not specified. This would basically mean that the subject id/identifier would be an attribute of the user. Otherwise, what is the use case for needing an expression here??~~
    - ~~Have an option to sync all LDAP groups for a user since it may not be obvious to the deployer which LDAP group(s) would be impacted when a user is updated in LDAP. Perhaps allow the loader_group_name column to be populated with something like "ALL_LDAP_GROUPS".~~
    - For SQL_GROUP_LIST, the real-time loader only works when grouperLoaderGroupsLike is specified. This was needed so that it knew which groups were owned by the job and could adjust their memberships appropriately. Now that there's loader metadata, perhaps SQL_GROUP_LIST and LDAP_GROUP_LIST can rely on that instead.