Auditing

Auditing exists for 3 reasons: point in time, user auditing, notifications

Point in time

"Point in time" (PIT) is knowing what the state of the registry was at some point in time in the past. A certain configurable amount of logs will be stored (e.g. last 1 month).

Design: we would like to use java to manage diffs. Triggers dont work well since it doesnt have all the information it needs (e.g. if there is a delete, what is the context id?)

Basically we have a shadow table for every table in the registry, with the same (or nearly the same columns), and a couple more (timestamp, and type of change, I (insert), U (update), D (delete). As data is changed, then the UDI records will be inserted into the audit tables. The PIT is calculated by considering the current state of data (in registry), and looking at the previous UDI's. Joining tables could be a pain, but it should be possible... A loader job can manage the cleaning up old data (if not stored indefinitely). Another feature could be an export in a point in time, to recreate the registry and do fast calculations.

User auditing

User auditing is auditing what users (or processes) do to the registry at a high level. e.g. Sue added a group on a certain day. It might not record that since that group was created, some privileges were created, the "base" type was associated with that group, etc.

User auditing

Notifications

These can be one table, which holds type (e.g. group, stem, etc), id, timestamp, action (e.g. insert, update, delete). Java will insert records into the change log. Then a loader process could read records off of this table in order of time, and handoff the data to the notification hook, or mark them as processed somehow in another table after the processing has occurred. This would be transactional, have correct ordering, would not miss records, and would be easy to implement.

There will nbe two tables for notifications. A "temp" table where data is inserted from all processes, then a loader daemon will move data from there into the real notification change log table. Basically no timestamp (to the microseconds) will be the same from one JVM (this is assured by code), but timestamps can be the same across JVMs, and they can be inserted in the wrong order. When data is moved over to the real change log table, a sequential, increasing, unique sequence number will be assigned to each row. This sequence will be the primary key. In the temp table, maybe all columns are the logical primary key.