Datomic Schema Agent

# Operation-Commands

This section describes in a high level overview the operations that will be created, their behavior, and the parameters and sub-operational behaviors comprising the major features. These operations, while resembling command line operations, are in fact not. They are a more general purpose behavior with the corresponding CLI commands described in a later section.

## Agent Configuration, setup, and maintenance operations

This is inherently internal or, could be implemented later or not at all and, not effect the operation of the Agent. These are nice to haves but, do not enable any additional functionality, with the exception of verify-datastore-operational-status.

### add-datastore

### update-datastore

### remove-datastore

### list-datastores

### verify-datastore-operational-status

## Data management operations

import/export, backup/restore, and, maybe for later on: visualization? statistical-analysis?

### capture-dataset

This is a core operation however it is generally useful as well for understanding a datomic-dataset-instance as it resides in a given moment in time. Therefore, it is provided as a packaged-operation exposed and available to the user to. The completed result of this operation is to connect to a given set of database server’s and database’s and, extract the entirety of the capture-target’s Transaction IDs, DATOMS for the EAVT, AEVT, AVET, and VAET indexes as well as history and db-stats information-structures. Once this information has been collected, it is aggregated into a single data-structure where additional identification, tracking and, meta properties are applied, enabling us to uniquely identify the dataset and its relationship to its constituent components, which enables it to become part of future operations.

#### convert-datoms-to-entities-model

One operation being considered, likely as a sub-operation of capture-dataset would be a convert-datoms model to an entity-model. This is useful both for the extract and import commands as well as the convenience of having a more human-centric view of the data. However, in addition, it might be a requirement or at the very least helpful if we decide that we want to perform operations between core data storage technologies (i.e. Datomic-RDF-Graph => Postgres-Relational).

### extract-entities

**Not sure why I have this in the sense of how does it differ from export-entities.** Will Hold for now but, I believe that this will be needed. The idea might have been with extract vs import – might be to have a database-instance-specific capture with extract, where all database--data-structure--instance specific identifiers are retained, perhaps for debugging purposes where export has some subsequent processing required to both generalize, link-retention/enforcement and, then, to be and maintain (via hashing) an import-ready data-structure. (the idea with the hashing is to enforce the integrity such that when it is imported into a new database-system, it does not perform a break in the data-structure.

### export-entities

### import-entities

### create-backup

### restore-backup

## Data migration operations

### generate-migration-plan

Generates a 'migration-plan' -- does this by producing a gap-computation between the destination schema and the current state of the migration-target systems and calculated-set of property differences between them.

### capture-migration-target-definitions

The capture-migration-target-definitions operation interrogates the target database instances to generate a model of their schemas, enabling the other side of a migration-plan to be generated and the computed migrations can then be applied. This information required to perform the capture-migration-target-defs is either a set of instance configurations defining the full set of parameters required to generate the connections to each database or, a collection of names as quoted strings referencing the Agent’s configuration-store named connection configurations in the overall (global) agent-config.

### produce-unsatisfiable-field-definitions-report (alternatively: alert-on-unsatisfiable-fields)

This generates a “report” that highlights which migrated destination fields where unable to be transformed do to either problems in generating a new object or in inferencing the destination target.

### edit-field-transforms

Allows the user to perform adjustments or enhancements to the calculated field-transformation operations. (and default values for new fields). Takes the user through all fields that will be created or transformed in some way and allows them to specify transformation strategies as well as defaults.

#### interactive

#### each-entity

#### overall-entity

### compute-destination-structure operation

This operation is to perform a computation of what properties should be on the target migration-destination, subsequently to be applied to each updated schema. This operation can be performed using the following methods:

#### within-single-instance

This was the original idea for this project. Go through the extracted data-set and compute a super set of properties, using the fact that they were tied to common entities as other members in the association to establish membership.

#### across-many-instances

This method takes a collection of data-sets and applies the same strategy as before, looking for additions and subtractions or changes of properties but, does it from a multitude of running database instances or extracted datasets.

#### from-schema-def-in-source

This is basically saying, use the current schema related properties in a provided source directory to determine an "absolute state of the world". This can either take a file-path to a clojure source directory containing schema-definition command-attributes or, a github URL.

// Im not sure that this comment is relative or accurate:

This then has the ramification of not allowing the destination schema structure from updating itself as a result of processing. The update means in terms of, if it found by processing the model changes linearly