Repository with history

DACI: How to implement a repository with history?

Status	RELEASED
Impact	HIGH
Driver	Chris Grote
Approver	Mandy Chessell
Contributors	Graham Wallis David Radley
Informed	
Due date	
Outcome	Progressed option 2 to a release state

Tips and info

Recommendations

We agreed to examine the potential of Option 2 in more detail, and have now ultimately taken that approach to a released state.

Background

A common scenario we come across with almost all metadata repositories we have seen is that they lack the ability to store historical information about metadata and respond to point-in-time inquiries. While Egeria's type system and APIs have been built from the beginning to support such history, we have not yet implemented a backend storage option that implements history.

Considering this comes up frequently as a common need, even to augment existing metadata repositories, providing such a historical store for metadata could be a somewhat narrow but nonetheless extremely common adoption point for Egeria.

Current state

We are currently considering implementation options for an initial approach to such a repository.

Data for decision support

- Identification of potential technologies to use as the backing store for such a repository.

Options considered

	Option 1: bi-temporal RDBMS	**Option 2: bi-temporal graph**	**Option 3: search index**
Description	Using a bi-temporal relational database like DB2	Using a bi-temporal graph store like Crux	Using a search index like Elastic
Rollout plan		Start with some initial proof of concept activities like building some of the basic methods in a repository connector.	Leaving as an alternative approach that was suggested, but no further details available.

Pros and cons	✓ **Native** Handles historical information natively at the storage layer, so should be simpler to implement point-in-time inquiry. ❗ **New approach** Takes a new approach to a backing store (relational) compared to our existing implementations (graph-based) ❗ **Commercial** We are unaware of any open source, native bi-temporal RDBMS, so this would put a dependency on licensed commercial software. ❗ **Schema** Requires a fixed schema, which raises questions about how to both handle efficient queries (not storing things as unqueryable blobs) but also manage history when the type system itself (schema?) may have changed over the course of that history (ie. deprecated attributes and types)	✓ **Native** Handles historical information natively at the storage layer, so should be simpler to implement point-in-time inquiry. ✓ **Similar to existing** Close alignment with our current repository approaches that are more graph-focused than relational. ✓ **Embedded option** Provides a simple option to run in an embedded capacity, which could be useful for demonstration purposes (not requiring additional infrastructure and components). ✓ **Pluggable backends** Implemented using pluggable characteristics for its own backends, including both open source and commercial options. ✓ **Schemaless** It sounds like each document in Crux is essentially schema-less (tuples / triples-based), so it may be feasible to store multiple versions of a type across the history of a given instance of metadata ❓	
Risks		⚠ **Scalability** The resource requirements that might be necessary for a "true production" rollout are unclear, or the volume to which it can scale. (We heard mention of "16 TB" (sounds plenty) ~~but also "10 million triples" (with history, and one triple per attribute value, per instance, this sounds small?)~~ – from subsequent conversations we confirmed that this is 10 *billion* triples rather than million, alleviating our immediate concerns.	
Estimated cost and effort			

FAQ

Q1.

A1.

References

Relevance	**Link**
Original GitHub issue	https://github.com/odpi/egeria/issues/2545
Discussion with Crux team	2020-11-27 Meeting notes

Follow-up action items

☐

Learn more: https://www.atlassian.com/team-playbook/plays/daci