Towards a Unified Query Language for Provenance and Versioning

Amit Chavan
Silu Huang
Aaron Elmore
Samuel Madden

Amol Deshpande
Aditya Parameswaran
DATAHUB: A COLLABORATIVE HOSTED DATA SCIENCE PLATFORM

The one-stop solution for collaborative data science and dataset version management

http://data-hub.org
DATAHUB: A COLLABORATIVE HOSTED DATA SCIENCE PLATFORM

• a dataset management system – import, search, query, analyze a large number of (public) datasets

• a dataset version control system – branch, update, merge, transform large structured or unstructured datasets

• an app ecosystem and hooks for external applications (Matlab, R, iPython Notebook, etc)

- DataHub: A Collaborative Data Analytics Platform

DataHub Architecture

Versioned Datasets, Version Graphs, Indexes, Provenance

I: Versioning API and Version Browser

II: Native App Ecosystem

III: Language Agnostic Hooks

Client Applications

DataHub Notebook

Ingest, Visualize, Query, etc.
Collaborative data science projects end up in dataset version management hell

- Many private copies of the datasets → Massive redundancy
- No easy way to keep track of dependencies between datasets
- Manual intervention needed for resolving conflicts
- No efficient organization or management of datasets
- No way to analyze/compare/query versions

Courtesy: XKCD
WHAT ABOUT GIT/SVN/...?

Analogous to management of source code before source code version control!

Many issues with directly using GitHub etc..
- Cannot handle large datasets or large # of versions (VLDB 2015)
- Datasets have regular repeating structure
- Querying and retrieval functionality is primitive

Temporal databases only support a linear chain of versions
NEED A RICH LANGUAGE FOR QUERYING AND RETRIEVAL

Querying in traditional VCS largely revolves around single version and metadata retrieval

No way to specify queries like:

• identify all versions derived from version A that satisfy property P

• identify all predecessor versions of version A that differ from it by a large number of records

• rank a set of versions according to a scoring function

• find the version where the result of an aggregate query is above a threshold

• find parent records of all records in version A that satisfy certain property
GOALS

To fully realize the DataHub vision, need a language that can:

- support all existing **VCS API**
- allow working with both **versions and data** seamlessly
- navigate the ad-hoc **derivation graph** of versions
- allow **declarative querying** of the data to the extent possible

Why a new language?

- Temporal query languages (e.g., TQuel) only work with a linear history of versions
- SQL is ill-suited to traversing a graph structure, and has a cumbersome aggregate syntax
- Several languages for workflow systems, but often quite specific to the platform
Generalization of Quel – a tuple calculus-based language developed for INGRES

Chosen primarily because of cleaner syntax

VQuel combines:

• full-fledged relational features and powerful aggregate constructs from Quel
• syntactic features from GEM, SQL, and path-based query languages
• iterator-based access to both versions and data items
“version”: immutable and consists of one or more datasets (files, relations) that are semantically grouped together

New versions created through the application of transformation programs or updates to one or more existing versions.

Version-level provenance is captured in the “version graph”
Queries written against a Conceptual Hierarchical Data Model
Example 1: What commits did Alice make after January 01, 2015?

V is an iterator over all the Versions

range of V is Version
retrieve V.all
where V.author.name = "Alice" and
V.creation_ts >= "01/01/2015"

Predicates are used to restrict the results returned
Example 2: Show the history of the tuple with employee id “e01” from Employee relation.

range_of V is Version
range_of R is V.Relations
range_of E is R.Tuples

retrieve E.all, V.commit_id, V.creation_ts
where E.employee_id = “e01” and
    R.name = “Employee”

sort_by V.creation_ts

R is an iterator over relations in a Version
E is an iterator over tuples in a Relation
AGGREGATES

Example 3: Among a group of versions, find the version containing most tuples that satisfy a predicate. For instance, which version contains the most number of employees above age 50?

range of V is Version
range of E is V.Relations(name = "Employee").Tuples
retrieve into T (V.id as id,
    count(E.id where E.age > 50) as c)
retrieve T.id
where T.c = max(T.c)

Aggregates can be used in both retrieve and where clauses

Restricts the tuples being considered in the counting

“retrieve into” implicitly defines an iterator

Evaluated once, used as a constant thereafter
Example 4: Find all versions within 2 commits of “v01” which have less than 100 employees.

range of V is Version(id = "v01")
range of N is V.N(2)
range of E is N.Relations(name = "Employee").Tuples

retrieve N.all
where count(E) < 100
AND MORE...

See paper for:

• Additional constructs for aggregates
• Partitioned aggregates – GROUP BY clause
• Joins across versions
• Additional constructs to traverse the version graph
• Querying fine grained provenance
THE ROAD AHEAD

Extensions

- Include user defined functions – e.g., custom “diff” functions for two versions
- Additional graph traversal operators

Engagement with users to refine the constructs

Implementation Challenges

- Data is stored in a compressed fashion, to exploit overlaps between versions
- Version graph can become very large in a “dynamic update” environment

- Need new query execution and optimization strategies
- Need scalable methods to handle the version graph
MORE ABOUT DATAHUB...

  *41st International Conference on Very Large Data Bases (VLDB), 2015.*

• Collaborative Data Analytics with Datahub (Demo). Anant Bhardwaj, Amol Deshpande, Aaron Elmore, David Karger, Sam Madden, Aditya Parameswaran, Harihar Subramanyam, Eugene Wu, and Rebecca Zhang.  
  *41st International Conference on Very Large Data Bases (VLDB), 2015.*

  *Conference on Innovative Database Research (CIDR), 2015.*
THANK YOU