Implementing the RDA Data Citation Recommendations for Long Tail Research Data

Stefan Pröll
Overview

- Introduction
- Recap of the WGDC Recommendations
- Long Tail Research Data
- SQL Prototype
- Git Prototype
- Conclusion
Data Driven Research

- Modern research is data driven
  - Results are based on data
  - But the results are still published in papers
- Data sets are often
  - Not available or accessible
  - Not cited
  - Ambiguous

→ Reproducibility is at risk
Citing data may seem easy
- from providing a URL in a footnote
- via providing a reference in the bibliography section
- to assigning a PID (DOI, ARK, …) to dataset in a repository

What’s the problem?
Main Challenges

- **Scalability**
  - More and more data sets
  - Growing amounts of data
  - **Granularity**

- **Infrastructure**
  - Sophisticated data management is not always available
  - Processes not defined well

- **Dynamics**
  - Frequent updates
  - **Evolving data**

- **Precise identification**
  - Ambiguity?
Granularity of Subsets

What about the **granularity** of data to be identified?
- Enormous amounts of data
- Researchers use specific subsets of data
- Need to identify precisely the subset used

Current approaches
- Storing a copy of subset as used in study -> scalability
- Citing entire dataset, providing textual description of subset
  - -> imprecise (ambiguity)
- Storing list of record identifiers in subset -> scalability,
  - not for arbitrary subsets (e.g. when not entire record selected)

Would like to be able to identify precisely the **subset of (dynamic) data used** in a process
Identification of Dynamic Data

- Citable datasets have to be static
  - Fixed set of data, no changes:
    - no corrections to errors, no new data being added
  - But: (research) data is **dynamic**
    - Adding new data, correcting errors, enhancing data quality, …
    - Changes sometimes highly dynamic, at irregular intervals
- Current approaches
  - Identifying entire data stream, without any versioning
  - Using “accessed at” date
  - “Artificial” versioning by identifying batches of data (e.g. annual), aggregating changes into releases (time-delayed!)
- Would like to identify precisely the **data as it existed at a specific point in time**
Research Data Alliance

WG on **Data Citation:**

**Making Dynamic Data Citeable**

WG officially endorsed in March 2014

Concentrating on the problems of

- large, dynamic (changing) datasets
- Focus! Identification of data!
- Not: PID systems, metadata, citation string, attribution, …
- Liaise with other WGs and initiatives on data citation
- (CODATA, DataCite, Force11, …)

- [https://rd-alliance.org/working-groups/data-citation-wg.html](https://rd-alliance.org/working-groups/data-citation-wg.html)
Basic Principle

Idea: Versioned data + timestamped queries

- Data: timestamped and versioned (aka history)
- Query: Timestamped

- Access: Re-execute query on versioned data with the appropriate timestamp.

- Trick: Assign the PID to the query
14 Recommendations

- Grouped into 4 phases:
  - Preparing data and query store
  - Persistently identifying specific data sets
  - Resolving PIDs
  - Upon modifications to the data infrastructure

- 2-page flyer
- More detailed Technical Report:
- Reference implementations
- (SQL, CSV, XML) and Pilots
Long Tail Research Data

Big data, well organized, often used and cited

Less well organized, "Dark data" non-standardised no dedicated infrastructure

Dynamic Data Citation for CSV Data

- **Goals:**
  - Ensure cite-ability of CSV data
  - Enable subset citation
  - Support particularly small and large volume data
  - Support dynamically changing data
  - Establish links between data set and subsets
  - Scalable approach without storing copies of data exports

- **Why CSV data?**
  - Well understood and widely spread
  - Small and big data settings
  - Simple and flexible
Advanced data infrastructure
- Large data sets
- Database driven
- Defined interfaces
- Trained experts available

Required adaptations
- Ingest CSV files
- Capture subset process
- Implement dedicated query store
- SQL Prototype
- Local workstations
  - Smaller data sets
  - Local storage and tools
  - Scripting languages
- Required adaptations
  - Data versioning, e.g. with Git
  - Store scripts versioned as well
  - Make subset creation reproducible
  - Document software and OS versions
  - Share repositories

Git Prototype
Prototype Implementations

- SQL based Prototype
  - A) Migrates CSV data into relational database

- Git based Prototypes
  - A) Git as backend only
  - B) Using branches for data and scripts

- Data backend responsible for versioning data sets
- Subsets are created with scripts or queries
Reproducible Subsets with SQL

- CSV files have the same structure as relational database tables
- Subsetting process via SQL SELECT statements

```
SELECT columns FROM table WHERE conditions HAVING conditions ORDER BY column
```
Data Citation – Deployment

- Researcher uses workbench or tool to identify subset of data
- Upon executing selection („download“) user gets
  - Data (package, access API, …)
  - PID (e.g. DOI) (Query is time-stamped and stored)
  - Hash value computed over the data for local storage
  - Recommended citation text (e.g. BibTeX)
  - Query string
- PID resolves to landing page
  - Provides detailed metadata, link to parent data set, subset,…
  - Option to retrieve original data OR current version OR changes
- Upon activating PID associated with a data citation
  - Query is re-executed against time-stamped and versioned DB
  - Results as above are returned
- Query store aggregates data usage
Reproducible Subsets with SQL Prototype

User interface
- Data upload
- Updates
- Subset creation
- Landing page display

Database backend
- Data versioning
- Event timestamping
- Migration

Query Store
- Store data set metadata
- Store query metadata
- Assign persistent identifiers (PID)
- Compute verification hashes
- Re-execute queries on demand
- Provide data for landing pages

Subsets
- Based on versioned data
- Execution time of the query
- Dynamically generated
Implementation Overview

• Presentation layer
  - Web interface

• Application server layer
  - CSV module
  - Query store module
  - Persistent identification module
  - Result set verification module

• Data server layer
  - Database module

• Technologies: Java 8, Maven 3, MySQL 5.7, Hikari CP, JSF, Primefaces, jQuery
Demo SQL Prototype

Videos available at: http://www.datacitation.eu/
Git as Data Backend

- **Git**
  - Distributed source code management software
  - Version control
  - Track changes
  - Ideal for text based file formats

- **Advantages of Git**
  - Local install possible
  - Available for all platforms
  - Repositories can be easily shared
  - Does not require central administration
  - Open source
Provide the same interface
- Data selection with GUI
- Git as backend
- Query store preserves CSV2SQL query
- Re-execution on top of CSV file revision

Git as Data Backend
- Ideal for text based formats
- Simple query translation via the interface
- Version all changes by committing
- Sharing via repositories (e.g. Github)
Subsets are created with a scripting language (e.g. R)
- Select columns, filter records and sort result set
- Script produces CSV file

Users store the subsetting script also in Git
- Subsetting process can be automatically executed
- The subsetting script is also stored in Git
- Metadata file describes script execution, language version, etc

Use Git to retrieve proper data set version and re-execute script on retrieved file

Advantage: Simple method, Integration with a Query Store
Disadvantage: Git commit history contains data set and script files
Reproducible Subsets with Git
Reproducible Subsets with Git Branches

- Using the Git branching model
  - Branches allow separation of data and scripts
  - Keeps commit history clean
    - Allows merging of data files
  - Use commit hash for identification
    - Assigned PID hashed with SHA1
    - Use hash of PID as filename
  - Orphaned branch for queries and metadata files
Reproducible Subsets with Git Prototype

Step 1: Select a CSV file in the repository

Step 2: Create a subset with a SQL query (on CSV data)

Step 3: Store the query script and metadata

Step 4: Re-Execute!
RDA WGDC Prototypes

- SQL Backend
- Git Backend
- Source code of all prototypes available at Github
- https://www.github.com/datascience
Conclusion

- **Query based data citation for evolving research data**
  - Enhances reproducibility
  - Relies on data versioning and query (script) timestamping
- **Implementation in small scale settings**
  - Git repositories can be easily shared
  - Metadata included
- **Implementation in large scale settings**
  - Versioning often already available
  - Interfaces for subsetting processes can be used for implementation
Thank You

Questions?
Comments?

Thank you very much for your attention!

stefan.proell@tuwien.ac.at
www.datacitation.eu
@stefanproell