Reproducible Database Queries in Privacy Sensitive Applications

Stefan Pröll [*] **Rudolf Mayer** [*] **Andreas Rauber** [**]

[*] *SBA-Research, Vienna, Austria (e-mail: sproell@sba-research.org, rmayer@sba-research.org).*
[**] *Institute of Software Technology and Interactive Systems at the Vienna University of Technology, Austria (e-mail: rauber@ifs.tuwien.ac.at)*

Abstract: Research databases allows scientists to gain sinsights in large datasets and deriving new knowledge from the data. In many cases the data stored in databases is sensitive and needs to be protected. Due to the increasing complexity of eScience settings and research projects, data is often integrated from different data sources stemming from competing data owners. In order to achieve the research goals, the data needs to be combined and analysed as a whole. As data owners of such sources may have potential conflicts of interest in certain aspects, a mechanism is needed which prevents the retrieval and or recombination of privacy related data while still full access to own data must be granted at all times. For enabling reproducible research, we present an approach which supports data citation and provides a mechanism for retrieving citable and privacy preserving datasets from large data sources.

Keywords: Reproducibility, Relational databases, Data handling systems, Data privacy, Data sets

1. INTRODUCTION AND MOTIVATION

Data is not only the product of computational research, but also crucial decisions are made on the basis of data. Whether or not an experiment was considered successful or a model is considered valuable is often settled on the resultssets delivered from statistical analysis of data. As data is not only a final product but is transformed, updated or changed in further process steps, each intermediate result needs to be tracebla and identifyable. Hence it is a fundamental requirement to know which exact data was involved in the decision making process and its intermediate steps. Data citation tackles the problem of uniquely identifying, referencing and citing datasets and their subsets in order to make them available at a later point in time. The goal of citing data is to attach a persistent identifier to each dataset retrieved from the database and use this identifier as a handle which allows to retrieve the exact same dataset again. Thus data citation enables the examination, revision or analysis of data and therefore constitutes evidence for decisions and how they have been made.

It is clear that the precise dataset needs to be available for long term in order to review and screen the data which was used in further process steps. For obvious reasons it is not sufficient exporting each individual dataset and store it as data dump in an archive file. For large data volumes this simple solution does not scale and without additional metadata the management of datafiles becomes a challenge. Identifying datasets can be achieved by applying fingerprinting and checksum mechanisms, but without the knowledge how the data was derived and how a specific subset was selected the reproducibility of data driven experiments is limited.

In many cases the datasets contain sensitive data and privacy protocols need to be applied. Implementing thorough permission schemes is essential, but the goal of providing a secure eScience environment becomes more challenging if several stakeholders with potentially competing interests need to exchange data. All participating partners obviously need access to the data they contributed, but providing access to aggregated results and compiled resultsets is a further requirement. The system needs to support analysis of the data but it needs to prevent the creation of datasets which contain sensitive data or allow the deduction of such information via skillful querying.

The goal of this work is identifying the requirements for enabling secure data citation for arbitrary subsets of integrated and potentially large data sources which maintain the privacy of the data during the complete data life cycle. The remainder of this paper is organized as follows: Section 2 provides and overview of the state of the art in data citation and reproducible research. Section 3 describes the scenario and identifies the requirements for achieving privacy aware data citation. Section 4 highlights the specifics of data citation regarding reproducible database queries conidering sensitive data in multi-user environments. Finally Section 5 gives a conclusion and an outlook on future work.

2. RELATED WORK

In the recent years data driven science and in-silico experimentation have produced remarkable results and con-

[⋆] Parts of this work are supported by the project DEXHELPP.

stituted e-Science as a completely new paradigm in many different disciplines Hey et al. (2009). With growing complexity of experiments it becomes increasingly difficult to reproduce the resultsMesirov (2010); Mayer et al. (2012) published in scientific journals and papers. Nevertheless reproducibility is the most important metric for valid researchLoscalzo (2012) and requires thorough documentation of all steps Schwab et al. (2000); Sandve et al. (2013). Different approaches exist in order to preserve research environments Dudley and Butte (2010) and capturing whole scientific workflows including all software dependencies and additional contextual information of experimentsStrodl et al. (2013); Antunes et al. (2014).

In order to completely understand and preserve scientific experiments, not only the processing steps, protocols, setups and descriptions of the tasks need to be available, also the data itself needs to be accessible and reusable in order to reproduce a scientific result. Data citation aims to make data referencable by providing methods which allow unique identification. Citation in general is a centura old tradition established by the need of identifying and acknowleding the work from peers. Citing articles and publications is a well established process which allows scientists to measure the impact of their work by using biblometrics. Currently there hardly exists a digital analogon for data citations which would provide researchers with credit for their data as well, but recent initiatives aim to alter this behavior and establish recognition for data products as well Callaghan et al. (2012). Sharing data is a popular topic in many disciplines, but still no uniform methods existBorgman (2012). But in many cases sharing data is neither a goal nor a legal possibility. But being able to reliably identify and access datasets is fundamental for reproducing experiments.

Keeping datasets available and acccessible is a challenge. So far datasets are often deposited on a storage system and referenced via URLs which point to the location of the dataset. The well known link rot phenomenon Rumsey (2002) is a threat to the accessibility of datasets as locations of files are prone to changes which causes such links to break. In order to overcome this issue centrally managed persistent identifiers systems Hans-Werner Hilse (2006) utilize redirection to resolve new locations of data files correctly.

As modern research is based on many digital artefacts, the need of addressing data grew within the last decade. Researchers work with various data types and formats, all this fragments are constituting to the final result, thus archiving and storing the data produced in research projects is a pressing issue Gray et al. (2002). Databases are used in many scientific settings and are used for managing large datasets in a reliable way. The query languae SQL can be used for retrieving versioned datasets of arbitrary granularity. In our earlier work we developed a data citation framework Pröll and Rauber (2013a) based on relational database management systems (RDBMS) and demonstrated how it can be applied to existing infrastructuresPröll and Rauber (2013b, 2014). The framework we presented allows to attach persistent identifiers to the queries instead of the exported data set. By applying versioning to the data and timestamped queries the result set valid at the time of the query execution can be retrieved at a later point in time.

3. INTEGRATING SENSITIVE DATA SOURCES (GAP-DRG): REQUIREMENTS

Health data is particularly sensitive to disclosure and therefore requires protection beyond standard security measures. In the scenario we introduce in this work, several independent organizations contribute sensitive patient data into one large database, which integrates the data in a datawarehouse fashion. Each organization obviously has access to the data which it contributed to the system. Additionally each stakeholder may perform defined analysis steps which do not allow the derivation of new knowledge which would not be available without the data from other stakeholders ownership. Hence the system needs to ensure that only those records may be retrieved, which are either owned by one particular party or are not classified as sensitive by other partners. Encrypting all data stored in the database was not an option as this approach would hinder querying the data. Hence an approach is needed which does not rely on encryption.

In general the data can either be retrieved via a SQL interface or via a Web interface translating user interactions into prepared statements. For obvious reaons, SQL interface access may be granted only for a limited set of users. SQL queries must be logged, annotated with and parsed prior to their execution.

The majority of users will only get access via an application interface, which is the preferred way of accessing the data. Although this method might seem restrictive, this mode provides several advantages.

As the interface is standardized, only a pre-defined set of operations on the database is permissible. These operations are predefined and allow the integration of privacy preserving methods in a streaight-forward way.

4. DATA CITATION

It is obvious that the data needs to maintained during the whole data life cycle in order to preserve the information for the long term. Database management systems (DBMS) such as PostgreSQL are used for managing large data sources and maintain the integrity of the data. In order to make the data citable the data schema has to be adapted and the database queries need to be made reproducible.

4.1 Preparing the Data Model

In many scenarios, data is not just static but it can also be highly dynamic. New records can be created, some data may get updated whereas older records may be deleted. For this reason the full history of all operations which either added, altered or deleted any record in the database system needs to be traced. Still versioning of databases alone does not allow retrieving a specific result set from a given point in time from a database. Additionally to the provenanceBuneman et al. (2006) of any record in the database, the actual query which was used for retrieving the dataset needs to be stored with additional metadata.

As most modern RDBMS PostgreSQL does support point in time recovery (PITR) which can be used for rolling back the data to a specific date. Although this method allows querying on top of the data as they were at any given moment in time, the approach is not feasible for retrieving historical data in an convenient fashion as it requires a potentially costly rollback operation to a specific data version which cannot be reused for other queries. It is to mention that the term version may be misleading in the context of data citation. In this paper a version denotes a specific state of the records in a result set, we do not refer to a data dump or export which is assigned a version number or the-like. The same query for instance will produce different versions of result sets, whenever a single record contained in the result was changed between to executions of a query. This being said, the solution does not scale for data citation which requires a more flexible approach. Other modules exist which enable "time-travel" within the data but those are highly vendor specific and not suitable for long term data retention. The support of such features may be ended[1] or a migration to a different system may become necessary.

In our earlier work, we described how a database schema can be adapted for enabling data citationPröll and Rauber (2013a). Depending on the frequency of changes, a different implementation needs to be chosen. In the scenario of this paper, updates occur only when errors are detected and corrected. Inserts occur at larger volumes and in batches. As a result, the data citation metadata can be integrated into the existing tables without the need for a separate history table. Enabling data citation for relational databases can be achieved with the following steps:

(1) Analyze existing tables
(2) Introduce metadata columns
(3) Initialize records
(4) Implement Query Store
(5) Adapt queries

In a first step the existing database schema needs to be analyzed and tables which need to be referencable need to be selected. Currently the GAP-DRG database consists of 47 tables and 8 views which require adaptation for data citation. A primary key is a fundamental requirement for differentiating the records. All but one table have a primary key column available, some tables use composite primary keys. The one table without a unique key requires the creation of an artificial primary key which can be created by using a sequence of numbers. In order to store all versions of the records, each table needs to be equipped with at least one timestamp column and an event column marking the state of each record. All insert, update and delete events are reflected via the state column of each table.

None of the tables in our scenario does have a suitable timestamp available, hence the tables need to be altered and the missing columns need to be added. In a second step we include two timestamp columns for each table in order to trace the valid time of each record: $valid\_from$, *valid_until*. This allows to measure the lifespan of a record state conveniently and allows to detect the most recent version quickly as this record does not have a *valid_until* timestamp set. Each record needs to expand the primary key and include the newly created columns for guaranteeing uniqueness within a table. Additionally metadata about the creator of each record needs to be stored in order to maintain the privacy of the records and prevent unauthorized access. Database indices are added to the timestamp columns and the event type column in order to increase the performance of the system.

In the third step, the existing records in the database need to be initialized, i.e. the timestamp column needs to be populated with an appropriate date. As no information about the insertion history of the data is available, all records will be initialized with the same timestamp. If no metadata about the creation time of the tables is available, additional documentation or system logs may be used in order to determine the appropriate date.

4.2 Adapting the Query Store

In order to persistently store the results of database queries and thus enable citations, a query store needs to be implemented in a fourth step. The concept of the query store is described in Pröll and Rauber (2014). In this work we need to extend the Query Store and adapt it to the requirements of this scenario. This includes a new hashing method based on unique sorting, privacy protection of records and query normalization. The query store records all query parameters with additional metadata such as timestamps, identifiers and information about the executing user. The re-execution of a query also requires to maintain the permission rights of the records as access to a specific portion of the data may be granted or revoked. This information needs to be captured and stored in the query store in order to reproduce who retrieved sensitive data and prove that access to a specific dataset was granted. For this reason the query store also serves as an audit trail and needs to be protected from manipulation. The goal of the query store is to attach persistent identifiers to query results and allow retrieving the same data again by re-executing a query against historical data. A persistent identifier (PID) uniquely identifies a resource for the long term by utilizing a managed infrastructure providing additional services which can be used for accessing the metadata and the object itself in a reliable way. Instead of attaching a PID directly to the exported dataset, the PID references the query which ultimately produces the dataset. The Query Store needs to detect whether a issued query is already stored persistently or if a new query needs to be inserted. In order to validate the re-executed query result for its correctness, a hash key is computed. A result set is only considered correct if and only if all records are included in the same sequence and ordering as in the original query.

It is essential to maintain the sequence of the records in the result set in order to detect if the result was correct. The ORDER BY statement itself does not guarantee stability of the sorting as the sorting sequence may be non-deterministic due to system internals. In order to guarantee a reproducible query result set sorting, the Query Store needs to utilize unique sorting criteria. Hence the Query Store uses the primary keys of the involved tables as sorting criteria in order to bring the result set

[1] Until PostgreSQL 6.3 the system used to have a time travel feature which was removed in later versions

into a reproducible sorting. The primary keys for each table can be retrieved by the database table metadata and thus be applied automatically. This method of primary key sorting is used for detecting inserted, changed or removed records within two executions of a query by calculating a hash key. The Query Store calculates the hash key by concatenating the fully qualified name of each column which is requested for inclusion in the SELECT statement. This allows detecting changes in the sequence of the columns in the result set. For detecting missing or added records in a result set, the Query Store iterates over the primary keys of the involved records computes a hash key from the involved keys. As the sorting is reproducible, the hash key can be calculated in a reliable manner.

Each query gets a persistent identifier assigned which allows referencing a specific dataset by a unique string. This string can be used for identifying and citing the dataset in reports or it can be processed by machines for automated data retrieval. The Query Store needs to resolve this identifier and re-execute the query in order to retrieve the dataset. Whenever a dataset is updated, for instance if a single record was corrected and updated, the result set needs to get a new identifier assigned. For this reason one identical query issued at different times may lead to multiple versions of the data set.

The Query Store also calculates a hash key for the query itself in order to detect duplicates. In order to detect such queries, the Query Store needs to normalize each query in order to detect deviations of queries which do not have an influence of the result set. Details on the normalization step are given in the Section 4.3. Furthermore, the Query Store checks whether the records of a result set have been updated between two executions of a query. Therefore an attribute storing the latest update of a record of the result is stored for each query. If no record has been updated between two executions of a query, the Query Store can immediately issue the result set known from a previous execution and thus can detect identical queries. If there was an update the Query Store assigns a new PID to the query and updates the metadata accordingly.

The Query Store also contains the metadata describing the permission rights and the ownership of the data. A data owner may always execute queries which only contain records where the owner is identical to the executing user. More complex permissions may be granted on query level and are evaluated before the data may be retrieved. As the permission might be revoked or newly granted, this information needs to be versioned as well.

4.3 Query Normalization

SQL is a very flexible language which allows expressing queries delivering the same results in multiple ways. In order to detect duplicate queries and calculate reliable query string hashes, the queries issued against the database system need to be normalized. In order to achieve reproducible SQL query strings, we generate an abstract syntax tree of the SELECT statements and sort the parameters and arrange parenthesis in an appropriate fashion. In order to remain independent from the actual database vendor, we adapted the existing SQL parser engine used by the open source DBMS FoundationDB[2] which can be used for parsing several SQL dialects. This parser can be used for generating abstract syntax trees from SQL statements. We implemented a custom Visitor pattern in order to store the tree in a relational table scheme. The parser library is capable of normalizing SQL queries and sort the resulting syntax tree in a reproducible fashion. This enables the Query Store to detect duplicate queries by calculating a hash of the normalized query strings. We transform all query parameters to lower case letters and remove trailing or leading spaces.

4.4 Citing SQL Result Sets

The fifth step involves adapting the query mechanism for retrieving the correct version of the data. In order to enable and use data citation the system needs to maintain the timestamps and update the state information of each record. Database specific methods such as triggers can be used for hiding the complexity of the Query Store and the data versioning completely from end users. As the system is based upon timestamped and versioned data and utilizes a query based approach, existing database queries used in applications, e.g. in prepared statements in the source code of applications, need to be adapted in order to retrieve the latest version of the data only. This can be achieved by adding the timing and event type columns to query criteria and ensuring that records marked as deleted are not included in the result set.

In order to retrieve a historical version of a dataset, the system needs to resolve the provided PID to the appropriate query. Then the system needs to rewrite the query and append the timing information of the original query execution time as an additional criterion to the original SELECT statement. By limiting the *valid_until* timestamp to the execution time and the event type to *inserted* and *updated*, the system retrieves the appropriate records only. Calculating the hash of the result set provides evidence for the correctness of the historic result set.

4.5 Complex Queries

The system we proposed so far works well with simple SELECT statements, unfortunately reality demands more complex queries. Modern RDBMS support a large variety of operations, aggregations and also randomized operations. Whenever a query produces non-deterministic results, the Query Store may not reproduce the exact same result sets due to the randomness of some operations. User defined functions (UDFs) can produce non-deterministic results as well. Therefore a manual step is needed where the supported UDFs need to be analysed and checked for non-deterministic operations. A user defined function can be considered deterministic if it produces the exact same result for the same input parameters[3]. Non-deterministic functions can not be guaranteed to be citable unless the function can be converted into a deterministic mockup. The cause of the non-determinism needs to be replaced in the mockup function by a deterministic counterpart, producing the same results as for the original query. Obviously this is a manual step which may not always be

[2] https://github.com/FoundationDB/sql-parser
[3] http://dev.mysql.com/doc/refman/5.1/en/create-procedure.html

possible to achieve. Also operations which are random or non-deterministic by definition, such as relative time specifications, seed values and all operations depending on external input beyond the control of the RDBMS can not be reproduced. The Query Store marks non-deterministic queries as not reproducible and provides this fact as metadata for the consideration of the user.

5. CONCLUSIONS

It is important to stress that due to the availability of all versions and their timing information, the solution we proposed in this paper is not depending on PostgreSQL or any other database vendor. Although more performant solutions may be provided by RDBMS internal tools, our focus was on providing a flexible solution indepenend from specific software solutions.

REFERENCES

Antunes, G., Bakhshandeh, M., Mayer, R., Borbinha, J., and Caetano, A. (2014). Using ontologies for enterprise architecture integration and analysis. *Complex Systems Informatics and Modeling Quarterly.*

Borgman, C.L. (2012). The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, 63(6), 1059–1078. doi:10.1002/asi.22634. URL `http://dx.doi.org/10.1002/asi.22634`.

Buneman, P., Chapman, A., and Cheney, J. (2006). Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, 539–550. ACM, New York, NY, USA. doi:10.1145/1142473.1142534. URL `http://doi.acm.org/10.1145/1142473.1142534`.

Callaghan, S., Donegan, S., Pepler, S., Thorley, M., Cunningham, N., Kirsch, P., Ault, L., Bell, P., Bowie, R., Leadbetter, A., et al. (2012). Making data a first class scientific output: Data citation and publication by nercâ s environmental data centres. *International Journal of Digital Curation.*

Dudley, J.T. and Butte, A.J. (2010). Reproducible in silico research in the era of cloud computing. *Nature biotechnology*, 28(11), 1181.

Gray, J., Szalay, A.S., Thakar, A.R., Stoughton, C., and Vandenberg, J. (2002). Online scientific data curation, publication, and archiving. In *SPIE Astronomy Telescopes and Instruments*, MSR-TR-2002-74, 6. Waikoloa, Hawaii. URL `http://research.microsoft.com/apps/pubs/default.aspx?id=64568`.

Hans-Werner Hilse, J.K. (2006). *Implementing Persistent Identifiers: Overview of concepts, guidelines and recommendations.* Consortium of European Research Libraries, London. URL `http://www.cerl.org/publications/report_on_persistent_identifiers`.

Hey, T., Tansley, S., and Tolle, K. (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research.

Loscalzo, J. (2012). Irreproducible experimental results causes,(mis) interpretations, and consequences. *Circulation*, 125(10), 1211–1214.

Mayer, R., Strodl, S., and Rauber, A. (2012). On the complexity of process preservation: A case study on anescience experiment. In *Proceedings of the 9th International Conference on DigitalPreservation (iPres 2012).*

Mesirov, J.P. (2010). Computer science. accessible reproducible research. *Science (New York, NY)*, 327(5964).

Pröll, S. and Rauber, A. (2013a). Citable by Design - A Model for Making Data in Dynamic Environments Citable. In *2nd International Conference on Data Management Technologies and Applications (DATA2013).* Reykjavik, Iceland.

Pröll, S. and Rauber, A. (2013b). Data Citation in Dynamic, Large Databases: Model and Reference Implementation. In *IEEE International Conference on Big Data 2013 (IEEE BigData 2013).* Santa Clara, CA, USA.

Pröll, S. and Rauber, A. (2014). A Scalable Framework for Dynamic Data Citation of Arbitrary Structured Data. In *3rd International Conference on Data Management Technologies and Applications (DATA2014).* Vienna, Austria.

Rumsey, M. (2002). Runaway train: Problems of permancence, accessibility, and stability in the use of web sources in law review citations. *Law Libr. J.*, 94, 27.

Sandve, G.K., Nekrutenko, A., Taylor, J., and Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLoS computational biology*, 9(10), e1003285.

Schwab, M., Karrenbach, M., and Claerbout, J. (2000). Making scientific computations reproducible. *Computing in Science & Engineering*, 2(6), 61–67.

Strodl, S., Mayer, R., Draws, D., Rauber, A., and Antunes, G. (2013). Digital preservation of a process and its application to e-science experiments. In *Proceedings of the 10th International Conference on Preservation of Digital Objects (IPRES 2013).*