Agile Research Data Management with Open Source: CaosDB

Daniel Hornung 1
Florian Spreckelsen 1
Thomas Weiß 1

1. IndiScale GmbH, Göttingen.

Abstract.
Research data management (RDM) in academic scientific environments increasingly enters the focus as an important part of good scientific practice and as a topic with big potentials for saving time and money. Nevertheless, there is a shortage of appropriate tools, which fulfill the specific requirements in scientific research. We identified where the requirements in science deviate from other fields and proposed a list of features which RDM software should fulfill to become a viable option.
Finally we analyzed the open-source RDMS CaosDB for compatibility with the proposed features and found that it fulfills the requirements.

1 Introduction

Research units, from small research groups at universities to large research and development departments are increasingly confronted with the challenge to manage large amounts of data, data of high complexity[1], [2] and changing data structures[3], [4]. The necessary tasks for research data management include storage, findability and long-term accessibility for new generations of researchers and for new research questions[4]–[6].

In spite of the advantages of implementing data management solutions[7], there is a lack of standard methods or even standard software so far for research data management, especially in the context of quickly evolving methods and research targets. In this article, we define the specific challenges for research data management (RDM) and propose features which suitable RDM software should have to (a) fulfill the practical needs and (b) be accepted by the potential users. We also demonstrate how the CaosDB[8] toolkit is a viable approach to satisfy all the proposed requirements.

1. Website: https://caosdb.org, source code: https://gitlab.com/caosdb/
2 Challenges for research data management

2.1 The scientific data lifecycle

Data which accrues in scientific research is more than just experimental readings, field notes or interview recordings. In order to fully represent the research journey and eventually enable reproducible science, the data from every research step may become relevant and thus should be consistently managed.

Figure 1 shows a schematic of different research steps during the research lifecycle, during which important data is generated. For full reproducibility, it is not sufficient however to simply store the data acquired at each point, but also to represent the semantic connections and make them searchable.

In more detail, the most relevant data source in scientific research are:

Prior publications An important part of good scientific practice (GSP) is to credit the influence of prior work, by the scientists themselves or third parties. Linking one’s work to previous publications and making these connections public also helps to assess reproducibility and may lead to fruitful data-reuse in unforeseen contexts.

Ideas and SOPs The data here consists mostly of text documents which describe thoughts, hypotheses and planned standard operating procedures (SOPs). These documents fill the gap between previous work and the next round of data acquisitions, they often also work as a blueprint for the data acquisition phase.

Lab data Environmental data, device settings, used SOPs and ingredients and other incidental data typically accrues during the course of experiments and was traditionally stored in paper laboratory notebooks. Currently, a lot of laboratories switch to electronic lab notebooks (ELNs) for the same purpose. While this data is often seen as second-class “metadata”,

we hold that since often conclusions can be drawn from it, it deserves the same handling as final instrument readings.

**Experimental results** These are what is often considered the main data. For analysis, it mostly still needs parameters from at least the previous step, to filter for special conditions, to compare settings or to verify that values are compatible with standard literature.

**Numerical simulations** Similarly to experimental results, data obtained from numerical methods can not be interpreted without knowledge about used software and parameters, possibly hardware conditions and input from laboratories or third-party data sources. Since bit-exact reproducibility is possible in theory, all relevant settings should be stored unchanged.

**Data analysis** When analyzing data from previous steps, storing not only the used programs, scripts and their parameters, but also the semantic connections enables later researchers to reconstruct which method was used, which assumptions were made and under which conditions the input data was gathered.

**Next publication** Formally the end of the lifecycle, but of course also the beginning of many new ones, a publication contains a number of statements which are supported by data from previous steps. A comprehensive research RDM system (RDMS) could quickly summarize, for each plot in a publication, how the data was analyzed and acquired and under which assumptions the experimental setting was planned.

This list focuses on experimental and laboratory centered disciplines, but of course in the humanities and theoretical sciences, there are equivalent steps which are equally important to preserve and link to each other.

### 2.2 Specifics of scientific research data management

Existing data management systems (DMS) often still follow the paradigms of relational databases[9], [10]: There is a number of types for the data, with each type having a number of possible properties. Each stored entity belongs to one type and has the properties defined by the type, occasionally it is also possible to leave a property empty or undefined. Searching or filtering in the stored data entities is possible by criteria which operate on the content of the properties and on the data types. These DMS have been widely successful in many areas such as finance, administration and high-tech industries[11], [12], but remain scarce in both academic and private sector research[12], [13].

We hold that the main reasons are:

**Interoperability** Scientists tend to work with their own custom-written software, which often requires files with data to be directly accessible to the OS via a – remote or local – file system. Many DMS store data files internally and make them available only via download. Also programmatic access (query, retrieve, update) to data DMS solutions via network APIs was uncommon until a few years ago, so that acquired data could not be instantly integrated in the DMS.

**Agility** Traditional DMS require users to define a data model and stick to it. All data to be entered has to conform to the data model as it was defined. Research however is defined
by having undefined outcomes, the research questions, experimental setup or analysis
methods change more often than not over the course of one investigation. A DMS based
on rigid SQL databases thus may soon become a liability instead of an asset.

Learning curve  Research is impossible without the contribution of many participants, with
different qualifications. With many DMS, it takes a lot of active learning before users can
benefit from the provided data.

Practical use  Systems which only store data, but do not provide short-term advantages, have
high entrance barriers everywhere. In academic research however, junior scientists with
short-term contracts have little incentive to invest time and money in systems which only
may pay out on longer timescales.

3 Proposed features

Under the assumptions from the previous sections, we propose the following core properties
which an RDMS should have:

Semantic linkage  To fully map the real-world environment of data entities to the RDMS, it
must be possible to link data sets with each other in a meaningful way. The default linking
possibilities and properties of the data types in the RDMS form the data model.

Deep search  The RDMS should have easily accessible search options not only for property
values of stored entities, but also for links to other entities and properties (and link) thereof.

Flexible data model  Researchers require an RDMS where the data model can be changed on
the fly, without the need to migrate or discard existing data. When the data model is
changed, for example due to new machines, protocols or evolving research questions, the
existing data must remain valid and usable.

Versioning  Mistakes during data acquisition happen, and it must be possible to correct existing
data sets. At the same time, this editing must be made transparent and the history of each
data set must be kept for future inspection.

File system integration  For interaction with third-party programs, raw data files must be avail-
able on standard file systems. Ideally the scientists’ workflows should remain unchanged
by the RDMS.

Open APIs  For machine interaction with third-party programs, the RDM must have a well-
documented API. In research contexts, these programs are often custom-written by scien-
tists without explicit computer science background, so extensive documentation is very
desirable.

3.1 User acceptance

Additionally, to increase the acceptance by potential users, we focus on two additional areas,
automated data integration and low-threshold search options.
### 3.1.1 Automation

Automation of repetitive data integration reduces error rates and frees users to concentrate on more challenging tasks. It is therefore desirable for an RDMS to have:

**Synchronization** The RDMS should make it easy for its administrators to integrate existing data sources (for example databases or file systems with structured folder hierarchies) into the RDMS: The RDMS should be synchronized automatically with data from these sources, which makes these data available in a unified manner via the RDMS interface. Note that the RDMS can not solve the conceptual problem of a single source of truth when synchronizing data from different sources, but it can at least highlight potential conflicts and where they first occurred to administrators.

**ELN integration** Research work in the lab is increasingly documented with electronic lab notebooks (ELNs)\[14], [15], which allow to conveniently enter device and experimental settings in a semi-structured way. This data is usually critical in the analysis of acquired raw data from instruments, e.g. for searching specific data sets or filtering by parameters. There should be a possibility that the RDMS integrates the ELN data and presents it like data from other sources.

**Workflow representation** While following one SOP, the laboratory workflow is often highly standardized, which makes it suitable for representation within the RDMS. The RDMS should support workflows with different states, which can only be switched in an admin-defined pattern. This simplifies the work for users, because they may e.g. only see the interfaces which are relevant for the current sample processing step.

### 3.1.2 Simplified search

To overcome initial reluctance by users, it is important to flatten the learning curve\[16]. Besides obvious requirements such as user-friendly documentation, we hold that it is especially important to provide simplified search options. The simplified search in an RDMS should give some early sense of achievement, so users can understand that an RDMS will lead to a simplification of their work.

### 4 CaosDB

We hold that CaosDB\[8], a flexible data management framework, fulfills the above-mentioned requirements. CaosDB was initially developed by one of our colleagues, Timm Fitschen, during his time at the Max Planck Institute for Dynamics and Self-Organization. In 2018, CaosDB was released under the AGPLv3 license on gitlab.com\[2]. Since 2020, CaosDB has found increased adoption in multiple research facilities.

#### 4.1 Data Model

CaosDB’s meta data model is shown schematically in Figure 2. The base type for everything is ENTITY, with the inheriting types PROPERTY (attributes of ENTITIES, may be list values and

\[2. \text{https://gitlab.com/caosdb} \]
Figure 2: The meta data model of CaosDB.

references to other EN TITIES), RECORDTYPE (templates for actual data sets) and RECORD. Actual data is typically stored in RECORDS, which inherit from one or more RECORDTYPES and thus have all the PROPERTIES defined therein. The RECORDTYPES may form a complex inheritance hierarchy themselves. FILE entities are similar to Records, but additionally are connected to files which may reside on conventional file systems or potentially in abstracted cloud storages. This approach to use files at their current locations instead of duplicating file content not only increases CaosDB’s scalability, but also lower the entrance barrier, since scientists can access the managed file in their traditional ways.

Details of this meta data model in CaosDB are elaborated on in [8], but it should be clear now already that CaosDB provides the Semantic linkage feature.

In CaosDB, the data model of the stored data refers to the RECORDTYPES and their PROPERTIES, which together describe the pattern to which newly created data sets should conform. The data model in CaosDB can be modified at any time, but the changes only take effect for data to be inserted after this modification. Existing data is not affected and remains unchanged. This property fulfills the proposed Flexible data model feature.

This possibility to completely change the data model, while not giving up on a general structure, places CaosDB between traditional SQL based relational databases and NoSQL\(^3\) approaches (c.f. Figure 3). While we described above why rigid SQL databases are not suited for use in dynamic research environments, giving no structure (the NoSQL paradigm) tends to lead to incoherent data which is hard to search\(^4\). A third approach, using graph databases to represent semantic information, has not found its way into general adoption to our knowledge, presumably because the query languages tend to become very unwieldy, compare the appendix 7 for an example.

4.2 Architecture and Libraries

CaosDB uses a client/server based architecture, as depicted in Figure 4a. CaosDB has is a REST API for simple access by traditional clients and a web interface for browsers, as well as a gRPC API which allows for more complex operations, such as atomic content manipulations. The existing client libraries\(^5\) and the open APIs provide the proposed Interoperability requirement.

One particularly useful client library component is the CaosDB Crawler framework. This extensible framework simplifies the work to synchronize external data sources with CaosDB

\(^3\) Popular examples for NoSQL database systems are CouchDB or MongoDB.

\(^4\) Missing structure in Data Lakes has lead to the tongue-in-cheek colloquialism “Data Swamp”.

\(^5\) A list of the available libraries with the respective source code repositories are given in the Appendix section 6.
CaosDB compared to other database approaches.

Figure 3: CaosDB compared to other database approaches.

Figure 4: (a) CaosDB’s server-client architecture with client libraries and backend components. Dotted elements are under development. (b) The crawler framework facilitates fast development of custom data integration from a diversity of sources.

The crawler workflow can be characterized as follows:

1. The crawler periodically checks its data sources for new or changed data stores, such as file systems or the content of other databases.

2. Each new data source is fed to a so-called CFood plugin for consumption. There is a choice of existing plugins, or administrators can write their own. The CFood plugin’s job is to build CaosDB entities from the consumed data and to specify Identifiables, which work as search patterns.

3. The crawler checks for each Identifiable if a corresponding entity exists already in CaosDB. If there is no corresponding entity, the entity as returned by the CFood plugin is inserted into CaosDB. If there is already an existing entity, the Crawler will attempt to merge the existing with the new entity and notify the data curators in case of merge conflicts.

This tool set provides the Synchronization requirement, and if ELNs are used as external data source, the ELN integration. Practical use of CaosDB crawler framework has previously been
demonstrated in [17] and ELN integration was implemented in [18].

4.3 Additional features

Deep search  CaosDB offers a simple semantic query language, which borrows some semantics from SQL, but has a focus on usability for non-technical users. The CaosDB query language makes deep search easy with expressions like the following:

\[
\text{FIND Analysis WITH quality_factor > 0.5 AND WITH Sample WITH weight < 80g}
\]

This convenient nesting of query expressions circumvents the JOIN operations from traditional SQL languages. A full documentation of CaosDB’s query language is available online\(^6\) or in CaosDB’s sources.

Search templates  CaosDB’s web interface provides customizable search templates which allow more advanced users to create their own templated queries, which can then be shared with novice users for simplified searches. In templated queries, users can insert custom strings into pre-defined locations of a search query, see Figure 5.

Versioning  When entities are modified in CaosDB, time and user of the change are recorded and CaosDB puts the previous version onto a history stack and amends the current version with link to the previous version. Over time, each entity may thus grow to a tree of linked versions, which can be retrieved via the web UI or programmatically through the APIs. This feature of CaosDB enables scientific research data management users to adhere to the principles of good scientific practice.

State management  In CaosDB, users may declare a state machine of states and allowed transitions. Users may then affix states to entities, and these states can then only be changed according to the rules of the state machine. In this way, users can implement a workflow representation which ensure that for example laboratory samples run through a specified list of preparation steps in order.

![Figure 5: A query template in CaosDB’s web UI. The user can enter a custom value into an input field and the template is then executed as a plain CaosDB query. Screenshot from https://demo.indiscale.com.](https://docs.indiscale.com/caosdb-server/CaosDB-Query-Language.html)
4.4 Availability and documentation

CaosDB is available on the public Git repository gitlab.com at https://gitlab.com/caosdb, a detailed list of CaosDB’s sub projects is given in the annex.

For the interested public, there is a live demo server at https://demo.indiscale.com, hosted by IndiScale GmbH. This demo server is actually running LinkAhead, a commercially supported distribution of CaosDB.

There are also Debian/Ubuntu packages to run precompiled LinkAhead/CaosDB for download at https://indiscale.com/download.

CaosDB’s sub projects each have their own documentation in their source directories. The documentation is also available online at https://docs.indiscale.com.

5 Conclusion

We found that scientific research has specific requirements to data management: Interoperability, agility, adequate learning curves and early practical use. Most data management system do not satisfy these requirements, so we set up a specific list of features which a scientific RDMS should have. The open source research data management framework CaosDB has the specified features and thus fulfills the requirements for a scientific RDMS.

Special attention has to be given to the aspect of synchronizing data from external sources (e.g., crawled files, ELNs) and records in the RDMS. Different sources can (usually by some error) have conflicting data, or entries in the RDMS can be changed manually by users after their insertion. In our experience, this problem can not be solved in a general and purely technical way. Instead, best practices have to be implemented as to where possible errors should be corrected and whether some sources have precedence above each other. A RDMS like CaosDB, together with the crawler framework, can help administrators identify inconsistencies in the case of two or more data sources. Through versioning, it is visible who and when maybe changed data manually. How to optimize the help in recognizing potential conflicts, and in the end curate data both in the RDMS and in the external sources, is subject of the authors’ ongoing research.

We hope that the open source license of CaosDB will inspire scientists to contribute to CaosDB.

6 Appendix: List of CaosDB libraries

The following libraries for programming client applications are publicly available:

Python https://gitlab.com/caosdb/caosdb-pylib The Python client library can be used for third-party applications and is the foundation for several other libraries:

Advanced Python tools https://gitlab.com/caosdb/caosdb-advanced-user-tools Additional high-level tools, including a legacy implementation of the CaosDB crawler.

**7 Appendix: Query language comparison**

As an example for nested queries in different query languages, we consider the search for female UK-based writers in a certain time period, whose given or family name starts with the letter M. We used the RDF query language SPARQL with Wikidata identifiers and CaosDB’s query language with fictional but plausible identifier names.

The SPARQL query is as follows:

```
SELECT DISTINCT ?item ?itemLabel ?givenName ?familyName
WHERE {
  ?item wdt:P31 wd:Q5; # Any instance of a human.
  wdt:P27 wd:Q145; # United Kingdom
  wdt:P21 wd:Q6581072; # female
  wdt:P106 wd:Q36180; # writer
  wdt:P569 ?birthday;
  wdt:P570 ?diedon;
  wdt:P734 [rdfs:label ?familyName];
  wdt:P735 [rdfs:label ?givenName].
FILTER(?birthday > "1870-01-01"^^xsd:dateTime
&& ?diedon < "1950-01-01"^^xsd:dateTime)
FILTER(regex(?givenName, "M.*") || regex(?familyName, "M.*"))
SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
}
```

In contrast, the CaosDB query looks like this:

```
SELECT given_name, family_name FROM Writer
WITH gender=f AND country=UK AND birthday > 1870 AND death < 1950
AND (given_name LIKE "M*" OR family_name LIKE "M*")
```

7. https://www.wikidata.org
8 Acknowledgements

We acknowledge the previous work on the CaosDB software by its main authors and independent contributors, especially Timm Fitschen.

9 Conflicts of interest

The authors work for IndiScale GmbH, which provides commercial support and other services for CaosDB and the derived free and open-source LinkAhead distribution. DH and FS contributed to the development of CaosDB.

10 Roles and contributions

Daniel Hornung: Conceptualization, Visualization, Writing – original draft

Florian Spreckelsen: Conceptualization, Writing – review & editing

Thomas Weiß: Conceptualization, Visualization

References


