

FAIR DATA AUSTRIA

Requirement Specification

Work Package 2

Cornelia MICHLITS¹ and Martin WEISE¹

¹Institute for Information Systems Engineering, Vienna University of Technology

1 Introduction

1.1 Scope of this Document

This document is used to derive the *functional specification* of the FAIR Data Austria Database Repository work package.

1.2 Overview

FAIR Data Austria is a cooperation project with Graz University of Technology, University of Vienna and Vienna University of Technology and aims in providing a infrastructure for knowledge transfer between universities, business and society.

The Vienna University of Technology is responsible for the implementation of a prototype for a database repository (structured data, SQL). A main focus is on the inclusion of the FAIR principles, i.e. research data should be findable, accessible, interoperable and reusable. The observance of these principles is necessary to use the recorded data collectively.

1.3 The Importance of a Database Repository

At the moment databases usually set up and used locally at each research unit. This local usage requires database administration skills and does not motivate to collect metadata, which is necessary to make data machine-readable, understandable and reusable. Moreover, data versioning is hardly used due to a lack of resources. The consequence is that reproducibility cannot be guaranteed.

In order to overcome these issues a private cloud hosted database repository is suggested. The database administration is outsourced to the repository and supported by experts in this field. Therefore, no local administrators are required. The generated Metadata makes the databases searchable and shareable. The uploaded data gets versioned and timestamped. Hence, reproducibility is guaranteed and data is cite-able at fine granularity.

1.4 Timeframe

January 2020 until December 2022 (3 years)

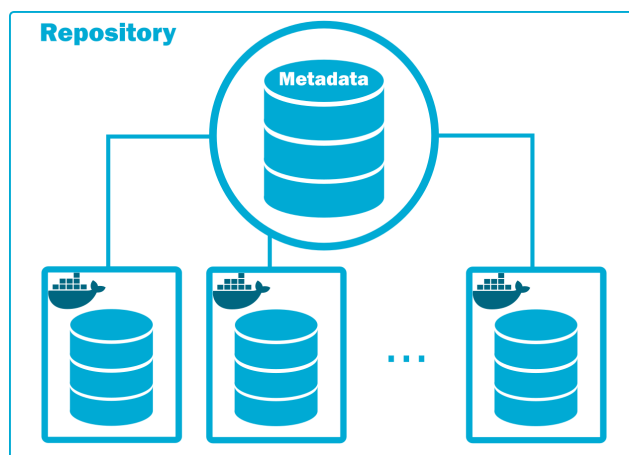


Figure 1: Schematic overview of the repository. Each database is encapsulated in a docker container. The meta-database contains the meta-information of the databases contained in the repository.

2 Overall project description

This project [1] contributes to the strengthening of the knowledge transfer between universities, business and society and supports the sustainable implementation of the European Open Science Cloud (EOSC). The implementation of the FAIR principles (“findable”, “accessible”, “interoperable” and “reusable”) plays a major role. Compliance is ensured by (1) integrated research data management (FDM), which is tailored to the discipline-specific and generic needs of the research groups, (2) by setting up and developing next-generation repositories for research data, code and other research outcomes and (3) by the Development of training and support services for efficient research data management. In this way, "FAIR Data Austria" forms complementary modules in the FDM area for the “Austrian DataLab and Services” and “RIS Synergy” projects.

For efficient research data management in accordance with the FAIR principles, it is essential to support the entire life cycle of research data - from generation to archiving - with specialist knowledge and the associated tools. This cannot be done in isolation. The project promotes collaboration between Austrian universities in developing coherent and robust research data services. This is how Austrian universities secure their role in the international research landscape.

3 The Repository

The repository should consist of the databases created by database users. Each database should be encapsulated in a docker container. This makes the system flexible and scalable. A major roles plays the so called *meta-database*, which should contain the metadata like table names, column names, SI units etc. of the databases running in the repository. The meta-database makes databases findable. Different levels of SQL-knowledge should be supported, e.g. forms to create tables, faceted-browsing to find databases and query builder for those who are not able to write SQL.

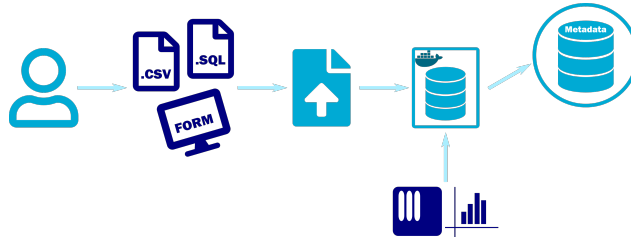


Figure 2: Database creation: There should be three possibilities to create a database. Either upload a csv file(s), support an SQL script or use a form. In the background a docker container with the database gets created. By default Postgres is used as database engine. The meta-information, e.g. database name, is written to the meta-database. Data can be uploaded automatically by a sensor via REST Interface.

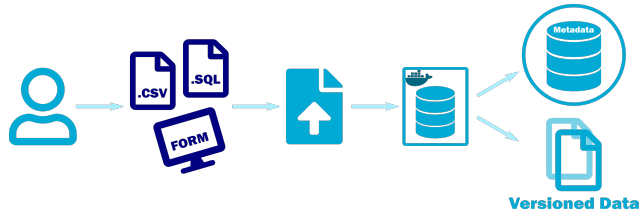


Figure 3: Update Database: Similar to the creation there are the possibilities of updating the database via csv, sql upload or by using a form provided in the UI. The system access the corresponding docker container updates and versions the data. Changes are reported to the meta-database.

3.0.1 Background Processes

A schematic overview of the most import processes (namely database creation, data uploads and searching databases) that should be implemented can be found in Figure 2, 3 and 4.

4 Interfaces

In the pre-meeting of 25.01. we discussed with colleagues of University Vienna to have RESTful interfaces.

4.1 Content Endpoint

4.1.1 Metadata Service

GET	/meta	“list all metadata resources”
POST	/meta	“create metadata resource”
GET	/meta/:id	“get metadata resource info”
PUT	/meta/:id	“update metadata resource”
DELETE	/meta/:id	“delete metadata resource”

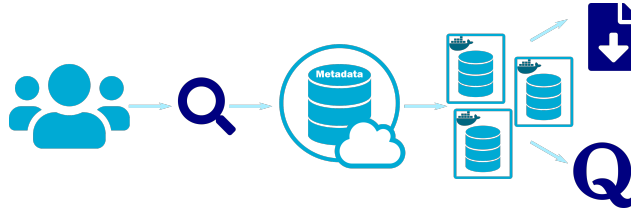


Figure 4: Find database: Using the web-interface the meta-database can be queried via SQL or faceted-browsing in order to find certain databases. For example, searching all databases containing personal data. Depending on the access rights the database can be queried or downloaded.

4.2 Container Endpoint

4.2.1 Container Managing Service

GET	/container	“view container resources”
POST	/container	“create container resource”
GET	/container/:id	“get container resource info”
PUT	/container/:id	“update container resource (e.g. "action":"start")”
DELETE	/container/:id	“delete container resource”

4.3 Database Endpoint

4.3.1 Database Managing Service

GET	/database	“list the created databases”
POST	/database	“creates a new database IN:databasename, DB engine, Owner, Creator, Publisher, PublicationYear, ResourceType, description, Separator OUT:Status, SQL for metadatabase”
GET	/database/:id	“get info about database including connection”
PUT	/database/:id	“update database resource”
DELETE	/database/:id	“deletes the corresponding database”

4.3.2 Table Service

GET	/database/:id/table	“view table resources”
POST	/database/:id/table	“create table resource”
GET	/database/:id/table/:id	“get table info”
PUT	/database/:id/table/:id	“update table resource (Andreas: this is not trivial!!)”
DELETE	/database/:id/table/:id	“delete table resource”

4.3.3 Query Service

POST	/database/:id/query	“store query (normalization > hash key computation)”
PUT	/database/:id/query	“execute query”
PUT	/database/:id/query/version/:timestamp	“execute query against timestamp”
PUT	/database/:id/query/version/:timestamp	“execute query against timestamp”

4.3.4 Data Manipulation Service

IN: dbid, tablename, values, boolean (for update) OUT: status

POST	/database/row?csv	"generate SQL statement from csv"
POST	/database/row?json	"generate SQL statement from json"
POST	/database/row?dump	"generate SQL statement from dump"
DELETE	/database/row/:id	"delete row resource"

4.3.5 Analyze Service

PUT	/database/analyze/datatypes	"determine data types"
PUT	/database/analyze/sql	"analyze SQL"
PUT	/database/analyze/ontology	"map ontology (keep original and mapped)"

4.3.6 Content Service

GET	/database/:id/download	"make temporary tarball and get download link"
-----	------------------------	--

4.3.7 DOI Service

GET	/doi/:id	"resolve DOI (not trivial, may include query re-execution, provide landing page)"
-----	----------	---

5 User Stories

This section covers the user stories, which are enumerated for better referencing in sprints and Gitlab.

5.1 Create database

- 5.1.1 "As a researcher without SQL knowledge I would like to create a database simply by using a graphical interface or by providing a CSV/Excel file / DB-dump. I am able to simply upload e.g. the CSV files and the system automatically creates a relational database that can be queried."
- 5.1.2 "As a database creator with a solid SQL background I would like to use a SQL interface in order to type in SQL commands for database creation, share my research findings and be connected to other researchers in that area."
- 5.1.3 "As a database creator I want to choose a certain RDBMS engine (MySQL, Postgres, MariaDB) for my database that provides the desired features I need." (low priority)
- 5.1.4 "As an unskilled SQL user I would like to have a supporting UI interface that helps me create tables."

5.2 Feed database

- 5.2.1 "As an unauthorized person or machine I am not able to access the database or insert data. So, the database owner keeps control of the added records."
- 5.2.2 "As an authorized person without SQL knowledge I want to use the graphical interface in order to store data or alternatively automatically by supplying a DB-dump, CSV/Excel file, i.e. upload the files to the system. The software makes me aware if the data is not schema conform. "
- 5.2.3 "As a researcher I want automate data insertion via a REST interface, e.g. using a sensor that send measurement data via internet. "
- 5.2.4 "As an authorized person with SQL skills I can simply insert the data in a SQL interface."
- 5.2.5 "As a database owner I would like to set an embargo period in order to have the sole access to the data for a given time."
- 5.2.6 "As a researcher I want to add data description and rich meta data such that other peoples can find my database."

5.3 Update database

- 5.3.1 "As an authorized person I can update or delete (also physically) data by either using an SQL interface, graphical interface, uploading CSV/Excel/ JSON files."
- 5.3.2 "As a machine I can update the database after an authorization step via REST interface."

5.4 Query database

- 5.4.1 "As a database user, I want to create queries via SQL interface directly so I can benefit of the flexibility of SQL."
- 5.4.2 "As a database user, I want to create queries via faceted browsing so I can set filters without having any skills in SQL."
- 5.4.3 "As a database user, I want to store my queries and respective metadata in a query store so I can reproduce my result sets by using the corresponding PID."
- 5.4.4 "As a database user, I want to obtain the result sets of executed queries as CSV/Excel file / JSON format so I can easily store and provide it."

5.5 Find database

- 5.5.1 "As a researcher I would like to find other for me relevant databases in the repository. Via a web interface I can easily access the meta data of the database owners and a description of the data set. The open meta data can be queried by using faceted browsing such that no SQL skills are necessary. For example, I am able to search by data owners and his meta data or by meta data of the database (column names, units, min/max/avg or range of values) and by data openness (open access, embargo period,)."

- 5.5.2 "As a database user I am able to find and access a specific version of data to run queries in a reproducible way and download the data for further use (e.g. statistic, ml, ...) as described below."

5.6 Download data

- 5.6.1 "As a researcher I want to download the database as ZIP consisting of CSV / JSON files and single tables as CSV / JSON files or alternatively as a database dump."

5.7 Manage database

- 5.7.1 "As an administrator I want to have an overview of the different numbers of Postgres, MySQL, MariaDB versions and some statistics (dashboard)."
- 5.7.2 "As an administrator I would like to reset passwords, lock users and grant access to certain database."
- 5.7.3 "As an administrator I am aware of the access rights of different users."
- 5.7.4 "As a system administrator, I want to manage Docker containers so I can maintain them."
- 5.7.5 "As a system administrator, I want to upgrade database engines in specific Docker containers so I can keep the functionalities of the repository up to date and provide new features of the engines."

5.8 Monitoring

- 5.8.1 "As a system administrator, I want to analyse and monitor logs in any component of the application so I can detect misbehavior."
- 5.8.2 "As a system administrator, I want to monitor containers so I can check the status and behaviour."

5.9 Merge databases

- 5.9.1 "As a researcher I would like to merge semantically similar databases, e.g. heart diseases in Austria and heart diseases in Germany to one database that can be queried."

5.10 Extensions

- 5.10.1 "As a database owner I would like to modify the schema in hindsight to make refinements and adjustments."
- 5.10.2 "As a researcher I am able to query the meta database in order to find relevant databases."
- 5.10.3 "As a researcher, I want to store my queries regarding the meta database in a query store so I can reproduce my result sets by using the corresponding PID."
- 5.10.4 "As a database owner I want to have the possibility to upload sensitive data that gets encrypted and still can be queried using SQL commands or graphical interface. So, in case of a data leak the records are useless."

6 Roles

A detailed description of their scope can be found in Table 1, in the following we give a short description of the roles:

System Administrator is responsible for supporting the infrastructure of the repository. He or she is responsible for grant- or release locks, viewing the dashboard, overview data versions, physically delete staff, block databases, set databases invisible or setting databases inaccessible. He or she should not be able to overwrite data in the database, including deletion;

Database Owner is the owner of a specific database. He or she creates one or multiple databases, releases data, block data, embargo period and access meta data;

Database User typically queries data (SQL, faceted browsing, landing page) and query data time series (reproducible);

Database Provider is considered to be a machine (e.g. a sensor measuring water pollution) automatically uploads data via REST interface as CSV or JSON.

Privileges	System Admin	Database Owner	Database Provider	Database User
SELECT	✓	✓	✓	✓
INSERT	×	✓	✓	×
UPDATE	×	✓	×	×
DELETE	×	✓	×	×
TRUNCATE	×	✓	×	×
REFERENCES	✓	✓	×	✓
TRIGGER	✓	✓	×	×
CREATE	✓	✓	×	✓
CONNECT	✓	✓	✓	✓
TEMPORARY	✓	✓	×	✓
EXECUTE	✓	✓	✓	✓
USAGE	✓	✓	×	✓

Table 1: Roles with use cases and permissions

7 Metadatabase

The meta-database should contain (all) the information required to make data understandable, findable and machine readable (SI units). At least it should support information of the data and the databases included in the repository, i.e. database, table and column information. Moreover statics of user access, access rights and information about the data itself (File encoding, type, version, provenance). Moreover, for each database there should be a contact person that can be reached to obtain further information or access (if the database is private or there is an embargo-period). A general (and open) problem is which information we receive about the user (and TISS). An ER-Schema can be found in Figure 5.

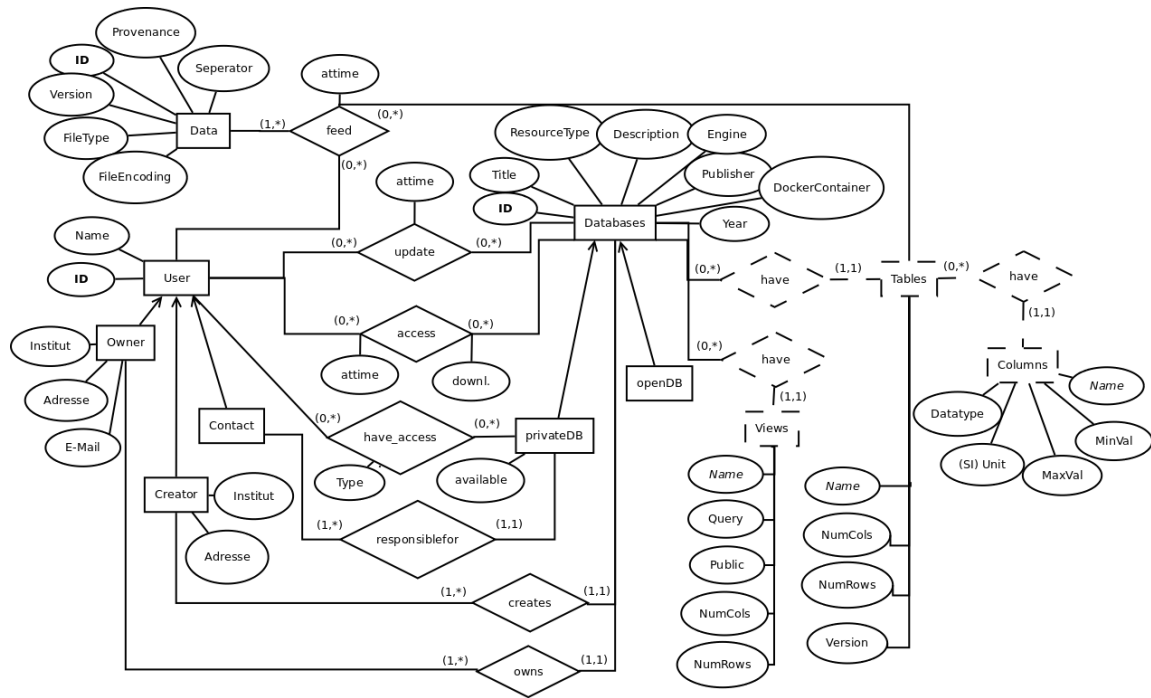


Figure 5: ER-Schema of the meta-database.

8 Microservice Architecture

createDB

generate URL, create Docker Container, start Docker Container, create Database, save URL, DBID, DB engine

IN: Databasename, DB engine, Owner, Creator, Publisher, PublicationYear, Resource type, description, Separator

OUT: status

determine_datatypes

determine columns of csv file, determine datatypes of csv file, ok user

IN: csv, delimiter, header (boolean) OUT: datatype for each column (json) Endpoint: /determine: determines table datatypes of given csv content

create_table

creates SQL statement for tables

IN: table name, column name, data types OUT: SQL statements for table creation

Endpoint: /create: creates sql statements for table creation

insert_update

inserts tuple or update, in case of update comparing if tuple already exists with earlier version (using ids)

IN: table name, values, boolean for update OUT: SQL statement for inserting or updating
Endpoint: /insert: inserts/updates values to corresponding table

execute_query

executes SQL queries either on a database or on the meta database, (check access rights and initial view - later if user clicks on found database?)

IN: Query, UserID, DatabaseID OUT: result of applied query (considering rights resp. view of user)

Endpoint: /execute: executes SQL query on corresponding db and returns the result

analyzeSQL

analyzes SQL statements in order to find out relevant information for meta database and produces SQL statements (inserts into meta database)

IN: SQL statement (this method searches for databasename, created tables, views in an SQL dump or SQL statement typed by a user directly into UI)

OUT: SQL statement for metadatabase inserts

determine_connection

gets as input database id, finds url, establishes a connection to the database and separates connection if no longer used

IN: databaseid OUT: connection

Endpoint: /determine: returns db connection for corresponding dbid

execute_databaseoperations

executes SQL statements like insert into, update, ... (access rights?)

IN: databaseid, SQL statement OUT: status

Endpoint: /execute: executes corresponding sql statements

store_query

save query to Query Store, finds query for given DOI and creates pid

IN: result set, query, execution timestamp OUT: pid (dbid + generic id)

Endpoints:

/save: saves query to corresponding query store

/find: finds query for given DOI

mapvocabularies

suggestion for vocabularies, ontology mapping

IN: usage (e.g. databasename), word OUT: group, suggestion (dynamic implementation)

9 Technical Requirements

9.1 Operating System

- Ubuntu 20.04 LTS

9.2 Technology

- Java 11 JRE and JDK

- Docker Engine v20+
- PostgreSQL 10 with additional extensions:
 - Temporal Tables

9.3 Testing Environment

Server at Uni Wien for Gitlab testing (static IP address is required and SSH public key):

HTTPS	dbrepo.phaidra.org
SSH	adminp7@dbrepo.phaidra.org ¹

Gitlab contains all code, the wiki documentaiton, etc..

HTTPS	gitlab.phaidra.org/fair-data-austria-db-repository/fda-services
GIT	git@gitlab.phaidra.org/fair-data-austria-db-repository/fda-services.git

...

10 Milestones

10.1 Sprint 0 (01.02.2021 to 22.02.2021)

- Preparation (what is already implemented?)
- Backend implementation for meta database (most important use-cases)
- Create database with x columns with data types as list (e.g. “int,timestamp,string,int,int” etc.)
- Import use cases to Gitlab

10.2 Sprint Review 0 (22.02.2021)

10.3 Sprint 1 (22.02.2021 to 15.03.2021)

- Roles (who can append data?, who can overwrite?) → JSON-Web-Token
- Test Deployment Server both at Uni Wien and TU Wien
- Docker Context ()

10.4 Sprint Review 1

10.5 Sprint 2

- tbd

11 Branching Strategy & Release

We follow the progressive-stability branching strategy [2] where the “master” branch is a release branch. All development must only merge into the development branch “dev” which is the main stable branch for all features. The branching tree therefore looks like:

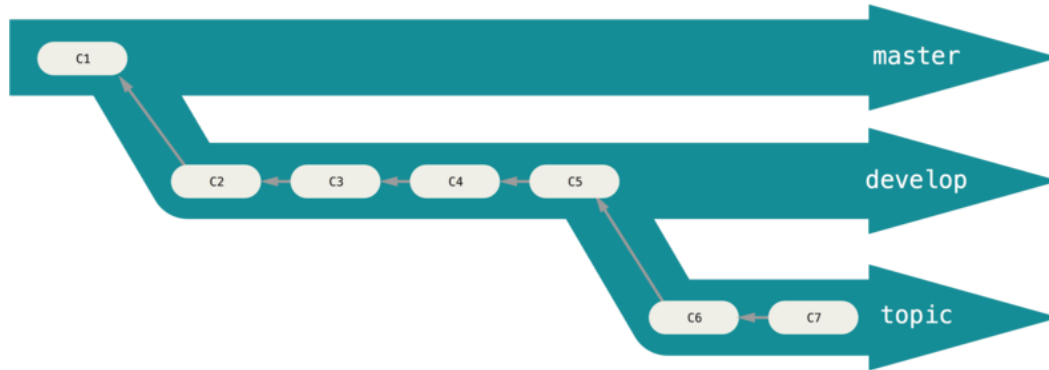


Figure 6: A “silo” view of progressive-stability branching

12 Organizational Environment

12.1 Project Owner

- Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber for *Vienna University of Technology*
- Dipl.-Ing. (FH) Raman Ganguly for *University Vienna*

12.2 Project Employees

In alphabetic order (sorted by last name and their roles within the team):

- Eva Gergely (Project manager)
- Markus Lindner (Back-end Developer)
- Cornelia Michlits (FAIR Principles, Metadata, Databases)
- Moritz Staudinger (Data versioning, Column-Store Databases)
- Kirill Stytsenko (Front-end Developer)
- Martin Weise (Back-end Developer)

12.3 Other

- Tomasz Miksa (provides us with test data, Vienna University of Technology)
- Christoph Jokubonis (system administrator, University of Vienna)

12.4 Meetings

- Andreas: <https://tuwien.zoom.us/j/98122298379>
- Team: <https://www.gotomeet.me/CorneliaMichlits/fair-data>

References

- [1] Technische Universität Wien. Fair Data Austria. [Online; accessed January 19th, 2021]. URL: <https://forschungsdaten.at/fda/>, 2020.
- [2] Git branching - branching workflows. [Online; accessed January 25th, 2021]. URL: <https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>, 2021.