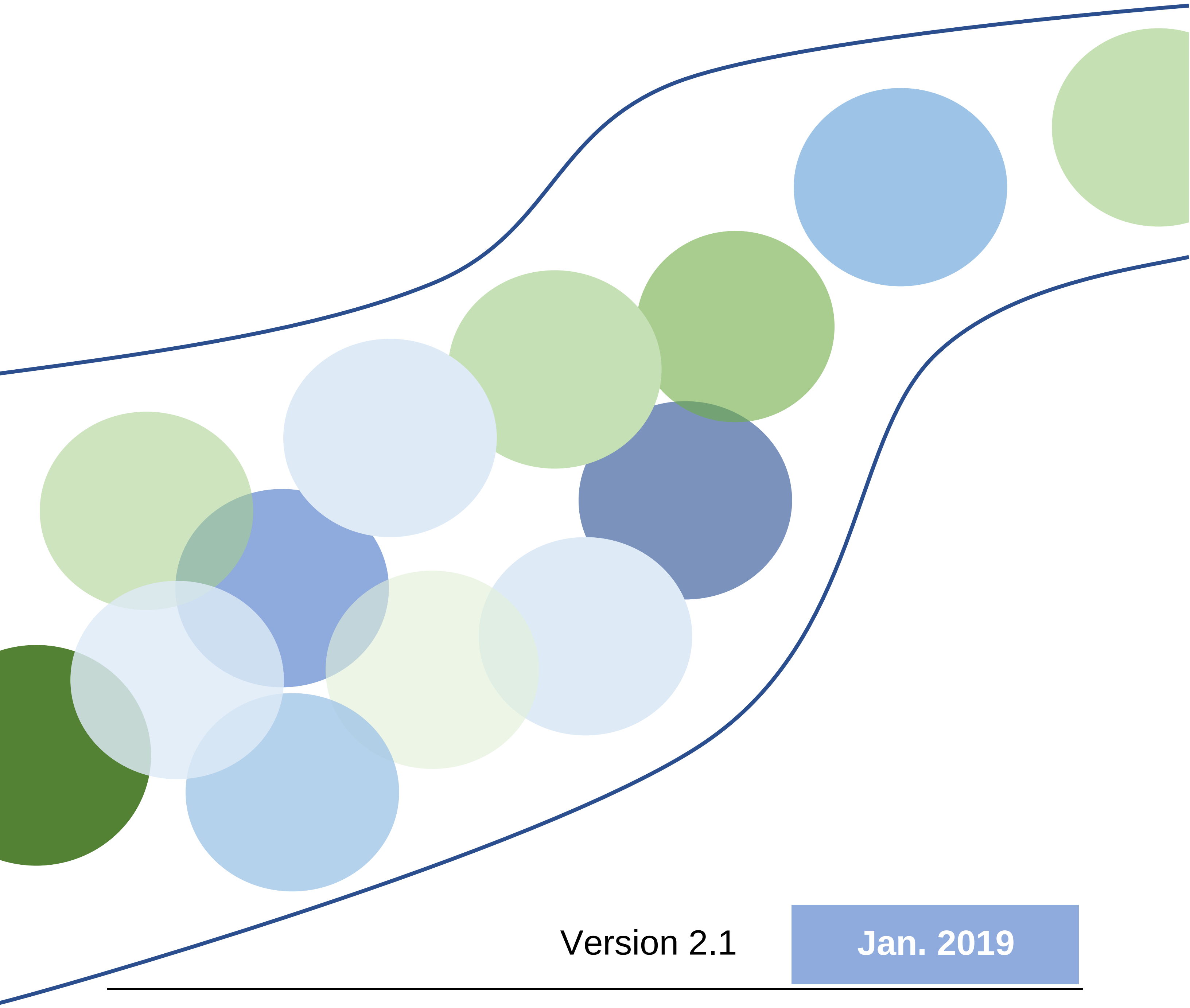
D6.5 REEEM Pathways Database

Technical Documentation



Version 2.1 **Jan. 2019**

Project Partners

Project Information

Project Number:	691739
Project Acronym:	REEEM
Project title:	Role of technologies in an energy efficient economy – model based analysis policy measures and transformation pathways to a sustainable energy system

Authors

Ludwig Hülk (RLI), Alexis Michaltsis (RLI), Ólavur Ellefsen (TOKNI), Mascha Richter (RLI)

History of changes

Version	Publication date	Change
0.1	11.01.2019	Initial version
0.2	18.01.2019	Reviewer version
1.0	25.01.2019	Beta version
2.0	28.01.2019	Release version
2.1	31.01.2019	Final version

Table of contents

Project summary

REEEM aims to gain a clear and comprehensive understanding of the system-wide implications of energy strategies in support of transitions to a competitive low-carbon EU energy society. This project is developed to address four main objectives:

(1) to develop an integrated assessment framework
(2) to define pathways towards a low-carbon society and assess their potential implications
(3) to bridge the science-policy gap through a clear communication using decision support tools
(4) to ensure transparency in the process

The REEEM project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 691739. This publication reflects only the views of its authors, and the European Commission cannot be held responsible for its content.

About this report

Project manual:

> *A flexible open-source and SQL-based Pathways Database will be set-up to service all models. It will be accessible through an interface on the project Web Platform. Existing databases within the Consortium will initially serve to populate the database, enabling open access as far as possible. Data gaps will be identified and filled to refine the models. The database will be updated to include additional technologies, demand categories, emission factors, etc. Additional data will be accessed through a range of public and private sources as well as drawing on data gathered for the case studies. Transparent data processing scripts and stand-alone tools will be developed to facilitate the communication between the various models used in REEEM, and to process basic to higher level datasets.*

Main challenges and tasks

In this project, a large group of modelling teams from different institutes are developing and using different software with different programming languages and modelling paradigms. This results in a large variety of data structures.

Thus, the project database has to meet different requirements. Besides complying with basic data security and access regulations as described in the Data Management Plan (D8.2), the structure must be flexible and database usage should be as automated as possible. In an integrated assessment model, data from different fields are used and created. This leads to challenges in the data classification and categorisation which were solved developing and implementing a flexible tagging system. Besides the technical aspects, the legal aspects were considered as well. The project aims at publishing the data sets under open licenses.

1. Database Setup

The database setup contains all scripts to create the REEEM **database structure**. The database setup is scripted using the database programming language *Structured Query Language* (SQL). It is highly oriented on the OpenEnergyPlatform (OEP) structure in order to assure compatibility between these databases.

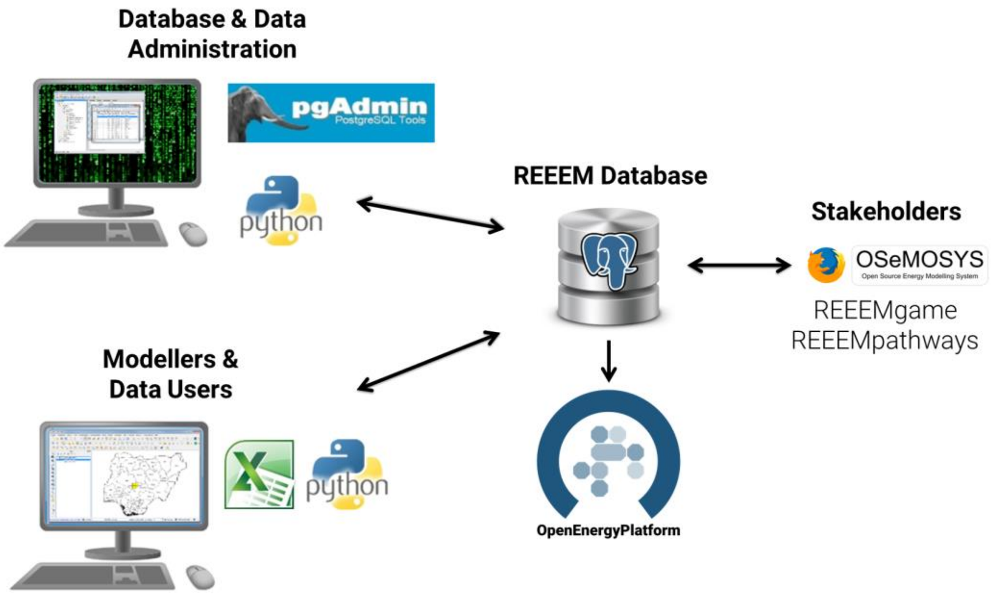


Figure 1 REEEM data management and dissemination © Reiner Lemoine Institut | CC BY 4.0

All database related code is under version control and publicly available under open licenses on GitHub. The repository is available on https://github.com/ReeemProject/reeem_db. To assure data security during the project, the REEEM database is not a public database; it's an internal project database. Using the code, one can create the entire setup of the internal database on another server or a local computer. The data itself is not part of the repository.

The database setup comprises the following topics, which are explained in detail in the next sections:

- Repository structure
- Schema Setup
- User Management
- Scenario Log
- Examples and Templates

Repository structure

The REEEM GitHub repository structure is described in following diagram. The project team created a GitHub project named ReeemProject. At the moment, the project has two separate repositories (reeem_db and reeem_game). It is possible to add more repositories, if necessary. The database repository is structured with different folders for the setup (database_setup), the data upload (database_adapter) and data processing and access (database_views).

All database operations are coded (in SQL and Python) to have a transparent and reproducible database setup. All scripts are licensed with an open license (AGPL-3.0) and can be reused and developed further. The database content (the data itself) is not included. The folder *Model_Data* is not part of the repository.

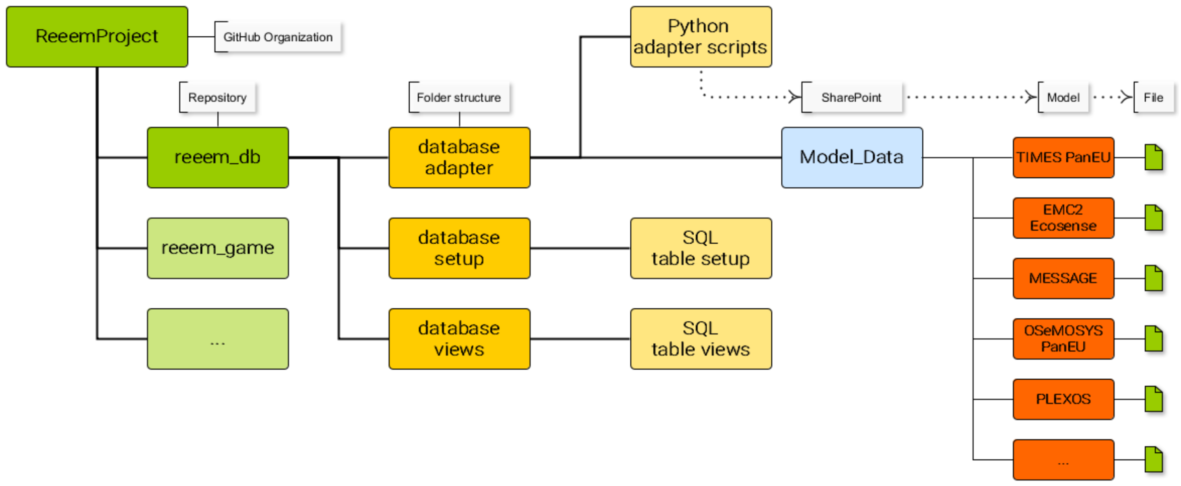


Figure 2 REEEM GitHub repository folder structure © Reiner Lemoine Institut | CC BY 4.0

Schema Setup

The PostgreSQL schema structure is based on the ISO 19115 MD_TopicCategoryCode (Section B.5.27) which is also used in the OEDB schema structure (v0.2). The ISO list has been complemented with energy specific topics(e.g., energy_demand) and project related schemas (e.g., game, pathway). In addition to these schemas, there are schemas which are software specific and are used for data visualisation (e.g., hdb_catalog, hdb_views).

Table 1 REEEM database schema descriptions

Schema name	**Description**
boundaries	legal land descriptions. examples: political and administrative boundaries
climate	processes and phenomena of the atmosphere. examples: cloud cover, weather, climate, atmospheric conditions, climate change, precipitation
economy	economic activities, conditions and employment. examples: production, labour, revenue, commerce, industry, tourism and ecotourism, forestry, fisheries, exploration and exploitation of resources such as minerals, oil and gas
energy_demand	consumption and use of energy. examples: peak loads, load curves
energy_grid	energy transmission infrastructure. examples: power lines, substation, pipelines
energy_supply	conversion (generation) of energy. examples: power stations, renewables
environment	environmental resources, protection and conservation. examples: environmental pollution, waste storage and treatment, environmental impact assessment, monitoring environmental risk, nature reserves, landscape
society	characteristics of society and cultures. examples: settlements, anthropology, archaeology, education, demographic data, recreational areas and activities, social impact assessments, crime and justice, census information
model_draft	modelling sandbox, temporary tables. examples: modelling raw data
reference	sources, literature
game	REEEMgame - Online Energy Systems Learning Simulation (D7.4)
pathway	REEEMpathways - Pathways Diagnostic Tool (D7.2)

Code: reeem_setup_schema.sql

User Management

The current concept of the user rights for the server is based on three main groups (admin, user, read) with additional groups for special schemas (like game). *Roles* (login & pw) are assigned to *groups* with defined *privileges*. Objects (tables, materialized views, views, sequences) are granted to *groups*. Usually this is done by default, but can be done manually.

"PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as either a database user, or a group of database users, depending on how the role is set up. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control who has access to which objects. Furthermore, it is possible to grant membership in a role to another role, thus allowing the member role to use privileges assigned to another role." source

Groups

REEEM users can have the following groups:

Table 2 Database user groups and access rights

Group	Rights
reeem_admin	Read, write and user management
reeem_user	Read and write
reeem_read	Read only
Reeem_game	Read all and write to own schema

REEEM user management

The project database has a multi-level user management with *admins* who take care of user rights, *users* who have unlimited access to the database content, and *visitors* with read access to selected parts. All user accounts are password protected. The permission rights of each data set are also stored in the metadata.
The REEEM user management is described in the following use case diagram:

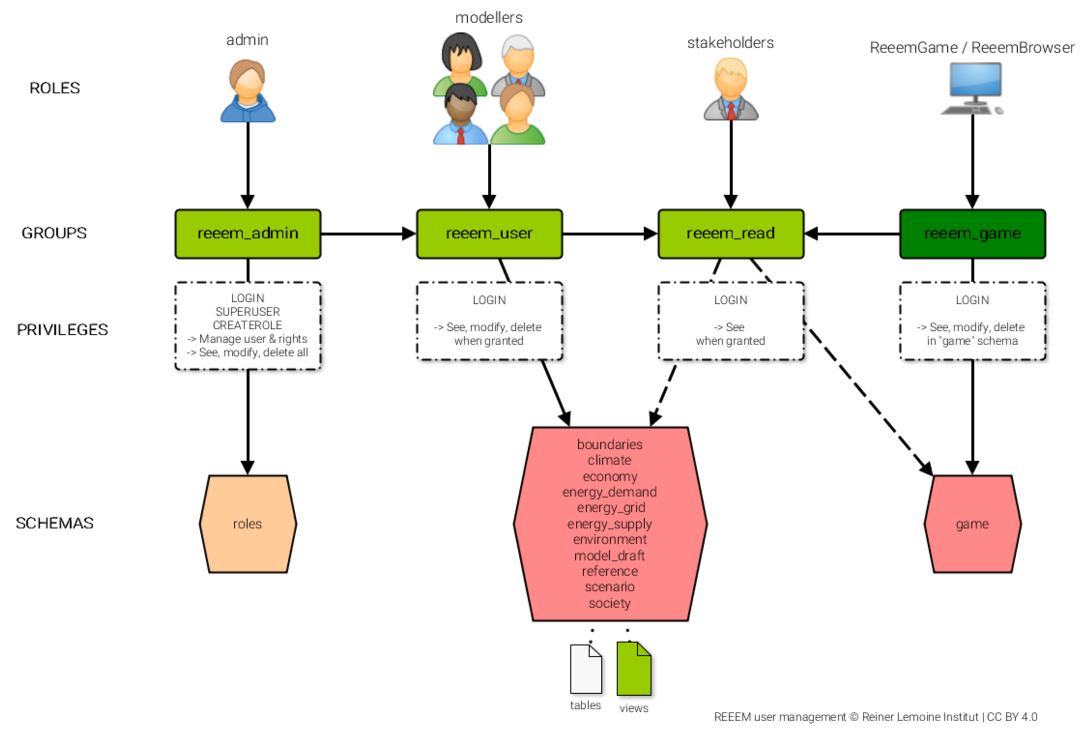


Figure 3 REEEM user management © Reiner Lemoine Institut | CC BY 4.0

Code: reeem_setup_user_managment.sql

Metadata

In the project database, the metadata is stored as a JSON string in a comment of each table. By creating a metadata string (following the FAIR Principles), additional information about each model is directly stored with the data.

- **General description** (e.g., title, description, spatial and temporal resolution)
- **License information** (e.g., sources, contributors, resulting (open) license)
- **Data description** (e.g., column name, column description, unit)

An example metadata string can be found here: reeem_test_table.sql

Scenario Log

The *Scenario Log* is a function to add an entry to the **scenario log table** (model_draft.scenario_log). It has been implemented for SQL and Python. It covers and documents the creation of the database structure (tables) and the import and export of data. All database scripts are using the scenario log function to ensure transparency.

Table 3 *Scenario Log* parameters

Inputs	Example	Outputs (to table)
• project	`REEEM`	• All **Inputs**
• version °	`V0.3`	• id (*)
• io °	`Input (output/setup)`	• entries (*)
• schema_name	`model_draft`	• user_name (*)
• table_name	`reeem_times_paneu_output`	• timestamp (*)
• script_name °	`reeem_db_setup_times_paneu.sql`	• metadata (*)
• comment	`Upload result data`	

(°) Generated in Python   (*) Generated from database system

Execution

In SQL-scripts:

```
-- scenario log (project,version,io,schema_name,
-- table_name,script_name,comment)
scenario_log('REEEM','v0.1.0','setup','model_draft',
'scenario_log','reeem_scenario_log.sql','Function test');
```

In Python-scripts:

```
# scenario log (con,project,version,io,schema_name,
# table_name,script_name,comment)
scenario_log(con, 'REEEM', fns['version'], fns['io'], db_schema,
db_table, os.path.basename(__file__), filename)
```

Code: Scenario Log table setup, Scenario Log function

The Scenario Log is used for different database operations:

- Create a table
- Insert data into table
- Update or change data
- Visualize and download data
- Delete a table/view

Examples and Templates

In order to increase reusability, examples and templates have been added to the repository. The programming language is SQL. The examples and templates are used to set up new models or users and test and document new developments.

Examples

Code: model_draft.test_table
Includes:

- test table
- access rights
- metadata description
- metadata validation
- insert test data
- select test data

Templates

Code: model_draft.model_data_template
Includes:

- table description
- access rights
- metadata
- scenario log

2. Database Adapters

Database adapters are a bundle of (Python) scripts. Each script reads in data from a local file and writes the data to a database table. Every script corresponds to one specific model. Each REEEM modelling team has a customized **data adapter**. The adapter does not affect the data itself but is used as an importer or exporter between the (online) database and a (local) file.

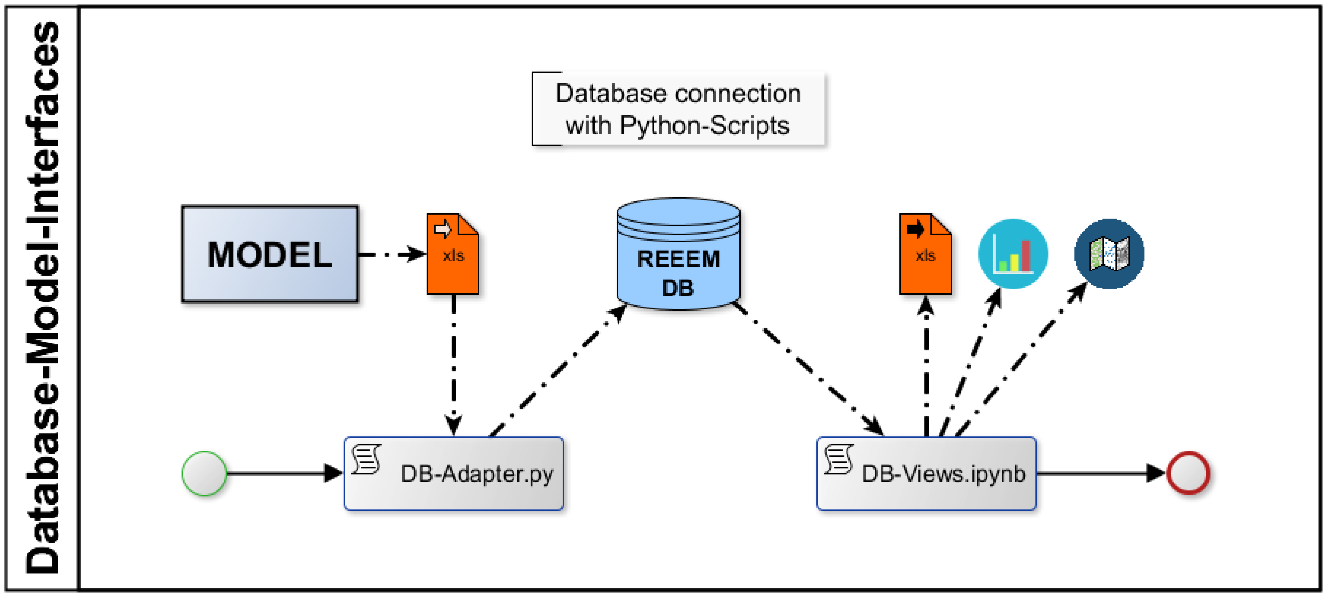

Database-Model-Interfaces
Database connection with Python-Scripts
MODEL
xls
REEEM DB
xls
DB-Adapter.py
DB-Views.ipynb

Figure 4 REEEM database connections © Reiner Lemoine Institut | CC BY 4.0

reeem.io script

The reeem.io script is needed for each adapter. It contains a method to establish a **database connection** to the REEEM PostgreSQL database. In addition it contains a **python logging** function and the described **scenario log** function.

Data

File names

In order to process the local data files automatically a naming convention for the data files was established.

[Date]_[PATHWAY]_[MODEL]_[FRAMEWORK]_[VERSION]_[I/O]

Example: `2018-01-16_BASE_TimesPanEU_FrameworkV2_DataV1_Input.xslx`

[Date] YYYY-MM-DD. (ISO 8601) Use the hyphen (-) to separate years, months and days.

[PATHWAY] Name of the pathway.

[MODEL] Name of the model or software.

[FRAMEWORK] FrameworkVn. It refers to the composition of connected models.

[VERSION] DataVn. Model run number.

[I/O] Input or output data of the model.

Data structure

Each table should have a similar data structure with the same columns and datatypes. In addition, some model data has additional columns (e.g., internal id). The data types are specific for PostgreSQL.

Table 4 REEEM table structure and columns

ID	Parameter name	Description	Data type
1	**id***	Unique table id	serial
2	**nid**	A unique id per sheet or file	integer
3	**pathway***	REEEM pathway (from filename)	text
4	**framework***	REEEM framework (from filename)	text
5	**version***	Data version (from filename)	text
6	schema***2**	Categorisation: Schema (e.g., supply, economy)	text
7	field***2**	Categorisation: Field (e.g., Power, Heat, Mobility)	text
8	category***2**	Categorisation: Category (e.g., Technology)	text
9	**region**	Country code (iso_2)	text
10	**year**	Year (format: YYYY)	integer
11	**indicator**	Parameter name	text
12	**value**	Parameter value	double precision
13	**unit**	Parameter unit (abbreviation)	text
14	**aggregation**	Single or sum (for a category)	boolean
15	**tag**	Data categorisation	hstore
16	**updated**	Timestamp from (now())	timestamp
17	**source**	Where does the value come from	text

***** These columns are added automatically (from the filename or system).
***2** Additional columns for data categorisation.

3. Data Classification

After the input and output model data is inserted into the database through the help of the database adapters, the next step is to describe the data by tagging each database row of every given model input and output data. The tagging process manifests in SQL data cleaning and tagging scripts. Each script corresponds to one model. While most models and databases have a hierarchic data classification system (schema, field, category), the REEEM project decided for a more flexible tagging system in addition. The reason for the tagging is to enable clustering of data across model borders and labelling of ambiguous terms.

Code: data-cleaning-and-tagging

Each model table has a column named tags. The fields of this column are of the PostgreSQL database data type named *hstore*. This data type can save key value pairs in a key value store (also often named associated *array*, *dictionary* or *hashmap*).

Data Structure

Description of the data structure of the hstore data type:

{“key_1”: “value_1”, “key_2” : “value_2”, ... “key_n” : “value_n” }

Keys

Each key can have one of the following names:

- model
- schema
- field
- category
- (custom)

One special case are custom keys, which can have multiple custom names, as long as they are not conflicting with the 4 other fixed key names.

Values

The values of the key value pairs are taken by tagging each data row in the database with model, schema, field, category and optional custom key values. This is done by grouping data rows and looking up the information given in their field, category and indicator columns, which have been provided by all research partners.

Model

The names of the used models in the REEEM project are the possible values of the model tag keys. All model names in the tags are written in lower case and white spaces are replaced by underscores.

Model tag example "TIMES PanEU":

```
model: times_paneu
```

Schema

The values of the schema key map to the 8 schema names described in the database schema setup section of this report. Some schemas are not considered, because no data was used from this area.

- boundaries
- climate
- economy
- energy_demand
- energy_grid
- energy_supply
- environment
- society

Field

Field values can be best described as sub schemas.

Field tag example:

```
schema: society
field: health
```

Category

Category values are values which are not an indicator but further deepen the definition of a data row.

Category tag example:

```
schema: society
field: health
category: pollution
```

Custom

Custom keys can have custom values and are used if further deepening of the key structure is desired.

Custom tag example:

```
field: costs
costs: investment   (custom key data row_n)
costs: variable     (custom key data row_m)
```

Multiple values

If needed, each key (except the model key) can have multiple values. In comparison to a strictly hierarchical categorisation system, this system is more flexible and allows tagging ambiguous terms. For this, each value is separated by a semicolon:

```
{“key”: “value_1;value_2;value_n” }
```

4. Database Usage

Database Views

Database views with Jupyter Notebooks

For pre-filtering and gaining inside information of the input and output data of the different models, frameworks and data versions used in the REEEM project the computational open-source environment named Jupyter Notebooks is being used. In order to access the data, the database can be selected and filtered using views. These views are written in SQL and are executed in the database. As additional service, the RLI has gathered visualisation scripts in Jupyter Notebooks.

Code: REEEM Jupyter Notebooks visualisation scripts

To run them on your computer you have to create a suitable environment. We recommend using Conda (which you have to install).

Link: Conda, package and environment management system

There are additional requirements to run the notebooks. Therefore you have to execute the following commands (only once - when you run them again you start with the "activate" command):

Create Conda environment

open a `cmd.exe` terminal

```
cd ...\reeem_db\database_views\reeem_jupyter\
conda env create -f requirement_reeem-vis.yml
conda info --envs
```

Run Jupyter Notebooks

open a `cmd.exe` terminal

```
cd ...\reeem_db\database_views\reeem_jupyter\
conda activate reeem-vis
jupyter notebook
```
(to execute the code in the notepads you have to push enter+shift)

Database Access

Corresponding to the REEEM Data Management Plan (DMP) the database (PostgreSQL) is hosted on a server located at the facility of the DTU project partner in Denmark.

To access the REEEM database, a suitable database management system (DBMS) and a valid user is required. For further details on the user management, please refer to the “User management” section in this document.

For security measures and because of licensing issues the REEEM project database access is publicly restricted to the project partners. In order to gain access to the entire project database, please contact Ludwig Hülk (RLI):

`ludwig.huelk@rl-institut.de`

As stated in the DMP, all openly licensed input and output data of the models and pathways (of the REEEMPathways tool) in the REEEM database are being publicly published on the OpenEnergyPlatform (OEP) until the end of the project and can then be identified under the tag “REEEM”. Furthermore, in the REEEM project the Open Source energy Modelling Base for the European Union (OSeMBE) is developed openly as well. The coupled open source engagement model and input data of OSeMBE is available on REEEM.org and the output model data will be available on OEP. Finally, all open support documentation from the different data processing and modeling activities are either available on GitHub or in the concerning open access publications.

For remote access to the REEEM PostgreSQL database system at DTU, we suggest using the browser based software pgAdmin4. For users not familiar with PostgreSQL, we further suggest the online documentation and available tutorials on PostgreSQL.

Tutorial: Database access with pgAdmin4

1. Get your personal access information from contact via email.
2. Install pgAdmin4
3. Create a connection to the DTU-server:
   [Browser] Servers -> right click -> Create -> Server
   - General
     - Name = DTU-Server (your choice / any name for the server)
   - Connection
     - Host = will be provided by email
     - Port = will be provided by email
     - Username = will be provided by email
     - Password = will be provided by email

   Save connection and login
   [In your browser] Servers -> right click -> Refresh

Tutorial: Explore database structure

1. [Browser] Databases -> reeem -> Schemas
   – Schemas are like folders, they structure and categorises the data
   – Schemas contain subfolders with the tables and (materialized) views

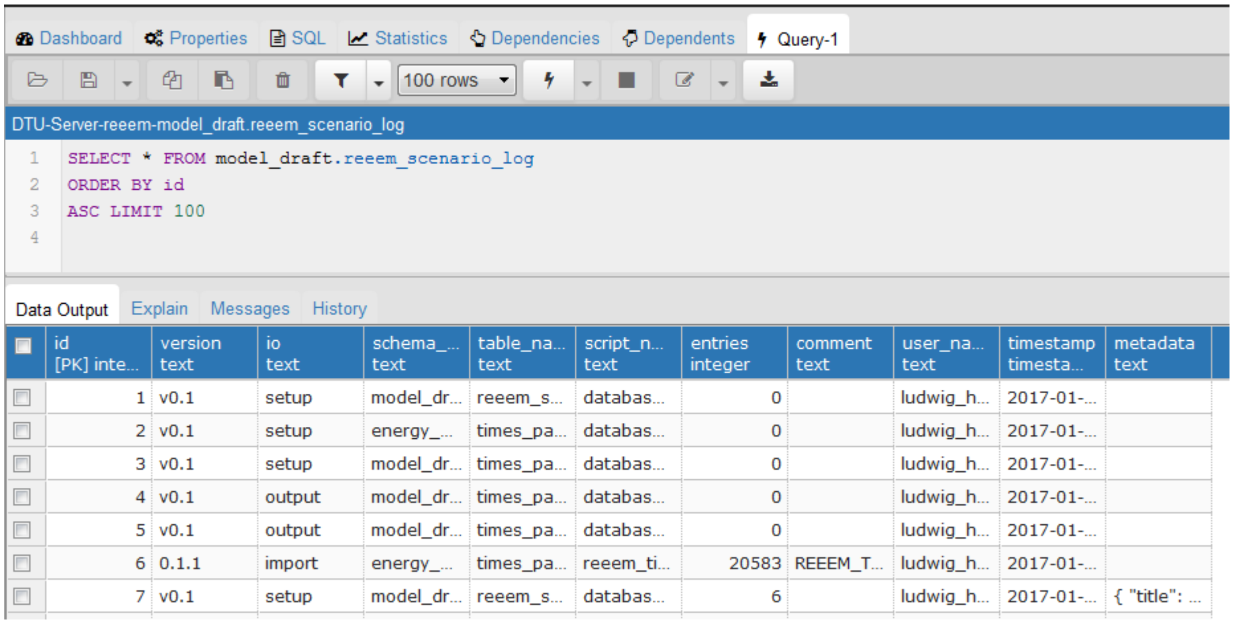
Tutorial: View the data of a table or (materialized) view

1. Open Table Data:

[Browser] (right click) on a table or view) -> View Data (you have following options):

   - All Rows
   - First 100 Rows
   - Last 100 Rows
   - Filtered Rows...

- If you open a table with > 10.000 rows it may take a while
- A Query-1 opens in the [Query-bar]
- It shows the Select-Code in SQL and the data



DTU-Server-reeem-model_draft.reeem_scenario_log

```
SELECT * FROM model_draft.reeem_scenario_log
ORDER BY id
ASC LIMIT 100
```

id [PK] inte...	version text	io text	schema_... text	table_na... text	script_n... text	entries integer	comment text	user_na... text	timestamp timesta...	metadata text
1	v0.1	setup	model_dr...	reeem_s...	databas...	0		ludwig_h...	2017-01-...	
2	v0.1	setup	energy_...	times_pa...	databas...	0		ludwig_h...	2017-01-...	
3	v0.1	setup	model_dr...	times_pa...	databas...	0		ludwig_h...	2017-01-...	
4	v0.1	output	model_dr...	times_pa...	databas...	0		ludwig_h...	2017-01-...	
5	v0.1	output	model_dr...	times_pa...	databas...	0		ludwig_h...	2017-01-...	
6	0.1.1	import	energy_...	times_pa...	reeem_ti...	20583	REEEM_T...	ludwig_h...	2017-01-...	
7	v0.1	setup	model_dr...	reeem_s...	databas...	6		ludwig_h...	2017-01-...	{ "title": ...

Figure 5 REEEM scenario log table

5. Links

Public GitHub repository
https://github.com/ReeemProject/reeem_db/