M.Sc. in Computational Data Science

Samuele Ceol (13140)

Data Preparation and Integration / Data Profiling

Project Report

A.Y. 2022/2023

Faculty of Computer Science

Contents

1 Introduction **1**

1.1 Description of the domain of interest 1

1.2 Dataset format 2

2 Data Profiling **3**

2.1 Initial data inspection and first data reformulation 3

2.2 Exploratory Data Analysis 5

2.2.1 agri_system_overview.csv 5

2.2.2 commodities.csv 6

2.2.3 connections.csv 6

2.2.4 factors.csv 6

2.2.5 connections.csv 7

2.2.6 farming_system.csv 7

2.2.7 impact_chain_model.csv 8

2.2.8 involved_experts.csv 8

2.2.9 resources.csv 8

2.3 DB design proposal and second data reformulation 8

2.4 IC discovery 11

2.4.1 Unique column combinations 11

2.4.2 Functional dependencies 11

2.4.3 Inclusion dependencies 12

3 Data Preparation and Integration **15**

3.1 Ontology 15

3.2 Mappings 16
3.2.1 Mapping pattern: Entity (MpE) 17
3.2.2 Mapping pattern: Relationship (MpR) 17
3.2.3 Mapping pattern: Relationship with Merging (MpRm) 18
3.2.4 Mapping pattern: Reified Relationship (MpRR) – Attribute case 19
3.2.5 Other mappings 19
3.2.6 SQL query simicolon error 19
3.3 Application description 21
3.4 SPARQL queries used by the application 22

4 Running the Application **24**
4.1 Restoring the PostgreSQL DB 24
4.2 Connecting Protégé to the Database 25
4.3 Setting up the Ontop SPARQL endpoint 26
4.4 (Optional) Creating the conda environment 26
4.5 Running the application 26

References **27**

List of Figures

1.1 Structure of an Impact Chain Model 2

2.1 Missing data from farming_system.csv 7
2.2 ERD of the subject domain 14

3.2 Mapping pattern: Entity (MpE) 17
3.3 Mapping pattern: Relationship (MpR) 17
3.4 Mapping pattern: Relationship with Merging (MpRm) 18
3.5 Mapping pattern: Reified Relationship (MpRR) – Attribute case 19
3.6 Ontop error raised by SQL query with trailing semicolon 20
3.1 Ontology represented via the graphical notation 23

4.1 DB restore page in pgAdmin 24
4.2 Successful DB connection in Protégé 25

Chapter 1

Introduction

1.1 Description of the domain of interest

The proposed project is shaped on the data collected for the Climate Risk Planning & Managing Tool for Development Programmes in the Agriculture & Food Sector (CRISP) project [1] funded by the Deutsche Gesellschaft für Internationale Zusammenarbeit GmbH [2].

Important in the understanding of the presented data is the concept of **Farming Systems**, which are described in [3] as: *"populations of farms that have broadly similar resource bases, enterprise patterns, household livelihoods and constraints, and for which similar development strategies and interventions would be appropriate. The biophysical, economic and human elements of a farm are interdependent, and thus farms can be analysed as systems from various points of view. [...] Regardless of their size, individual farm systems are organised to produce food and meet other goals through the management of available resources [...] within the existing social, economic and institutional environment".*

The presented data seeks to describe farming systems belonging to different macro regions (the broad definitions of which can be found in [3]) using the concepts and terminology related to the impact chain method developed by the Eurac Climate and Disaster Research Group and described in the corresponding vulnerability sourcebook [4] and risk supplement [5]. In this context, another relevant notion is the one of **Impact Chain Models**, which are described in [4] as: *"conceptual models describing climate impact as cause-effect relationships within a socio-ecological system"* and, in even simpler terms, are defined in [5] as: *"analytical tool[s]*

that help you better understand, systemise and prioritise the factors that drive vulnerability in the system under review". The general structure of an impact chain model can be observed in Figure 1.1.

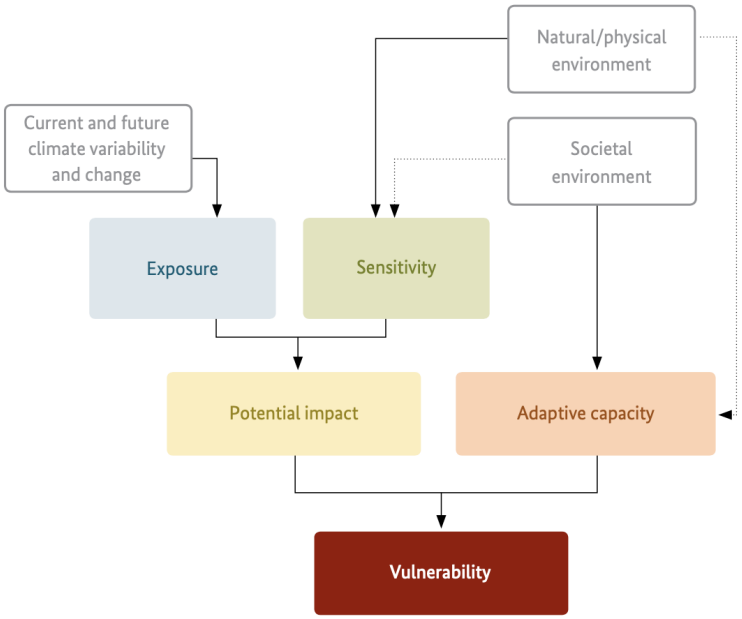


Figure 1.1: Structure of an Impact Chain Model

In this context, the underlying data associated with each given farming system presents the relevant **Factors** (which can be seen as the building blocks for the Impact Chain Model) and ties them together by means of a set of **Connections**.

Additionally, a wide array of indicators are provided with the purpose of describing in detail the farming system at hand. Such indicators include information on livelihood sources, produced commodities, landscapes, related countries, etc.

Finally, much of the presented data is associated with **bibliographic resources**, which allow to tie a piece of information to its original formulation.

1.2 Dataset format

The initial dataset related to the project at hand is made up of a set of 22 `.xlsx` files, each containing information related to an individual farming system. The data has been privately shared for the completion of this project and can be accessed by authorised users from [6].

Chapter 2

Data Profiling

2.1 Initial data inspection and first data reformulation

In order to develop an initial idea on the structure of the interested data, the 22 source .xlsx files (found under `CRISP Dataset/XLSX data`) are manually inspected. In this context, it is observed that each individual file (representing a different farming system) is organised into a set of (up to) ten different sheets, each containing a different category of information. Because of this reason (and in order to generate a more easily explorable dataset), the individual sheets are used as the starting point for the generation of a set of .csv files (found under `CRISP dataset/CSV/Original csv`). In this regard, data belonging to the same sheet and different source files is concatenated into a single document. From this process, a set of eight .csv files is generated:

- `agri_system_overview.csv` contains most of the general information related to a specific farming system. This include its name, an overall description, the countries in which it appears and some numerical information related to its characteristics. Additionally, it has been decided to add to this document the fields containing the citation keys of the resources related to the various attributes of a given farming system. This fields are identified by the prefix `bib_` (e.g. `bib_cultivated_area`).
- `commodities.csv` contains the details of the commodities that are generated in the context of a given farming system.
- `factors.csv` contains the (hazard, impact, vulnerability, ...) factors related to the impact

chain models that characterise a given farming system, together with the corresponding labels and an associated descriptions.

- `connections.csv` contains the connection information linking together the various factors that are part of a given impact chain model. Each connection is represented by an origin and target factor, a description and a set of descriptive tags and bibliographic resources.
- `farming_system.csv` contains the detailed information of the various farming system. In this context, each farming system is represented by a name (which sometimes also contains the related macro region identifier) a description, a (set of) bibliographic resource(s), information on the locations it encapsulates, livelihood sources, landscapes, agro-ecological zone, (total, cultivated, irrigated, farm) area, (total, cattle, agricultural) population and dominant livelihood source.
- `impact_chain_model.csv` contains a description (and associated bibliographic resources) for each of the described impact chain models (one per farming system). Additionally, it contains additional location information to the one found in `farming_system.csv`.
- `involved_experts.csv` contains the experts involved in (the data collection for) a given farming system. Each expert is represented by a name and surname, the domain for which it was consulted, the organization to which it belongs to, the date of the involvement, the email and a citation key.
- `resources.csv` contains the consulted resources. Conceptually, each source document should contain in the resource sheet the complete set of papers used for gathering all information related to a given farming system. Each resource is represented by a citation key, a title, a resource type, a URL, a publication date and a list of authors' names.

It can be noticed how the `IC description` sheet was not translated into a dedicated `.csv` file. This was due to the fact that the information found in this sheet was already contained in the impact chain model one.

Additionally, an `ID` column (following the naming of the original .xlsx files) was added to certain `.csv` files (e.g. `commodity.csv`) as a temporary attribute to avoid losing the association between the interested tuples and the related farming system source file they belonged to.

Finally, at this stage a new source file named `M49_Codes.csv` was generated. This file contains

all the UN location data used to identify the geographical locations involved in the dataset at hand. This information were obtained from the official website of the united nations [7].

2.2 Exploratory Data Analysis

The generated set of `.csv` files allows for the execution of an initial exploratory data analysis which might provide further insights on the data at hand and might help guiding the successive database design phase. In this context, the `ProfileReport` class of the `pandas_profiling` library is utilised. The set of profiling reports (which is found in the `PP_dump` folder) can be easily constructed by using the following code:

```
sources = ['agri_system_overview', 'commodities', 'connections', 'factors', '
    farming_system', 'impact_chain_model', 'involved_experts', 'resources']

for source in sources:
    df = pd.read_csv(f"CRISP Dataset/CSV/Original csv/{source}.csv")
    profile = ProfileReport(df, title=source)
    profile.to_file(f"PP_dump/{source}.html")
```

Listing 2.1: Profiling reports generation

In this section, only the most relevant (and non-trivial) observation made from this initial analysis phase for each generated file will be reported. For instance, saying that a farming system's name is highly correlated with its corresponding description is an example of a trivial information that would be omitted from this section. In other words, a comment will be made only on those information that offer a deeper insight on the data at end. The following observations can be made starting from the generated reports:

2.2.1 agri_system_overview.csv

The file contains a total of *19* features and *22* entries. Immediately, it is possible to notice that each entry maps to a single source *.xlsx* file (and therefore to a single farming system). When observing the `Name of agri system` feature, it can be observed how some values are repeated. This already suggests that to uniquely identify a given agricultural system (i.e. a farming system) using only its' related name is not sufficient. The countries related to an individual entries are expressed as lists in plain text format, this will require a normalisation step when this data will be carried over at the database level. The other features, containing

either numerical data related to a given system or the resource information tied to a given field both present a vast number of missing values. Finally, some constant values can be observed among citation keys appearing in the `bib_overview` column.

2.2.2 commodities.csv

The file contains a total of *15* features and *74* entries. Once again, the `name` column describing a given commodity is provided for all entries but presents numerous duplicate values. This suggests that, at the database level, it will need to be paired up with additional features in order to be part of a primary key. The NCBI Taxonomy descriptors are provided only for some of the file entries (they are not provided in 17 rows). In a general sense, information tied to a given commodity are present for almost all rows, the notable exception to this trend is the `Max THI` column, the values of which are absent in 17 entries.

2.2.3 connections.csv

The file contains a total of *8* features and *2528* entries. This source file is substantially larger when compared to the previously observed ones. Each connection is tied to the original source file by the `FS Name` attribute. Even though some values for this attribute are missing, they can be easily filled since the connection appear in the `.csv` file grouped by the farming system to which they belong. The `connection` column, containing an identifier for each connection is made up of mostly missing values (and will likely be discarded when constructing the database). All other attributes contain fundamentally no missing values (with the exception of one empty row). In this sense, each connection is "fully" described.

2.2.4 factors.csv

The file contains a total of *9* features and *2411* entries. Each factor is tied back to the origin farming system by the `Type column`. The `Label` attribute (describing the name of a given factor) contains 292 distinct values. In this context, it is expected to have repeated labels for the same factor described in the context of different farming systems. From the report, it can also be observed that the attributes `Link to external database` and `Developer and name of external database` are fundamentally empty and will therefore be ignored in the following phases of the project. It can be noted how there is an high correlation between the `Type-1` and `Type-2` attributes. Manual inspection reveals that quite a few rows are completely empty and

derive from how the data was structured in the original files. This is not necessarily an issue in this initial exploration phase, but it will require to be addressed when preparing the data for the database.

2.2.5 connections.csv

The file contains a total of *8* features and *2528* entries. This source file is substantially larger when compared to the previously observed ones. Each connection is tied to the original source file by the `FS Name` attribute. Even though some values for this attribute are missing, they can be easily filled since the connection appear in the `.csv` file grouped by the farming system to which they belong.

2.2.6 farming_system.csv

The file contains a total of *23* features and *22* entries. Once again, we have that each row corresponds to a source file. In this context, it can be observed that (as opposed to what has been seen in the `agri_system_overview.csv`) the `Farming System` column is fully filled and contains no duplicate values. Conceptually, this attribute could be used to uniquely identify a given tuple. Additionally, location data is more granular and also includes information expressed using the M-49 and ISO Alpha-3 codes and refers to Region, Sub-Regions and Macro regions in addition to the already observed country related information. The data is fairly complete, with most of the missing values belonging to the location data (this can be visually observed in Figure 2.1). Fortunately, the choice of using the `M49_Codes.csv` source file should address this issue.

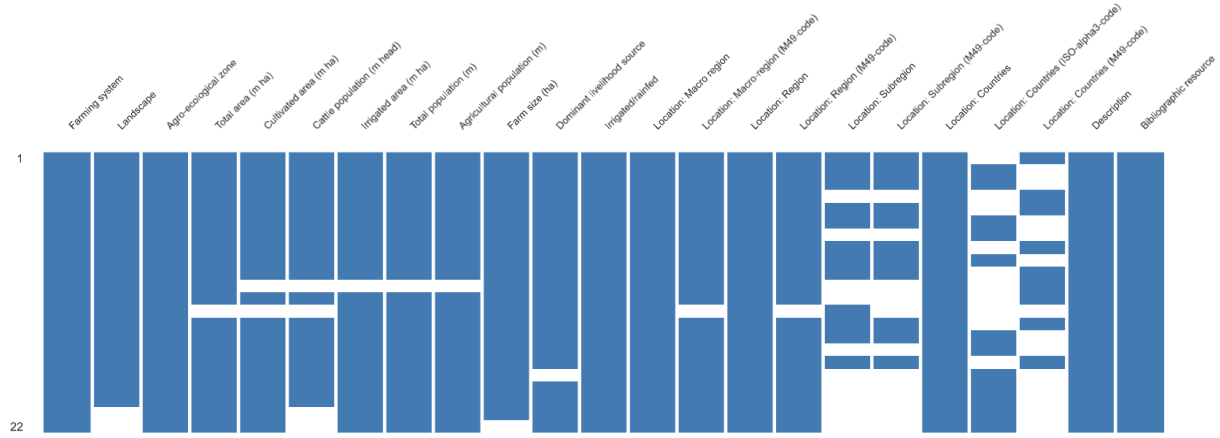


Figure 2.1: Missing data from farming_system.csv

2.2.7 impact_chain_model.csv

The file contains a total of *13* features and *22* entries. In this context, a one-to-one correspondence of a given ICM to the corresponding farming system can be observed. It can also be noticed how most of the attributes present in this files refer back to already existing location data found in `farming_system.csv` and (only partially) in `agri_system_overview.csv`. It can be safely assumed that the information extracted from this file at DB design time will be related to the ICM `description` (which contains no missing values) and (for only two entries) the `bibliographic resource` attribute.

2.2.8 involved_experts.csv

The file contains a total of *8* features and *29* entries. A set of 9 and 8 distinct values can be observed for the `First name` and `Last name` attributes respectively, suggesting that the same expert have contributed in providing information for multiple farming systems (here identified in the context of the `ID` column). The data related to this file is fairly complete, with only one value missing from the `Organisation/Position` attribute. An exception to this completeness comes with the `Citation` attribute (indicating the citation key related to a given expert). In this context, the attributes present a missing value for 22 entries.

2.2.9 resources.csv

The file contains a total of *7* features and *933* entries. By observing the number of distinct values for the `Title` attribute (equal to 823) it can be inferred that there exist some duplicate publications among the different farming systems. This is further confirmed by the distinct count of the `Citation` attribute (830), which represent the citation key for a given resource. It can also be noticed how the choice of representing Authors using lists of name could pose a problem with regards to the generation of a normalised database.

2.3 DB design proposal and second data reformulation

After the exploratory data analysis (which provided an initial idea of the data at hand), the individual files were manually inspected with the purpose of coming up with a DB design that could support the information contained within them. In a general sense, integrity constraints were inferred by observing the structure of the data. In this context, the proposed integrity

constraints algorithms (discussed in Section 2.4) were used only *after* designing the database architecture in order to confirm the choices made at design-time. Although this choice might seem counter-intuitive, it was justified by the highly heterogeneous nature of the data at hand which would have made it impossible to execute such algorithms before the conclusion of an appropriate data pre-processing step. Designing a database that would appropriately host the proposed dataset in a normalised format required the creation of a total of 40 tables. The (final) design, complete of all table and attribute names and with notation highlighting the primary and foreign keys is provided in Figure 2.2. A large part of the proposed relations are fundamentally junction tables that are required to model the many-to-many relationship that are part of the underlying data (e.g. `FS livelihood sources`, `FS landscapes`, etc.). Additionally, the database also presents a good number of single field tables which are used to uniquely store the name identifiers of various entities which are then referenced as foreign keys in other tables (e.g. `Agro-ecological zone` used as foreign key in `Farming System`). Additionally, notice that the `Authors` (and the corresponding `Resource authors`) table were omitted from the final DB architecture (both are represented using dotted lines). This choice is justified by the fact that the authors' names in the original source files are presented in an extremely heterogeneous format. Processing this data to make it suitable for the proposed database would have added a substantial amount of labour to the already extensive pre-processing phase required to complete this project. In this context, the two omitted tables can be seen as future work with regards to the presented architecture.

Following the proposed DB design, the data is further divided into a set of `.csv` files matching the corresponding DB tables. Within this newly created files, matching values written in different formats are standardised using either bulk edit approaches such as regular expressions (e.g. to format numerical values) or using predefined mappings (which are for the most part highlighted in the `Mappings.xlsx` file).

In the `commodities.csv` file, changes were made to the following fields:

- Max THI: The > symbol was removed
- Min temperature: When dealing with ranges, the minimum value was selected
- Average temperature: When dealing with ranges, the average value was computed. Unit of measure symbols and words were removed.

- Max precipitation: Unit of measure symbols were removed.
- Average precipitation: When dealing with ranges, the average value was computed. Unit of measure symbols were removed. The > symbol was removed.
- Min elevation: The < symbol was removed
- Max elevation: The > symbol was removed. Unit of measure symbols and words were removed.

The `Livelihood source` attribute was added to the `.csv` file related to the `Commodities` table in order to link the `Livelihood sources` table with the corresponding commodities a livelihood source generates in the context of a given farming system. In this regard, some missing information had to be added to the the `Livelihood sources` table in order for the foreign key in the `Commodities` table to hold.

In the `farming_systems.csv` file, average values were computed for the `Farm size (ha)` when dealing with ranges.

The few `sources` (`resources.csv` file) without a specified year were assigned the resource year of 2021 (the year the CRISP project started) since this was likely to match the year in which they were consulted. This mostly refers to websites for which a unique publication year could not be assigned. Some of the sources were mentioned in the factors and connection section of the corresponding source files but were ultimately missing from the corresponding resources tab. When possible, said resources were manually retrieved online and added as entries in the resources table. The resources for which it was not possible to univocally retrieve a given title and url were added to the resource table with only the citation key and the year.

2.4 IC discovery

As previously mentioned, in the context of this project the IC discovery algorithms were executed *after* the DB design was created (and the corresponding data reformulation phase was concluded). In this context, the output of such algorithms was meant to indicate whether the content of the newly generated files conflicted in any way with the integrity constraints imposed by the database.

2.4.1 Unique column combinations

The code for the implemented UCC discovery solution is associated with the `UCC_discovery()` method located in the `DP/discovery_algorithms.ipynb` document. The proposed solution, which follows the pseudocode presented during the course, represents an implementation of the Bottom-Up Apriori approach which uses the efficient candidate generation discussed in Section 3.1 of [8]. The full result generated by this implementation and related to all source files containing more than one attribute is located in the `DP/IC_dump` directory (`UCCs.txt` document).

The content of this result document highlights that for some tables, unique column combinations containing fewer attributes than the proposed primary keys for the corresponding table were found. In this context, it is taken into account that a minimal UCC found by the algorithm does not automatically constitute a valid primary key when designing the database architecture. This is due to the fact that the underlying data might suggest key candidates that are logically unsuitable for the role (e.g. using a description field as primary key). Such results are manually inspected and the choice of primary keys proposed in 2.3 is further confirmed.

2.4.2 Functional dependencies

For the discovery of exact functional dependencies, an algorithm following the principles of TANE [9] has been implemented starting from the pseudocode presented during the course. Such implementation is represented by the `FD_discovery()` method located in the `DP/discovery_algorithms.ipynb` document. The document resulting from the execution of the algorithm on the underlying data can be found in the `DP/IC_dump` directory (`FDs.txt` file). Unfortunately, strictly following the presented pseudocode implementation leads to some FDs not being discovered during the search process. In particular, the proposed implementation

seems to be failing to generate FDs where both the left- and right-hand side of the dependency are primary keys, and the RHS has a smaller set of attributes than the corresponding LHS.

An example of such behavior can be observed in the context of the `farming_systems.csv` file by taking as LHS candidate the set of attributes {`Description, Farming System`}. In this context, the proposed implementation correctly generates the FDs
{`Description, Farming System`} $\rightarrow$ `Agro-ecological zone` and
{`Description, Farming System`} $\rightarrow$ `Farm size (ha)`.

On the other hand, the FD
{`Description, Farming System`} $\rightarrow$ `Total area (m ha)` is not generated since the attribute `Total area (m ha)` is also a primary key. The source of this behavior (which would require further investigation in order to be fixed) is likely to be found in the `FD_prune()` method, were the pruning of keys takes place.

To avoid this issue, the full set of dependencies is generated using the TANE implementation offered by Metanome and stored in the `DP/IC_dump/FDs_metanome.csv` file. Once again, the resulting dependencies are manually inspected and evaluated against the proposed DB design. Clearly, the knowledge that lackluster data might have influenced their generation is kept into account during this process.

2.4.3 Inclusion dependencies

In the `discovery_algorithms.ipynb` document two implementations for the discovery of unary inclusion dependencies can be found. The first (`unary_IND_discovery()`) method mirrors the solution discussed during the course. In this context, the method takes a set of interested attributes ($\mathbb{U}$), the related set of values found in the instances ($\mathbb{V}$) and the binary relation ($\mathbb{B}$) computed from the two with regards to the values contained in the instances at hand and computes (for the given domain) the set of discovered unary INDs. Additionally, the method generates a printout for the non trivial dependencies found during the search.

On the other hand, the method `unary_IND_discovery_simplified()` accepts two specific columns (belonging to different tables) and checks for unary inclusion dependencies solely among the two attributes passed as parameters. In this context, the discovery of a valid IND is reduced to a simple `set` comparison (performed using the `issubset()` method). Given the fact that the discovery of integrity constraint is being performed *after* the DB design phase, the inclusion

dependency discovery algorithm mainly serves the purpose of verifying that each of the chosen foreign key constraint holds with regards to a given dependent and corresponding referenced attribute. In other words, the purpose of the `unary_IND_discovery_simplified()` is to verify that that the data contained in the newly created `.csv` files does not conflict with the proposed DB architecture. This second method has therefore been used to ensure that the attributes that were part of foreign keys were actually contained in the referenced attributes. In this case, a full dump containing all the test unary INDs has not been created. On the other hand, a few examples highlighting how the method can be used are found in `discovery_algorithms.ipynb`.

As an additional activity, the Metanome tool has been used to generate a complete document which also includes the n-ary inclusion dependencies stemming from the data sources at hand. In this context, the Binder [10] algorithm has been used to complete this task. The full set of results stemming from this execution can be found in the `IC_dump` folder (`IND_metanome.csv` document)

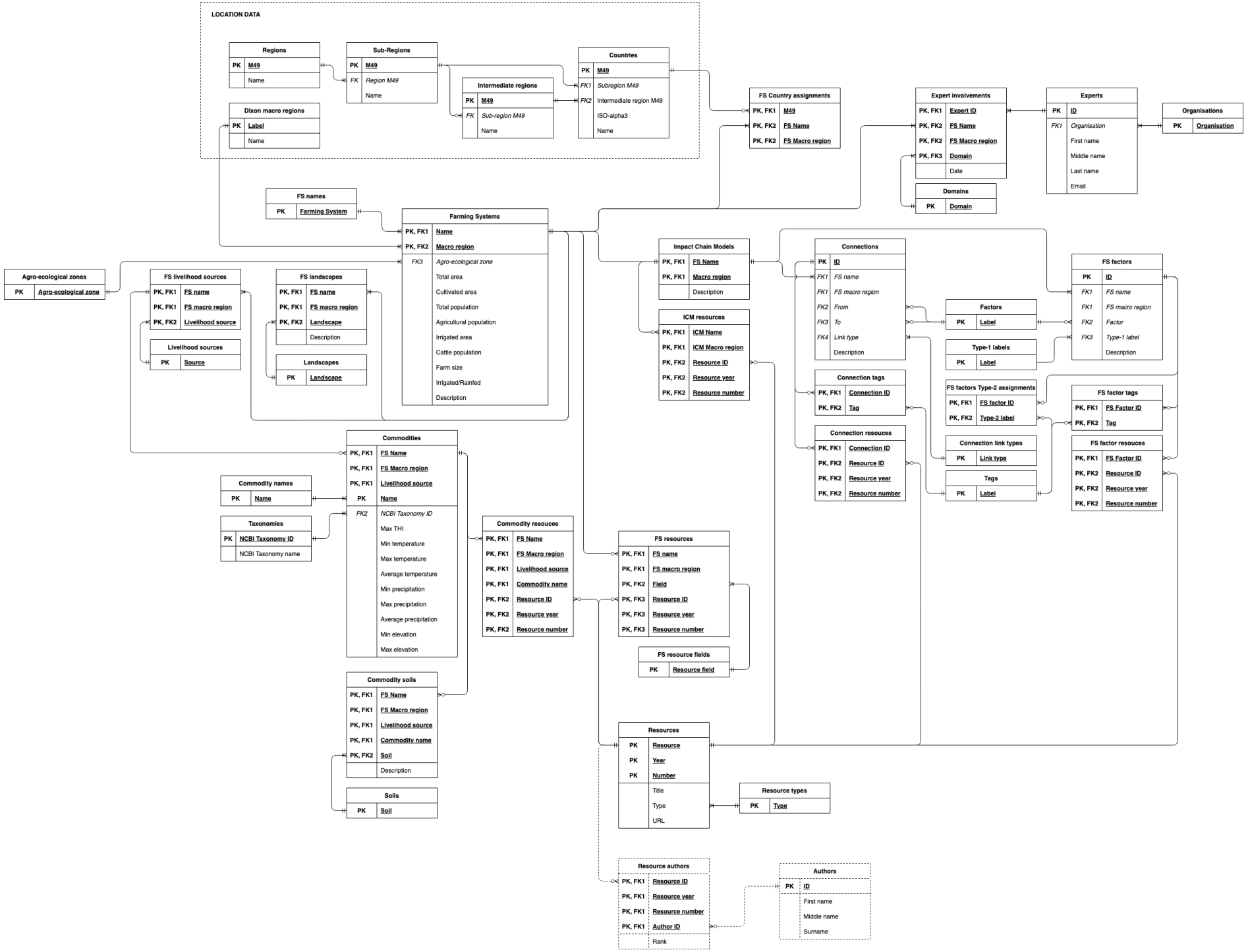


Figure 2.2: ERD of the subject domain

Chapter 3

Data Preparation and Integration

3.1 Ontology

Starting from the proposed database architecture, an ontology design is proposed. The proposed ontology, which is highlighted using the graphical notation in Figure 3.1, is made up of 20 classes, 26 object properties and 59 data properties. All ontology-related files can be found in the `DPI/Ontology` directory. Notice that this also includes the additional `Dixon_FS-materialized.ttl` file which represents the materialised ontology saved using the turtle syntax. In a general sense, the ontology follows for the most part the structure seen in the relational database. Naturally, in this context most junction tables seen in the original DB (e.g. `Commodity resources`) are represented as object properties rater than classes. Additionally, it can be noticed how a class hierarchy is established among the entities generated from tables containing location data due to their shared fields. One of the major differences with regards to how the data is structured in the underlying database is represented by the `Factor` and `Factor_descriptor` classes. In this context, it can be noticed how both of these classes are essentially generated from the same underlying table (`FS factors`). Furthermore, the generated instances for the `Factor` class stem from the execution of a group by statement on the underlying `FS factors` table. The choice of modelling factors and the corresponding descriptors separately was justified by the fact that in some instances a single factor (recognised by a given name and type-1 label) was present (with potentially different descriptions and resources) more than once for a given farming system. Because of this reason, in these instances it was not possible to uniquely tie the `From` and `To` fields of a given connection to a single factor's instance. This problem is addressed

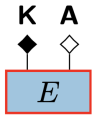
in the proposed design, were repeated factor entries are represented by multiple instances of the `Factor_descriptor` class. In this context, it can also be noticed that there exists some redundancy in the aforementioned `From` and `To` fields related to a given `Connection` instance. In this context, we have that these information are represented by two corresponding data and object properties. Introducing this slight redundancy was required due to the fact that (in some instances) the factors mentioned within a given connection instance were not found as a related `Factor` instance within the same farming system. In these situations, having only the `connectionFrom` and `connectionTo` data properties would result in missing data for a given connection.

Once again, it can be noticed how the (two) classes related to the authors' information (and corresponding link to the related resources) are represented with dotted lines, indicating that they were ultimately not included in the final version of the ontology.

3.2 Mappings

In order to map classes, data and object properties to the underlying data source (i.e. the relational database), a set of mapping assertions is proposed. The mappings are written using the concrete mapping language used by the Ontop system (as opposed to the more widely used R2RML mapping language). All the presented mapping assertion are manually written using the Protégé mapping manager. The choice to manually write the mappings from scratch rather than using the bootstrapping functionality offered by the Ontop system is derived from a desire to better understand the process of designing a given mapping assertion starting from a formulated ontology. All mappings can be accessed under `Ontop Mappings` → `Mapping manager` in Protégé after opening the project located in the `DPI/Ontology` directory or more directly by opening the `DPI/Ontology/Dixon_FS.obda` file. When possible, the introduced mapping patterns are utilised when writing assertions in the context of the presented ontology and underlying database. Naturally, the inherent complexity brought upon by working on real-world data does not necessarily allow for all parts of the ontology to be mapped using existing patterns. In this section, the different mapping patterns utilised in the context of this work are presented together with the assertions on which they are applied. Additionally, fundamental differences between a given mapping assertion and the corresponding (closest) pattern are also highlighted.

3.2.1 Mapping pattern: Entity (MpE)



Relational schema and constraints:
$T_E(\underline{\mathbf{K}}, \mathbf{A})$

Mapping assertion:
$s : T_E$
t : :E/{K} rdf:type C_E .
{ :E/{K} d_A {A} . }$_{A \in \mathbf{K} \cup \mathbf{A}}$

Ontology axioms:
{ $\exists d_A \sqsubseteq C_E$ }$_{A \in \mathbf{K} \cup \mathbf{A}}$

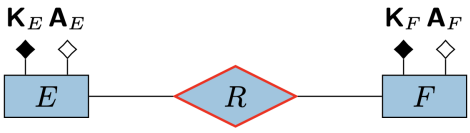
For the application of the mapping pattern, we observe the following:

- This fundamental pattern considers a single table T_E with primary key $\underline{\mathbf{K}}$ and other relevant attributes $\mathbf{A}$.
- The pattern captures how T_E is mapped into a corresponding class C_E.
- The primary key $\underline{\mathbf{K}}$ of T_E is used to construct the objects that are instances of C_E, using a template :E/{K} specific for C_E.
- Each relevant attribute of T_E is mapped to a data property of C_E.

Figure 3.2: Mapping pattern: Entity (MpE)

This simple `Entity` pattern (Figure 3.2), which ties a given entity with a single underlying table, has been used in the proposed ontology to map the following classes: `Expert`, `Landscape`, `Soil`, `Resource`, `Farming system`, `Livelihood source`, `Impact Chain Model`, `Connection`, `Commodity` (involves a left join with the `taxonomies` table in order to get the `ncbi_taxonomy_name` attribute), `Factor` (involves a `group by` on the attributes `fs_name`, `fs_macro_region`, `factor`, `type_1_label`), `Factor descriptor`, `Dixon macro region`, `Country`, `Intermediate region`, `Sub-region`, `Region`.

3.2.2 Mapping pattern: Relationship (MpR)



Relational schema and constraints:
$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$
$T_R(\underline{\mathbf{K}_{RE}}, \underline{\mathbf{K}_{RF}})$

Mapping assertion:
$s : T_R$
t : :E/{K$_{RE}$} p_R :F/{K$_{RF}$} .

Ontology axioms:
$\exists p_R \sqsubseteq C_E$
$\exists p_R^- \sqsubseteq C_F$

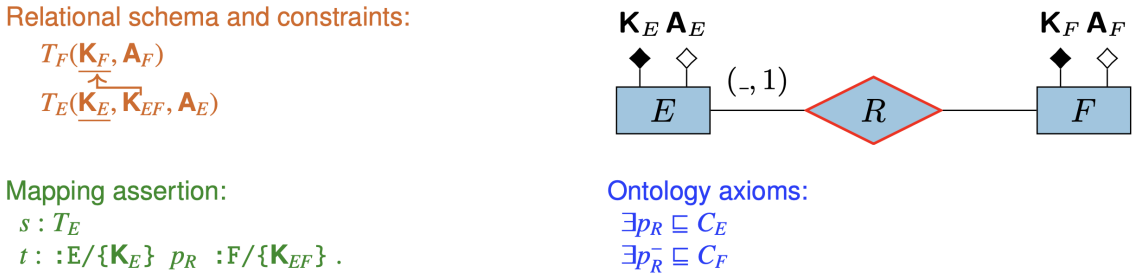
For the application of the mapping pattern, we observe the following:

- This pattern considers three tables T_R, T_E, and T_F.
- The primary key of T_R is partitioned into two parts $\mathbf{K}_{RE}$ and $\mathbf{K}_{RF}$ that are foreign keys to T_E and T_F, respectively.
- T_R has no additional (relevant) attributes.
- The pattern captures how T_R is mapped to an object property p_R, using the two parts $\mathbf{K}_{RE}$ and $\mathbf{K}_{RF}$ of the primary key to construct respectively the subject and the object of the triples in p_R.

Figure 3.3: Mapping pattern: Relationship (MpR)

The `Relationship (MpR)` mapping pattern (Figure 3.3) was used for some of those junction tables present in the underlying database (with primary keys partitioned into two parts) that were ultimately represented as object properties in the ontology. In this context, the following data properties are mapped using this pattern: `commoditySourcedFrom`, `icmSourcedFrom`, `connectionSourcedFrom`, `FDSourcedFrom`.

3.2.3 Mapping pattern: Relationship with Merging (MpRm)



For the application of the mapping pattern, we observe the following:

- Such pattern is characterized by a table T_E in which the foreign key $\mathbf{K}_{EF}$ to a table T_F is disjoint from its primary key $\mathbf{K}_E$.
- The table T_E is mapped to an object property p_{EF}, whose subject and object are derived respectively from $\mathbf{K}_E$ and $\mathbf{K}_{EF}$.

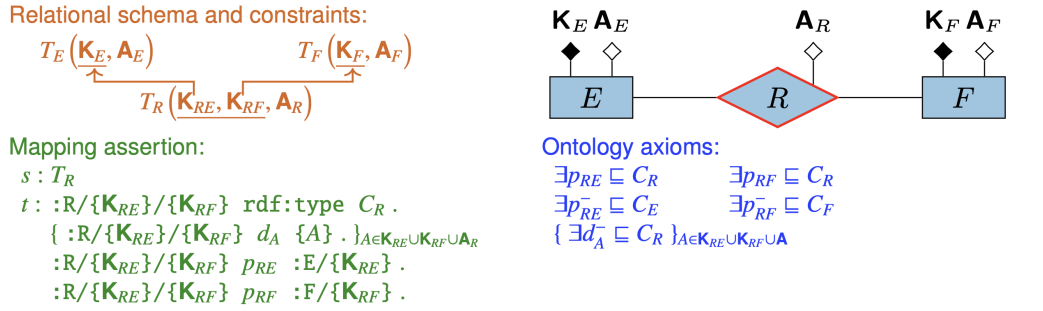
Figure 3.4: Mapping pattern: Relationship with Merging (MpRm)

The `Relationship with Merging (MpRm)` mapping pattern (Figure 3.4) is used for all those instances of object properties represented by a foreign key constraint in the underlying database. The following object properties are represented using this mapping pattern: `countrySituatedInSubRegion`, `countrySituatedInIntermediateRegion`, `intermediateRegionSituatedInSubRegion`, `subRegionSituatedInRegion`, `characterisedByFactor`, `describedByConnection`.

Additionally, for the following object properties it can be noticed how K_{EF} is part of the composite primary key of T_E: `producesCommodity`, `hostsFarmingSystem`, `reliesOnLS`, `takesPlaceInLandscape`, `foundInCountry`, `growsInSoil`.

Finally, for one object property the full primary key of T_E is also a foreign key referencing the primary key of T_F (the two tables are in a one-to-one relationship): `subjectToICM` (T_E = `impact_chain_models`, T_F = `farming_system`)

3.2.4 Mapping pattern: Reified Relationship (MpRR) – Attribute case



For the application of the mapping pattern, we observe the following:

- The pattern applies to a table T_R whose primary key is partitioned in (at least) two parts $\mathbf{K}_{RE}$ and $\mathbf{K}_{RF}$ that are foreign keys to additional tables, and there are additional attributes $\mathbf{A}_R$ in T_R.
- Since T_R corresponds to a conceptual element that has itself properties (corresponding to $\mathbf{A}_R$), to represent it in the ontology we require a class C_R whose instances have an IRI :R/{$\mathbf{K}_{RE}$}/{$\mathbf{K}_{RF}$}.
- The mapping ensures that each components of the relationship is represented by an object property (p_{RE}, p_{RF}), and that the tuples instantiating them can all be derived from T_R alone.

Figure 3.5: Mapping pattern: Reified Relationship (MpRR) – Attribute case

The `Reified Relationship (MpRR)` mapping pattern (Figure 3.5) was used for the two classes (and corresponding object properties) for which a reification transformation was applied. In this context, the associated classes are: `Expert involvement` (together with the `involvementByExpert` and `involvementInFS` object properties), `Farming System sourcing` (together with the `sourcedForFSField` and `FSFieldSourcedFrom` object properties),

Conceptually, this mapping pattern would have been applied also to the `Autorship` class if it had been ultimately introduced in the ontology.

3.2.5 Other mappings

Three of the mappings related with the `Factor` class were not found to found to match any of the presented patterns. The interested mappings are: `connectionFrom`, `connectionTo`, `describedByFD`.

3.2.6 SQL query simicolon error

When materialising the triples via the generated mappings (`Ontop` → `Materialize triples`), some queries would raise the error highlighted in Figure 3.6 when saved in a given mapping with a trailing semicolon.

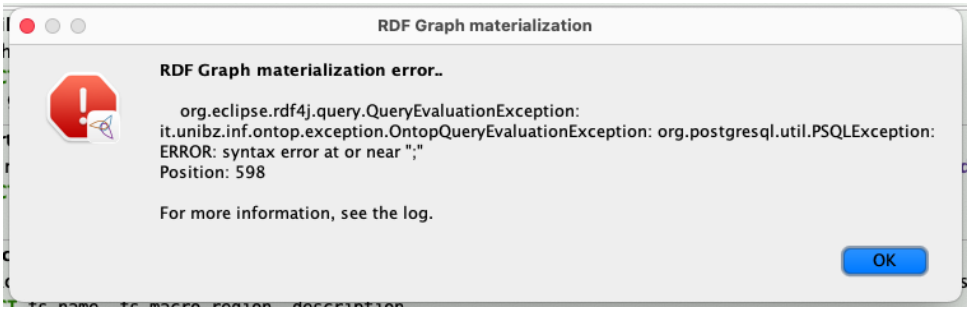


Figure 3.6: Ontop error raised by SQL query with trailing semicolon

Some examples of queries raising this error where the ones related to the `Commodity` and `Factor mappings`. Interestingly enough, the error was present when formulating the source SQL query with the semicolon:

```
SELECT  c.fs_name, c.fs_macro_region, c.livelihood_source,
        c.name, c.ncbi_taxonomy_id, t.ncbi_taxonomy_name,
        c.max_thi, c.min_temperature, c.max_temperature,
        c.average_temperature, c.min_precipitation, c.max_precipitation,
        c.average_precipitation, c.min_elevation, c.max_elevation
  FROM global_schema.commodities c
  LEFT JOIN global_schema.taxonomies t
        ON c.ncbi_taxonomy_id = t.ncbi_taxonomy_id;
```

But otherwise absent if an equivalent reformulation (with semicolon) was provided:

```
SELECT  c.fs_name, c.fs_macro_region, c.livelihood_source,
        c.name, c.ncbi_taxonomy_id, c.max_thi,
        c.min_temperature, c.max_temperature, c.average_temperature,
        c.min_precipitation, c.max_precipitation, c.average_precipitation,
        c.min_elevation, c.max_elevation,
        (
            SELECT t.ncbi_taxonomy_name
                FROM global_schema.taxonomies t
                WHERE  c.ncbi_taxonomy_id = t.ncbi_taxonomy_id
        )
    FROM global_schema.commodities c;
```

Additionally, the system did not prompt any errors if a (non-equivalent) version of the first query featuring an inner join (with trailing semicolon) was used. Event though most mappings were not affected by this issue. It was nonetheless decided to remove the trailing semicolon entirely from the SQL queries related to all the written mappings.

3.3 Application description

In the context of this project, a web app written in the Python programming language is proposed as an interface to access the described ontology and related data. The application makes use of the `PyWebIO v1.7` library [11]. The library offers a straightforward way to build web-based GUIs without requiring additional knowledge on web development frameworks. In this context, UI elements are created using the offered output functions (e.g. `put_markdown()`, `put_image()`, etc.) and are grouped together in output content containers called `scopes`. The data used to populate such containers is obtained by executing SPARQL queries on the active endpoint. In this regard, the `sparql-dataframe v0.4` package is used to automatically convert the `SPARQLWrapper` results obtained from executing the queries into Pandas dataframes.

From a usability standpoint, the application can be seen as an interactive markdown document. In this context, the user is initially asked to choose a macro region of interest. The application will then present a complete list of farming systems associated with the selected region. Upon selecting a given farming system, the following set of information are presented:

- The list of the **experts** involved in describing the given farming system, together with their role, email and organisation.
- A `description` of the farming system.
- A set of (mostly numerical) indicators, together with associated icons.
- The `resources` associated with the presented description and indicators.
- The `landscapes` found in the farming system.

At the bottom of the page, the application will show a "What else?" section that the user can use to generate additional information. In this regard, the user can:

- Generate a searchable table containing the **countries** found in the farming system.
- Create a **livelihood source** section containing the set of livelihood sources on which the farming system relies on. Additionally, a subsection is created for each livelihood source indicating its related **commodities**. In this regard, each commodity is displayed in a dedicated tab together with the related indicators (each associated with a relevant icon), the set of **soils** in which the commodity is produced and the **resources** from which the commodity information have been obtained.

- Build the **impact chain model** section which will contain its description and a tabbed pane containing the following two tabs:
  - A **Factors** tab containing a list of all factors associated with the selected farming system. The list can be searched based on the factors names and type-1 labels. Additionally, an entry can include up to two collapsable UI element containing the related **descriptors** and **connections**. Each descriptor can contain a description and one or more type-2 labels and tags and can be presented with the related resources.
  - A **Connections** tab including the full list of connections between factors. In this context, each connection has the format `Link_from` → `Connection_type` → `Link_to`. Additionally, a connection might contain a description, a (set of) tag(s) and a set of related resources. Once again, the list of connection can be freely searched.

All the **resources** presented in the various section of the application will include a citation key, a publication year and a resource type indicator. Additionally, each resource might also include a title and a url.

Notice that the application will dynamically add and remove sections and fields depending on the available information for a given farming system.

3.4 SPARQL queries used by the application

The described application uses a total of 19 SPARQL queries in order to obtain the required data. In this context, the individual queries used by `DPI/Application/app.py` are sometimes represented as multiple (string) fragments and built dynamically when the application is executed depending on the task at hand. The full set of queries can be explored in the `Ontop SPARQL` tab in Protégé by opening the `DPI/Ontology/Dixon_FS.owl` project or more directly by inspecting the `DPI/Ontology/Dixon_FS.q` file.

To avoid excessive clutter, it has been decided not to include the queries in the presented report.

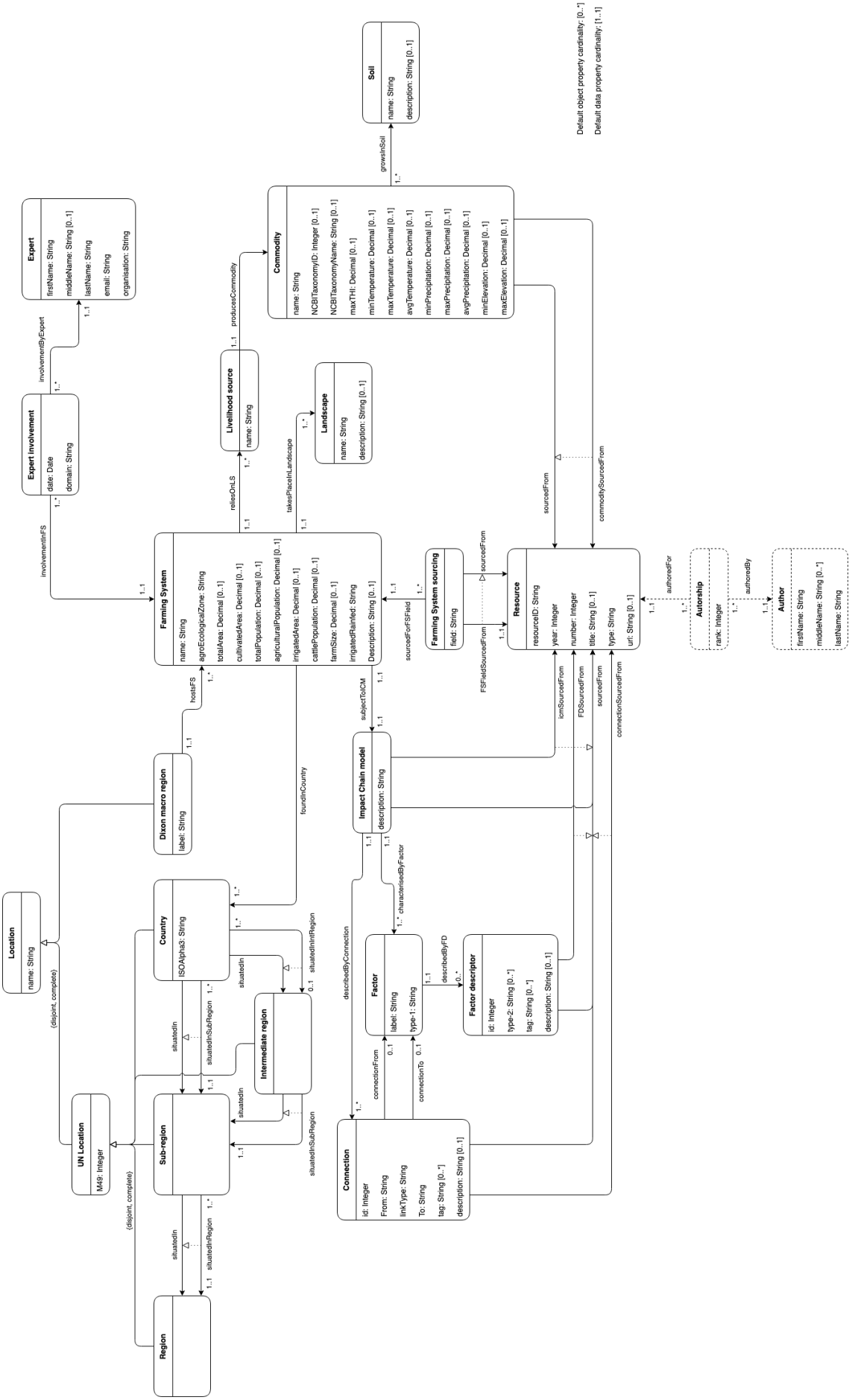


Figure 3.1: Ontology represented via the graphical notation

Chapter 4

Running the Application

4.1 Restoring the PostgreSQL DB

The presented database can be restored from the `DB_dump.sql` file found in the `DP` directory. In order to do so, open pgAdmin and create a new database (`Create` → `Database...`). Right click on the newly created database and select `Restore`. Then, select `Custom or tar` on the `Format` entry and provide the path to the `DB_dump.sql` file under the `Filename` voice. Finally, press `Restore` and wait for the process to finish. The restore page and the related settings are highlighted in Figure 4.1. If the process completes successfully, the relevant tables should be made accessible under `Schemas/global_schema/Tables`.

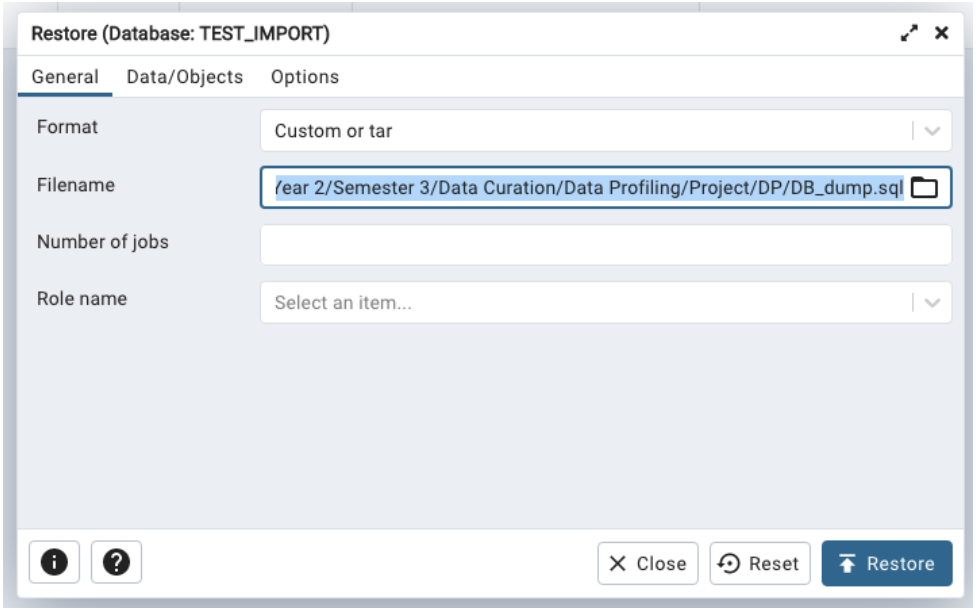


Figure 4.1: DB restore page in pgAdmin

4.2 Connecting Protégé to the Database

In the context of this project, `Protégé v5.5.0` and the related `Ontop ODBA Protégé plugin v4.2.2` were used.

In order to connect Protégé with the generated DB, open Protégé and go to `Protégé` → `Settings...` → `JDBC Drivers` and click `Add`. Under `Class name` select `org.postgresql.Driver`, provide a description and upload (under `Driver file (jar)`) the relevant driver by selecting the `postgresql-42.5.1.jar` file located in the `DPI/PostgreSQL JDBC Driver` folder.

Then, go to `Ontop mappings` → `Connection parameters` and insert the relevant DB parameters. Verify that the connection has been successfully established by pressing the `Test Connection` button. A successful DB connection is shown in Figure 4.2.

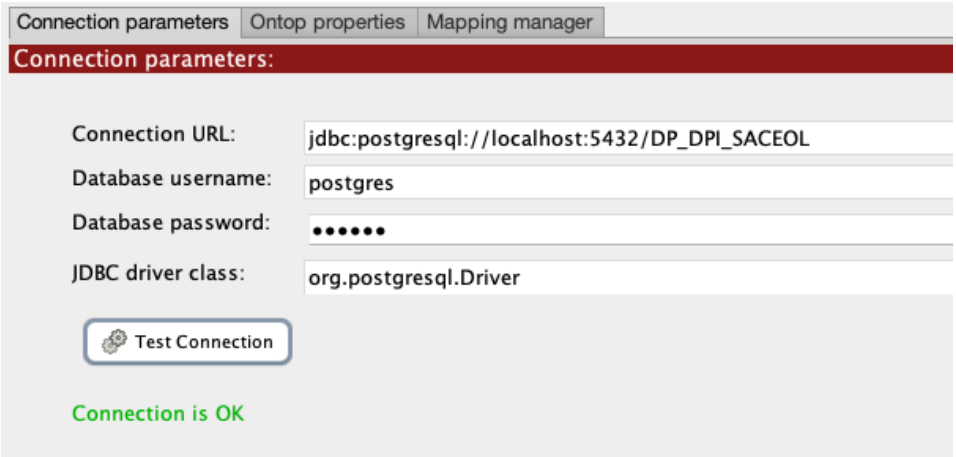


Figure 4.2: Successful DB connection in Protégé

4.3 Setting up the Ontop SPARQL endpoint

In order for the application to issue queries to the ontology (and the underlying database), a SPARQL endpoint needs to be made available. In this regard, the files related to the `Otop CLI v5.0.0` have been downloaded from [12] and can be found in `DPI/ontop_cli`.

In order to successfully start the endpoint, copy to `DPI/ontop_cli/input` the `Dixon_FS.owl`, `Dixon_FS.obda` and `Dixon_FS.properties` files found in the `DPI/Ontology` folder. Then, open a terminal, travel to the `ontop_cli` directory and exetute the following command:

```
./ontop endpoint \
    --ontology=input/Dixon_FS.owl \
    --mapping=input/Dixon_FS.obda \
    --properties=input/Dixon_FS.properties
```

A successful execution of the Ontop SPARQL endpoint can be verified at `http://localhost:8080`

4.4 (Optional) Creating the conda environment

If needed, the application can be executed within a conda environment. To do so, after having installed Anaconda, open a terminal and move to the `DPI/Application` directory. From within the directory, run the following command:

```
conda env create -f environment.yml
conda activate DP_DPI
```

4.5 Running the application

Once the SPARQL endpoint has been succesfully started, open a new terminal window and move to the `DPI/Application` directory (skip this step if you created and activated the conda environment and you are already in the `DPI/Application` directory). From the terminal, run the following command:

```
python3 app.py -p PORT_NR
```

Where `PORT_NR` is the port number were the application will be executed on (the default value is 8081). Upon successful execution, the application should open automatically in the browser.

References

[1] *CRISP project landing page*. https://www.eurac.edu/en/institutes-centers/institute-for-earth-observation/projects/crisp.

[2] *GIZ landing page*. https://www.giz.de/en/html/index.html.

[3] Dixon and Gibbon. *Global Farming Systems Study: Challenges and Priorities to 2030*. http://www.international-food-safety.com/pdf/GFSS.pdf.

[4] Marc Zebisch, Stefan Schneiderbauer, Kerstin Fritzsche, Philip Bubeck, Stefan Kienberger, Walter Kahlenborn, Susanne Schwan, and Till Below. "The vulnerability sourcebook and climate impact chains – a standardised framework for a climate vulnerability and risk assessment". In: *International Journal of Climate Change Strategies and Management* 13.1 (Feb. 2021), pp. 35–59. DOI: 10.1108/ijccsm-07-2019-0042. URL: https://doi.org/10.1108%2Fijccsm-07-2019-0042.

[5] Giz and EURAC Research. *The Vulnerability Sourcebook - Risk supplement*. https://www.adaptationcommunity.net/publications/risk-supplement-to-the-vulnerability-sourcebook/.

[6] *CRISP data access website*. https://crisp.inf.unibz.it/.

[7] *Standard country or area codes for statistical use (M49)*. https://unstats.un.org/unsd/methodology/m49/.

[8] Ziawasch Abedjan and Felix Naumann. "Advancing the Discovery of Unique Column Combinations". In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM '11. Glasgow, Scotland, UK: Association for Computing

Machinery, 2011, pp. 1565–1570. ISBN: 9781450307178. DOI: 10.1145/2063576.2063801. URL: https://doi.org/10.1145/2063576.2063801.

[9] Yká Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. "Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies". In: *The Computer Journal* 42.2 (1999), pp. 100–111. DOI: 10.1093/comjnl/42.2.100.

[10] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. "Divide & Conquer-Based Inclusion Dependency Discovery". In: *Proc. VLDB Endow.* 8.7 (Feb. 2015), pp. 774–785. ISSN: 2150-8097. DOI: 10.14778/2752939.2752946. URL: https://doi.org/10.14778/2752939.2752946.

[11] *PyWebIO Documentation*. https://pywebio.readthedocs.io/en/latest/.

[12] *Ontop CLI*. https://github.com/ontop/ontop/releases.