

Master Thesis

Efficient Database Model to Represent Numerical Research Data of Material Flow Analysis

Mohammad Mahadi Hasan

14 May 2018

Albert-Ludwigs-Universität Freiburg
Faculty of Engineering

Chair of Algorithms and Data Structures
Professor Dr. Hannah Bast

Supervisor

Professor Dr. Hannah Bast

First Reviewer

Professor Dr. Hannah Bast

Second Reviewer

Junior Professor Dr. Stefan Pauliuk

Table of Contents

Abstract

Zusammenfassung

Declaration

Acknowledgement

1. Introduction.....	1
1.1. Motivation and Problem Statement.....	1
1.2. Related Work and Research Gap	2
1.3. Goal and Approach.....	3
1.4. Structure of the Thesis	4
2. Background	5
2.1. Relational Database (RDB).....	5
2.1.1. Relational Database Management System (RDBMS).....	6
2.1.2. MySQL Database Architecture	6
2.1.3. Structured Query Language (SQL).....	8
2.1.4. RDB Schema, Tables and Constraints.....	8
2.2. The Semantic Web	10
2.2.1. Semantic Web and Linked Open Data (LOD).....	10
2.2.2. RDF – Data Model	11
2.2.2.1. Triples	12
2.2.2.2. URIs, Vocabularies & Ontologies	13
2.2.2.3. Serialization Formats	14
2.2.3. SPARQL – Query Language	14
2.3. Triple Stores	16
2.3.1. Architecture	16
2.3.2. Apache Jena.....	17
3. Construction of Relational Database	18
3.1. MySQL Database Design Process	18
3.1.1. Creation of Schema, Tables, and Constraints.....	18
3.1.2. Loading Data into Database	20

3.2. Implementation of the Web Interface.....	21
3.2.1. Back-End Development.....	22
3.2.1.1. Database Connection	22
3.2.1.2. Data Access.....	23
3.2.1.3. WCF Data Service with Ajax Call.....	24
3.2.2. Front-End Development	25
3.2.2.1. Data Presentation with JqGrid	25
3.2.2.2. Interface Styling with Bootstrap	25
3.2.3. Hosting with IIS.....	26
3.3. Structural Analysis of Web Interface	26
4. Construction of Triple Store	30
4.1. Mapping Relational Database to RDF	30
4.1.1. Mapping Languages and Platforms	30
4.1.2. D2RQ.....	31
4.1.2.1. D2RQ Overview and Features	32
4.1.2.2. D2RQ Mapping Process	32
4.2. Loading Data into Triple Store.....	34
4.2.1. Apache Jena TDB	34
4.2.2. Loading Triples into TDB	35
5. Database Performance Analysis	36
5.1. Experimental Settings	36
5.2. Results Evaluation.....	37
6. Discussions	39
6.1. Future Work	40
6.2. Conclusion.....	40
List of Figures.....	43
List of Listing.....	44
List of Tables	45
Appendix.....	46
Bibliography	51

Abstract

The research group of Industrial Ecology Freiburg (IEF) produces research results consisting material stock and flow data between industrial sectors and regions. Most of these data are stored in .mat or .csv files on a local machine which makes data accessibility difficult to other users. Usually these datasets are big, and it is difficult to re-generate a specific part (based on specific region or industrial sector) on request from another interested user. An efficient data storing and retrieving mechanism could provide a solution to this problem. While, Relational Databases (RDB) have been meeting this criterion for decades, Triple Store is a new kind of database which has grown in popularity in recent years due to its applications towards Semantic Web Technology which assures that data to be shared and reused.

In this thesis, we provide a study on both relational databases and triple stores. We start with constructing a relational database from our dataset and design a web interface so that users can retrieve data by their own. To examine the offerings that triple stores provide us, we design a triple store to store the same dataset. In relational databases where data is stored in relational tables, triple stores store data as semantic triples using the Resource Description Framework (RDF); a semantic web standard for publishing and sharing data on the Web. Therefore, we transform our entire dataset from the relational database to RDF and load into the triple store. Finally, we do the performance evaluation on both database models in terms of query processing time running equivalent test queries. This evaluation also includes the additional offerings triple stores and relational databases provide which summarize the argument to choose more suitable database model for industrial ecology research data.

The implemented online platform of IEF database can be found at-
<http://www.database.industrialecology.uni-freiburg.de/>

Zusammenfassung

Die Forschungsgruppe Industrielle Ökologie Freiburg (IEF) erstellt Forschungsergebnisse, die Materialbestands- und -flussdaten zwischen Industriebereichen und Regionen umfassen. Die meisten dieser Daten werden in .mat- oder .csv-Dateien auf einem lokalen Computer gespeichert, was die Datenverfügbarkeit für andere Benutzer erschwert. In der Regel sind diese Datensätze groß und es ist schwierig, auf Anfrage eines anderen interessierten Nutzers einen bestimmten Teil (basierend auf einer bestimmten Region oder einem bestimmten Industriezweig) neu zu generieren. Ein effizienter Mechanismus zum Speichern und Abrufen von Daten könnte eine Lösung für dieses Problem bieten. Während relationale Datenbanken (RDB) dieses Kriterium seit Jahrzehnten erfüllen, ist Triple Store eine neue Art von Datenbank die in den letzten Jahren an Popularität gewonnen hat, da sie für Anwendungen zur Semantic Web-Technologie sicherstellt, dass Daten geteilt und wiederverwendet werden können.

Diese Arbeit beinhaltet eine Studie zu relationalen Datenbanken und Triple Stores. Wir beginnen mit der Konstruktion einer relationalen Datenbank aus unserem Datenbestand und entwerfen eine Webschnittstelle, so dass Benutzer Daten selbst abrufen können. Um die Angebote von Triple-Stores zu untersuchen, entwerfen wir einen Triple-Store, um den gleichen Datensatz zu speichern. In relationalen Datenbanken, in denen Daten in relationalen Tabellen gespeichert sind, speichern Triple-Stores Daten unter Verwendung des Resource Description Framework (RDF) als semantische Tripel, ein semantischer Webstandard für das Veröffentlichen und Teilen von Daten im Web. Daher transformieren wir unseren gesamten Datensatz von der relationalen Datenbank in RDF und laden ihn in den Triple-Store. Schließlich führen wir die Leistungsbewertung für beide Datenbankmodelle in Bezug auf die Abfrageverarbeitungszeit durch. Diese Auswertung beinhaltet auch die zusätzlichen Angebote von Triple-Stores und relationalen Datenbanken, die analysieren, ob Triple-Store ein geeigneteres Datenbankmodell für industrielle Forschungsdaten darstellt.

Die implementierte Online-Plattform der IEF-Datenbank kann auf folgender Webseite abgerufen werden: <http://www.database.industrialecology.uni-freiburg.de/>

Declaration

I hereby declare that this Master Thesis has been composed by me based on my own work, that I have not used any sources other than those specified, and that all passages those have been taken literally or meaningfully from published writings have been specified as such. Furthermore, I declare that this thesis has not been fully or partially presented for any other test.

Place, Date

Signature

Acknowledgement

I thank Professor Dr. Hannah Bast for providing me the opportunity to work on such an interesting topic for my thesis. I acknowledge the time she has extended to me to supervise me with my thesis. I express my sincere gratefulness to Junior Professor Dr. Stefan Pauliuk for his mentoring, and guidance during the thesis. His review and comments on the drafts have been very instrumental in improving the quality of the thesis.

My deepest gratitude to my family and friends for their continuous encouragement, and support, those allowed me to concentrate fully on my thesis. I appreciate the cooperation and assistance of my classmates and colleagues.

A special thanks to Ms. Stefanie Klose for assisting me with the necessary translation.

List of Abbreviations

IEF	Industrial Ecology Freiburg
RDB	Relational Database
GUI	Graphical User Interface
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
DBMS	Database Management System
RDBMS	Relational Database Management System
DL	Description Logic
FOAF	Friend of a Friend
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
LOD	Linked Open Data
OWL	Web Ontology Language
WWW	World Wide Web
UMIS	Unified Materials Information System
MFA	Material Flow Analysis
OBO	Open Biological and Biomedical Ontology
CSV	Comma-Separated Values
API	Application Programming Interface
RDFS	Resource Description Framework Schema
IDE	Integrated Development Environment
WCF	Windows Communication Foundation
AJAX	Asynchronous JavaScript and XML
IIS	Internet Information Services
ISWC	International Semantic Web Conference
URI	Universal Resource Identifier
URL	Universal Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Chapter 1. Introduction

In this chapter, we discuss about the thesis concept, the motivation behind it, goals, and the approaches. In *section 1.1*, we present the motivation that led us to construct this thesis. We present two case scenarios which state the problem. *Section 1.2* introduces the related approaches to deal with the problem. It also covers the research gap we have identified. *Section 1.3* focuses on the goals and approaches, and the scope of this thesis. Lastly, we discuss the overall structure of the thesis in *section 1.4*.

1.1. Motivation and Problem Statement

Researchers in industrial ecology often find it difficult to access and re-use the results of their quantitative analyses [1]. The limitation of the existing data sharing mechanisms imposes a great challenge to them to share their knowledge or findings to other researchers in this field. Similarly, it creates challenges for the other potential users to access the data.

Let us present here two case scenarios:

Researcher A of Industrial Ecology Freiburg group studied on the “Global multiregional steel cycle model”. From his research, he collected data from different industrial sectors around the world and at the end, he came up with a dataset of steel cycle data (stock data belongs to one sector and flow data between sectors/regions) of different industrial sectors of all the countries within the time period between 1700 and 2008 [2]. For example, this data is stored in .mat format and can be extracted in CSV format by running a Python script.

Scenario one: Let us assume, Researcher B from same research field is interested to access this specific dataset. Instead of asking for the whole dataset, he is particularly interested on certain part of it (could be based on sector/s or country or a specified time period etc.). In response to Researcher B’s query, Researcher A has to write a custom script to extract the requested data. That’s not all, researcher A has to repeat this same process again and again every time he gets requests from other interested individuals; which is often time consuming and complex repeating process. This case scenario brings the necessity of having a well-defined database driven online platform where anybody can query the dataset and download it based on their interest.

Scenario two: The second scenario arises when Researcher A thinks about linking his dataset with another dataset or even more than one datasets. While working on steel cycle data, Researcher A found it interesting to include the findings from another researcher who is working on environmental impact caused by mass use of steel. To be precise, Researcher A wants to connect with open world data by collaborating his dataset with external resources

available on internet. Thus, his dataset become a part of global data cloud, precisely Linked Open Data Cloud.

Considering these two cases, we intend to design our thesis to facilitate data sharing and linking. Earlier, WWW was basically limited to web of inter-linked documents. Recent advancements of WWW provide us lot more opportunities than just documents interlinking. It leads us with the idea of data to be shared and re-used across different applications and community boundaries. This is where the idea of Semantic Web comes into play. Semantic Web is basically a web of data where data is structured following Semantic Web standards; such as RDF.

1.2. Related Work and Research Gap

“We are not capable of fully achieving this due to the current state of tools used in IE [Industrial Ecology] and current community practices. Although we deal with a vastly interconnected world, we are not so good at efficiently interconnecting what we learn about it.”

Industrial Ecology 2.0, 2010, Chris Davis, Igor Nikolic, Gerard P.J. Dijkema [4]

The need of sharing and re-using research data throughout industrial ecology community has been a hot topic these days. “Concerns have been raised that industrial ecology lacks a major online presence and exists primarily as an offline community” [3]. This is because, there is a lack of an online platform where researchers can share their data and knowledge.

There have been few online tools that provide this kind of platform for material flow analysis; one of the major Industrial Ecology methods. One of them is Unified Materials Information System (UMIS): An Integrated Material Stocks and Flows Data Structure [5]. UMIS is a merely graphical approach, however, it is not linked or has established a data structure neither as relational database nor in another format such as RDF.

Another approach is the STAN data structure [6] where Material Flow Analysis (MFA) data are stored in an XML structure which contains both clear text and binary data.

It has also discussed how semantic web and linked open data can play an important role to facilitate the research among industrial ecologists [4]. Unfortunately, there has not been enough effort made regarding this. We did not really find any research result of material flow analysis that has been generated or published following the Semantic Web standards. That also implies that the lack of useful and ready to use tools could be one of the reasons behind the adoption of linked data or semantic web of data in Industrial Ecology.

One of the main problems is, most of the research data are in the tabular format and they are structured in a way that is closed to the relational database model. Therefore, the necessity of mapping of tabular or relational data to RDF triples has built up. There have been few

approaches to address this mapping process. James A. Overton in his OBO (stands for Open Biological and Biomedical Ontology) Tutorial [7], has described such methodology to convert biological data from CSV using OBO namespace.

Paper [8] provides a detailed procedure to produce RDF graph from a relational database using Jena API which is similar to the techniques that we are going to cover in *chapter 4* of this thesis. This mapping process largely depends on the type of data we are going to transform. In this case, we will use our own dataset of material flows and stocks.

1.3. Goal and Approach

The goal of this thesis was to deal with the questions that arose in case scenarios in motivation and problem statement part (*section 1.1*). The relational database mostly works in a closed loop, present data from a specific database set up in a server. In contrast, triple store is able to provide a data cloud by connecting data sources. We have tried to assess which database model present more suitable way to address these case scenarios.

To deal with the first case scenario, we studied on relational database and its scopes. We investigated its design process and what advantages and disadvantages it imposes while we want to model our industrial ecology dataset. We used MySQL database because of its excellent design structure and rich technological support to handle relational database. We developed a web interface for users to query the database and produce their own dataset and download it in CSV format.

To deal with the second scenario, we investigated on the idea of the semantic web and its components; which led us to translate the data from relational database (tables with rows and columns) to RDF dataset (list of triples in the form of subject-predicate-object). There are some powerful tools available to translate RDB to RDF. D2RQ is one of them which translates RDB to RDF via direct mapping or using a mapping language. Using D2RQ, we transformed the entire dataset from relational database to RDF triples and we store these triples in a widely used RDF Triple store: Apache Jean TDB.

Once we have same data in both databases, we run several test queries to do the basic performance analysis in terms of query processing time. The architectural and design complexity of both database models will be examined within the scope of this thesis. Apart from these, we have checked the additional limitations and offerings of both databases.

1.4. Structure of the Thesis

The remainder of this thesis is structured as follows.

- In *Chapter 2* (Background), we discuss the background of the relational databases along with its components. We also talk about Semantic Web with its core technologies and Triple Stores.
- *Chapter 3* (Construction of Relational Database) reflects the relational database in further depth. This chapter covers the design process of the database with creating the database schema, tables, and constraints and loading CSV data into it. Later we discuss about the application development consisting the Front-End and Back-End implementation process of the web interface.
- In *Chapter 4* (Construction of Triple Store), we discuss the mapping process of data from the relational database to RDF triples. We talk about D2RQ and its working principles. We load these triples into the triple store and query with SPARQL.
- In *Chapter 5* (Performance Analysis), we cover the evaluation part which is mainly on experimental settings and results evaluation.
- *Chapter 6* (Discussion) provides a conclusion of the thesis and suggests some areas for improvement in future work.

Chapter 2. Background

Humans began to store information long ago, even before the computers were invented [9]. A database is an electronic storing system of structured information where information is stored and organized in a way that it can be easily accessed and managed. With the emergence of computers, the world of database and data model has changed rapidly [10].

The data model that determines the logical structure of a database is defined as database model. It also determines how data will be stored and accessed. Relational databases that follow the relational model, is the most popular and has been using successfully for past 50 years. It is kind of logical database model which uses a table-based format. Triple Store is another kind of database model which gained popularity recently. It stores data as a list of triples following the Semantic Web Standard; such as Resource Description Framework (RDF).

In *section 2.1*, we discuss relational database and its components. We discuss about Relational Database Management Systems (RDBMS); a platform to operate the relational database and Standard Query Language (SQL) that used to query RDB and one of the most common RDBMSs: MySQL. *Section 2.2* covers the semantic web and its components. We talk about the relationship between Semantic Web and Linked Open Data. Later, we discuss RDF: a data model upon which the web of semantic data is built. In *section 2.3*, we talk about Triple Store and its architecture.

2.1. Relational Database (RDB)

A relational database (RDB) is a collection of data sets organized and stored in following some pre-defined relations between tables. This idea is based on the relational model specified by Edgar F. Codd in 1970 in his famous article “A Relational Model of Data for Large Shared Data Banks” [11].

In a relational database, each table stores records (Tuples) and each record has a unique value identified by a field (attribute). Each of these tables within relational database shares at least one column field (referred as foreign key) with another table to establish relationship. Following a well-defined relationship between these tables, data can be extracted by querying with query language like SQL; A structured query language to query relational databases.

Most relational database management systems use the SQL to access the database. However, the basic is same yet different RDBMS vendors use a different kind of syntax. In this thesis, we used the SQL syntax variation of MySQL database. We discuss more about SQL in *section 2.1.3*.

Relational databases are a rich source of information worldwide. Our daily online activities are somehow directly or indirectly interacting with one or multiple databases at the same time. These operations can be either deleting, inserting or updating some data from database; hence a transaction with the database. This is one of the tasks of RDBMS.

2.1.1. Relational Database Management System (RDBMS)

RDBMS is a database management system that is used to create and maintain relational databases. RDBMS runs its own server and provides multi-user access to multiple databases at the same time. It helps database administrator to create user-based roles for specific users so that data privacy and security is also maintained when needed. Most major database management systems are relational. Popular examples include Microsoft Access, SQL Server, Oracle Database, MySQL, PostgreSQL etc.

2.1.2. MySQL Database Architecture

MySQL is an open-source relational database management system and probably the second most popular RDBMS after Oracle [12]. One of the main reasons behind its popularity is the architectural design. The MySQL architecture describes how the different components within MySQL system are related to each other. The architecture is basically a client-server system. The three-layer architecture consists of the application layer, MySQL server layer, and storage engine layer (*figure 1*).

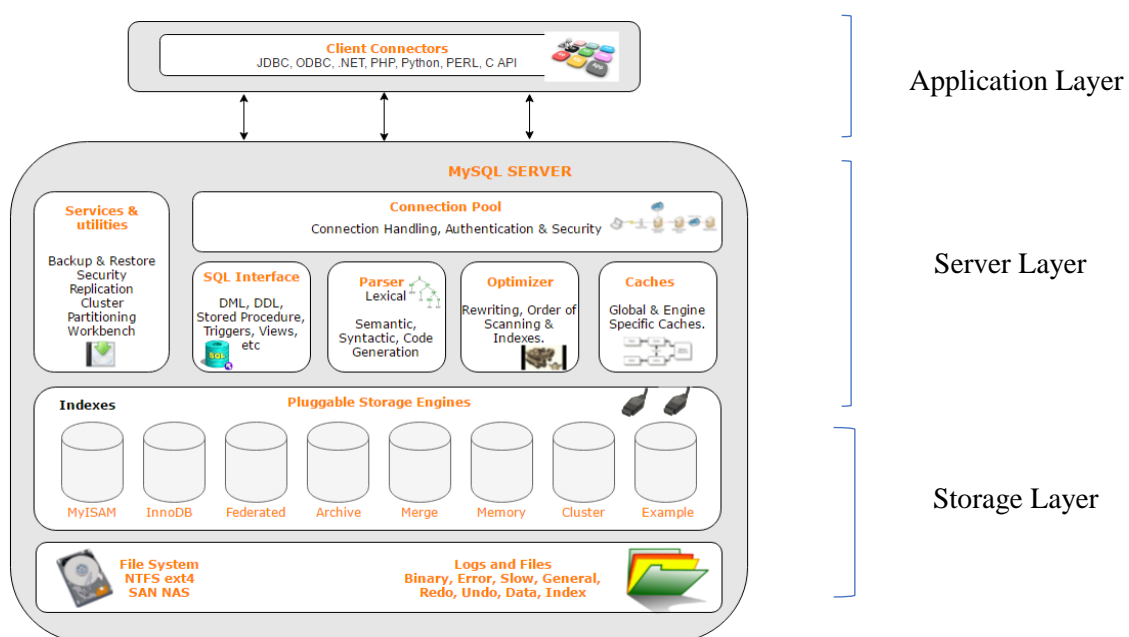


Figure 1: MySQL layer architecture

The topmost layer in MySQL architecture is the application layer. This is the client-side layer which basically handles the connection string, authentication (for example, checking the username, password, and hostname) and most importantly security (user-based role; means which user can operate what kind of operations).

The second layer is the MySQL server layer. This layer is called the brain of the overall architecture. It provides different kind of sub-components including different services, SQL interface, SQL parser, Optimizer etc. Any kind of query statement is executed in this layer.

The last layer is the storage engine layer. The pluggable storage engine and rich options to choose any of these engines depending on the requirements, made this type of database popular among users. MySQL provides several storage engines. Among them, InnoDB is most widely used. We also chose InnoDB for this thesis.

Indexes:

Indexes are used to find a data entry quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the expected rows [13]. The larger the table, the more time it will take for query processing.

Let us consider following example:

```
SELECT * FROM table WHERE id=1;
```

In a naive implementation without defining the indexes, the query will go through every row and column. After traversing all the records, it will return the rows that match the query. This will require a full-table scan and it will be slow; $O(n)$.

The optimization here is to add an index, in our case we used the table primary key as the index. This allows running the query against only the table indexes rather than all the column data. Therefore, MySQL can quickly determine the position using a faster search algorithm. Most MySQL indexes are stored in B-trees [14] which allows data selection faster with SQL SELECT query.

2.1.3. Structured Query Language (SQL)

SQL (Structured Query Language) is a standardized programming language is used for performing operations in relational databases. It was one of the first commercial languages for Edgar F. Codd's relational model [11]. SQL operations like select, insert, update, delete, create, alter etc are used to access the relational database and retrieve subset or whole dataset from the database storage. Among these operations, most important ones are collectively referred to as **CRUD** (Create, Read, Update, Delete). SQL is also able to perform different mathematical operations within the query. For example,

```
SELECT AVG(TotalAmount)
FROM [Order]
```

This simple query gives the average of the values from the column name “TotalAmount” of “Order” table.

2.1.4. RDB Schema, Tables and Constraints

RDB Schema

Relational database involves database schema, a specification of how data will be stored in the database. A database schema is nothing, but a set of formulas imposed on a database. These formulas define how data will be stored as names, data types, relationships and constraints to guarantee the real-time consistency of data.

The creation of a relational database schema is carried out with two general steps. One is creating the logical schema and the other one is creating the physical schema. The logical schema is hand sketch of the database. That means to draw the schema showing the tables, columns, and relationships with other tables and can be created using any kind of modelling tools or spreadsheet and drawing software. The physical schema is the generation of tables, columns, and relationships within tables in the relational database management software.

All kinds of relational database components are created by the CREATE statements. The first component created in a database schema is the database itself followed by creating a table. For instance, consider a database that contains information about people. The following CREATE statement creates a database schema called people.

```
CREATE DATABASE 'people';
```

RDB Tables:

A relational database consists of at least two inter-related tables (by definition). The database table is a set of data elements of vertical columns and horizontal rows. The number of columns is always specified where number of rows can be infinite depending on the data volume.

As mentioned before, tables are also created with SQL create statement (CREATE TABLE). To create a table, the general SQL statement will be like

```
CREATE TABLE database_name.table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
    columnN datatype  
);
```

Listing 1: SQL query syntax to create a table

RDB Constraints:

Constraints are important components of database schema and they are used to specify rules for the data in a table. There are several types of constraints and they are usually defined within the CREATE TABLE statement. It is also possible to assign constraints later with SQL ALTER statement.

The following constraints are commonly used in SQL:

- NOT NULL - It means that a column cannot have a NULL value.
- UNIQUE - Ensures that all the values in a column are unique; no duplication is allowed.
- PRIMARY KEY - This is basically a combination of a NOT NULL and UNIQUE. It is a very important constraint in relational database as defining primary key is the prerequisite to relate a table to other tables. Primary key uniquely identifies each row in a table and it can also be used as INDEX.
- FOREIGN KEY - It is used to establish linking between tables. It matches with the primary key of another table and relate multiple tables.
- CHECK – This is used to impose any specific rules or condition that a column value has to meet.
- DEFAULT - Sets a default value for a column when no value is given or specified.
- INDEX - It is important to create and retrieve data from the database very quickly.

Among these constraints, we have a rich use of primary and foreign keys in our dataset. We define one column from each table as primary key and use this primary key as index. We also specified columns as foreign keys to establish relationship with other tables. In *Chapter 3*, we will discuss more about these constraints and talk about how they are assigned.

2.2. The Semantic Web

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

T. Berners-Lee, J. Hendler, O. Lassila, et al.; 2001, The Semantic Web [16]

The Semantic Web is the extension of the current web where information is given in a well-defined meaning and leads to a better collaboration between computers and humans [11]. It is considered to be the next evolutionary stage of World Wide Web (WWW). According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [15]. The term was first coined by Tim Berners-Lee while he was defining the web as "web of data" that will be machine-readable.

While the representation of WWW today is based on linking documents which are mainly human-readable, the development of Semantic Web leads towards the development of a *Web of Data* that makes Web content more *machine-readable*.

2.2.1. Semantic Web and Linked Open Data (LOD)

The idea of Semantic Web isn't just about making data available on the web, it is also about making links between data so that a person or machine can explore the web of data. Furthermore, Semantic Web needs access to these data and the relationships among data have to be made available too which eventually create a Web of Data. This collection of interrelated datasets on the Web can also be referred to as Linked Open Data (LOD).

Linked Open Data lies at the heart of Semantic Web and similarly, the Semantic Web idea significantly stands on the availability of LOD. A classic example of LOD can be DBpedia¹: a dataset with the contents of Wikipedia where the data contents are structured following the semantic web standards. DBpedia is not only included Wikipedia data, it also incorporates links to other datasets on the Web such as Geonames². With those extra links, applications can gather extra knowledge from other datasets; which is basically one of the major advantages that LOD offers. The Semantic Web researchers has been developing a set of linked data technologies through the standardization process of W3C. We will understand it more having a look on web structure.

¹<http://wiki.dbpedia.org/>

²<http://www.geonames.org/>

Classic Web and Web of Data

The classic WWW can be described as a web of documents where documents are inter-linked. As visualized in figure 2(a), the web is made of documents (nodes) and directed links (edges) between these documents. The documents are backed by an individual data source.

Usually, these documents are logically structured using the Hypertext Markup Language (HTML). In HTML, connection between documents can be made using the `` tag.

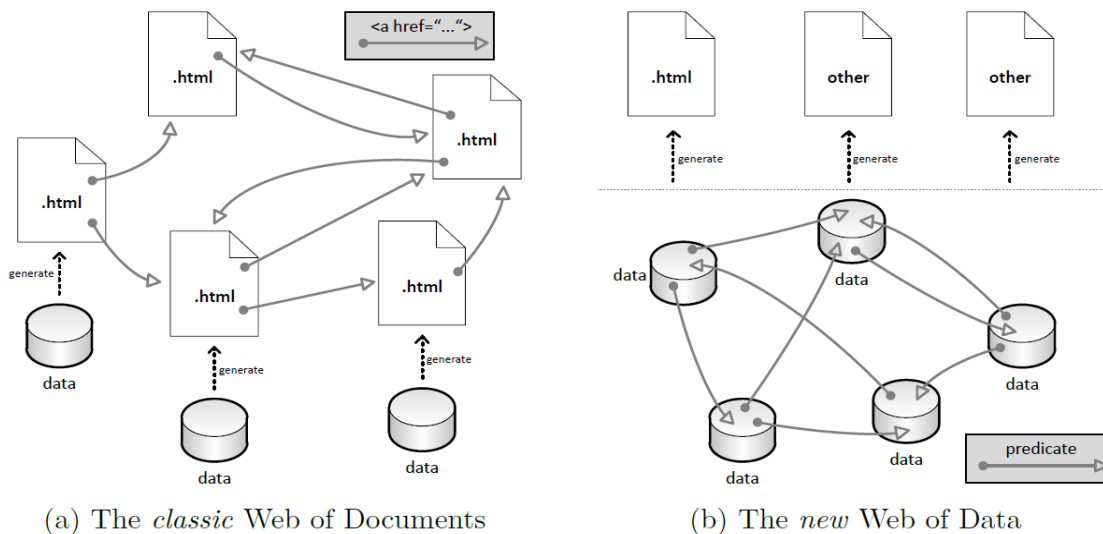


Figure 2: Classic Web and Web of Data¹

Figure 2(b) visualizes how the Web of Data is made of. Here, data sources are considered as nodes and links between data sources are the edges.

In contrast to the general links of the WWW, the Web of Data uses typed links, called properties (or predicates). That means predicates establish the relationship between two data nodes. Therefore, it's important that all these data sources follow a standard mechanism so that relation can be established. This mechanism is provided by the Resource Description Framework (RDF), we discuss this term in next section.

2.2.2. RDF – Data Model

Resource Description Framework is a "framework for representing information on the Web" [17]. It was designed to provide a common way to describe information, so that information can be read and understood by computer applications rather than human. RDF provides a flexible way to describe things in the world – such as people, locations, or abstract concepts – and how they relate to other things [18].

¹Image source: <https://www.inf.fu-berlin.de/inst/ag-se/theses/Holst13-RDF-structure.pdf>

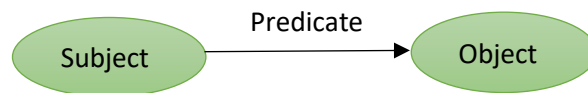
In RDF, a description of a resource is represented as several *triples* (section 2.2.2.1). RDF uses globally unique identifiers *URIs* and *vocabularies* that consists domain specific sets of terms (section 2.2.2.2). Different serialization format exists to represent RDF (section 2.2.2.3).

2.2.2.1. Triples

The basic element of RDF is triples. A triple is a set of three entities that form a statement about semantic data in the form of subject-predicate-object expressions. For example,

Harry Potter → *has author* → *J. K. Rowling*

This triple is consisting of a subject ("*Harry Potter*"), a predicate ("*has author*"), and an object ("*J. K. Rowling*"). Together they express a complete sentence. In general, triples represent a relationship where predicate denotes sets the relationship between subject and object. A set of triples is called an RDF graph.



Consider another simple example the sentence:

http://dbpedia.org/page/Moby-Dick is a novel by writer Herman Melville

This sentence has the following parts:

Subject (Resource)	<i>http://dbpedia.org/page/Moby-Dick</i>
Predicate (Property)	Writer
Object (literal)	"Herman Melville"

The RDF graph of this triple will be-

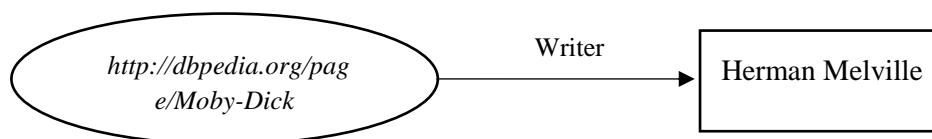


Figure 3: Simple RDF graph

The RDF standard [17] by W3C has defined these three components as follows:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference
- the object, which is an RDF URI reference, a literal or a blank node

Where literals are strings with optional language tags, and blank nodes are also strings.

2.2.2.2. URIs, Vocabularies & Ontologies

Uniform Resource Identifiers (URIs) are short strings that identify resources in the web. They ensure that the resources are available under a variety of naming schemes and access methods such as HTTP.

Semantic Web uses the data with explicit semantics, that is, meaning. Therefore, RDF has to provide the means to use globally unambiguous terms and identifiers. For this purpose, RDF uses URIs. Consider the following example:

```
<http://dbpedia.org/resource/Berlin>          .. (1)
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> .. (2)
<http://dbpedia.org/class/yago/CapitalsInEurope> .. (3)
```

Listing 2: Simple triple with URI's

$$\begin{array}{c} \textit{subject} \rightarrow \textit{predicate} \rightarrow \textit{object} \\ (1) \rightarrow (2) \rightarrow (3) \end{array}$$

This triple refers that resource (1) has a relationship (2) with the resource (3). The resource `<http://dbpedia.org/resource/Berlin>` was defined and therefore, the resource is available at dbpedia.org to represent the capital city of Germany; Berlin. DBpedia have created all their resource URIs with the common prefix of `<http://dbpedia.org/resource/>` like this example. A URI prefix like this also called a namespace [19].

As we see here, it is about modelling the entities into triples. This is done according to a meta-model, that is made partly of RDF data model and part of domain-specific conceptualizations, called vocabularies (or ontologies). “There is no clear division between what is referred as vocabularies and ontologies” [20]. The term “ontology” is used for more complex, and more formal collection of terms, where “vocabulary” is only for basic use.

The role of the vocabularies on the Semantic Web is to help data integration between data sets. While integrating, data ambiguities may exist in different datasets if the same type of data is expressed using different terms.

Let us make it clear with an example, we used the same problem statement that we discussed in the *motivation section 1.2*. Let us assume, Researcher A wants to integrate his data to Researcher B’s dataset. For example, Researcher A's dataset has the term “*country name*” while researcher B used “*region name*” to define the same thing. Therefore, integration between these two datasets become necessary. To make the integration complete, an extra definition should be added to the RDF data, pointing that the relationship described as “*country name*” is same as “*region name*”. To define this extra piece of information, we use vocabulary.

2.2.2.3. Serialization Formats

RDF defines an abstract data model of triples that form RDF graphs. These graphs have to be serialized to store as files. Different serialization formats exist for RDF. Among them, some of the important formats are- Turtle (Terse RDF Triple Language) [21], N-Triples, [22], RDF/XML [23], RDF/JSON [24] etc. We have used the Turtle notation throughout this thesis.

2.2.3. SPARQL – Query Language

SPARQL is the standardized query language for RDF, just like SQL is the standardized query language for relational databases. Like SQL, SPARQL also follow the same “SELECT...FROM...WHERE...” query structure. But there are also important differences. As discussed in *section 2.1*, relational databases work with a set of relational tables, connected by primary and foreign keys. SQL is used here to specify joins between tables and collect subsets of rows. In contrast to this, we saw in *section 2.2.2.1* that RDF data forms a graph structure; hence SPARQL is used to specify graph patterns to select the subset of data that satisfied these patterns.

A SPARQL query comprises, in order:

- Prefix declarations: Abbreviating URIs.
- Dataset definition: The RDF graph(s) are being queried.
- A result clause: The information to be returned from the query.
- The query pattern: To specify what to query.
- Query modifiers: Ordering or rearranging the query results.

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```

Listing 3: Sparql query pattern

Let us have a look at a basic example-

Assume, we have the following RDF triples in our database.

```
:id1 foaf:name "André Schürrle"  
:id1 foaf:based_near : Dortmund  
:id2 foaf:name "Nils Petersen"  
:id2 foaf:based_near : Freiburg
```

Listing 4: Triples with foaf

These triples imply two football players named “*André Schürrle*” who plays for Dortmund and “*Nils Petersen*” who plays for FC Freiburg. Here, we want to find the names of all the people from this dataset. The SPARQL query would look like:

```
SELECT ?name  
WHERE {  
  ?x foaf:name ?name  
}
```

Listing 5: Simple SPARQL query 1

This query is only selecting the values assigned to the variable `?name`, therefore the final answer is “*André Schürrle*” and “*Nils Petersen*”.

Let us look at another query:

```
SELECT ?name  
WHERE {  
  ?x foaf:name ?name .  
  ?x foaf:based_near : Freiburg .  
}
```

Listing 6: Simple SPARQL query 2

In this case, the predicate has a constant value of `foaf:based_near` and the object has a constant value of `: Freiburg` which can only match to one of our RDF triples. So, the result will be “*Nils Petersen*”.

2.3. Triple Stores

Usually, data can be stored electronically in two ways: in simple files on hard disks or through data storage systems like database. Same goes for RDF data too. RDF data can be stored in two different ways-

- either in *files*, where triples are stored following one of the serialization formats (*section 2.2.3*), or
- in special kind of databases designed for triples, called *triple stores*.

Triple Store (also called RDF triple store) is a specialized database management system for RDF triples. Popular triple stores include Apache Jena¹, OpenLink Virtuoso², Sesame³, Allegro-Graph⁴ etc.

2.3.1. Architecture

Triple stores have three possible architectures:

- In-memory: which stores the triples in main memory. It is fast, efficient but expensive.
- Native Store: which provides persistent storage systems with own implementation of databases. For example- Jena TDB, Sesame Native, Virtuoso, AllegroGraph, Oracle 11g etc.
- Non-native Store: which provides persistent storage using a third-party RDBMS. For example- Jena SDB backed by MySQL database.

For our thesis, we have chosen Apache Jena TDB, a native RDF store by Jena. In the next section, we will talk about Jena architecture and the architecture of Jena TDB will be covered in *chapter 4*.

¹<http://jena.apache.org/>

²<http://virtuoso.openlinksw.com/>

³<http://www.openrdf.org/>

⁴<http://www.franz.com/agraph/allegrograph/>

2.3.2. Apache Jena

Apache Jena is a framework for developing semantic web in the form of java libraries [25]. Jena is an open source project developed by the researchers at HP Laboratories in the Bristol city in UK. *Figure 4* shows the architecture of Jena.

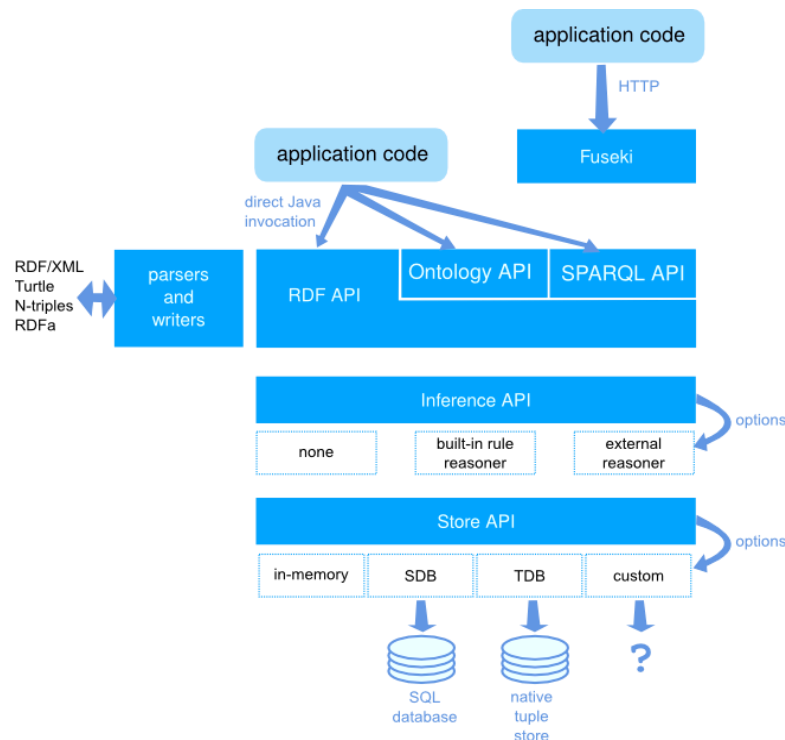


Figure 4: Apache Jena Architecture¹

RDF triples and graphs are accessed through Jena's RDF API. It also has the functionality for adding and removing triples to the graphs and finding triples that match particular patterns based on the query. The SPARQL API has the responsibility to manage the SPARQL, both for query and update. There are two ontology languages for RDF: RDFS and OWL. OWL is much more expressive than RDFS. Both languages are supported in Jena through the Ontology API.

The inference API provides a number of rule engines to impose semantics rules. These semantic rules can be used to define the class and subclass relationship. For example, if class C is a sub-class of class B, and B a sub-class of A, then C is a sub-class of A. These rules can be built-in rulesets for OWL and RDFS, or custom defined. Apart from that, to perform similar kind of operations with different reasoning algorithms, this API can also be connected to external reasoners such as description logic (DL) engine.

The API Store is used to communicate with the graph model in the RDF Store and it can also interact with a variety of databases by the specific database drivers.

¹Image source: https://jena.apache.org/about_jena/architecture.html

Chapter 3. Construction of Relational Database

In this chapter, we discuss about the design and implementation process of IEF database on MySQL. In *section 3.1*, we talk about MySQL database design steps based on the structure of dataset we are using. *Section 3.2* is the implementation process of the web interface. This part is basically divided into two sub-parts; one is the development of Back-End architecture and other one is the development of the Front-End architecture. Within the Back-End part, we discuss about the server-side design; mechanisms to communicate with the database. The client-side interface design will be covered in the Front-End section which performs the task of data presentation. Later, we talk about the hosting mechanism that we are using to host our application. In *section 3.3*, we will provide a structural overview of our application.

We have used different tools to program and build the application. As database platform, we have used *MySQL Workbench*: a visual database design tool for the MySQL database system management and we have used *Microsoft Visual Studio*: an integrated development environment (IDE) for the web interface design.

3.1. MySQL Database Design Process

We start with the description of creating the database schema, required tables, and relations between tables. The relation between tables is defined by assigning the database constraints (discussed in *section 2.1.4*). After that, we load data from CSV files into the database.

3.1.1. Creation of Schema, Tables, and Constraints

Any kind of relational database design starts with creating the database schema. We start with creating a database schema using SQL CREATE statement. We have named the database as “iefdatabase”.

```
CREATE DATABASE 'iefdatabase';
```

The next step is to create the required tables. Before doing so, we need to look at our dataset stored in multiple CSV files. All these CSV files of IEF dataset can be found in the website of Industrial Ecology Freiburg group¹. There are six different CSV files containing the data of country, unit specification, dataset (list of available datasets), process list, stock data and flow data between processes. We can consider these CSV files as six data tables that we need to create in MySQL. *Figure 5* gives an overview how the data look like with a simple example.

¹<http://www.industrialecology.uni-freiburg.de/>

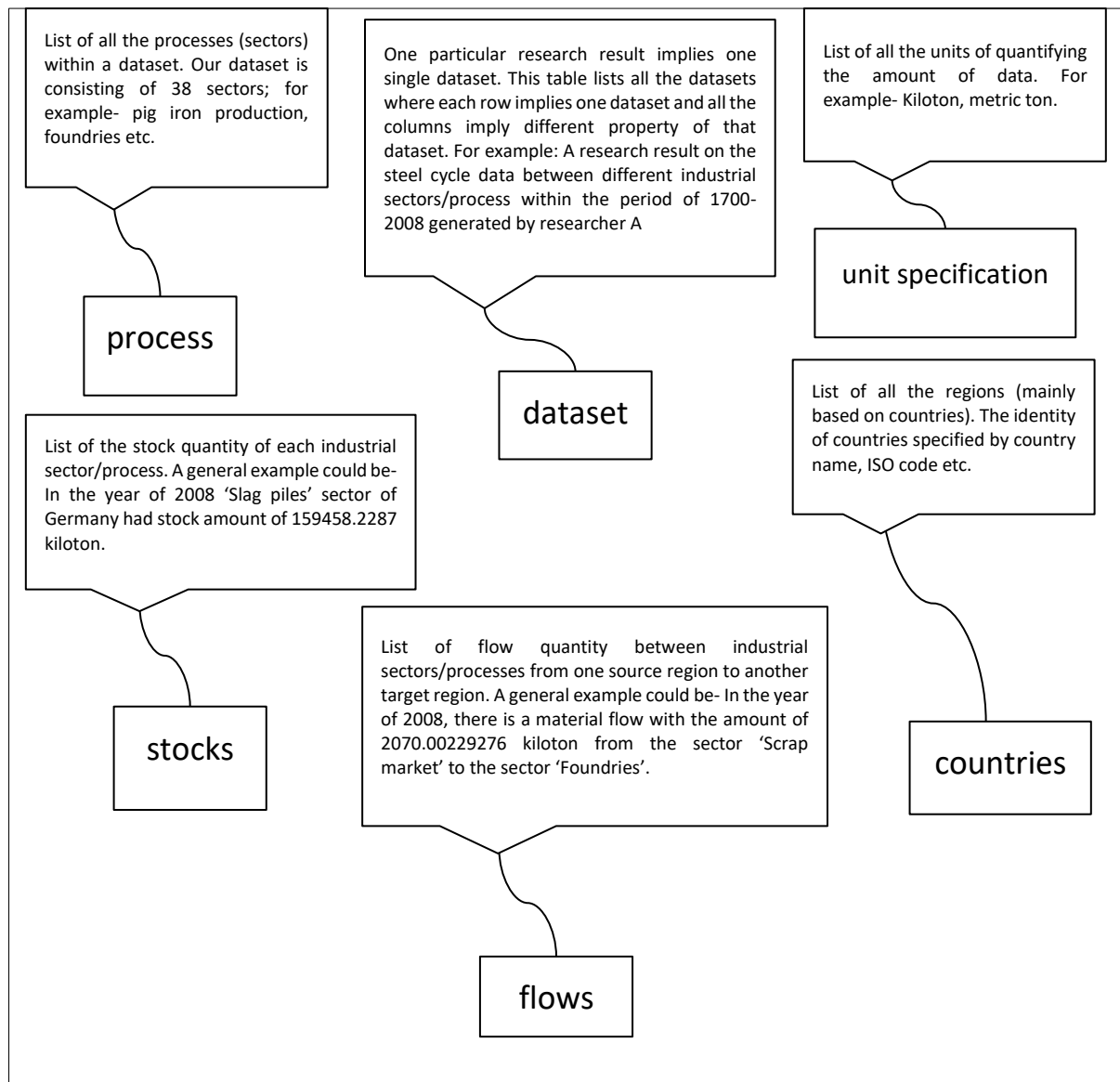


Figure 5: Dataset structure of Industrial Ecology Freiburg

MySQL Workbench provides a nice user interface to upload data from CSV or JSON files directly to the database; even without creating the database tables first. But it is always good practice to create the tables with suitable column name and datatype rather than doing it directly as automated table column names will take the csv column header by default.

The query structure to create these tables is already discussed in *section 2.2.1*. All the primary and foreign keys defined in these queries are basically representing how these tables are interrelated and how they will share their individual table information. After assigning the necessary constraints, we get six primary keys for six tables and eleven foreign keys between these tables.

The following figure provides an overview of our database schema which consists six tables with a number of column fields, primary keys, and the foreign keys relationship between these tables.

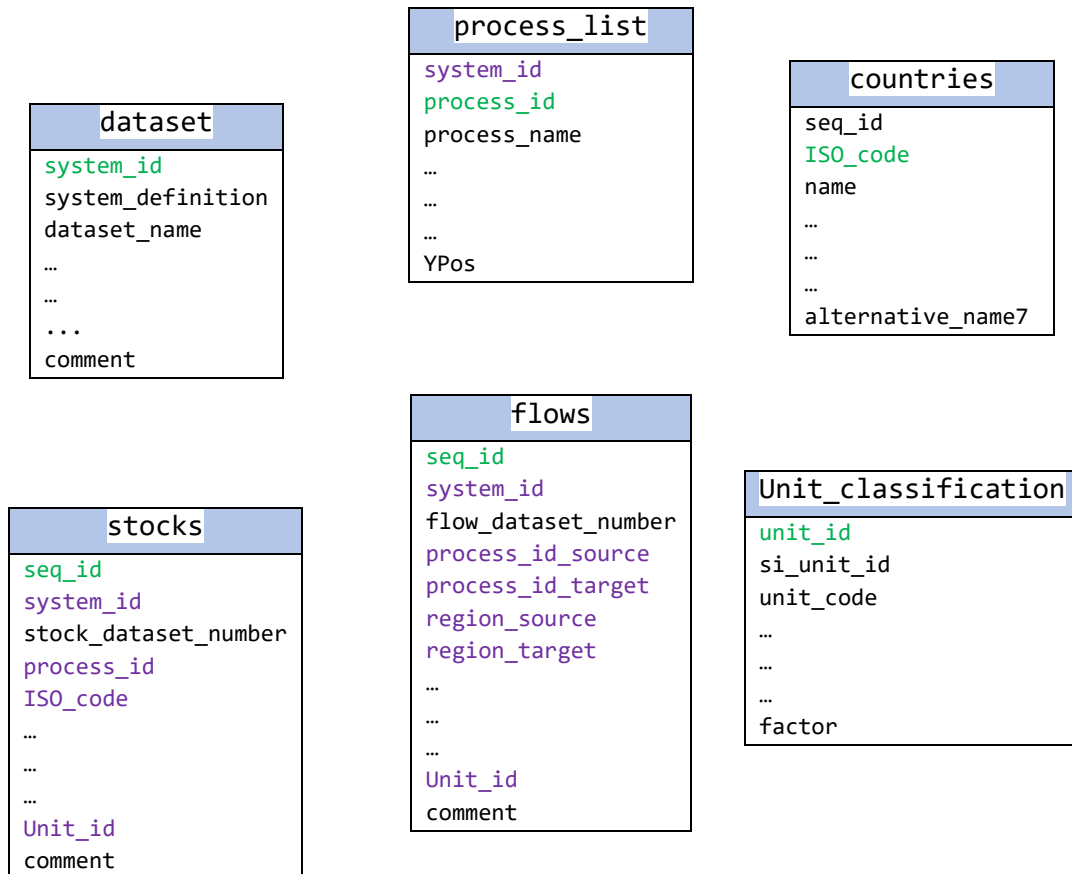


Figure 6: Tables with columns and constraints among them

All the blue coloured column fields are the primary keys of following table and all the purple coloured column fields are the foreign keys.

3.1.2. Loading Data into Database

There are several ways to load data into the database. We have used MySQL workbench interface to upload data directly from CSV files. In this process, we had to load one table data from one CSV file at a time. It is important to carefully uploading the files; sometimes any special character from data can cause the error. In our case, we faced error in the case of uploading the column value consisting the special character used for local country names (for example, *Ísland* for Iceland). In case of this kind of error, we might need to introduce special encoding format that is defined in the workbench manual or we can simplify the record by replacing the special characters contained values with alternative values.

3.2. Implementation of the Web Interface

A database-backed web interface usually consists of two-layer design architecture. A Back-End: mechanisms to communicate with the database and a Front-End: mechanisms to design the user interface to present data in an interactive way. In this section, we discuss these two-side implementation process with the sub-parts.

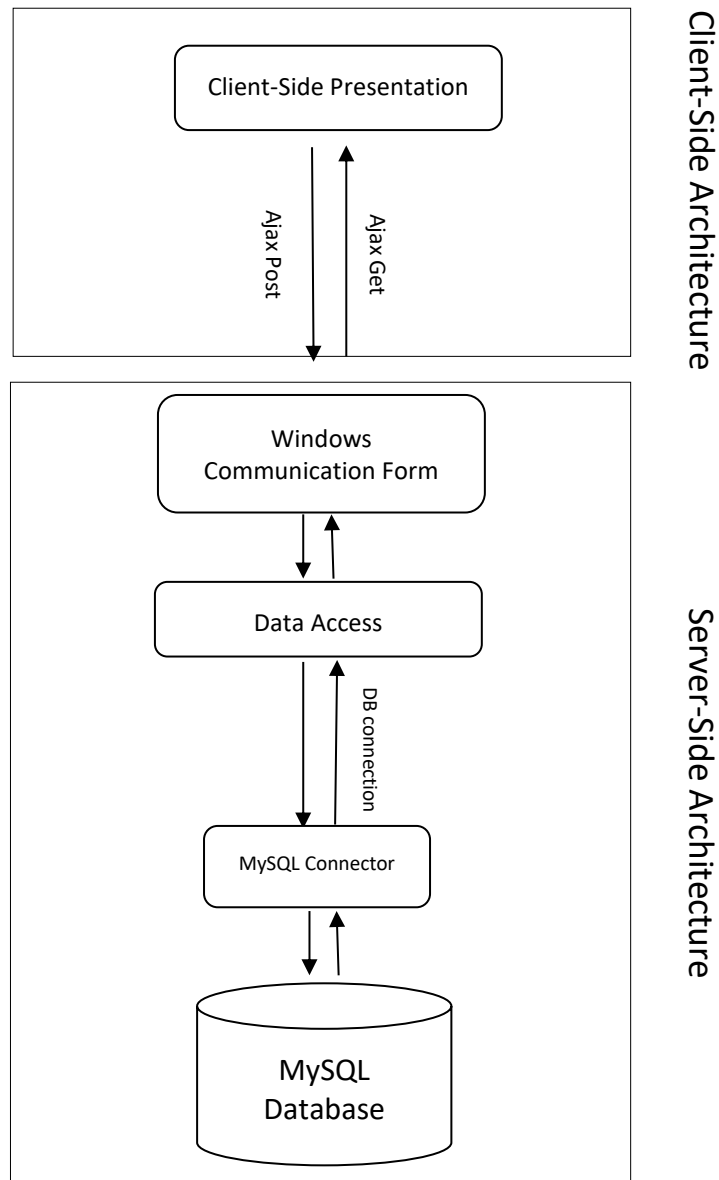


Figure 7: Architectural overview of Web Interface

3.2.1. Back-End Development

The Back-End development delivers smooth functionality to the users to get information from the database right into the browser. This is something that users do not see it or directly interact with it as with client-side technologies. It involves designing the infrastructure of the application, implementing algorithms and logic, and working with data.

Our IEF Database is built on ASP.NET¹; an extension of .NET² that is used to write web-based applications and websites. We chose C# as the server-side programming language. Although, Microsoft SQL Server is the default database platform for .NET and ADO.NET³ is a data access technology that provides communication between database and code, we used MySQL database as discussed before.

To use MySQL database with the application, we will use MySQL connector; a fully-managed *ADO.NET* driver for *MySQL*. First, we have to establish connection with the database (*section 3.2.1.1*). The next part will be accessing the database (*section 3.2.1.2*). Lastly, we will return the data as JSON format using a web service (*section 3.2.1.3*) which works as a communication bridge between client-side and server-side. The figure bellow shows the architecture our back-end design.

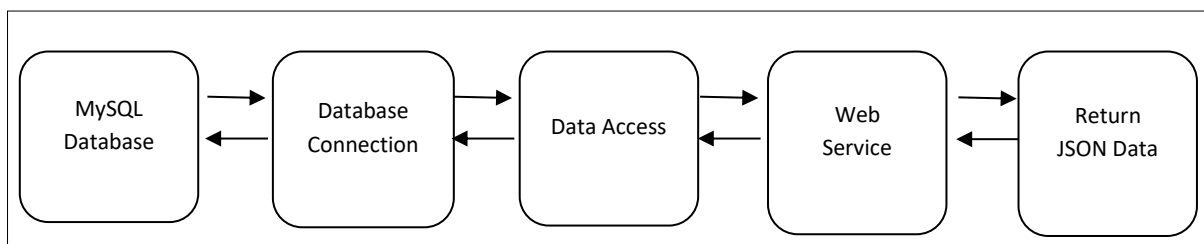


Figure 8: Back-End design architecture

3.2.1.1. Database Connection

Connection to a MySQL database starts by using a *MySqlConnection* object. The *MySqlConnection* constructor takes a connection string which consists of a number of parameters. The connection string provides necessary information to make the connection to the MySQL database.

¹<https://www.asp.net/>

²<https://www.microsoft.com/net/>

³<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview>


```

public void Initialize()
{
    server = "localhost";
    database = "iefdatabase";
    uid = "root";
    password = "iefdatabase";
    string connectionString;
    connectionString = "SERVER=" + server + ";" + "DATABASE=" + database + ";" + "UID="
+ uid + ";" + "PASSWORD=" + password + ";";

    connection = new MySqlConnection(connectionString);
}

```

Listing 7: MySQL database connection code

The *MySqlConnection* constructor returns a connection object which is used for different database operations. It opens the connection for the operations to take place and close it after the operations are performed. Sometimes an attempt to perform an `Open ()` on a connection object can fail due to unexpected reasons and generate an exception that can be handled using standard exception handling method.

3.2.1.2. Data Access

A Data Access class takes request from client side via web service. It checks if the database connection is open or not and then execute the requested query on the database.

A query is executed by the *MySqlDataAdapter*: a method that executes the SQL statement and retrieves one or more result sets from the database. The *MySqlDataAdapter* object has two main methods: (i) *Fill* method which reads data into the `DataTable`¹, and (ii) The *Update* method which writes data to the database. Our application is built to present data on user request. Therefore, this adapter method mainly serves the purpose of taking the query request, execute it and *fill* the queried data into `DataTable`.

```

public DataTable stocksDataTable(parameter)
{
    DataTable table = new DataTable();
    string query = "";
    if (cn.open_connection() == true)
    {
        MySqlCommand cmd = new MySqlCommand(query, cn.connection);
        MySqlDataAdapter adapter = new MySqlDataAdapter(query, cn.connection);
        DataSet ds = new DataSet();
        adapter.Fill(table);
        cn.close_connection();
    }
    return table;
}

```

Listing 8: Data access code for queuing the data and store the data in Data Table

¹`DataTable` is an asp.net controller of in-memory table data

Once data is available inside DataTable, our next step is to serialize this table data and send it to the client-side script for representation. This whole process is carried by one of the services provided by the .NET framework. Here, we are using data service named Windows Communication Form (WCF).

3.2.1.3. WCF Data Service with Ajax Call

WCF Data Service basically exposes data via web service accessed over a protocol like HTTP. It fetches data from the database directly or via data handler class (our case is described in *section 3.3.1.2*), then serialize the data and send back to the client-side script. The whole procedure takes place based on AJAX call.

```
[OperationContract]
[WebInvoke(Method = "POST", BodyStyle = WebMessageBodyStyle.WrappedRequest, ResponseFormat = WebMessageFormat.Json)]

public string DataSet(parameter)
{
    clsDataHandling objDataClass = new clsDataHandling();
    DataTable dt = null;
    dt = objDataClass.DataSet(); // the data table from data access class

    //Serialization Code goes here

    return stringBuilder.ToString();
}
```

Listing 9: Data Service code to serialize the data in JSON format

We have used AJAX call because of its asynchronous behaviour. This is a very important concept as it stops the entire web page to reload again in response to client request, rather, it partly refreshes the web page without losing the given input data of other webpage controllers like text field, checkbox etc.

Ajax is not a programming language or a tool, but a client-side script that communicates with WCF Data Service without the need for a postback or a complete page refresh. For our application, we load data from the data service using a \$.post(); an AJAX HTTP POST request.

```
$.ajax({
    url: "DataService.svc/DataSet ",
    data: '',
    dataType: "json",
    type: "POST",
    contentType: "application/json; charset=utf-8",
    success: successFunction
});
```

Listing 10: Ajax code to call a data service and retrieve data from that service

Number of properties jointly perform the call with data service.

- url - A string containing the URL to send the request.
- data - An object or string sent to the server with the request as a parameter.
- success - A callback function that is executed if the request succeeds.
- dataType - The type of data expected from the server. In our case, we are using JSON data format.

Once Ajax call is initiated, this script communicates with WCF with the given url.

3.2.2. Front-End Development

Front-End part is basically the design of interface consisting different HTML controllers. A scripting language like JavaScript and Front-End library like JQuery have been used here to bind the data in these controllers. CSS and bootstrap CSS library were used to design these controllers as well as the whole interface.

3.2.2.1. Data Presentation with JqGrid

JqGrid is an Ajax enabled stylish and featured tabular data presentation control. It can be integrated with any of the server-side technologies. ASP.NET provides a good mechanism to integrate with JqGrid.

For our web interface, we designed four JqGrids. First one shows the list of datasets, the second one shows the stocks data, the third one for the flows data and the final one is also for flows data with different filter parameter.

What we have done is, we created WCF data service with required HTTP handlers, and the functions inside the http handlers fetches the records and writes in the HTTP Response. The HTTP response would be a JSON string following the JqGrid data format. Our JqGrid reads the JSON response and display it in the JqGrid. In url section, we provided the location of specific HTTP handler within WCF data service.

3.2.2.2. Interface Styling with Bootstrap

Bootstrap is a free front-end framework for faster and easier web development. It provides a set of stylesheets with basic style definitions for all the key HTML components/controllers. The reason we have used bootstrap is mainly for its responsive appearances from device to device. A significant interested user may want to use the web application from different types of devices. The responsive feature of bootstrap makes it easy. Therefore, we used this

framework for designing the overall interface layout with different necessary HTML controls like- table, dropdowns, text fields etc.

3.2.3. Hosting with IIS

Industrial Ecology Freiburg group has already configured a virtual machine to host their implemented web applications. This machine runs on Windows Server. It offers a secure, manageable platform for hosting web applications. This platform is called Internet Information Services (IIS). This IIS is configured to host web applications.

We provide the physical address of application folder and the host name. In our case, our host address is <http://www.database.industrialecology.uni-freiburg.de/> and, our implemented web applications can be found here.

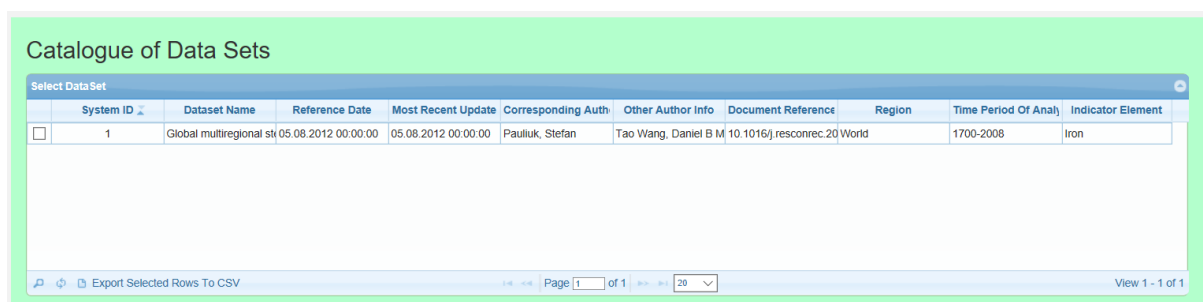
3.3. Structural Analysis of Web Interface

In this section, we discuss about the structure of our interface. As we mentioned before, there are four data grids generated from the database. These data grids present different sets of data queried with four different SQL queries. We used the term “DataGrid” quite often within this section, DataGrid is a asp.net controller used to present data as a table.

Dataset DataGrid: First data grid presents the list of available datasets (*figure 9*) in the database. All other data grids populated on upon the selection of one the dataset from this list.

For detailed SQL queries, please look at the *Appendix* section.

Query 1: Select all the dataset available in the database from dataset table:



System ID	Dataset Name	Reference Date	Most Recent Update	Corresponding Auth	Other Author Info	Document Reference	Region	Time Period Of Anal	Indicator Element
1	Global multiregional st	05.08.2012 00:00:00	05.08.2012 00:00:00	Pauliuk, Stefan	Tao Wang, Daniel B M	10.1016/j.resconrec.20	World	1700-2008	Iron

Figure 9: Grid with list of datasets available in the database

On the bottom left side of every DataGrid, there is an export button which lets users to export selected rows in CSV format.

Stock DataGrid: Once we select a particular dataset from the above list, the list of processes appears in the stock dropdown (*figure 10*). We then select the specific process and data appear in the second grid. There are two text fields where we can add two different parameters as tag, Year and Country. Data within the data grid therefore filtered according to the values in the tag sections. For selected process, year and country we got the data in the grid with following query-

Query 2: Select all the stocks within the dataset (identified by the id) and filter the data with year (2007, 2008) and country (Austria, Germany, Switzerland and Italy) from stock table;

The screenshot shows a web interface for 'Processes with stocks'. It includes a 'Choose Stock Process' dropdown set to 'Slag piles'. Below, there are filter sections for 'Year and Region'. The 'Year' filter has tags for '2008' and '2007'. The 'Region' filter has tags for 'Austria', 'Germany', 'Switzerland', and 'Italy'. At the bottom, a table titled 'Select Stock Process' displays filtered data.

stock dataset num	process name	country name	year	age cohort	indicator element	aspect of dataset	value	error type	error value 1	error value 2	data quality	unit id	comment
<input type="checkbox"/> 762	Slag piles	Austria	2007		slag	Stock	14364.0962395					kt	
<input type="checkbox"/> 763	Slag piles	Austria	2008		slag	Stock	14742.1002813					kt	
<input type="checkbox"/> 5340	Slag piles	Germany	2007		slag	Stock	157162.243482					kt	
<input type="checkbox"/> 5341	Slag piles	Germany	2008		slag	Stock	159458.228723					kt	
<input type="checkbox"/> 7302	Slag piles	Italy	2007		slag	Stock	59069.7802223					kt	
<input type="checkbox"/> 7303	Slag piles	Italy	2008		slag	Stock	60640.8354917					kt	
<input type="checkbox"/> 13515	Slag piles	Switzerland	2007		slag	Stock	2070.50257304					kt	
<input type="checkbox"/> 13516	Slag piles	Switzerland	2008		slag	Stock	2136.16484003					kt	

Figure 10: DataGrid with stock data between processes

Flow DataGrid: Flow data is represented between two processes. Process from first dropdown lists are defined as source process and second dropdown process is defined by the target process (*figure 11*). We select one source process and the target process. The other parameters work as above. The query structure is like-

Query 3: Select all the flows within the dataset (identified by the id) and filter the data with year (2005) and source process 'scrap market' and target process 'foundries' from flow table;

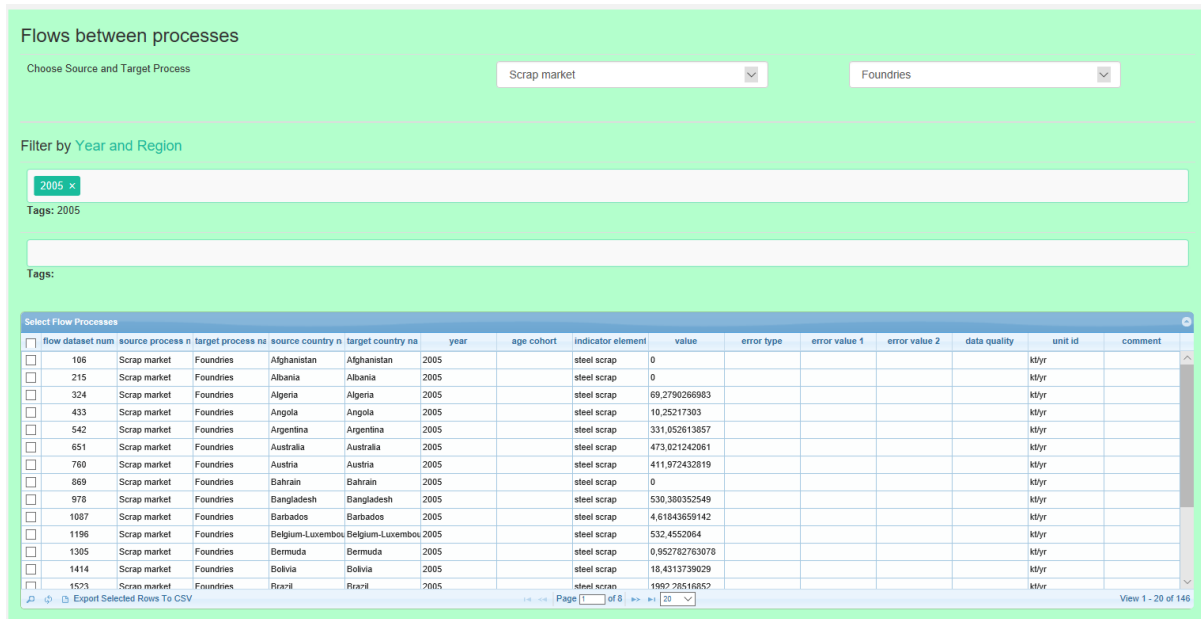


Figure 11: DataGrid with flow data between processes

Flow DataGrid for Sankey Diagram: The fourth data grid has basically the same data as in data grid three. Instead of querying based on source and target process, we query all the flows from all the process depending on a specific year and a specific country (figure 12).

Query 4: Select all the flows within the dataset (identified by the id) and filter the data with year “2001” and country “Germany” from flow table;

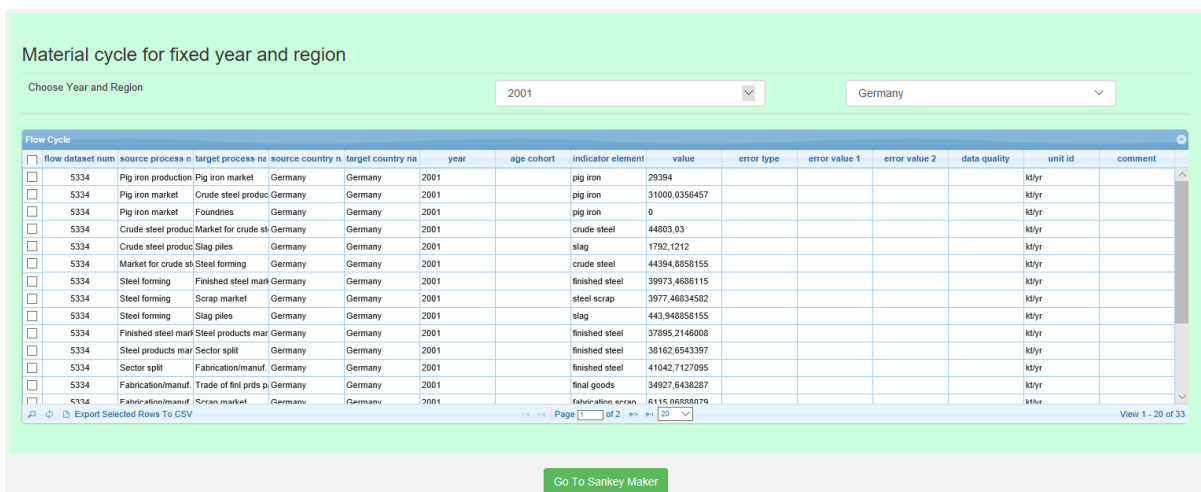


Figure 12: DataGrid with material cycle data for fixed year and region

On the bottom of this data grid, there is a button named “Go To Sankey Maker”. This button directs to “Circular Sankey” tool [23] which produces a diagram to visualize the data. *Figure 13* shows the generated Sankey diagram driven from our queried data.

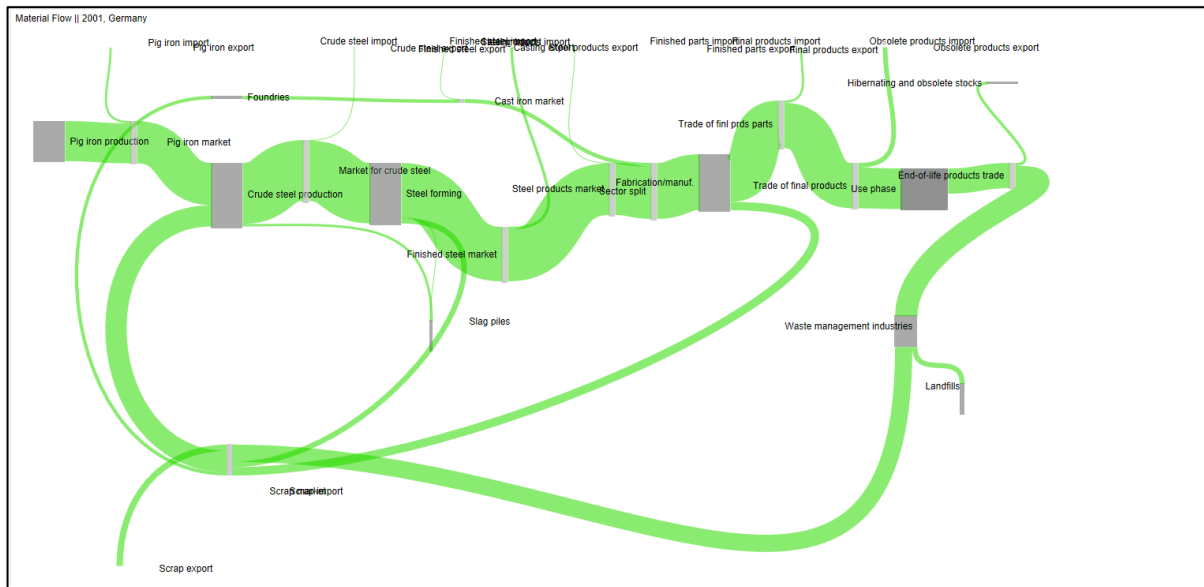


Figure 13: Unpolished Sankey diagram, as generated by the D3 application

Chapter 4. Construction of Triple Store

The majority of data on the current Web is stored in relational databases [27]. We already discussed in *section 2.2* about the semantic web and its useful application, especially if data from different sources can be exchanged or integrated. This integration is only possible when data is structured in semantic web standards such as RDF. Therefore, it is important to impose mapping technologies between relational database and RDF [28].

In this chapter, we basically highlight the procedure of constructing the triple store. In *section 4.1*, we discuss the process of mapping relational database to RDF triples. *Section 4.2* covers discussions on the Apache Jena TDB architecture; a component of Jena RDF storage, designed for storing triples directly from RDF file. Here, we also talk about the process of uploading the triples into it.

4.1. Mapping Relational Database to RDF

In this section, we discuss about different mapping languages and platforms. We transform the entire dataset from MySQL to RDF following one of the mapping techniques; D2RQ: a mapping language and platform for treating non-RDF relational databases as virtual RDF graphs [29].

4.1.1. Mapping Languages and Platforms

Over the years, many approaches have been introduced to transform relational database to RDF. “Depending on the requirements, these approaches introduced mapping languages that range from simple and pragmatic to highly specific or general-purpose” [28]. The W3C RDB2RDF Working Group¹ has introduced two standards to map relational databases to RDF. One is Direct Mapping [30] and the other one is via R2RML (Relational Database to RDF Mapping Language) [31].

The Direct Mapping is the default way of representing a relational database as RDF. It is implemented based on the structure of the database schema itself. In [32], a direct mapping is proposed. It maps relational tables to classes in an RDF vocabulary and table attributes to properties in the vocabulary. R2RML is a language for expressing customized mappings from relational databases to RDF. R2RML mappings are expressed as RDF graphs and written down in Turtle syntax. There exist some other techniques to transform relational data to RDF [28]. We will use D2RQ for this purpose and our next section covers further details on it.

¹<http://www.w3.org/2001/sw/rdb2rdf/>

4.1.2. D2RQ

D2RQ exposes relational data as RDF. It uses the D2RQ Mapping Language to describe the relationship between a relational database schema and RDFS vocabularies or OWL ontologies [33]. It takes any JDBC database as input and provides the output linked data in different serialization format. *Figure 14* shows the architecture of D2RQ.

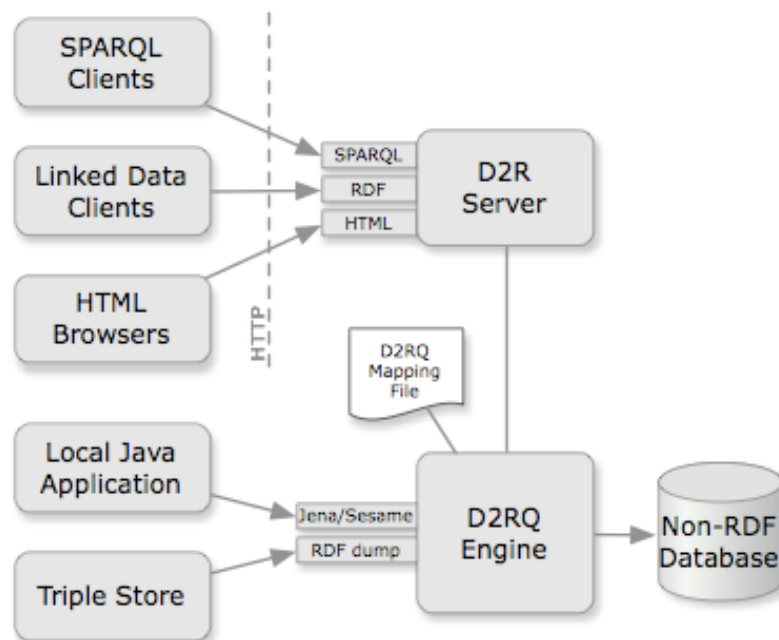


Figure 14: Architecture of D2RQ³

A D2RQ Engine accesses a Non-RDF database (for example MySQL) to create a mapping file with mapping language¹. The mapping language basically describes the relationship between an ontology (or vocabulary) and a relational data model. It also creates an RDF dump and provides RDF APIs which can be embedded with Java applications through Jena or Sesame APIs. The RDF dump file can be stored in any triple store. Another important feature D2RQ is the D2RQ server². It is an HTTP server which provides an HTML view to explore the mapped database.

¹<http://d2rq.org/d2rq-language>

²<http://d2rq.org/d2r-server>

³Image source: <http://d2rq.org/>

4.1.2.1. D2RQ Overview and Features

The D2RQ platform serves different tools to offer RDF-based access to the content of relational databases. All these tools are stored in D2RQ folder directory and can be run from the command line. Among these tools, we used *generate-mapping* and *dump-rdf* to transform our MySQL data to RDF dump file and we tested the mapping file with the *D2R-server* tool.

The ***generate-mapping*** tool creates a D2RQ mapping file by analysing the relational database schema. D2RQ supports a number of relational databases like Oracle, MySQL, PostgreSQL, SQL Server etc. Connection to one of these databases is established via a jdbc driver with authorized username and password of the selected database. After specifying some other required parameters, the output default mapping file is created. This mapping file can be used directly or after some customization to create RDF dump file in one of the supported serialization formats of D2RQ: TURTLE, RDF/XML, RDF/XML-ABBREV, N3, and N-TRIPLE.

The ***dump-rdf*** tool is used to dump the contents of the entire database into a single RDF file. This process can be done using the custom mapping file generated with the *generate-mapping* or it can be done directly without using any mapping file. In that case, it will automatically generate the default mapping file and use it. This dump file can be used as a dataset with a list of triples and can be loaded into any triple store (*Section 4.2*).

The ***D2R-server*** takes the mapping file as input parameter and provides a web interface where RDF data generated from mapping file can be browsed and searched. The default port address for running the server is 2020 and the default URL location for D2R server is *http://localhost:2020/* unless specified otherwise. In this location, we get a HTML web interface where we can see the data as RDF format from RDB. This interface also provides a SPARQL endpoint and an editor to write SPARQL query on mapping file itself.

4.1.2.2. D2RQ Mapping Process

In this section, we present the mapping process of our MySQL database with the D2RQ tools that have been discussed above. We already designed a MySQL database and specified the required constraints: primary and foreign key columns in tables (*section 3.1.1*) from the given dataset; which is a pre-requisite for this transformation process. First, we create the mapping file using the *generate-mapping* tool. Then we will create the RDF dump with *dump-rdf* tool and lastly, we browse the mapping file to run some SPARQL test query using the *D2R-server* tool.

All these tools run from the command line. First, we run the *generate-mapping* tool.

```
>generate-mapping -u username -p password -o mapping.ttl jdbc:mysql:///iefdatabase
```

-u username	The login name of the MySQL database user.
-p password	The password of the MySQL database user.
-o mapping.ttl	The generated mapping will be stored in this file in Turtle syntax. Any other supported file format can be used here.

And the JDBC driver class with the MySQL scheme name that we are going to map.

The generated mapping file indicates the default mapping from MySQL database. It automatically generates the common used prefixes and the connection parameters of MySQL database. The default mapping file can be found in following link¹

The second step is to create the RDF dump file using the mapping.ttl file that we generated in the first step.

```
>dump-rdf -f TURTLE -o iefdatabase.ttl -b http://foo.example/DB/vocab/mapping.ttl
```

-f format	The RDF syntax to use for output file format. We are using TURTLE here.
-b baseURI	The base URI we are using here is the <i>http://foo.example/DB/vocab/</i> . By default, it takes <i>http://localhost:2020/</i> .
-o iefdatabase.ttl	Name of the output file. The output iefdatabase.ttl is a single file consists of all the triples generated from the selected MySQL database schema.

So, we have a single file (a big file depending on the size of MySQL database) that has all the triples. We use this file as input dataset to load into the triple store which will be discussed in our next chapter. We will now check the generated mapping file with the *D2R-server* tool.

```
>d2r-server mapping.ttl
```

This command line will open a server at the address *http://localhost:2020/* where we can browse and debug our database. We can also run SPARQL queries against it.

¹<http://www.industrialecology.uni-freiburg.de/>

4.2. Loading Data into Triple Store

In the *section 4.1*, we mapped our designed MySQL database and produced a single RDF dump file containing all the triples. These triples expose our entire relational database as subject-predicate-object statements. Now, we load these triples into a triple store. We already discussed the basic architecture of our chosen triple store: Apache Jena in *section 2.3.2*.

4.2.1. Apache Jena TDB

TDB is a component of Jena that is used as native RDF storage. It can be used as a high-performance RDF store on a single machine [34]. It has a simple design that provides performance improvements over traditional triple stores, particularly in the area of read/write performance to the node table [35]. TDB is stored in a single directory in the filing system backed by a dataset. To store a triple internally, TDB creates three composite indexes while the triples are being loaded. They are Subject-Predicate-Object (SPO), Object-Subject-Predicate (OSP) and Predicate-Object-Subject (POS) [36]. It does not require any triple table as each index has all the information about a triple. Just like MySQL database, TDB also uses indexing to avoid full table scan.

Nodes of each triple are stored internally in TDB by 64-bit node identifiers. The triples are stored in three triple indexes as a series these identifiers, or NodeIDs. Each NodeID is a unique reference to the node table and they are created during the load process of triples. NodeIDs are comprised of 8 bits of type information (data type, for example; integer, date etc..), and 56 bits of disk address for the literals that can be encoded in 56 bits or less.

A complete TDB dataset consists of

- The node table
- Triple and Quad indexes
- The prefixes table

The node table stores the representation of RDF terms. It consists of two mappings: one is from Node to NodeId and another is from NodeId to Node. The Node to NodeId mapping is used during data loading and it is heavily used in query processing because same NodeId can appear in many query results. For NodeId to Node mapping, the default storage of the node table is a sequential access file and a B+ Tree for the Node to NodeId mapping.

Triple indexes are used for the default graph and **Quads** are used for the named graphs. The only difference is that triples are held as 3-tuples of NodeIds in triple indexes where quads are held as 4-tuples of NodeIds. But their other functions are same and handled in the same manner.

The **prefixes table** stores index for Graph->Prefix->URI mapping. It provides the mechanism for Jena API to serializae the triples in RDF/XML or Turtle.

4.2.2. Loading Triples into TDB

Being a native triple store, TDB serves our purpose by storing the triples from RDF file. This storing process will be carried out by with a simple Java application.

```
public class TriplesLoader {
    public void Load () {
        String directory = "C:\\CodeRepository\\ief database\\TDB Directory";
        Dataset dataset = TDBFactory.createDataset(directory);
        String source = "C:\\CodeRepository\\ief database\\Files\\data.ttl";
        Model tdbModel = dataset.getDefaultModel();
        FileManager.get().readModel(tdbModel, source);
        TDB.sync(dataset);
    }
}
```

Listing 11: Triples Loading function in TDB

The class *TDBFactory* creates and connects to a TDB-backed graph or an RDF dataset. For this, the directory location has to be specified. If a directory is empty, the TDB files for indexes and node table are created. Once the dataset is connected, TDB loads the triples from the RDF file. If the directory contains files that have been created before, TDB connects to the data already there and add new triples.

Once the triples are loaded into store, we can execute SPARQL queries. The sample code in *listing 12* does this job.

```
public void SparqlQueryExecutor() {
    String directory = "";
    Dataset dataset = TDBFactory.createDataset(directory);
    String sparqlQueries = "";
    String prefix="prefix vocab: <http://foo.example/DB/vocab/>";
    Query query = QueryFactory.create(prefix+sparqlQueries);
    QueryExecution qexec = QueryExecutionFactory.create(query, dataset);
    try {
        ResultSet results = qexec.execSelect();
        for (; results.hasNext() ; )
        {
            // do something
        }
    } finally { qexec.close() ; }
}
```

Listing 12: SPARQL query execution function in TDB

Chapter 5. Database Performance Analysis

In this chapter, we present the performance evaluation of both relational database and triple store. In *section 5.1*, we describe the experimental setting which includes data preparation and application design to access both the databases. In *section 5.2*, we report our performance analysis of both databases in terms of query execution time. We also talk about the additional comparative advantages that both databases offer within this section.

5.1. Experimental Settings

For our experiment, we have worked on two different datasets. First dataset (we have named it as IEF dataset) is the dataset consisting the material stock and flow data of industrial ecology Freiburg research group. The second dataset (we have named it as ISWC dataset) is the sample dataset with the information about conferences, papers, authors and topics from the ISWC 2002 conference¹.

The IEF dataset on relational database consists 6 data tables. Each table has its own primary key. There are 11 foreign key constraints between these tables. The total number of rows of this dataset is 589165. After mapping with D2RQ, we initially got roughly 10 million triples. D2RQ is designed to ignore the *empty* string column value while mapping. Our database in MySQL has number of *null* column value which took part in the mapping process. So, we ran an update query on MySQL to replace all the *null* column values with the *empty* “ ” with following query:

UPDATE database.table SET column = “ ” where column is null;

We have executed this query for all the columns within tables and finally got 7596007 triples after mapping. Jena TDB took approximately 95 seconds to load these triples in TDB store².

The ISWC dataset comparatively smaller which consists of 9 tables. It also has one primary key for each table and there are 11 foreign key constraints between tables. This dataset has 96 rows in total. D2RQ mapping produced 322 triples from this dataset and TDB took approximately only 1 second to load these triples.

So, we have now two different datasets stored in MySQL database and TDB triple store. The next step is to access these datasets through a java application to run similar queries on both databases to retrieve some data. The list of queries can be found in our *appendix* section.

¹<http://iswc2002.semanticweb.org/>

² The TDB loading performance largely depends on the machine configuration. We performed the testing on a 64-bit windows machine on Intel core i5 2.70GHz processor, 8 GB of RAM, and 256GB SSD.

5.2. Results Evaluation

We ran number of queries on both databases. Each of these queries are designed in both SQL and SPARQL syntaxes. For each dataset, we ran four SQL queries and equivalent SPARQL queries to retrieve same data from MySQL and Jena TDB respectively. The respective query execution time have been listed in table 1 and 2.

For IEF dataset, we ran four queries (*query 1- query 4*) in MySQL database. For similar query result, we ran the equivalent SPARQL queries (*query 5- query 8*). The queries can be found in the *appendix* section.

SQL Queries on MySQL		SPARQL Queries on Jena TDB	
	Query Execution Time (ms) (average of 10 run)		Query Execution Time (ms) (average of 10 run)
<i>Query 1</i>	9	<i>Query 5</i>	145
<i>Query 2</i>	127	<i>Query 6</i>	1673
<i>Query 3</i>	80	<i>Query 7</i>	11448
<i>Query 4</i>	16	<i>Query 8</i>	10257

Table 1: Query execution time comparisons for ief dataset

For ISWC dataset, we again ran four queries (*query 9- query 12*) in MySQL database. For similar query result, we ran the equivalent SPARQL queries (*query 13- query 16*).

SQL Queries on MySQL		SPARQL Queries on Jena TDB	
	Query Execution Time (ms) (average of 10 run)		Query Execution Time (ms) (average of 10 run)
<i>Query 9</i>	3	<i>Query 13</i>	132
<i>Query 10</i>	3	<i>Query 14</i>	7
<i>Query 11</i>	2	<i>Query 15</i>	5
<i>Query 12</i>	2	<i>Query 16</i>	12

Table 2: Query execution time comparisons for iswc dataset

The Query execution time on both datasets shows that MySQL works faster than TDB. One of the main reasons is the architectural differences. MySQL stores data in storage engine and use indexing to identify records. To respond to a query, MySQL only looks for the indexes rather looking at all the columns.

On the other hand, to respond to SPARQL query on TDB, we used *QueryExecutionFactory* to execute query over the Dataset.

```
QueryExecutionFactory.create(query,dataset);
```

All this does is creates an execution that can execute the query, but it does not execute the query yet. Query execution needs to call one of the *exec()* methods which basically depend on the query type e.g. *execSelect()* for SELECT queries or *execUpdate()* for the UPDATE queries. This query execution method does not finish until results are fully enumerated. This enumeration process largely depends on the number of data records are available in the *ResultSet* (rows returned by the query). The IEF dataset consists of 7.5 million triples as discussed before, which also consists of a number of predicates (column name). Therefore, this enumeration process takes longer time which results in longer execution time in TDB.

```
ResultSet results = qexec.execSelect();
long numResults = ResultSetFormatter.consume(results);
```

Another important performance issue is the variation of query execution time in TDB. We ran the SPARQL queries in two different ways. First, we ran one query per application build and then we ran all the queries using a *for loop* within a single build. We listed the query results in table 3 and 4. Within *for loop*, just after the first query, all the remaining queries took less execution time (especially for the smaller dataset; table 4). One explanation could be that the TDB loads all the triples in the main memory when *QueryExecutionFactory* is invoked for the first query and keep it available during the build time. Therefore, the remaining queries can read the data faster from main memory.

	One query at a time Execution time (ms) (average of 10 run)	All queries with a loop Execution time (ms) (average of 10 run)
<i>Query 5</i>	145	145
<i>Query 6</i>	1874	1673
<i>Query 7</i>	13422	11448
<i>Query 8</i>	12646	10257

Table 3: Query execution time variations in Jena TDB for ief dataset

	One query at a time Execution time (ms) (average of 10 run)	All queries with a loop Execution time (ms) (average of 10 run)
<i>Query 13</i>	132	132
<i>Query 14</i>	106	7
<i>Query 15</i>	111	5
<i>Query 16</i>	122	12

Table 4: Query execution time variations in Jena TDB for iswc dataset

Chapter 6. Discussions

In this final chapter, we first provide an overall discussion of this thesis paper. Later we present potential future work (*section 6.1*) and lastly, we conclude with outlining a conclusion (*section 6.2*) of the work presented in this thesis.

The focus of this thesis was to study the efficient database model to represent research data of industrial ecologists. For this purpose, we studied on relational database and investigated further on triple stores which are specially built to handle data as RDF triples. There are some certain advantages of using triple store for storing the IEF dataset we are working on. We discuss these findings more elaborately here.

```
<"stocks17011" "type" "stocks">
<"stocks17011" "stocks_country" "Germany">
<"stocks17011" "stocks_aspect_of_dataset" "Stock">
<"stocks17011" "stocks_indicator_element" "slag">
<"stocks17011" "stocks_process" "Scrap market">
<"stocks17011" "stocks_system_id" "1">
<"stocks17011" "stocks_unit_id" "kilo ton">
<"stocks17011" "stocks_value" "444.0907">
<"stocks17011" "stocks_year" "1906">
```

Listing 13: A list of triples to define a stock data

Here are some triples to describe one stock record from our IEF dataset. From these triples, we get a complete information of a stock id 17011. For example, the second triple says that this stock id implies the country "Germany".

In a relational database, there would be a need for several tables to store the information about this stock record. Maybe, a table with the list of processes, a table with the list of countries, a table for unit id's, etc. We also need to establish links between these tables with required primary and foreign key constraints, and so on. Therefore, it would need multiple joins between tables to get the full information which becomes complex with the number of tables increase.

Apart from this problem, there may be a need to look for an extra bit of information and this information might not be available in these tables. This is where triple stores are valuable. It allows existing data to collect external data from other ontologies and resources, which look like URLs / URIs and return serialized information. For example, if someone wants to query "the ISO code and local name of Germany" from the same database. Even if there exists no such data in the dataset, to answer to that query, one could use the "*stocks_iso_code*" property as defined by some resource and "Germany" resource from dbpedia.org's data sources.

As everything is stored as statements in triple stores with three pieces of information, new predicates (or property) can be added to the database as much as needed without changing any schema, which connects data from different sources. In other words, a small dataset can be enlarged into a bigger and complete dataset.

One of the main purposes of a set of RDF triples is to build, or to extend, an ontology which defines a common vocabulary for researchers who need to share information. From this point of view, we tried to discuss such methodology to build an ontology to represent research data of material flow analysis. The best way to build this kind of ontology is to reuse other developed ontologies available on the web, that means if one needs to build an ontology, it is important to integrate several such existing ontologies. Unfortunately, we did not find any such ontology that has been published to help us construct our own ontology for the kind of research dataset we are using.

The unavailability of this kind of ontology put us a big challenge to develop a complete ontology from our dataset. There are some terms (for example, process names) that exist in our dataset which are not defined clearly in any public dataset. But, it is important to have a clear reference to this kind of term as based on these process names the entire dataset is formed. For example, the process name “pig iron production” has its own meaning. We did find http://dbpedia.org/page/Pig_iron in DBpedia but “pig iron” does not refer to “pig iron production”. We believe, this is an important issue that industrial ecologists have to address by defining this kind of entities accordingly. Once these entities are defined and available as resources referred by url to some ontology, property, resource, etc, it is possible to join any data against any other data which will also facilitate the process of our target ontology construction.

6.1. Future Work

In this section, we present some areas of improvement and potential future work relating to the work presented in this thesis.

While querying the relational database with SQL, we performed several joining between tables. Any kind of joining between tables increase query processing time significantly depending on the size of tables. Therefore, efficient query plan, in other words, query optimization becomes important. This query optimization for better query performance is largely determined by the order in which the tables are joined. For example, when joining 3 tables A, B, C of size 10 rows, 100 rows, and 10000 rows respectively, a query plan that joins B and C first can take significantly more time to execute than the query plan that joins A and C first. This is an area of improvement that we should address. Some of the tables of our dataset have huge number of rows, therefore, query optimization will be crucial to improving the performance of the designed database.

As discussed in *section 3.3*, our implemented web interface allows using "year" and "country" tags to narrow down the query results. The current system allows to input these tags as an array list. The year string format is like- (2002,2005,1905...) which is unable to handle a certain range like -(1905-1999). That means, if a user wants to pull out the data with a range of 100 years, s/he must manually select the years one by one which is not a good practice. Another problem is the country tags. It also works similarly like "year" tag. The country string format looks like- ("Germany", "Italy", "Austria", "Switzerland" ...). This string format cannot answer the query if a user wants data based on a "continent", for example, Europe, Asia etc. Again, selecting 50 country of Europe one by one is not a good solution either. A possible solution could be constructing another database table defining a variety of region-based terms and relate these entries to the country table by defining foreign key between country and newly constructed table.

One of the scopes of our thesis was addressing the transformation techniques of data from the relational database to RDF triples. To do this, we followed the direct mapping process of D2RQ (*section 4.2*). This mapping technique uses the table and column names of the database to generate new vocabulary terms unless we customize the mapping file. For example, the column name "iso_code" from "countries" table become "countries_iso_code" in the vocabulary list. Often these default vocabularies become long string and do not provide a standard meaning. But, it is important to use standardized vocabularies to facilitate interoperability between data catalogues published on the Web. To make it easy, we should either use some public and well-known existing vocabularies wherever possible or we should only define new terms if the required terms are unavailable in existing vocabularies. This is an area for improvement that we should look deeper to build an ontology from a dataset like we used.

Another important future work follows the construction of a complete ontology for industrial ecology dataset. We did show the process of constructing the RDF triples from the relational database with D2RQ mapping and this database is built from the dataset itself, yet we did not build the complete ontology. We need to proceed one step further. To do this, we have to customize the generated mapping file (*section 4.1.2.2*) by adding more custom properties like linking the entities with external resources. The challenges behind this process already discussed in above para and in the *discussions section*. We believe, this could be an interesting research topic to build a domain-specific complete ontology for the industrial ecology group from their own research findings.

6.2. Conclusion

The aim of this thesis was to efficiently model the numerical research data of industrial ecology in a database. The main problem with these research data is their sizes. The single dataset that we used in this thesis has over half a million rows (relational). It is obvious that there will be more dataset like this (could be even bigger). Our target was to model these datasets in a database in a way that certain part of data can be retrieved by users through a web interface with a minimum effort. For this purpose, we designed a relational database to store our dataset and a web application on top of it.

Simultaneously, we looked into the use of triple stores as optional of our current MySQL database for improving the ability to query the data. One major fact that we considered is the performance of triple store in terms of query processing time and another important fact was to study the additional advantages that triple stores offer us to industrial ecology dataset. Triple stores store RDF triples, unlike relational database where data is stored in the tabular format. So, we discussed a mapping process of transforming our relational table data into RDF triples. We used Apache Jena TDB to store these generated triples and investigated the query processing time and more importantly some comparative features that both database models offer us.

Our findings suggest that taking only the query processing time into account, both database models do not outperform each other by a significant margin. Rather, it is more important to focus on the design complexity of both databases and what extent both models can provide richer use of industrial ecology dataset. In the previous chapter, we showed that although relational database performs very steadily to assure data accessibility and the design complexity is simpler, yet triple store grows database by pulling data from external ontologies and resources and thus provides more information from a dataset than the dataset itself has. This could be a major advantage for the industrial ecologists to follow such standards to publish their research data and take advantage of collaborating and re-using of data to get a richer picture of their research findings.

List of Figures

Figure 1: MySQL layer architecture	6
Figure 2: Classic Web and Web of Data.....	11
Figure 3: Simple RDF graph.....	12
Figure 4: Apache Jena Architecture.....	17
Figure 5: Dataset structure of Industrial Ecology Freiburg	19
Figure 6: Tables with columns and constraints among them.....	20
Figure 7: Architectural overview of Web Interface.	21
Figure 8: Back-End design architecture.....	22
Figure 9: Grid with list of datasets available in the database	26
Figure 10: DataGrid with stock data between processes	27
Figure 11: DataGrid with flow data between processes	28
Figure 12: DataGrid with material cycle data for fixed year and region	28
Figure 13: Unpolished Sankey diagram, as generated by the D3 application	29
Figure 14: Architecture of D2RQ	31

List of Listings

Listing 1: SQL query syntax to create a table.....	9
Listing 2: Simple triple with URI's	13
Listing 3: Sparql query pattern	14
Listing 4: Triples with foaf	15
Listing 5: Simple SPARQL query 1	15
Listing 6: Simple SPARQL query 2	15
Listing 7: MySQL database connection code	23
Listing 8: Data access code for queuing the data and store the data in Data Table.....	23
Listing 9: Data Service code to serialize the data in JSON format.....	24
Listing 10: Ajax code to call a data service and retrieve data from that service	24
Listing 11: Triples Loading function in TDB	35
Listing 12: SPARQL query execution function in TDB.....	35
Listing 13: A list of triples to define a stock data	39

List of Tables

Table 1: Query execution time comparisons for ief dataset	37
Table 2: Query execution time comparisons for iswc dataset	37
Table 3: Query execution time variations in Jena TDB for ief dataset.....	38
Table 4: Query execution time variations in Jena TDB for iswc dataset.....	38

Appendix

Query 1:

```
SELECT System_ID SI, System_definition SD, Dataset_name DN,  
Reference_date RD, Most_recent_update MRU, Corresponding_author_info CAI,  
Other_author_info OAI, Document_reference DR, region R,  
Time_period_of_analysis TPA, Indicator_element IE FROM  
ief_database.dataset
```

Query 2:

```
select stocks.system_id SI, stocks.stock_dataset_number SDN,  
stocks.process_id PID, proc.process_name PN, stocks.country_iso_code CIC,  
countries.country_name C, stocks.year T, stocks.age_cohort AC,  
stocks.indicator_element IE, stocks.aspect_of_dataset AD, stocks.value  
V, stocks.error_type ET, stocks.error_value_1 EV1, stocks.error_value_2  
EV2, stocks.data_quality DQ, stocks.unit_id UID, stocks.comment CM from  
ief_database.stocks stocks inner join ief_database.process_list proc on  
stocks.process_id = proc.process_id inner join ief_database.countries on  
stocks.country_iso_code = countries.ISO_code where stocks.process_id = 18
```

Query 3:

```
SELECT distinct f.system_id SI, f.flow_dataset_number FDN,  
f.process_id_source PIDS, (select p.process_name from  
ief_database.process_list p where p.process_id = 15) as PNS,  
f.process_id_target PIDT, (select p.process_name from  
ief_database.process_list p where p.process_id = 16) as PNT,  
f.region_source CICS, c.country_name CS, f.region_target CICT,  
c.country_name CT, f.year T, f.age_cohort AC, f.material IE, f.value V,  
f.error_type ET, f.error_value_1 EV1, f.error_value_2 EV2, f.data_quality  
DQ, f.unit UID, f.comment CM FROM ief_database.flows f inner join  
ief_database.countries c on (f.region_source = c.ISO_code or  
f.region_target = c.ISO_code) where f.process_id_source = 15 and  
f.process_id_target = 16 and f.year in(2005)
```

Query 4:

```
SELECT distinct f.system_id SI, f.flow_dataset_number FDN,  
f.process_id_source PIDS, (select p.process_name from  
ief_database.process_list p where p.process_id = PIDS) as PNS,  
f.process_id_target PIDT, (select p.process_name from  
ief_database.process_list p where p.process_id = PIDT) as PNT,  
f.region_source CICS, c.country_name CS, f.region_target CICT,  
c.country_name CT, f.year T, f.age_cohort AC, f.material IE, f.value V,
```



```
f.error_type ET, f.error_value_1 EV1, f.error_value_2 EV2, f.data_quality
DQ, f.unit UID, f.comment CM FROM ief_database.flows f inner join
ief_database.countries c on(f.region_source = c.ISO_code or
f.region_target = c.ISO_code) where f.year in(2001) and
(f.region_source in (276) or f.region_target in (276))
```

Query 5:

```
SELECT DISTINCT ?dataset_system_id ?dataset_system_definition
?dataset_dataset_name ?dataset_reference_date
?dataset_most_recent_update ?dataset_corresponding_author_info
?dataset_other_author_info ?dataset_document_reference
?dataset_region ?dataset_time_period_of_analysis
?dataset_indicator_element ?dataset_comment
WHERE {
  ?s vocab:dataset_system_id ?dataset_system_id .
  ?s vocab:dataset_system_definition ?dataset_system_definition .
  ?s vocab:dataset_dataset_name ?dataset_dataset_name .
  ?s vocab:dataset_reference_date ?dataset_reference_date .
  ?s vocab:dataset_most_recent_update ?dataset_most_recent_update .
  ?s vocab:dataset_corresponding_author_info
?dataset_corresponding_author_info .
  ?s vocab:dataset_other_author_info ?dataset_other_author_info .
  ?s vocab:dataset_document_reference ?dataset_document_reference .
  ?s vocab:dataset_region ?dataset_region .
  ?s vocab:dataset_time_period_of_analysis
?dataset_time_period_of_analysis .
  ?s vocab:dataset_indicator_element ?dataset_indicator_element .
  ?s vocab:dataset_comment ?dataset_comment .
}
ORDER BY ?dataset_system_id;
```

Query 6:

```
SELECT DISTINCT ?stocks_seq_id ?stocks_system_id
?stocks_stock_dataset_number ?stocks_process_id
?stocks_ISO_code ?stocks_year ?stocks_value ?stocks_unit_id
WHERE {
  ?x vocab:stocks_seq_id ?stocks_seq_id .
  ?x vocab:stocks_system_id ?stocks_system_id .
  ?x vocab:stocks_stock_dataset_number ?stocks_stock_dataset_number .
  ?x vocab:stocks_process_id ?stocks_process_id .
  ?x vocab:stocks_ISO_code ?stocks_ISO_code .
  ?x vocab:stocks_year ?stocks_year .
  ?x vocab:stocks_value ?stocks_value .
  ?x vocab:stocks_unit_id ?stocks_unit_id .
  FILTER (?stocks_year = 2005)
}
ORDER BY ?stocks_seq_id;
```

Query 7:

```
SELECT DISTINCT ?flows_seq_id ?flows_system_id ?flows_flow_dataset_number
?flows_process_id_source ?flows_process_id_target
?flows_region_source ?flows_region_target ?flows_year ?flows_value
?flows_unit_id
WHERE {
  ?x vocab:flows_seq_id ?flows_seq_id .
  ?x vocab:flows_system_id ?flows_system_id .
  ?x vocab:flows_flow_dataset_number ?flows_flow_dataset_number .
  ?x vocab:flows_process_id_source ?flows_process_id_source .
  ?x vocab:flows_process_id_target ?flows_process_id_target .
  ?x vocab:flows_region_source ?flows_region_source .
  ?x vocab:flows_region_target ?flows_region_target .
  ?x vocab:flows_year ?flows_year .
  ?x vocab:flows_value ?flows_value .
  ?x vocab:flows_unit_id ?flows_unit_id .
  FILTER (?flows_year = 2005)
};
```

Query 8:

```
SELECT DISTINCT ?flows_seq_id ?flows_system_id ?flows_flow_dataset_number
?flows_process_id_source ?flows_process_id_target
?flows_region_source ?flows_region_target ?flows_year ?flows_value
?flows_unit_id
WHERE {
  ?x vocab:flows_seq_id ?flows_seq_id .
  ?x vocab:flows_system_id ?flows_system_id .
  ?x vocab:flows_flow_dataset_number ?flows_flow_dataset_number .
  ?x vocab:flows_process_id_source ?flows_process_id_source .
  ?x vocab:flows_process_id_target ?flows_process_id_target .
  ?x vocab:flows_region_source ?flows_region_source .
  ?x vocab:flows_region_target ?flows_region_target .
  ?x vocab:flows_year ?flows_year .
  ?x vocab:flows_value ?flows_value .
  ?x vocab:flows_unit_id ?flows_unit_id .
  FILTER (?flows_year in (2005, 2003))
};
```

Query 9:

```
select concat(FirstName, LastName) as name, email, Phone from
iswc.persons;
```

Query 10:

```
select distinct paper.Title, concat(person.FirstName, person.LastName) as
name, org.Name from iswc.papers paper inner join iswc.rel_person_paper
personpaper on paper.PaperID=personpaper.PaperID
inner join iswc.persons person on personpaper.PersonID=person.PerID inner
join iswc.rel_person_organization po on person.PerID=po.PersonID
inner join iswc.organizations org on po.organizationID = org.orgID;
```

Query 11:

```
select distinct concat(person.FirstName, person.LastName) as name,
org.Name from iswc.persons person inner join iswc.rel_person_organization
po on person.PerID=po.PersonID
inner join iswc.organizations org on po.organizationID = org.orgID;
```

Query 12:

```
select distinct paper.Title, concat(person.FirstName, person.LastName) as
name, org.Name from iswc.papers paper inner join iswc.rel_person_paper
personpaper on paper.PaperID=personpaper.PaperID
inner join iswc.persons person on personpaper.PersonID=person.PerID inner
join iswc.rel_person_organization po on person.PerID=po.PersonID
inner join iswc.organizations org on po.organizationID = org.orgID where
person.FirstName="Andreas";
```

Query 13:

```
SELECT DISTINCT ?name ?email ?phone WHERE {
  ?person rdf:type foaf:Person.
  ?person foaf:name ?name .
  ?person foaf:mbox ?email .
  Optional {?person iswc:phone ?phone}
}
```

Query 14:

```
SELECT DISTINCT ?paperTitle ?authorName ?organizationName WHERE {
  ?paper dc:creator ?author .
  ?author foaf:name ?authorName.
  ?paper dc:title ?paperTitle .
  ?author iswc:has_affiliation ?organization .
  ?organization rdfs:label ?organizationName.
}
```

Query 15:

```
SELECT DISTINCT ?organizationName ?personName WHERE {  
  ?person rdf:type foaf:Person.  
  ?person foaf:name ?personName.  
  ?person iswc:has_affiliation ?organization .  
  ?organization rdfs:label ?organizationName  
}
```

Query 16:

```
SELECT DISTINCT ?paperTitle ?authorName ?organizationName WHERE {  
  ?paper dc:creator ?author .  
  ?author foaf:name ?authorName.  
  ?paper dc:title ?paperTitle .  
  ?author iswc:has_affiliation ?organization .  
  ?organization rdfs:label ?organizationName.  
  FILTER (?authorName = "Andreas Eberhart")  
}
```

Bibliography

- [1] Edgar Hertwich; Niko Heeren; Brandon Kuczenski; Guillaume Majeau-Bettez; Rupert J. Myers; Stefan Pauliuk; Konstantin Stadler; Reid Lifset: Nullius in Verba 1: Advancing Data Transparency in Industrial Ecology. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/jiec.12738>
- [2] Stefan Pauliuk; Tao Wang; Daniel B Müller: Steel all over the world: Estimating in-use stocks of iron for 200 countries. URL: https://www.researchgate.net/publication/257326646_Steel_all_over_the_world_Estimating_in-use_stocks_of_iron_for_200_countries
- [3] Edgar G. Hertwich: Consumption and the Rebound Effect: An Industrial Ecology Perspective. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1162/1088198054084635>
- [4] Chris Davis; Igor Nikolic; Gerard P.J. Dijkema: Industrial Ecology 2.0. URL: <https://onlinelibrary.wiley.com/doi/full/10.1111/j.1530-9290.2010.00281.x>
- [5] Rupert J. Myers; Tomer Fishman; Barbara K. Reck; T. E. Graedel: Unified Materials Information System (UMIS): An Integrated Material Stocks and Flows Data Structure. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/jiec.12730>
- [6] Material Flow Analysis database. URL: <http://www.stan2web.net/>
- [7] James A. Overton; OBO Tutorial. URL: <https://github.com/jamesaoverton/obo-tutorial/blob/master/docs/processing-data.md>
- [8] Jamal Bakkas; Mohamed Bahaj Generating of RDF graph from a relational database using Jena API. URL: <https://pdfs.semanticscholar.org/93b3/40fa0f818f859276a2dce8155338868a4647.pdf>
- [9] Date, C. J. (2003). An Introduction to Database Systems, Fifth Edition. Boston, MA: Addison Wesley.
- [10] Kristi Berg; Dr. Tom Joseph Seymour; Richa Goel: History of Databases. URL: https://www.researchgate.net/publication/298332910_History_Of_Databases
- [11] E. F. Codd: A Relational Model of Data for Large Shared Data Banks URL: https://www.researchgate.net/publication/220423134_A_Relational_Model_of_Data_for_Large_Shared_Data_Banks

- [12] DB-Engines Ranking of Relational DBMS. URL: <https://db-engines.com/en/ranking/relational+dbms>
- [13] MySQL Index. URL: <https://dev.mysql.com/doc/refman/5.7/en/mysql-indexes.html>
- [14] Berners-Lee, T.; Miller, E. The Semantic Web Lifts Off. URL: http://www.ercim.eu/publication/Ercim_News/enw51/berners-lee.html
- [15] "W3C Semantic Web Activity". World Wide Web Consortium (W3C). November 7, 2011. Retrieved November 26, 2011.
- [16] T. Berners-Lee; J. Hendler; O. Lassila: "The semantic web". In: Scientific American 284.5 (2001)
- [17] Jeremy J. Carroll; Graham Klyne: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. URL: <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>
- [18]. Tom Heath; Christian Bizer: Linked Data: Evolving the Web into a Global Data Space. URL: <http://linkeddatabook.com/editions/1.0/>
- [19] T. Bray; D. Hollander; A. Layman: Namespaces in XML. 1999. URL: <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- [20] World Wide Web Consortium W3C. Vocabularies. Accessed on Feb 6, 2013. URL: <http://www.w3.org/standards/semanticweb/ontology>
- [21] "RDF 1.1 Turtle: Terse RDF Triple Language". W3C. 9 January 2014.
- [22] "RDF 1.1 N-Triples: A line-based syntax for an RDF graph". W3C. 9 January 2014.
- [23] "RDF 1.1 XML Syntax". W3C. 25 February 2014.
- [24] "RDF 1.1 JSON Alternate Serialization (RDF/JSON)". W3C. 7 November 2013.
- [25] A. S. Foundation. (2012, 30 August 2012). Apache Jena. URL: <http://jena.apache.org>
- [26] Visualization tool: Online Circular Sankey Maker. URL: <http://www.visualisation.industrialecology.uni-freiburg.de/frmHome.aspx>
- [27] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. SIGMOD Record, 2004.

- [28] Hert, Matthias; Reif, Gerald; Gall, Harald: A Comparison of RDB-to-RDF Mapping Languages. URL: http://www.zora.uzh.ch/id/eprint/53354/1/20111217103857_merlin-id_2523.pdf
- [29] C. Bizer; A. Seaborne: D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In Proceedings of the 3rd International Semantic Web Conference, 2004
- [30] M. Arenas; A. Bertails; E. Prud’hommeaux; J. Sequeda: Direct mapping of relational data to RDF. W3C Working Draft 29 May 2012. URL: <http://www.w3.org/TR/2012/WD-rdb-direct-mapping-20120529/>.
- [31] S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language. W3C Working Draft 29 May 2012; URL: <http://www.w3.org/TR/2012/WD-r2rml-20120529/>.
- [32] T. Berners-Lee: Relational Databases on the Semantic Web. URL: <http://www.w3.org/DesignIssues/RDB-RDF.html>, 2009. Last visited July 2011
- [33] Souripriya Das; Seema Sundara; Richard Cyganiak: R2RML: RDB to RDF Mapping Language. URL: <https://www.w3.org/TR/r2rml/>
- [34] Apache Jena – TDB. URL: <https://jena.apache.org/documentation/tdb/>
- [35] D2R Server: Accessing databases with SPARQL and as Linked Data. URL: <http://d2rq.org/d2r-server>
- [36] Alisdair Owens; Andy Seaborne; Nick Gibbins: Clustered TDB: A Clustered Triple Store for Jena. URL: <https://eprints.soton.ac.uk/266974/1/www2009fixedref.pdf>