A Decision Support System for Degree Coordination

Rúben Alexandre Afonso Gama Anágua

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisors: Prof. Helena Isabel de Jesus Galhardas
            Prof. Nuno João Neves Mamede

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Helena Isabel de Jesus Galhardas
Member of the Committee: Prof. Cláudia Martins Antunes

November 2018
Acknowledgements

I am very grateful to my thesis advisors Prof. Helena Galhardas and Prof. Nuno Mamede, for their continuous guidance and support during this past year, and for their availability to help me out when needed. I am also grateful to the professors that have indirectly contributed to this document by providing me with the necessary knowledge to write this dissertation.

I would also like to thank my family, for playing a part in raising me, supporting me, and always believing in me. In particular, I will always very thankful to my mother for raising me in circumstances more challenging than usual, and for always being proud of me.

Last but not least, I am especially grateful to my friends, as they have supported me and helped me keep my mind clear.

Lisboa, November 29, 2018
Rúben Alexandre Afonso Gama Anágua
Dedicated to my mother for raising me, to the rest of my family for being there when I needed them, and to my friends for supporting and encouraging me.
Resumo

As instituições do ensino superior estão a aumentar a aposta no planeamento estratégico das suas atividades e cursos nos dias que correm, aumentando a necessidade de um sistema informatizado que as apoie neste processo. Uma das ferramentas que faz parte deste planeamento estratégico é a publicação de relatórios acerca do desempenho académico dos cursos oferecidos por tais instituições por coordenadores de curso. Estes relatórios permitem às partes interessadas determinar os factores que levam a um desempenho académico insuficiente por parte dos estudantes. No entanto, em muitas instituições, os dados para estes relatórios são obtidos a partir de um processo demasiado manual. Isto significa que vários coordenadores de curso têm de assegurar a consistência entre nomes de ficheiros, conteúdo de ficheiros, e fórmulas utilizadas nesses ficheiros todos os semestres, numa pilha de ficheiros que aumenta de forma constante e que leva a um processo ineficiente e propenso a erro humano. Propomos um sistema de apoio à decisão que extrai, periodicamente, dados académicos a partir desses ficheiros, transforma-os, carrega-os para um armazém de dados, e gera novos ficheiros Excel com os resultados relevantes, aumentando os níveis de automação e fiabilidade do processo.
Abstract

Over time, higher education institutions are increasing their emphasis on strategic planning of their activities and degrees, which raises demand for a computerized system to help them in this process. One of the tools currently used for strategic planning is the release of summarized reports about academic performance of the degrees offered by such institutions by degree coordinators. These reports allow stakeholders to clearly pinpoint the factors leading to insufficient student performance. However, in many institutions, data for the summarized reports are compiled in a process heavily relying on manual labor. This means that degree coordinators must ensure that, every semester, file names, file data, and formulas are consistent, in an always-increasing pile of files that leads to an inefficient process, which is prone to human error. We propose a decision support system to periodically extract academic data from these files, transform them, load them to a data warehouse, and generate new Excel files with relevant results, increasing the automation and reliability levels of this process.
Palavras Chave

Keywords

Sistema de Apoio à Decisão
Armazém de Dados
Ensino Superior
Desempenho Académico
Coordenação de Grau Académico

Decision Support System
Data Warehouse
Higher Education
Academic Performance
Degree Coordination
# Contents

1 Introduction .................................................. 1
  1.1 Motivation ................................................. 1
  1.2 Objectives ................................................ 2
  1.3 Solution .................................................. 2
  1.4 Document Outline ........................................ 3

2 Basic Concepts ................................................. 5
  2.1 General concepts ......................................... 5
    2.1.1 Data Warehouse ...................................... 6
    2.1.2 Extract, Transform, Load (ETL) ....................... 6
    2.1.3 Data Analysis and Multidimensional Models .......... 8
    2.1.4 Reporting ............................................ 11
  2.2 Domain-specific concepts ................................. 12
    2.2.1 Students ............................................. 12
    2.2.2 Data Sources ......................................... 13
    2.2.3 Reports .............................................. 13

3 Related Work .................................................. 15
  3.1 Business Intelligence Tools ............................... 15
    3.1.1 Pentaho .............................................. 16
    3.1.2 SQL Server Business Intelligence .................... 18
    3.1.3 Qlik .................................................. 19
    3.1.4 Discussion ............................................ 19
3.2 Decision Support Systems in Education .......................................................... 20
3.2.1 Methodological Framework for Data Warehousing and UBT Case Study .... 20
3.2.2 The Higher Education Information System in Croatia .............................. 23
3.2.3 Other Case Studies on Educational Data Warehousing ......................... 24
3.2.4 Discussion ................................................................................................. 27

4 The Degree Coordinator’s Decision Support System ......................................... 29
4.1 Goals .............................................................................................................. 29
        4.1.1 Business Requirements and Supported Queries ................................. 29
4.2 Architecture ................................................................................................. 33
4.3 Multidimensional Model .............................................................................. 33
        4.3.1 Dimension Tables ............................................................................... 34
        4.3.2 Fact Tables ......................................................................................... 36
4.4 Data Sources ................................................................................................ 37
        4.4.1 Updating ............................................................................................ 39
4.5 Schema Mappings ......................................................................................... 40
4.6 Populating the Data Warehouse ................................................................... 40
        4.6.1 Verification stage ................................................................................ 41
        4.6.2 Extract, Transform, Load (ETL) Process Overview ......................... 42
        4.6.3 Process Degrees ................................................................................. 42
        4.6.4 Process Admissions .......................................................................... 43
        4.6.5 Process Curricular Plans ................................................................. 45
        4.6.6 Process Grades .................................................................................. 46
        4.6.7 Other Transformations ..................................................................... 48
4.7 Querying the Data Warehouse ....................................................................... 49
        4.7.1 Run Course Queries ......................................................................... 49
        4.7.2 Run Generational Queries ............................................................... 51
List of Figures

2.1 Typical data warehousing architecture ................................................................. 5
2.2 Excerpt of the data source containing the tables of grades for the courses taught in the first semester of 2014/2015 as part of LEIC-T ................................................................. 14
3.1 PDI’s drag-and-drop interface. Example taken from the Pentaho website .............. 17
3.2 Methodological framework for data warehousing on higher education (Aljawarneh 2015). . 21
3.3 Representation of the proposed iterative data warehouse design method (Aljawarneh 2015). 22
3.4 Star schema of the data mart for academic performance (Aljawarneh 2015) .............. 23
3.5 Bassil’s snowflake schema, implemented in Microsoft Access 2010 (Bassil 2012) ....... 25
3.6 Architecture of a school’s data warehouse system and interaction with stakeholders (Kurniawan and Halim 2013). ................................................................. 26
4.1 Architecture of the developed decision support system ............................................. 33
4.2 Multidimensional model represented in the form of a diagram ................................ 35
4.3 Excel file containing degree data for LEIC-T and a similar degree, taught in another campus in the same institution ................................................................. 37
4.4 Excerpt of an Admissions file valid for LEIC-T ..................................................... 38
4.5 Excerpt of a Curricular plan file valid for LEIC-T .................................................. 39
4.6 Diagram describing the process developed to populate the warehouse ..................... 41
4.7 Diagram describing the ETL subprocess regarding degrees ..................................... 43
4.8 Diagram describing the ETL subprocess regarding student admissions .................... 44
4.9 Diagram describing the ETL subprocess regarding curricular plans ......................... 45
4.10 Diagram describing the ETL subprocess regarding student grades .......................... 47
4.11 Diagram describing the process used to run all supported queries ............................ 49
4.12 Diagram describing the process used to run all course-related queries .......................... 50
4.13 Excel file output by the subprocess associated with the fourth query .......................... 50
4.14 Graphical representation of the PDI transformation associated with our sixth query .... 51
4.15 Excel file output by the subprocess associated with query Q12 of Table 4.1 ............... 51
4.16 Diagram describing the process used to run all generation-related queries ................. 52
4.17 Dialog box as shown by Spoon when running the top-level PDI job .......................... 55

5.1 Execution time of the populator process and its components, by number of input genera-
tions of LEIC-T .................................................................................................................. 61
5.2 Execution time of the query executor process and its components, by number of in-
put generations of Bachelor of Science in Computer Science and Engineering at IST-
Taguspark (LEIC-T) ........................................................................................................ 62
5.3 Execution time of the query executor process, excluding Q13 of Table 4.1, and its compo-
nents, by number of input generations of LEIC-T ....................................................... 64

A.1 Multidimensional model represented in the form of a diagram, including field data types . 72
A.2 Multidimensional model as a diagram, including example field values ....................... 73

B.1 Main PDI job used to build the data warehouse ............................................................ 74
B.2 PDI transformation that handles Excel files containing data on degrees .................... 75
B.3 PDI transformation that handles Excel files containing data on admissions (split vertically) 76
B.4 PDI transformation that handles Excel files containing data on curricular plans (split verti-
cally) ................................................................................................................................ 77
B.5 PDI transformation that handles Excel files containing data on grades ..................... 78
B.6 PDI transformation that populates the num_enrollments attribute ............................ 78
B.7 PDI transformation that populates the enrolled_semester1 attribute ......................... 79
B.8 PDI transformation that populates the f_student_degree table ................................ 79
B.9 PDI transformation that populates the f_student_activity table ................................. 80
B.10 PDI transformation that populates the first_passed attribute .................................. 80
B.11 PDI transformation that populates the is_new attribute ........................................... 80

vi
List of Tables

2.1 Storage model and measure comparison of major OLAP architectures .......................... 10

3.1 Comparison of features among the considered BI stacks ................................. 20

4.1 Queries and respective granularity of Excel output files ................................. 31
4.1 Queries and respective granularity of Excel output files ................................. 32
4.2 Types of Excel files serving as data sources for the data warehouse ....................... 38
4.3 Mappings between source fields and target fields of the schema ......................... 40

C.1 Number of seconds required for the execution of the components of the populator process, by number of years of LEIC-T input data ............................................ 81
C.2 Number of seconds required for the execution of the components of the query executor process, by number of years of LEIC-T input data ............................................ 81
C.3 Statistics related to the generated sets of input files by number of years of LEIC-T input data ............................................................................................................. 82
Acronyms

**ACID**  Atomicity, Consistency, Isolation, and Durability

**BI**  Business Intelligence

**BIRT**  Business Intelligence and Reporting Tools

**DBMS**  Database Management System

**DSS**  Decision Support System

**ECTS**  European Credit Transfer and Accumulation System

**ETL**  Extract, Transform, Load

**GPA**  Grade Point Average

**HOLAP**  Hybrid OLAP

**IDE**  Integrated Development Environment

**IST**  Instituto Superior Técnico

**ISVU**  Croatian Higher Education Information System

**JSON**  JavaScript Object Notation

**KPI**  Key Performance Indicator

**LEIC-T**  Bachelor of Science in Computer Science and Engineering at IST-Taguspark

**LINQ**  Language Integrated Query

**MDX**  Multidimensional Expressions

**MOLAP**  Multidimensional OLAP

**OLAP**  Online Analytical Processing

**PAZ**  Pentaho Analyzer

**PDD**  Pentaho Dashboard Designer
PDI  Pentaho Data Integration
PRD  Pentaho Report Designer
PSW  Pentaho Schema Workbench
QUC  Course Unit Quality
RDBMS  Relational Database Management System
ROLAP  Relational OLAP
SQL  Structured Query Language
SSAS  SQL Server Analysis Services
SSIS  SQL Server Integration Services
SSMS  SQL Server Management Studio
SSRS  SQL Server Reporting Services
UBT  University of Business and Technology
XML  Extensible Markup Language
1.1 Motivation

The use of Decision Support Systems (DSSs) is becoming widespread in areas related to Business Intelligence (BI). Common DSSs are based on the analysis of data stored in previously set up databases, resulting in intuitive dashboards that display relevant information about the areas of an organization that need to be improved the most. These systems represent an efficient and valuable method to support decision making (Sauter 2010).

Higher education is one of the areas where DSSs can be used. The performance of the offered courses and the enrolled students depends on the conditions they are subjected to, such as the assigned professors and the balance of workload in the schedules of the students. These systems allow the end user, who can be a degree coordinator, to find out what courses need to be revamped or improved, or what conditions should be provided to students to increase their rate of success.

Currently, coordinators of academic degrees collect and extract large amounts of data regarding the performance of their students as part of their activities at the end of each semester. Historical data, regarding previous semesters, must also be kept, to keep track of the evolution of students and Key Performance Indicators (KPIs) related to them. Examples of KPIs relevant to the evolution of students are Grade Point Average (GPA) and the percentage of passed courses.

The process of collection and extraction of data is mostly manual, typically resorting to programs such as Microsoft Excel to organize the extracted data and transform it according to a set of formulas. As such, the creation of these reports may not be an efficient process due to heavy reliance on manual labor. The reports for most degrees are also built manually, and rely on several worksheets, to store data regarding grades, admission lists, and degree plans, rather than having such data stored in a single data repository. All of these worksheets add up over time, as they must be kept as historical data. Furthermore, file names, file data and formulas may become inconsistent, a scenario that degree coordinators wish to avoid as this raises confusion and decreases efficiency. While this process does not require an underlying system, it is inflexible, inefficient, and prone to human error.

Even though several examples of DSSs for higher education have been presented, few have been translated into actual development. Moreover, some of those are specific to certain universities or are
CHAPTER 1. INTRODUCTION

not directly related to the scope of our project – an academic degree and its students. As such, there is an open problem in this area, which consists of a generic platform that is able to process a given set of data sources containing academic data (typically in the form of Excel files), and then to automatically convert them into a data repository that allows visualization of transformed data via predefined queries. The implemented prototype does not directly generate data visualizations – instead it outputs Excel files with the query results, and then the features available in Microsoft Excel can be used to generate the actual charts and tables present in the degree coordinator’s reports.

1.2 Objectives

This project aimed to create a DSS to automate the process of extracting data, manipulating it, and obtaining Excel files used to create data visualizations, with the final goal of providing all the necessary data to create reports on student performance every semester. This system contains a structured data repository containing current and historical data, that degree coordinators will be able to query via graphical tools, producing Excel files containing data regarding indicators related to the evolution of academic performance in a given degree and its courses.

1.3 Solution

The development of a prototype for the application meeting the objectives described in Section 1.2 consists of four phases: designing a data warehouse to store the collected data, building the designed data warehouse, developing a process to periodically load data to the warehouse from a set of data sources, and applying some measures over the data, automatically producing Excel files with the desired results. These data sources are worksheets related to degree admissions, student grades, and curricular plans. The final infrastructure will be based on a typical data warehouse architecture, shown in Figure 2.1 and presented in Section 2.1.

As such, this project contributes to the evaluation of the performance of students in higher education degrees, for the assessment of the areas requiring future changes or improvements. A set of predefined queries, which generate different Excel files, will be provided, and the project should allow end users (the degree coordinators) to add more, without requiring a significant learning curve.

This project focuses on automating the generation of query results, leaving the possibility of integration with institution-wide frameworks, such as FenixEdu\(^1\), open. Future integration would enable the possibility of having different student performance information shown to degree coordinators, managers, and the students themselves in an intuitive way.

\(^1\)http://fenixedu.org/dev/overview/
Additionally, the development of the application should prioritize open-source or free third-party tools. This will open the possibility for other institutions to take advantage of it, as well as enabling future improvements by other developers without requiring them to purchase software.

1.4 Document Outline

This document is organized as follows:

- Chapter 2: detailed background on concepts about business intelligence and our specific domain;
- Chapter 3: overview on the third-party tools that can be used and specific proposals and implementations of data warehouses and DSSs in higher education;
- Chapter 4: description of the solution of the project and its architecture;
- Chapter 5: methodology of validation of the implemented solution;
- Chapter 6: conclusion to this document.
Basic Concepts

This section explains the concepts related to our domain, and it is split into two subsections: one containing general, wider concepts regarding BI, and another regarding domain-specific concepts. Section 2.1 explains general concepts related to BI. These concepts will then be related to the various steps in a data warehouse architecture constituting a DSS. Subsection 2.2 overviews more specific concepts related to higher education.

2.1 General concepts

The purpose of a DSS is to help decision making processes. To achieve this purpose, an architecture based on a data warehouse system, such as the one described on Figure 2.1, is built and used.

![Figure 2.1: Typical data warehousing architecture](image)

The central component of this typical architecture is the data warehouse. The first of four parts of this subsection will provide a general overview of this concept. The second part will explain the processes that information goes through, starting from initial sources, such as operational systems and flat files, and ending with a data warehouse. The main process is named ETL, and the concepts of data cleaning and data staging will be visited. The third part is based around data analysis and multidimensional models, providing an overview of Online Analytical Processing (OLAP) and data mining. We will visit the concepts of dimensions, dimension tables, fact tables, and database schemas to support the multidimensional models. Examples of operations that can be done with OLAP cubes will be seen, along with the major server architectures of OLAP. The final part refers to reporting and describes a set of
2.1.1 Data Warehouse

A data warehouse is a central repository of data collected from multiple sources, according to a unified schema (Han et al. 2012). It is considered a core component of BI as it facilitates access to current and historical data, and enables fast and simple strategic decision making (Ponniah 2010). As such, a data warehouse is an instrumental component of a DSS, connecting multiple, potentially inconsistent and redundant data sources to organization-wide query and analysis tools.

Data warehouses have four relevant characteristics: they are subject-oriented, integrated, non-volatile and time-variant (Inmon 2002). *Subject orientation* depends on the type of the organization (Han et al. 2012). Subject areas should be intuitive and obvious to the developer and to the end business user (Kimball and Ross 2013) (for instance, student, degree, and grade would be good subject areas for our domain). *Integration* means that data from disparate sources must be extracted and transformed to a singular, intercompatible format. *Non-volatility* means that data, once entered into the warehouse, should not change – data can be loaded and accessed, but it is should not be updated often (Inmon 2002). The last characteristic is *time variance*, meaning that every unit of data in the data warehouse is accurate at a given time instant (Inmon 2002). It can be implemented by replacing old data or by appending data to old data, not replacing it. The former would imply volatility in the data warehouse, thus the latter option is the acceptable solution. As such, the usual way of implementing time variance is by including a time stamp with the time of generation or upload of a record.

2.1.2 Extract, Transform, Load (ETL)

The key steps of an ETL process are extracting data, transforming it, and loading it to a data warehouse.

All three of those basic steps are essential: first, to change data into information that can be used, the data must be captured. After this extraction process, data should be selected based on user requirements. Data must undergo several transformations, and only then it can be converted into strategic data, which is data that can be analyzed by an organization to improve its performance. Transformed data is still not useful to end users until moved to the data warehouse repository; thus, data loading is also an essential step. Unsurprisingly, it is not uncommon for a team to spend 50% to 70% of their dedicated time on the development of ETL functions (Ponniah 2010).

Third-party ETL tools are commonly used, as they simplify the development (through automatic integrated metadata or prebuilt connectors for source and target systems, for instance), decreasing the overall costs of a project, and allow professionals with lower programming skills to perform ETL
2.1. GENERAL CONCEPTS

processes effectively (Kimball and Caserta 2004). Examples of third-party tools include Pentaho Data Integration\(^1\) and Microsoft SQL Server Integration Services\(^2\). Hand-coded ETL may, however, result in improved flexibility.

The first key step, data extraction, takes data from one or more data sources, which may be part of physically and logically incompatible systems. Such systems may be implemented in different operating systems, using different hardware, and using different communication protocols (Kimball and Caserta 2004). Data are frequently stored in a wide range of formats, commonly including, but not being limited to relational or non-relational databases, flat files, JavaScript Object Notation (JSON), and Extensible Markup Language (XML). Data may also have to be obtained from external sources, via screen scraping or web crawling. Extracted data are then usually converted into a common format to ease the transformation step, and they go through a validation process, to confirm that the obtained values match a given domain.

The second key step is data transformation. A common stage of this step is data cleaning: data must be cleaned to be accurate and of high quality. Accurate data must be correct, unambiguous, consistent and complete (Kimball and Caserta 2004). To be correct, data values should describe the objects they represent truthfully; to be unambiguous, data can only be interpreted with a single meaning; to be consistent, the values and descriptions in data use a common convention to express their meaning; and to be complete, the values and descriptions in data are defined for every instance that they should (all instances that must be defined are indeed not null). Examples of data cleaning methods are elimination of duplicate records (error discovery), and normalization of inconsistent data (data correction).

Standardization of data elements forms the second part of data transformation. Data elements from different sources may have different field lengths and may be represented by different data types, so these must be standardized. Semantics can also be standardized by merging synonyms and resolving homonyms (Ponniah 2010).

Summarization is often necessary, as keeping data at the lowest level of detail in a data warehouse is not always feasible and such data may never be queried. For instance, sales data detailing every transaction may be irrelevant; so, these data can often be transformed into a summarization of sale numbers by day and by product (Ponniah 2010).

The data are now cleaned, standardized, and summarized. As such, they are ready for the final key step in an ETL process – data loading. Data loading is split in two parts: the initial load to the data warehouse (which is often slow and big in size), and periodic data revisions, which are called incremental loads. Full refreshes may also occur on demand, and they consist in erasing the contents of one or more tables, reloading them with fresh data afterwards (Ponniah 2010).

\(^1\)http://www.pentaho.com/product/data-integration
\(^2\)https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services
A final notion related to ETL is *data staging*, which is optional. A decision to consider when building an ETL process is whether to stage or not to stage data. Data can be entirely processed in memory, which gets data to the ultimate target as fast as possible, or they can be part of a physical staging area, with the ability to recover from failures without restarting the entire process (Kimball and Caserta 2004).

This physical staging area, as shown in Figure 2.1, is an intermediate storage area to support the ETL process. It is usually implemented in the form of tables in relational databases and may have its contents erased before or after running an ETL process. Data should be staged (written to disk), after each key step in the ETL pipeline (Kimball and Ross 2013). As such, if a fatal error occurs during loading, we can take the staged, transformed data, and get back to loading straight away. Staging tables can be added, dropped, or modified with no warning by the ETL team, and these actions should have no negative consequences (such as broken reports) to the end users (Kimball and Caserta 2004).

### 2.1.3 Data Analysis and Multidimensional Models

In a typical architecture of a data warehouse system (Figure 2.1), we consider OLAP and data mining as two methods for data analysis. We will discuss OLAP in detail, starting with OLAP cubes, dimension tables, and fact tables. Database schemas to be used with OLAP, OLAP operations and the three major OLAP server architectures will also be discussed.

Figure 2.1 shows OLAP cubes as the connection between the data warehouse, and the layer of reporting tools, used by the end users. These cubes are arrays of data of various dimensions. A university may wish to summarize data by degree, semester, and course to compare student performance. For this case, a data cube, with degree, time (by semesters) and course as dimensions, can be formed – and this multidimensional array is called an OLAP cube.

As such, *dimensions* are perspectives with respect to which a business wishes to keep records (Han et al. 2012). *Dimension tables* are associated with dimensions and further describe these. For example, a dimension table for the most critical dimension in any analytical system – time – may contain the attributes *year*, *month*, and *day*.

Usually, multidimensional models revolve around a central theme, represented by a *fact table* (Han et al. 2012). A fact table contains keys to the related dimension tables and facts, which can be defined as measures and metrics of a business process. For example, a fact table for the data warehouse of a university can include *num_students* (number of students), and *avg_grade* (average grade) as facts.

These models follow a *database schema*, which is a set of entities and the relationships between them (Han et al. 2012), existing in the form of a star schema, a snowflake schema, or a fact constellation schema.
2.1. GENERAL CONCEPTS

The star schema is the most common paradigm. This schema revolves around a central fact table, containing the bulk of the data, and a set of dimension tables, one for each dimension (Han et al. 2012). These tables are displayed in a radial pattern around the central fact table, resembling a star.

The snowflake schema is a variant of the star schema, splitting data into additional tables, to minimize redundancy in dimension tables (however, this has a performance cost) (Han et al. 2012). For example, a dimension table in a star schema for the location dimension may include the attributes street, city, province, and country, besides the primary key. In a snowflake schema, we could consider a separate city dimension table, with the attributes city, province, and country. This table would be referenced by the location dimension table via a foreign key. The central fact table would be connected to location, which would be, in turn, connected to the city dimension table.

Fact constellations are a paradigm where multiple fact tables exist, sharing some dimension tables. For complex data warehouses, this is a popular schema; however, for data marts (subsets of the data warehouse focusing on a single subject), the star schema and the snowflake schema are the popular options because they are geared toward modeling a single subject.

Before jumping into the operations that can be used with OLAP cubes, it is important to define a concept hierarchy. A concept hierarchy is a sequence of mappings from low-level, specific concepts, to higher-level, more general concepts. For example, the location dimension had street, city, province and country as attributes. Each street can be mapped to a city, which can be mapped to a province, which can, in turn, be mapped to a country, forming a concept hierarchy.

OLAP cube operations exist to allow interactive and intuitive querying and analysis of the data at hand (Han et al. 2012), by materializing different views of data. Examples of popular OLAP operations are:

- Roll-up: navigates from detailed data to more general data, by performing aggregation on a data cube by rolling up a concept hierarchy (grouping data by a higher-level concept instead of a low-level concept in the hierarchy) or by dimension reduction (considering less dimensions);
- Drill-down: it is the reverse of roll-up. Drill-down operations are performed by stepping down a concept hierarchy (from a high-level concept to a lower-level concept), or by the introduction of additional dimensions;
- Slice: performs a selection on one dimension of the data cube, resulting in a subcube (a subset of the initial cube);
- Dice: performs a selection on two or more dimensions.

Lastly, the differences between OLAP paradigms should be discussed. There are three major OLAP
server architectures: ROLAP, MOLAP, and HOLAP (Han et al. 2012; Ponniah 2010). The differences in storage methods and measures are displayed in Table 2.1.

<table>
<thead>
<tr>
<th>Storage model/measure</th>
<th>MOLAP</th>
<th>HOLAP</th>
<th>ROLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregations</td>
<td>Cube</td>
<td>Cube</td>
<td>Relational tables</td>
</tr>
<tr>
<td>Detail-level values</td>
<td>Cube</td>
<td>Relational tables</td>
<td>Relational tables</td>
</tr>
<tr>
<td>Query performance</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Storage cost</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Cube maintenance cost</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

Relational OLAP (ROLAP) servers use relational databases to store and manage warehouse data. When queried, the ROLAP server consults the underlying database, retrieving the necessary data for answering it. This is very scalable, keeping data storage costs to a minimum. However, ROLAP is limited by the capabilities of Structured Query Language (SQL), not being suitable for complex analysis. Moreover, this limits query performance with the ROLAP architecture. Multidimensional OLAP (MOLAP) uses specialized multidimensional data cubes instead. Data are pre-computed and summarized, resulting in faster analysis, at the cost of higher load times and higher storage costs, which makes the architecture unsuitable for large systems. Data cubes support more complex analysis but, unlike ROLAP, do not support the dynamic addition of dimensions. Hybrid OLAP (HOLAP) represents an attempt to combine the strengths of both approaches, representing a middle-term architecture, as Table 2.1 shows. It allows data to be split into a relational partition (as ROLAP) for the lower levels of aggregation with are not frequently required, and a specialized, multidimensional partition (as MOLAP) for the higher levels of aggregation. For summarized information of low detail, HOLAP can then take advantage of fast analysis (due to the underlying data cubes), and it can hold large amounts of data as well (because lower levels of data are stored in relational databases).

OLAP cubes are not always required – some systems may only require one-dimensional queries, such as “What is the average grade of the Programming course?”. However, the following query would be much more useful: “What is the average grade of the Programming course, in the last three years, by degree, broken down by individual semesters, compared to the average grade of the Advanced Programming course?”. This query, like many others, is not feasible without OLAP cubes. End users went beyond basic queries and now must have the ability to analyze data in a wide variety of ways, along any number of dimensions, at any level of aggregation. They must have the ability to drill down and roll up along the hierarchies (sets of parent-child relationships, such as year-month-day, where a parent summarizes its children) of each dimension (Ponniah 2010). A data warehouse can only be considered complete with a system allowing multidimensional analysis.

OLAP may be complemented by data mining. Data mining represents a change of paradigm – OLAP is user-driven, and data mining is data-driven (Ponniah 2010). It is the computing process of discovering
valuable, non-obvious knowledge and patterns from a large amount of data (Han et al. 2012). Data mining tools take pre-processed data, such as the data present in the data warehouse architecture of Figure 2.1, and attempt to find additional hidden information that would have not been found otherwise.

2.1.4 Reporting

Data reporting corresponds to the presentation layer of a typical data warehouse system, as described in Figure 2.1. It collects data and submits reports of key elements related to the performance of an organization, providing instrumental information to trigger improvements in different aspects.

When compared to queries, reports are inflexible, as they are preformatted and rigid. End users have less control over the received reports than formulated queries, and, as such, a good set of guidelines is instrumental to ensure that the generated reports are useful to the end users (Ponniah 2010). The following guidelines should apply to any reporting tool:

• Preformatted reports: provide a library of preformatted, clearly described reports, with intuitive browsing functionalities.

• Parameter-driven predefined reports: provide some flexibility by allowing users to set their own parameters and subtotals.

• Easy report development: provide the end users with the ability to develop their own reports easily with a simple report-writing tool. These reports should complement preformatted reports and parameter-driven preformatted reports.

• Delivery options: provide a wide range of options to deliver reports, such as websites, or mass delivery of emails to an entire department.

• Data manipulation options: allow the users to perform various operations on data, such as pivoting and sorting.

• Presentation options: allow presentation of information in various formats, such as graphs, charts, maps, and tables.

The process of data reporting organizes information in order to allow end users to monitor how the different areas of an organization perform (Anderson 2015). As such, reporting is a very valuable component of a data-driven organization, but not sufficient. Reporting must be followed by a phase of analysis, which transforms data assets of the past into competitive insights. These insights drive business decisions, supported by DSSs, made to improve different areas in an organization.
2.2 Domain-specific concepts

Many degree coordinators compile a report analyzing the academic performance of their degrees. To compile such reports, a range of data sources related to students and courses in the degree is extracted, and certain measures are applied over those data to get key performance indicators in an intuitive form.

This subsection takes a look at student-related concepts and explains a credit accumulation system, then details the data sources used for the reports describing student performance, and ends with an overview of the currently available reports in our degree of reference, LEIC-T.

2.2.1 Students

One key measure of the performance of a student is the European Credit Transfer and Accumulation System (ECTS). This system represents a standard method of comparison between students of the volume of learning and the workload associated to it, which is an estimation of time taken for that purpose. Each full-time academic year is composed of 60 ECTS credits, divided by a set of courses. Bachelor's degrees using this system are designed to have a duration of three to four full-time academic years, meaning that a student has to accumulate 180 to 240 ECTS credits to complete those degrees.

In the context of a course, we will consider the following types of students:

- **Enrolled student**: enrolled in a course of a given degree;
- **Evaluated student**: delivered all mandatory elements of evaluation, regardless of grades in such elements;
- **Non-evaluated student**: failed to deliver one or more mandatory elements of evaluation;
- **Passing student**: evaluated with a final passing grade;
- **Failing student**: evaluated, but failed the course, not getting a final passing grade.

In the context of a degree, other three types of students are to be considered:

- **Admitted student**: accepted by the university to take the degree, in a given year;
- **Inactive student**: not evaluated in any course they have enrolled in, or has not enrolled in any course, in a given semester;
- **Active student**: evaluated in at least one course in a given semester.

---

A student that has been considered as active on the previous semester, but is inactive on the semester under analysis is considered a *withdrawal*; the inverse, a student transiting from inactive to active, is considered a *comeback*.

Students may be evaluated in up to two academic periods: the *regular academic period* is open to all students, and the *special academic period* is only available to specific groups of students, such as high performance athletes and student workers.

### 2.2.2 Data Sources

Every year, new students are accepted to degrees (the admitted students defined above). Information on when a student started a given degree, and why such student was accepted is very relevant to pinpoint what are the characteristics of the students doing better in a certain degree. Thus, one of the data sources is the yearly list of admitted students and their reason of admittance. Examples of common reasons of admittance are a high secondary school GPA, for students admitted by a degree’s general quota, and student transfers between degrees or institutions.

Secondly, every semester, tables of grades for the functioning courses constitute another data source – each grade lets us know if a student has passed the course (and how well they did), has been evaluated, or if they have just enrolled and given up, being instrumental to any kind of analysis.

An example of a set of tables of grades used in our degree of reference can be seen in Figure 2.2. This Excel file contains a set of data sheets, one for each course taught in the first semester of the 2014/2015 school year, where the grades of all enrolled students are shown. In the case of this particular data sheet, named “PO” (the short name in Portuguese for this course), the grades related to the Object-Oriented Programming course are displayed. The “Nota final” column indicates the final grade of a student, and we can see that the first enrolled student passed the course with a grade of 15 out of 20, and the second student has failed the course (as “RE” is used as a shortened version of a failing grade in these files).

The final source of data is the curricular plan of each degree, providing information on what courses constitute part of the degrees, what academic years of the degrees these courses correspond to, and how many ECTS credits are assigned to them. This curricular plan is not necessarily static, so historical data on it must be kept.

### 2.2.3 Reports

The main goal of these reports is to quickly and efficiently find out what needs to be improved. This may occur at the course level, with specific courses having low passing rates consistently, or at the degree
Figure 2.2: Excerpt of the data source containing the tables of grades for the courses taught in the first semester of 2014/2015 as part of LEIC-T

level, when students are collectively performing worse than before.

However, manual labor may be required to the point that creating these reports is not that efficient. Many degrees have their reports built manually, having many worksheets for the different tables of grades, admission lists, and curricular plans, rather than actual databases. These worksheets add up, as historical data must be kept. Consistency of file names, text, data and formulas, must be manually ensured, every semester, as different files, file names and conventions are all exposed to the end-user. This represents a labor-intensive, error-prone process.

Depending on specific situations of certain degrees, new measures may be deemed relevant, such as the evolution of the GPA of students. These measures should be transformed to visualizations in tables or charts, for an easier analysis on what exactly needs improvement. However, this represents an effort that is impractical without resorting to a data warehousing approach. This was the case with LEIC-T, where new measures considered to be useful in more recent reports were left out because adding them would require an unreasonable amount of effort.
3

Related Work

This section reviews relevant work regarding data warehousing and higher education and is organized in two parts: the different tools that can be used in this area (Section 3.1), OLAP and data warehousing in education (Section 3.2), where a general framework is discussed and several applications of DSSs in higher education are presented.

3.1 Business Intelligence Tools

An instrumental part for the development of this project is the set of tools used to develop our data warehouse-based system. This set of tools, commonly designed as a BI stack, was chosen for the implementation of the ETL process, and the presentation of data stored in the data warehouse. Features slated for future prototypes, such as OLAP analysis the automated generation of reports, must be compatible with the work already performed as part of this initial prototype.

The BI suite to use must be free or have a free edition or license, for the public or for academic purposes. This license must be obtainable automatically, without requiring manual approval, as the risk of a future developer not being able to use the platform must be low to none. This requirement excludes some influential players such as SAP\(^1\) and IBM\(^2\). WebFOCUS\(^3\) is another relatively popular platform but it is paid as well.

Other factors to consider are flexibility, learning curve, popularity, whether the code is open-source or not (as free software is not necessarily open-source), and cross-system compatibility.

Tableau, currently positioned as market leaders according to Gartner\(^4\), was also discarded, as its free edition, for academic purposes, requires manual approval by the responsible company. We cannot be certain that future developers will be successfully be approved by this manual process, thus this platform cannot be considered. SAS\(^5\) has a free University Edition, but it runs on a virtual machine. Installation is heavily preferred to improve flexibility and allow an actual client-server architecture. Oracle

---

2. [https://www.ibm.com/](https://www.ibm.com/)
5. [https://www.sas.com](https://www.sas.com)
is also very popular, but its ETL tool, the Oracle Data Integrator, is paid, and there are no free alternatives from Oracle.

Three BI stacks meeting the requirements above were found: Pentaho, SQL Server Business Intelligence, and Qlik.

In this section, we will describe these three BI stacks. We will provide a general overview on the free edition of each stack, and take a brief look at the tools used by each stack for ETL, OLAP analysis, and reporting.

3.1.1 Pentaho

Pentaho’s BI suite consists in a wide set of open-source pillars that work independently from each other, but can be combined to form a full BI stack. Pentaho’s free community edition includes a full BI suite of core tools, along with the ability to install several community-driven server plugins, which are installed on top of the Pentaho platform. These tools were all created with Java, making the suite compatible with all popular operating systems. Two popular examples are Ctools, a set of open-source tools mitigating some limitations of the core community edition, such as the development of dashboards, and Saiku, an analysis suite based on a RESTful server connecting to existing OLAP systems to deliver a set of configurable and intuitive analytics. Ctools is a collection of dashboard-oriented open-source tools, including a framework and an editor for dashboards, being an alternative to Pentaho Dashboard Designer (PDD), the enterprise edition plugin. Saiku is a data analysis tool based on a server connecting to existing OLAP systems, where data, that business users are interested in, can be looked at from a range of perspectives, with varying levels of detail.

The tool to use for the ETL process is PDI, codenamed Kettle. This tool allows the developer to create their data transformation steps and jobs (sequences of transformation steps), necessary for the ETL process. The graphical interface, illustrated in Figure 3.1 is based on drag-and-drop interaction, not requiring any lines of code. This inevitably limits flexibility as the developer is limited to a set of graphical elements rather than a full-fledged programming language; however, scripting and the creation of custom components are possible and intended for more advanced users. PDI is composed of four sequential core components:

- Spoon: the main component, used for the actual modeling and development of ETL.
- Pan: to execute transformations modelled with Spoon

---

6 http://www.pentaho.com/
7 https://help.pentaho.com/Documentation/6.1/0R0/CTools/CTools_Overview
8 https://meteorite.bi/products/saiku
9 https://help.pentaho.com/Documentation/8.0/Products/Dashboard_Designer
3.1. BUSINESS INTELLIGENCE TOOLS

- Kitchen: to orchestrate and execute jobs modelled with Spoon, creating a large business process composed of smaller jobs and transformations
- Carte: to set up dedicated web servers, in a single node or in a cluster of nodes, used to run and monitor jobs and transformations.

Figure 3.1: PDI’s drag-and-drop interface. Example taken from the Pentaho website

OLAP cubes can be created and tested with Pentaho Schema Workbench (PSW)\textsuperscript{10}, a tool that accesses fact and dimension tables in a Relational Database Management System (RDBMS) and outputs XML files representing metadata models. These models are compatible with Mondrian\textsuperscript{11}, the codename for Pentaho’s OLAP engine, and they are to be used by an OLAP server, such as Saiku.

For reporting purposes, the subset of dashboard-related tools in Ctools or Pentaho Reporting\textsuperscript{12} can be used.

Pentaho is commonly used with four types of RDBMSs: MySQL, SQL Server, Oracle, and PostgreSQL. PostgreSQL is an object-RDBMS with the advantages of being extensible, fully compliant with SQL standards, and it is guaranteed to be compliant with Atomicity, Consistency, Isolation, and Durability (ACID), a set of properties that guarantee that database transactions are valid regardless of external conditions. However, it is more complex than other open-source alternatives, suffering from low performance and low popularity. The most popular and best performing MySQL is the best choice of

\textsuperscript{10}https://mondrian.pentaho.com/documentation/workbench.php
\textsuperscript{11}https://community.hds.com/docs/DOC-1009853-mondrian
\textsuperscript{12}https://community.hds.com/docs/DOC-1009856-pentaho-reporting
a management system for this particular project, as compliance with SQL standards or ACID is not re-
required (though, MySQL may be ACID compliant under specific circumstances), and implementation of
security, replication and integration features with web servers is usually more straight-forward.

3.1.2 SQL Server Business Intelligence

Microsoft SQL Server\textsuperscript{13} is a popular, closed-source RDBMS developed by Microsoft. Its Express edition
is free, like the more complete Developer edition. The Enterprise edition, the one with the most features,
is paid, but it can be obtained for free via the Microsoft Imagine\textsuperscript{14} program. As such, we will consider
the features offered by the Enterprise edition. SQL Server 2017 is natively compatible with Windows
and Linux, and is compatible with Mac OS via Docker. It is the first version of SQL Server to be com-
patible with Linux, but popular applications such as SQL Server Management Studio (SSMS)\textsuperscript{15}, are still
Windows-only.

The tool used to model the ETL process is SQL Server Integration Services (SSIS). Much like PDI, the
ETL process can be modelled without writing a single line of code. However, it is possible to write
code to define custom tasks, transformations, log providers, and objects, to increase flexibility.

OLAP is handled by SQL Server Analysis Services (SSAS)\textsuperscript{16}. This tool supports relational OLAP,
multidimensional OLAP, and hybrid OLAP, unlike Mondrian, Pentaho’s analysis engine, which is based
on relational OLAP. The supported query languages for OLAP cubes are Language Integrated Query
(LINQ), a component of Microsoft’s .NET framework, and Multidimensional Expressions (MDX).

The final big component of Microsoft’s SQL Server services suite is SQL Server Reporting Services
(SSRS)\textsuperscript{17}. As we would expect given the tool’s name, it is a server-based application to generate reports.
A drag-and-drop interface is also available, not requiring the developer to write code. It is, however,
based on old technology, leading Microsoft to the development of Power BI\textsuperscript{18}.

Power BI is based on more recent technology and it also has a free, limited version. It is more
limited than SSRS, as paid editions of Power BI are not free for students. It also has a drag-and-drop
interface and it is more intuitive than SSRS.

SSIS, SSAS, and SSRS all run exclusively on Windows at this time, limiting the ability of choosing
this solution for other operating systems. Power BI is also Windows-only, but it includes a cloud-based,
system-independent version.

\textsuperscript{13}https://www.microsoft.com/en-us/sql-server/sql-server-2017
\textsuperscript{14}https://imagine.microsoft.com/
\textsuperscript{15}https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms
\textsuperscript{16}https://docs.microsoft.com/en-us/sql/analysis-services/analysis-services
\textsuperscript{17}https://docs.microsoft.com/en-us/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports
\textsuperscript{18}https://powerbi.microsoft.com/
3.1. BUSINESS INTELLIGENCE TOOLS

When compared to MySQL, SQL Server has three significant drawbacks: it is not open-source, compatibility with other operating systems is either very recent or non-native, and it is not entirely free.

3.1.3 Qlik

Qlik Sense and QlikView are the main products of Qlik\(^{19}\) and constitute an alternate approach to a BI suite. They have free editions, but the full stack of Qlik software can be obtained for free, through an application via their academic program\(^{20}\). However, this process is not automatic, and as such we will restrict ourselves to describing the free editions of Qlik Sense and QlikView. The free edition of QlikView has a very relevant limitation: it can only open applications developed on the same system. As such, it is not possible for unlicensed users to make changes to the application.

Unlike most BI stacks, Qlik does not use an underlying RDBMS nor database queries; instead, it loads all data into memory thanks to their associative engine\(^{21}\), and persistent data can be stored using proprietary QVD files. This makes Qlik a more intuitive program for business users when compared to more traditional BI stacks, with the drawbacks of not being able to handle large amounts of data or very complex ETL due to limitations with the provided tools.

ETL processes, and OLAP-based star schemas and snowflake schemas can be modelled with QlikView. We have seen before, however, that complex ETL may not be practical with a QlikView-based approach.

For OLAP analysis, the used tool is Qlik Sense. Unlike the previously analysed stacks, Qlik does not natively support MDX queries.

QlikView Reports, part of QlikView, is used to design reports. Its basic premise is to drag objects from the base QlikView application to another page, which has the layout desired by the user at a better resolution. This page can be then exported to PDF, for instance.

3.1.4 Discussion

Table 3.1 compares the three analyzed BI stacks, resorting to a set of factors that have been deemed relevant for the scope of this project.

The biggest limitation of Qlik is that its free edition can only open documents that were designed by the same installation of it (thus, free editions cannot open documents designed in other computers). It is

\(^{19}\)https://www.qlik.com/
\(^{20}\)https://www.qlik.com/us/company/academic-program
\(^{22}\)Based on https://www.capterra.com/business-intelligence-software/#infographic
Table 3.1: Comparison of features among the considered BI stacks

<table>
<thead>
<tr>
<th>Feature</th>
<th>Pentaho</th>
<th>SQL Server BI</th>
<th>Qlik</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-source</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Opens from any system</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Uses underlying RDBMS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Windows, Mac OS, Linux</td>
<td>Windows</td>
<td>Windows</td>
</tr>
<tr>
<td>Popularity rank</td>
<td>#19</td>
<td>#7</td>
<td>#3</td>
</tr>
<tr>
<td>ETL tool</td>
<td>PDI</td>
<td>SSIS</td>
<td>QlikView</td>
</tr>
<tr>
<td>OLAP/analysis tool</td>
<td>PSW, Saiku</td>
<td>SSAS</td>
<td>Qlik Sense</td>
</tr>
<tr>
<td>Reporting tool</td>
<td>Pentaho Reporting, Ctools</td>
<td>SSRS, Power BI</td>
<td>QlikView Reporting</td>
</tr>
</tbody>
</table>

possible to obtain an academic edition that can do so, for free, but it requires manual approval. This fact alone excludes the possibility of using Qlik, as the future development of applications on this ecosystem could be threatened.

Both Pentaho and SQL Server BI would be suitable for our domain. Pentaho has the advantages of being open-source and supporting Mac OS and Linux, while Microsoft’s BI stack is more popular and robust. Considering this, Pentaho seems to be the most suitable choice, unless a critical limitation with its tools is experienced; in this case, the more comprehensive Microsoft SQL Server stack would be the best solution.

To take full advantage of the fact that Pentaho is open-source, it would make the most sense to use an underlying open-source RDBMS with it. MySQL and PostgreSQL are the most popular open-source choices; MySQL performs better than PostgreSQL, and integration of this former system is usually simpler than integration of PostgreSQL. Moreover, the advantages of PostgreSQL are not crucial to our project, making MySQL the most sensible choice.

3.2 Decision Support Systems in Education

This section aims to analyze the current state of DSSs in education. In Subsection 3.2.1, we will describe a general framework for data warehouses in higher education, along with a case study of an implementation based on that framework. In Subsection 3.2.2, a nation-wide information system for higher education will be presented. Lastly, in Subsection 3.2.3, multiple examples of usage of data warehouses in higher education will be mentioned.

3.2.1 Methodological Framework for Data Warehousing and UBT Case Study

While it is widely accepted that data warehouses currently constitute crucial parts of the decision-making capabilities of a university (Aljawarneh 2015), few institutions have reliable data warehouse models, as
3.2. DECISION SUPPORT SYSTEMS IN EDUCATION

the field is not financially attractive, unlike activities such as banking and manufacturing (Baranović et al. 2003). There is no standard framework nor set of guidelines to properly design a DSS for higher education. In an attempt to fill this gap, Aljawarneh proposed a general framework and a set of guidelines to build data warehouses for higher education information systems (Aljawarneh 2015).

Designing and implementing a DSS is complex due to two major reasons. First, modelling a data warehouse takes time, causing higher positions of an organization to be reluctant to assign scarce resources to a project that is not having any results on a short term. Secondly, orienting an organization to a new DSS requires a revamp on their business processes. Like other projects, requirements can change over time, and such requirements cannot always be met by the DSS (Aljawarneh 2015).

With the above two reasons in mind, Aljawarneh defined two instrumental guidelines: an agile development methodology (iterative and incremental, encouraging rapid response to alterations in requirements), and the assignment of different priority levels to changes in specified requirements. These two guidelines constitute the basis of the proposed framework, represented in Figure 3.2.

![Figure 3.2: Methodological framework for data warehousing on higher education (Aljawarneh 2015).](image)

The proposed framework consists of four procedural steps, the first of which is problem observation. This step aims to identify two distinct types of problems: first, existing problems that could be solved with an architecture based on a data warehouse by observing current business processes, and second, potential problems that may significantly effect the decisions taken regarding the architecture to implement, by interviewing the potential end users, such as degree coordinators.

The second step, development, is composed by the specification of requirements, logical design, and physical design. Requirements specification is performed in three sequential rounds: the first round is user-oriented, where an initial document is created as a result of interviewing business executives
and end users, and requirements are prioritized according to their feedback, ensuring their engagement in the process; the second round is oriented to the observation of business processes, resulting in a requirements document with more detail; the final step is performed by examining source data to be used by the DSS, creating additional technical requirements, and resulting in the final and most detailed business requirements document. Logical design is the modelling of a data warehouse, with the result of this being a dimensional model for the warehouse. After logical design comes physical design, where a Database Management System (DBMS) is used to create the physical fact and dimension tables, where the primary key of each dimension references a foreign key in a fact table, and a primary key of a fact table is the composition of all primary keys of the related dimension tables, ensuring referential integrity.

Development is followed by a validation step, where the resulting system is tested to ensure that the requirements have been fulfilled, which may include performance measures. If some of the requirements have not been fulfilled, a revision may be required. As such, the development step may be revisited, starting from the logical design, as the requirements specification did not change. However, it is also possible that a few details were missed or left ambiguous in previous phases, and requests to change requirements may also occur. This forces changes in the business requirements specification, as shown by Figure 3.3. This is a normal process in an agile development methodology, where multiple iterations of refinements are expected.

Figure 3.3: Representation of the proposed iterative data warehouse design method (Aljawarneh 2015).

The final procedural step is the conclusion, where the resulting DSS is deployed to a production environment and readied for usage.

An example of application of this framework is University of Business and Technology (UBT), located in Jeddah, Saudi Arabia (Aljawarneh 2015). The first step consisted in the observation of business processes and in interviewing the director of IT at UBT, to discuss limitations of their traditional system.

Requirements for the data warehouse were specified via interviews (user-oriented), observation (oriented to business processes), and document examination (oriented to the operational source). As a result of the specified requirements, two data marts, for course registration and academic performance, were modelled as star schemas, as part of the logical design process. The latter is represented by Figure 3.4. The dimensions of time and student were the same for both data marts, meaning that they were conformed dimensions. Modelled facts include the GPA, a useful measure to reveal problems on
some courses or departments. The physical design stage was implemented with SQL Server as the underlying DBMS.

The validation step consisted in querying the system, basing the queries on questions asked by the end users of this system, to ensure that the defined business requirements had been met. After some iterations on development and validation, the system was concluded, being deployed for usage by the end users. This system served as a tool for building ad hoc reports, with information such as historical evolution of GPA and registration by degree and year.

![Star schema of the data mart for academic performance](http://www.isvu.hr/javno/hr/index.shtml)

**Figure 3.4:** Star schema of the data mart for academic performance (Aljawarneh 2015).

### 3.2.2 The Higher Education Information System in Croatia

The Croatian Higher Education Information System (ISVU)\(^{23}\) is a nation-wide information system, launched in 2000, to respond to the need of digitizing higher education institutions and of providing a seamless picture of the higher education panorama in Croatia. This project is funded and supervised by the Croatian Ministry of Science, Education, and Sports, and its data warehouse module stores information on students, professors, courses, curricular plans, student enrollments, and exams, with the ability of automatically generating reports on facts of these fields.

In 2003, Baranovic et al. (Baranović et al. 2003) presented the data warehouse behind this system. The requirements were specified according to two main goals: having a quick and efficient tool for data analysis, and having an intuitive and constantly available service for coordinators and administrative staff to generate reports. With these goals in mind, a multidimensional model, composed by a set of star schemas, was created, supporting queries on degree enrollment, course enrollment, course attendance, and exam taking.

\(^{23}\)http://www.isvu.hr/javno/hr/index.shtml
The only data source for this system was a relational database containing all required academic-related information at a nation-wide level. Even though all necessary information was stored in that database, querying it was a slow and inefficient process. As such, an ETL process was developed, copying the most relevant tables to a smaller replica of the relational database (the staging area of this architecture), transforming it, and loading it to a database based on a multidimensional model. The database replica and the multidimensional database were implemented in SQL Server, using Windows 2000 Server as the operating system. This multidimensional database is the source for the MOLAP server and its OLAP cubes, which interacts with the user via a web-based application. The choice of MOLAP as the server architecture led to improved query performance, with the drawbacks of a longer ETL process and higher storage requirements. The latter was not very relevant, as the data warehouse was relatively small.

In 2009, Mekterovic et al. (Mekterović et al. 2009) presented some improvements to the data warehouse, considering changes in requirements due to the introduction of the Bologna process\(^{24}\), and an evolution in other business rules. A more robust data cleaning step was introduced to the ETL process, to ensure consistency and accuracy of data.

Regarding data analysis and presentation, the developed web-based application for the presentation of data supported three types of queries:

- **Predefined queries**: the authenticated user could select a query from an existing set of queries that had already been defined;

- **Detailed ad hoc queries**: the authenticated user could choose a set of constraints via text boxes and combo boxes (drop-down lists where one option can be selected). This would then be dynamically translated to a parameterized SQL query, thanks to a scripting language;

- **Summarized ad hoc queries**: the authenticated user could perform a query through a drag-and-drop interface allowing some basic OLAP cube operations, such as drilling down dimension hierarchies and filtering by specific fields.

### 3.2.3 Other Case Studies on Educational Data Warehousing

Bassil (Bassil 2012) proposed a data warehouse design and corresponding ETL to be used in the typical information system of a university, representing its business activities, such as student enrollment, payment schemes, a method of archiving the records and achievements of *alumni*, and a management system to distribute assets, such as computers and printers, through different departments. The chosen

\(^{24}\)http://ec.europa.eu/education/policy/higher-education/bologna-process_en
multidimensional model was a snowflake schema, represented in Figure 3.5. This snowflake schema is centered around the Transcript fact table, directly connected to the dimension tables of Student and Instructor via foreign keys (that constitute the primary key of the fact table). Other dimension tables are indirectly connected to the fact table thanks to foreign keys – for instance, the Receipt dimension table contains a foreign key to the Account dimension table, which in turn contains a foreign key to Student.

![Figure 3.5: Bassil's snowflake schema, implemented in Microsoft Access 2010 (Bassil 2012)](image)

Goyal et al. (Goyal and Vohra 2012) proposed several applications of data mining in higher education. One of those applications is the prediction of student performance. The final grade of a student is predicted from a set of features based on existing data from a data warehouse, by using techniques such as neural networks. Matamala et al. (Matamala et al. 2011) provides a more in-depth analysis of the usage of neural networks over a data warehouse to measure student performance. The authors used the Pentaho BI stack to build charts on several variables, such as the expected passing rate of students in a course, in future years. A clustering process was performed to split the students into groups, enabling the possibility of identifying what students had the highest probability of passing a given course, taking into account certain variables, such as their region of residence, and whether they were enrolled in the course for the first time.

The use of data warehouses and data mining techniques to predict academic performance is also the focus of Kurniawan et al. (Kurniawan and Halim 2013), albeit not in the scope of higher education. Resorting to data mining techniques, the final grade of a student in a given school course is predicted using four variables: test grades, attendance, discipline, and delivery of assignments. Values for these variables are stored in an existing data warehouse system. If the predicted grade is insufficient, the academic stakeholders (teachers, directors, counsellors, parents, and the student themselves) are in-
formed of the reasons that led to the insufficient value and receive recommendations on how to improve it. Figure 3.6 describes this process: in short, data mining algorithms fetch the required information from a multidimensional data warehouse, predict the final grade of a student, and inform the stakeholders described above. The data warehouse is periodically updated with educational data from the school’s internal web-based system. As a result, stakeholders, when informed of the predicted grade, are still in time to act, being able to help the student to achieve a satisfactory grade.

![Figure 3.6: Architecture of a school’s data warehouse system and interaction with stakeholders (Kurniawan and Halim 2013).](image)

Sinaga et al. (Sinaga and Girsang 2016) implemented a data warehouse for university accreditation, where comprehensive reports on the activities of the university are provided to BAN-PT, an Indonesian accreditation agency for higher education. BAN-PT was created with the intent of enforcing quality standards in Indonesian higher education. Examples of information requested by this accreditation body are the average GPA of students in the last five years, the average waiting time for alumni to find a job, and the yearly ratio of candidate students to the total number of open spots for new students in the available degrees. The aim of the data warehouse is to provide the means to quickly generate the necessary charts and tables with such information. The logical design for the data warehouse created to provide the requested information is a multidimensional model, consisting on a set of three star schemas: one for student registration, one for student enrollment in courses, and one for lecturers. The ETL process was implemented using PDI, introduced in Subsection 3.1.1, with the extraction of
data from multiple sources (relational databases, Excel files, and Microsoft Word files) being considered as the biggest challenge.

### 3.2.4 Discussion

In this section, a general framework to build a data warehouse systems for higher education was presented and validated by a case study, and the most relevant examples of applications of data warehouses in the field were introduced, including an information system for higher education in Croatia. The usefulness of data mining for the prediction of academic performance was also mentioned, and it could work as a future expansion module for the project to develop.

However, the introduced examples have many shortcomings. The most salient shortcoming is the lack of detail – while most of the provided examples provided sufficient information regarding the logical design, with schematic diagrams being presented, knowing everything else is difficult: details on tools used to build the data warehouses are often omitted, the validation stage is not performed or not documented, examples of usage are insufficient or non-existent, and information on ETL, analysis and reporting is frequently vague. Furthermore, these examples do not come from particularly reputable sources and most of them are only usable in particular use cases, such as the system introduced by Sinaga et al. This system is only useful for reporting information related to specific fields, of specific forms, for a specific accreditation body, not being useful for any other applications. A final shortcoming is the unavailability of source code for any of the discussed examples, which means that we cannot reuse the code or even look at implementation details.
In this chapter, we will describe the first functional prototype of the decision support system that aims to provide key performance indicators related to the students and courses of a degree, aiding the decisions and suggestions of degree coordinators. A more in-depth look at the goals, the business requirements, and the process utilized to obtain them, will be taken in Section 4.1.

The general architecture of the implemented DSS is described in Section 4.2. Its central piece is a data warehouse, which was created according to the multidimensional model presented in Section 4.3. To populate it, different types of Excel files are used as data sources, which are described in more detail in Section 4.4. The correspondences between the elements in these data sources and the multidimensional model are described in Section 4.5.

The processes of populating the data warehouse and visualizing data by performing queries over the data warehouse are described in Sections 4.6 and 4.7, respectively. Lastly, specific tool-related details regarding the implementation are provided in Section 4.8.

4.1 Goals

The purpose of this project was to create a functional prototype of a DSS with the main goal of automating the processes performed by degree coordinators to gather and transform data related to student performance in the degrees they coordinate. This new system aims to boost the reliability of this process and to sharply reduce its learning curve, allowing easy expandability for future developers as well.

The guidelines of the created system and the defined business requirements are documented in Section 4.1.1.

4.1.1 Business Requirements and Supported Queries

Kimball’s approach for defining business requirements was followed, in a four stage process: preplanning, collecting the business requirements, conducting data-centric interviews, and a debriefing stage, where requirements are documented (Kimball and Ross 2013).
As part of the preplanning stage, the approach of face-to-face interviews to gather business requirements was selected, ensuring an in-depth look at what is needed, why is that needed, and what might be needed in the future. It was also defined that the primary end user would be the coordinator of our degree of reference, LEIC-T, and that the resulting system should be usable by other degree coordinators as well, whether they are from the same institution or not.

In the second stage, which regarded the collection of business requirements, the actual face-to-face interviews with the primary end user were conducted. At this stage, the input formats to support, the output formats, the collection of queries to support, and metrics to evaluate the system were all defined. Additionally, past spreadsheets and reports were brought to the interviews by the primary end user, facilitating the communication of what is expected of this system.

The third stage, regarding conduction of data-centric interviews, was done in cooperation with both the primary end user and a subject matter expert, to ensure that the required core data existed and was accessible, and to evaluate the potential drawbacks of the drafted multidimensional model, along with the feasibility of performing the required queries over a data warehouse built with this model in mind.

The last stage consisted in creating documentation for the business requirements and presenting such documentation, to ensure the alignment of everyone involved with the project. As a result, it was also defined that adding new generations at once on an average machine should take, at most, 30 minutes, with 11 generations of LEIC-T used as input, with the execution time increasing with the number of input generations in a linear fashion. The requirement for execution of all queries (filtered to LEIC-T), was to ensure that execution time would be under one hour for 11 generations of LEIC-T in the data warehouse, with the total execution time also increasing linearly with the number of input generations.

To evaluate the system’s correctness, it was stated that all 19 queries had to be implemented, and that, where comparable, the query results should match the results of past reports of the primary end user, who stated, however, that it is likely that multiple minor errors would have occurred in such reports.

Additionally, it was specified that the DSS would support the input Excel file types defined in Section 4.4, process them and populate a data warehouse based on the multidimensional model of Section 4.3, and support a set of 19 queries, defined in Table 4.1, with the results being written to a set of Excel files with the specified levels of granularity. Queries of type “C” are course queries, being computed in the context of each course, and queries of type “G” are generational queries, computed in the context of each generation.
### Table 4.1: Queries and respective granularity of Excel output files

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Query</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sheet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>File</td>
</tr>
<tr>
<td>Q1</td>
<td>C</td>
<td>Number of enrolled students (considering all students or first time enrollments)</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q2</td>
<td>C</td>
<td>Number of non-evaluated students and non-evaluated / enrolled student ratio</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q3</td>
<td>C</td>
<td>Number of evaluated students that failed to pass the course and their ratios against enrolled students and evaluated students</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q4</td>
<td>C</td>
<td>Number of evaluated students that passed the course and their ratios against enrolled students and evaluated students</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q5</td>
<td>C</td>
<td>Number of students that have already enrolled previously (number of enrollments is higher than 1)</td>
<td>By school year, semester, and generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q6</td>
<td>C</td>
<td>Number of students by grade (considering exact grade values or grade ranges)</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By course</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Two files (value + range)</td>
</tr>
<tr>
<td>Q7</td>
<td>G</td>
<td>Number of inactive students and percentage of inactivity</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q8</td>
<td>G</td>
<td>Number of withdrawals and comebacks</td>
<td>By school year and semester</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q9</td>
<td>G</td>
<td>Number of students completing a degree by number of semesters since admission (considering the number of ECTS credits obtained or completion of all courses in the curricular plan valid at the time)</td>
<td>By number of semesters since admission</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single file</td>
</tr>
<tr>
<td>Q10</td>
<td>G</td>
<td>Number of students by GPA (considering rounded integer GPA values or ranges) at a given semester</td>
<td>By GPA value and range</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By semester (value + range)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By generation</td>
</tr>
</tbody>
</table>
Table 4.1: Queries and respective granularity of Excel output files

<table>
<thead>
<tr>
<th>#</th>
<th>Type</th>
<th>Query</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Row</td>
</tr>
<tr>
<td>Q11</td>
<td>G</td>
<td>Number of students by cumulative GPA (considering rounded integer GPA values or ranges) until a given semester</td>
<td>By GPA value and range</td>
</tr>
<tr>
<td>Q12</td>
<td>G</td>
<td>Average number of enrollments needed to pass a course for the first time</td>
<td>By course</td>
</tr>
<tr>
<td>Q13</td>
<td>G</td>
<td>Number of students by number of courses left to complete the degree</td>
<td>By number of courses left</td>
</tr>
<tr>
<td>Q14</td>
<td>G</td>
<td>Number of students by number of courses passed, for the first time, at a given semester</td>
<td>By number of passed courses</td>
</tr>
<tr>
<td>Q15</td>
<td>G</td>
<td>Number of students by range of obtained ECTS credits at a given semester</td>
<td>By ECTS credit range</td>
</tr>
<tr>
<td>Q16</td>
<td>G</td>
<td>Number of students by range of cumulative obtained ECTS credits until a given semester</td>
<td>By ECTS credit range</td>
</tr>
<tr>
<td>Q17</td>
<td>G</td>
<td>Number of withdrawn students by total number of passed courses</td>
<td>By number of passed courses</td>
</tr>
<tr>
<td>Q18</td>
<td>G</td>
<td>Number of students not passing a course (considering failures or non-evaluations) and passing all others at a given semester</td>
<td>By course</td>
</tr>
<tr>
<td>Q19</td>
<td>G</td>
<td>Number of students by number of courses passed for the first time and number of enrollments in the given semester</td>
<td>By number of passed courses and number of enrollments</td>
</tr>
</tbody>
</table>
4.2 Architecture

The architecture of the implemented DSS can be broken down into three components: the set of data sources, the data warehouse populated using the aforementioned sources, and an analysis component, which provides the desired data visualizations. This general architecture can be seen in Figure 4.1.

![Figure 4.1: Architecture of the developed decision support system](image)

More specifically, the set of data sources for this system consists of four different types of Excel files, provided with varying periodicities to the data warehouse. This data warehouse forms the central piece of this architecture, and data are loaded to it via an ETL process over the data sources (Section 4.6). The analysis component provides a set of Excel files as output, obtained by performing SQL queries over the data warehouse (Section 4.7). These files can then be used to generate data visualizations.

4.3 Multidimensional Model

The multidimensional model of the data warehouse is a fact constellation, composed by five dimension tables and four fact tables. The relational schema of the tables is as follows (primary keys are underlined and foreign keys are denoted by “FK”):

- **d_degree** (degree_id, short_name, full_name, ects_to_complete, total_years)
- **d_student** (student_id, institutional_id, admission_stage)
- **d_time** (time_id, year, semester)
- **d_curricular_plan** (plan_id, degree_id, min_time_id)
  - degree_id: FK(d_degree)
  - min_time_id: FK(d_time)
- **d_course** (course_id, plan_id, short_name, full_name, year_number, semester_number, ects, is_new)
  - plan_id: FK(d_curricular_plan)
- **f_student_activity** (student_id, degree_id, time_id, active)
student_id: FK(d_student)
degree_id: FK(d_degree)
time_id: FK(d_time)

• f_student_admission (student_id, degree_id, time_id, enrolled_semester1)
  student_id: FK(d_student)
degree_id: FK(d_degree)
time_id: FK(d_time)

• f_student_degree (student_id, degree_id, degree_all_courses_time_id, degree_all_ects_time_id)
  student_id: FK(d_student)
degree_id: FK(d_degree)
degree_all_courses_time_id: FK(d_time)
degree_all_ects_time_id: FK(d_time)

• f_student_evaluation (student_id, course_id, time_id, regular_grade, special_grade, final_grade, state, num_enrollments, first_passed)
  student_id: FK(d_student)
course_id: FK(d_course)
time_id: FK(d_time)

This schema is also represented graphically in Figure 4.2. Primary keys are represented in bold and foreign keys are italicized. Associations between tables and their respective cardinalities are also represented. Alternatively, a full representation, including field data types unique/not null constraints, is also available in Figure A.1.

The aforementioned dimension and fact tables are detailed further in the following subsections. An example which includes example values for fields is also available in Figure A.2.

### 4.3.1 Dimension Tables

Information in this data warehouse is mainly structured in five different dimensions.

The d_degree dimension table stores information regarding the degrees that are to be considered by the system. This dimension is needed as the analysis procedure involves filtering data by degree, as degree coordinators are more interested in visualizing data from this perspective due to their positions. Each degree has a name, in its abbreviated form and in its full form, the minimum number of ECTS credits required for completion of the degree, and the number of years for which the degree is designed for (for example, our degree of reference, LEIC-T, is designed to have a duration of 3 years¹).

¹https://fenix.tecnico.ulisboa.pt/cursos/leic-t/curriculo
Figure 4.2: Multidimensional model represented in the form of a diagram
CHAPTER 4. THE DEGREE COORDINATOR’S DECISION SUPPORT SYSTEM

The student dimension is represented by the \textit{d\_student} dimension table. Each student is identified by its institutional ID (which is always unique and never set to null) and, optionally, by the stage of admission to the degree.

The time dimension is represented by the \textit{d\_time} dimension table. It is the most crucial in any analytical system. The chosen time granularity is the semester, as a semester is the minimum amount of time needed by a student to finish a course and to have an associated grade. Therefore, semesters and years are the units of time considered by our system.

The curricular plan of a degree is not necessarily static over time. For instance, our degree of reference, LEIC-T, had seven different curricular plans since 2006. Completing the minimum required number of ECTS credits in a degree is not a guarantee that the degree was indeed finished, as new mandatory courses that have not yet been passed by these students may have been created in the meantime, and a student must pass all of them to be a holder of the degree under analysis. As such, curricular plans are also stored on the database, thanks to the \textit{d\_curricular\_plan} dimension table. Each curricular plan is associated with a degree and a minimum semester of validity (for example, the current curricular plan of LEIC-T is valid since the first semester of 2015/2016, and until a new curricular plan arises), via surrogate keys identifying the degree and time dimensions, respectively.

The \textit{d\_course} table represents the course dimension. Each course is associated with a curricular plan, and it has a name, in its abbreviated form and in its full form, a year and semester number (for instance, a course can be taught in the first semester of the first year of the degree), a number of ECTS credits, and a Boolean \textit{is\_new} field. This field is set to true if the respective course has the lowest year and semester numbers for a given abbreviated course name, in a given curricular plan, and set to false otherwise. In the scope of our system, each course is associated with a single curricular plan (which is, in turn, associated with a single degree), and to a single year/semester pair of the degree, which means that different courses with the same name may exist.

4.3.2 Fact Tables

Four fact tables are part of this data warehouse, and they contain foreign keys to dimensions defined in Section 4.3.1 and facts or measures.

The data warehouse keeps track of a student’s activity in a given degree, at a given semester, by using a fact table named \textit{f\_student\_activity}. It contains a Boolean field named \textit{active}, which states whether a student was evaluated in at least one course, considering the provided degree and semester. Thanks to this table, the system can filter out inactive students (a concept defined in Section 2.2.1).

The \textit{f\_student\_admission} fact table keeps a record of all admitted students, the degrees that they were admitted to, and when admittance of such students occurred. A field named \textit{enrolled\_semester} is
available to filter out students that did not enroll in all available courses of the first semester (considering
the curricular plan valid at the time).

In the $f_{\text{student\_degree}}$ fact table, each student and degree is associated with two fields: $\text{degree\_all\_courses\_time\_id}$ and $\text{degree\_all\_ects\_time\_id}$. The former will be set to the lowest time for which the student has completed all available courses in the specified degree; the latter will be set to the lowest time for which the student has met or surpassed the minimum requirement of ECTS credits declared in the degree dimension.

The $f_{\text{student\_evaluation}}$ fact table contains data related to grades. Grades obtained in the regular academic period, in the special academic period, and final grades are all available for each course that a student has enrolled in, at a given semester. Additional facts are also available: $\text{state}$ determines whether a student has passed a course, failed it, or has not been evaluated (according to the definitions in Section 2.2.1), $\text{num\_enrollments}$ is the number of times that a student had enrolled in the associated course until the associated semester, and $\text{first\_passed}$ is set to true if that entry represents the first time that a student has passed the associated course, or set to false otherwise.

### 4.4 Data Sources

The four types of Excel files serving as data sources are summarized in Table 4.2, and they must be loaded into the data warehouse in the listed order. For instance, it is not possible to use an Excel file with admission information for a degree that has not been loaded previously into the data warehouse as input.

The Degrees file must be the first to be provided to the DSS. It includes the short and long versions of the degree name, along with the total number of ECTS credits required for completion of the degree, and the number of school years that the degree is designed for. No periodic updates are needed, unless the existing degrees change or new degrees are introduced. The manually created Excel file used in our prototype is shown in Figure 4.3.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Short name</td>
<td>Long name</td>
<td>ECTS to complete</td>
<td>Number of years</td>
</tr>
<tr>
<td>2</td>
<td>LEIC-T</td>
<td>Licenciatura Bol</td>
<td>180</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>LEIC-A</td>
<td>Licenciatura Bol</td>
<td>180</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4.3: Excel file containing degree data for LEIC-T and a similar degree, taught in another campus in the same institution.

The Admissions file type is provided every year as input to the DSS, as it is assumed that students are only admitted in the first semester of each school year. Two fields, the short degree name and the
### Table 4.2: Types of Excel files serving as data sources for the data warehouse

<table>
<thead>
<tr>
<th>Type</th>
<th>Periodicity</th>
<th>File fields</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees</td>
<td>Initial load</td>
<td>Short degree name, Long degree name, ECTS credits to complete, Number of years</td>
<td>One row per degree</td>
</tr>
<tr>
<td>Admissions</td>
<td>Yearly</td>
<td>Short degree name (in file name), School year (in file name), Institutional student ID, Admission phase</td>
<td>One row per student</td>
</tr>
<tr>
<td>Curricular plan</td>
<td>Semesterly at most</td>
<td>Short degree name (in file name), School year (in file name), Semester (in file name), Short course name, Long course name, Awarded ECTS credits on completion, Year number, Semester number</td>
<td>One row per course, year, and semester numbers</td>
</tr>
<tr>
<td>Grades</td>
<td>Semesterly</td>
<td>School year (in file name), Semester (in file name), Short course name (in sheet name), Institutional student ID, Short degree name, Regular season grade, Special season grade, Final grade</td>
<td>One sheet per course, with one row per student</td>
</tr>
</tbody>
</table>

School year, are parsed from the file name. Regarding the file content, this file has one row per admitted student. Each row includes a student ID and, optionally, their admission phase, as seen in Figure 4.4.

![Figure 4.4](image1.png)

**Figure 4.4:** Excerpt of an **Admissions** file valid for LEIC-T

The **Curricular plan** file contains information about the curricular plan of a degree. Both short and long versions of a course name, along with the number of ECTS credits it awards on completion and the numbers of the year and semester in which it functions are provided as part of the plan. The degree to which the courses are inserted, the school year, and semester are the same for all courses in the curricular plan. Therefore, these details are provided as part of the file name. An example of an Excel file of this type can be seen in Figure 4.5.

![Figure 4.5](image2.png)

**Figure 4.5:** Excerpt of a **Curricular plan** file valid for LEIC-T.
Lastly, Grades files are provided every semester. A Grades file includes a sheet for each course. Each sheet name matches the short name of the corresponding course. For each sheet, each row contains grading information regarding a specific student. As such, each row includes a student ID, the short degree name in which the student is enrolled, the final grade, and, optionally, grades for the regular season and special season, if they exist. The provided final grade is not necessarily a number: the strings "NA" (not evaluated) and "RE" (failed) are also accepted: in those cases, the different grade fields that constitute part of the f_student_evaluation table are set to null and the state field of the same table will contain the respective string. These files are the same files presented in Section 2.2.2 and Figure 4.3, not requiring any changes to be accepted by this system.

In our current use case, which is to assist the coordination of LEIC-T at Instituto Superior Técnico (IST) by interacting with this DSS, only the Grades files are automatically generated (via Fenix\(^2\), the campus management system used by IST). Admissions files are sent to the degree coordinator by request, and the remaining file types, Degrees and Curricular plan, are created manually by the end users. However, curricular plan details can be scraped from the degree website\(^3\).

### 4.4.1 Updating

The implemented prototype avoids insertion of duplicate data by only inserting a row if it does not exist yet. Duplicate rows are logged for future analysis by the end user and do not reach the loading stage. As a result, data sources used for previous insertions to the data warehouse may be reused or may be removed from the input file set, with no practical effect in the data warehouse itself.

It is also possible to update existing rows. For instance, a Grades file may contain an incorrect grade for a given student. In that case, the grade can be corrected in the Excel file, and the entire file can be submitted again, resulting in an updated row with the corrected grade. The remaining rows, which

---

\(^2\)https://ciist.ist.utl.pt/projectos/fenix.php

\(^3\)https://fenix.tecnico.ulisboa.pt/cursos/leic-t/curriculo
already exist in the data warehouse, remain unaffected as they are not inserted to the data warehouse.

### 4.5 Schema Mappings

The correspondences between the columns of the source Excel files (explained in Section 4.4) and the elements of the multidimensional model described in Section 4.3 are presented in Table 4.3.

<table>
<thead>
<tr>
<th>Source file types</th>
<th>Source fields</th>
<th>Target fields</th>
<th>Target tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees</td>
<td>Short degree name</td>
<td>short_name</td>
<td>d_degree</td>
</tr>
<tr>
<td></td>
<td>Long degree name</td>
<td>long_name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ECTS credits to complete</td>
<td>ects_to_complete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of years</td>
<td>total_years</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>degree_id (auto-incrementing)</td>
<td></td>
</tr>
<tr>
<td>Admissions</td>
<td>Short degree name (in file name)</td>
<td>degree_id (obtained from name)</td>
<td>d_student</td>
</tr>
<tr>
<td></td>
<td>School year (in file name)</td>
<td>time_id (by considering semester 1)</td>
<td>f_student_admission</td>
</tr>
<tr>
<td></td>
<td>Institutional student ID</td>
<td>institutional_id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Admission phase</td>
<td>admission_phase</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>student_id (auto-incrementing)</td>
<td></td>
</tr>
<tr>
<td>Curricular plan</td>
<td>Short degree name (in file name)</td>
<td>degree_id (obtained from name)</td>
<td>d_course</td>
</tr>
<tr>
<td></td>
<td>School year (in file name)</td>
<td>min_time_id</td>
<td>d_curricular_plan</td>
</tr>
<tr>
<td></td>
<td>Semester (in file name)</td>
<td>short_name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Long course name</td>
<td>full_name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ECTS credits on completion</td>
<td>ects</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Year number</td>
<td>year_number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semester number</td>
<td>semester_number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>plan_id (auto-incrementing)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>course_id (auto-incrementing)</td>
<td></td>
</tr>
<tr>
<td>Grades</td>
<td>Short course name (in sheet name)</td>
<td>course_id (obtained from name)</td>
<td>f_student_evaluation</td>
</tr>
<tr>
<td></td>
<td>School year (in file name)</td>
<td>min_time_id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semester (in file name)</td>
<td>student_id (obtained from name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Institutional student ID</td>
<td>regular_grade</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Short degree name</td>
<td>special_grade</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Regular season grade</td>
<td>final_grade</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Special season grade</td>
<td>state</td>
<td></td>
</tr>
</tbody>
</table>

### 4.6 Populating the Data Warehouse

The process of populating the data warehouse from a set of data sources is composed by two stages: a verification stage (Section 4.6.1), that creates the database tables if they have not been created yet
and ensures that valid input Excel files exist, and an ETL stage (Section 4.6.2), responsible for making
the developed set of data transformations run sequentially. A diagram of this process is shown in Figure
4.6.

It is composed by two stages: an initial stage (Section 4.6.1), that creates the database tables if
they have not been created yet and ensures that valid input Excel files exist, and an ETL stage (Section
4.6.2), where the developed ETL subprocesses run sequentially. This architecture is represented in
Figure 4.6.

![Diagram of the process developed to populate the warehouse](image)

**Figure 4.6: Diagram describing the process developed to populate the warehouse**

### 4.6.1 Verification stage

The first step in this stage is to verify whether the multidimensional model presented in Section 4.3
already exists in the database. If it does not exist yet, the tables are created via an SQL script containing
the table definitions. Additionally, if the time dimension is not populated, a stored procedure[^4] is called
to generate year and semester combinations, resulting in a populated \( d\_time \) table, with all supported
year/semester pairs.

Second, the specified directory (provided as an argument when running the top level process) and
its subdirectories are scanned to ensure that valid Excel files exist and can be scanned. If no valid input
Excel files are provided, the ETL stage, described in Section 4.6.2, will not be executed.

4.6.2 ETL Process Overview

The ETL process is represented in the second part of Figure 4.6. It is composed by two groups of subprocesses: the first group processes the different types of input Excel files, and the second group populates new tables and new attributes in existing tables of the data warehouse. Some subprocesses depend on data from previous subprocesses, leading to the decision of running them sequentially. Data already present in the data warehouse is ignored, and not reinserted into the data warehouse.

The first group contains four subprocesses handling a different Excel file type provided as input. These subprocesses are described in detail in Sections 4.6.3 to 4.6.6. Each process contains an extraction stage, where data are extracted from the Excel files of the correct type, a transformation stage, where data are validated, correctly formatted, and checked against existing database tables for duplicates, and a loading stage, which populates database tables with the transformed data.

The second group has the responsibility of populating the remaining tables and attributes of the data warehouse, with the main purposes of decreasing complexity of eventual queries and increasing query performance. It focuses on grouping existing data and performing new calculations over it to populate new tables and attributes, resulting in improved read performance. As such, this group must be executed only after all data from the input Excel files are loaded to the data warehouse – thus, execution of the first group must be finished before the second group starts executing.

This second group contains six subprocesses, which are described in Section 4.6.7. Each subprocess populates a new table or a previously unpopulated attribute of an existing table. As is the case with the transformations of the first group, each has an extraction stage, a transformation stage, and a loading stage. However, in this case, data are extracted from the data warehouse rather than Excel files, and transformation of data is a less complex process, as data from the data warehouse have already went through the transformation stages of the first group.

Graphical representations of the subprocesses, as obtained from PDI (the tool used to create them), can be seen in Appendix B.

4.6.3 Process Degrees

This transformation is represented in Figure 4.7. The respective PDI representation is shown in Figure B.2. It only interacts with one dimension and only basic data validation is performed, making it simpler than the other transformations to be discussed.

The extraction stage is performed by fetching the degree information for each input Excel file, described in Table 4.3. Unlike all other transformations, this subprocess does not fetch any data from the input file names.
4.6. POPULATING THE DATA WAREHOUSE

In the next stage, data transformation, rows are validated and invalid rows are logged and ignored. Duplicate entries, which have been defined as entries with the same degree short name, are resolved in favor of the last entry with that degree short name, as all other rows are logged and ignored. Next, the \textit{d\_degree} dimension of the data warehouse is queried to remove entries that are already present in the database. In this case, entries are considered as already present in the database if all fields (excluding the auto-incrementing ID) match.

The final step consists in loading all remaining rows to the \textit{d\_degree} dimension table.

4.6.4 Process Admissions

This transformation handles student admissions and interacts with multiple dimension tables and the \textit{f\_student\_admission} fact table. It is represented in Figure 4.8, and its full PDI representation can be seen in Figure B.4.

The extraction stage is performed by reading all rows from each input Excel file, each representing an admitted student, and by fetching the school year of admission and short degree name from the respective file name.

The provided years of admission and short degree names are all validated to make sure they are present in the respective dimension tables. Validation on the degree name is performed by querying
CHAPTER 4. THE DEGREE COORDINATOR’S DECISION SUPPORT SYSTEM

Figure 4.8: Diagram describing the ETL subprocess regarding student admissions

\[d_{degree}\] and getting the last ID with that short degree name. If no such ID exists, all rows related to that degree are ignored and written to a log file. It is then verified that the provided year of admission exists in the \[d_{time}\] dimension table by selecting the ID that matches with the first semester of that year. Similarly, if no such ID exists, the related rows are ignored and logged. The returned IDs for the degree and time dimensions are kept for eventual insertion in the \[f_{student\_admission}\] fact table.

Entries with the same institutional ID are considered as duplicate, and they are resolved in favor of the last entry with that institutional ID, with other rows being logged and ignored as well. Rows that could not be transformed to become sets of valid values are also removed and written to a log file. Some additional validation is performed to ensure that entries already existing in the \[d_{student}\] and \[f_{student\_admission}\] will not be inserted again.

Validated student information is loaded to the \[d_{student}\]. For each student, the auto-incrementing student ID is retrieved and joined with the already obtained \[degree\_id\] and \[time\_id\].
4.6.5 Process Curricular Plans

This transformation handles curricular plans, each being associated with a degree and a minimum year and semester. The ETL subprocess is represented in Figure 4.9. The respective PDI transformation is represented in Figure B.4.

In the extraction stage, each input Excel file’s content is read and associated with the minimum year/semester and the degree short name, which is parsed from the respective file name.

The parsed file name data is validated by querying the \( d_{\text{time}} \) and \( d_{\text{degree}} \) dimension tables to ensure that the provided time and degree information is present in the database. This validation step also serves as a way to retrieve the respective primary keys of these dimensions, which will be useful for a future insertion to \( d_{\text{curricular\_plan}} \). If either ID does not exist, the related rows are ignored and written to a log file.

Validation at the row level is required as well. Each row represents a course, and each course functions in a given year number and semester. This year number cannot be lower than 1 nor higher...
than the total number of years of the associated degree. This number of years is provided by the `total_deg_years` field of the associated degree. Additionally, it is also verified that the sum of ECTS credits of all declared courses in a curricular plan matches the total number of ECTS credits of the degree, as it is assumed that all courses are mandatory. This number is provided by the `total_ects` field of the associated degree. Therefore, the `d_degree` dimension table is accessed to perform these verifications, and invalid curricular plans are ignored and logged.

The transformation also ensures that no duplicate curricular plans will be inserted – if new courses are present (a course is considered new if at least one of its associated fields differs), then we are certain that it is a new curricular plan. However, a new curricular plan using only existing courses may be introduced, and this is why the database is again accessed to verify that no other curricular plan is associated with exactly the same list of courses.

After all invalid curricular plans are ignored, the remaining ones are inserted to the `d_curricular_plan` dimension table. The auto-incrementing IDs of the curricular plans are fetched, as they will be used to associate the inserted curricular plans with courses. The final step is the insertion of courses, associated with the respective retrieved `plan_id` field values, to `d_course`.

4.6.6 Process Grades

Student grades are handled last, as they are associated with degrees, students, and courses, which are handled by the other transformations. This transformation is represented in Figure 4.10, and its respective PDI representation can be seen in Figure B.5.

In the extraction stage, each input Excel file row is associated with the sheet name it is contained in (the short course name), and the school year and semester associated with the grades, which is parsed from the respective file name.

The next stage is the transformation stage, which begins with a sequence of validations. The first step is to validate the provided school year and semester pairs by querying `d_time`. Rows associated with pairs not present in the database are ignored and written to a log file. The second step is the validation of the course name; for this purpose, the short names of each degree and each course are obtained to find the respective course ID in the database. There may be multiple course IDs for the same degree name and course name – which is why the school year and semester are also important to find what curricular plan is in effect at that time, resolving that issue. There may also be no matching course ID in the database – in this case, the related rows are ignored and logged. The final validation concerns the students, as they must have been admitted to the associated degree to be associated with a grade. As such, the database is again accessed to ensure that each student is present in the database, and that they are associated to the correct degree.
After all validations described above, grade values are normalized. Grades are usually positive numbers, can also be provided as "NA" or "RE" strings, which mean that a student has not been evaluated or failed the course, respectively. However, it will be difficult to make mathematical operations on the grade values if they can also be strings, so a state field will let us know if a student has passed a course, failed it, or has not been evaluated, and the grade values are restricted to being just numbers or null values.

Some final validations are performed afterwards: first, duplicate rows are resolved in favor of the last valid one (the last row is kept, while all others are removed), incorrectly formatted rows are also remove, and the f_student_evaluation table is queried to ensure that no rows already existing in the database will be inserted. All removed rows during these validations will be written to a log file for future analysis by the degree coordinator.

The final step consists in the loading stage, where student grades are inserted to the f_student_evaluation fact table.
4.6.7 Other Transformations

This group of ETL subprocesses was introduced in Section 4.6.2 and it has the purpose of improving the read performance of the data warehouse, along with reducing the complexity of certain types of queries.

Two of these subprocesses are responsible for populating fact tables. One of these populates $f_{\text{student\_degree}}$, which, as explained in Section 4.3.2, stores the semester at which a student has obtained the required amount of ECTS credits in the scope of a degree (or it stores a null value if the student has not done so), and the semester at which a student has passed all courses in the scope of a degree (with the default null value also applying here). For this purpose, two different SQL queries are performed: the first fetches the semesters at which students have completed the minimum ECTS requirement of their degrees, and the other fetches the semesters at which students have completed all courses present in the curricular plan valid at that time. These results are then merged into a single stream, which is used to populate the aforementioned database table.

The other fact table being populated during this stage is $f_{\text{student\_activity}}$. An initial SQL query fetches all students, the degrees they have enrolled in, and the semesters in which activity is considered. For each student/degree combination, the semesters to be considered are all semesters between the time of admission and the time of degree completion. Therefore, this transformation depends on $f_{\text{student\_degree}}$. Then, each trio of student/degree/semester returned by the first query is used as input to a second query, which returns a Boolean value stating whether the student was active in the given degree, at the given semester. Finally, the transformation is ready to populate $f_{\text{student\_activity}}$ with the data fetched from both queries.

The remaining four subprocesses populate new attributes on existing tables. The first sets the num_enrollments field of $f_{\text{student\_evaluation}}$, which represents the number of times that the student has enrolled in a course with the same name until the semester under analysis.

The second remaining subprocess populates the enrolled_semester1 field of $f_{\text{student\_admission}}$. This field contains a Boolean field, which is set to true if a student has enrolled in all courses in their first semester since admission, or is false otherwise. To obtain this value, two SQL queries are executed and then joined, returning all student grades in their first semester since admission. Then, the courses declared in the curricular plan valid at the time for the degree to which they were admitted, taught in the first semester of the first year, are selected, and the students having enrolled in all of these get their respective enrolled_semester1 value set to true, while the remaining students have their value set to false.

After populating the enrolled_semester1 attribute, another subprocess has the goal of setting the Boolean first_passed field of $f_{\text{student\_evaluation}}$. For this purpose, all evaluations in which the state is set to "AP" (i.e., the student has passed the course) are fetched, and then past evaluations with the
state also being set to “AP” for the same student and course are searched for. If such past evaluations exist, this attribute is set to false, or it is set to true otherwise.

Lastly, the *is_new* field of *d_course* is populated in this stage as well. For this purpose, all courses in the same curricular plan are obtained, grouped by name to find out the minimum semester number at which they are taught, and finally, the *is_new* field is set to true if the course entry matches the calculated minimum semester number, or false otherwise.

### 4.7 Querying the Data Warehouse

The process of gathering output data from the warehouse can be split into two phases: one manages course queries over the data warehouse, and another manages queries regarding generations, as seen in Figure 4.11. Each subprocess gathers all possible results for all queries in an individual degree, as provided to the DSS when running this process.

![Figure 4.11: Diagram describing the process used to run all supported queries](image)

Details on the queries used by these subprocesses can be found in Table 4.1. For instance, the first query in the aforementioned table is of type "course"; therefore, executed by the first subprocess), and outputs a single Excel file split into sheets, each representing a different course. Each worksheet contains a row per school year/semester pair (as long as the respective course was taught in that year/semester).

The subprocess responsible for executing course queries is presented in detail in Section 4.7.1; the subprocess used to run generational queries is presented in Section 4.7.2.

#### 4.7.1 Run Course Queries

Running course queries on this system’s data warehouse is a two stage process: first, the system gets all courses defined in any curricular plan of the degree, and then, all query subprocesses run for each course. This architecture is represented in Figure 4.12.

![Figure 4.12: Architecture of the course query subprocesses](image)

Thus, in the initial stage, an SQL query responsible for getting all course short names ever defined in a curricular plan associated to the selected degree is executed. These course names are then retrieved by a higher level process, which forwards the results to the second stage.
In the second stage, a lower level process is called to run all associated subprocesses, each associated to a query (one of the first six of Table 4.1), and write their output to Excel files. Each subprocess structure includes an extraction stage, where the data warehouse is queried, an optional transformation stage, where extracted data is joined with other queries or transformed to the desired formats, and a loading stage, where the results are written to a single Excel file. An example of this single output Excel file can be seen in Figure 4.13, which was created as a result of the execution of the fourth query of the aforementioned table, considering our degree of reference. This lower level process runs once for every course name retrieved in the first stage.

However, one of the course queries (sixth query of Table 4.1) is an exception as it does not output just to a single Excel file. Instead, the associated subprocess executes an SQL query to get the number of students for each grade value for one course, writes the results to an Excel file, and then groups the results into grade ranges that may be more relevant to business users, which are written to a separate
4.7. QUERYING THE DATA WAREHOUSE

Excel file. This behavior can be seen on Figure 4.14, which is the graphical representation, as obtained from PDI, of the transformation associated with this query.

![Graphical representation](image)

Figure 4.14: Graphical representation of the PDI transformation associated with our sixth query

### 4.7.2 Run Generational Queries

The process of generating results for generational queries is more complex, as there are two different levels of granularity from the perspective of output Excel files:

- **Level 1**: Single file, one sheet per generation;
- **Level 2**: One file per generation, one sheet per semester since admission of that generation.

For queries (as defined in Table 4.1) of the aforementioned level 1 of granularity, the two stage process of Section 4.7.1 also applies here: in the initial stage, all years with at least one student admitted to the current degree are fetched (as long as they are within the ranges defined when running the top-level process), and, as part of the second stage, a subprocess is called once per fetched admission year, and it runs all generation queries of this level of granularity and outputs the results to Excel files. An example of such output files can be seen in Figure 4.15, where the results of query Q12 (as per Table 4.1) are listed for the degree of reference.

![Excel file output](image)

Figure 4.15: Excel file output by the subprocess associated with query Q12 of Table 4.1

For queries of the deeper granularity level 2, a nested loop is required. For each admission year previously fetched, the system runs an inner loop to fetch a number of semesters since admission...
(the lowest of the maximum number is the highest semester for which the data warehouse has any information, and the semester of admission plus the maximum number of semesters defined when running the top-level process). Then, each combination of admission year and semester is fed to another lower level process, which executes all transformations associated with queries of this level of granularity, and writes a different Excel file for every generation, and a different sheet in each file for each semester considered in the same generation.

The concatenation of the subprocesses above results in the diagram represented in Figure 4.16.

![Diagram](image)

Figure 4.16: Diagram describing the process used to run all generation-related queries

### 4.8 Implementation Details

Taking into consideration that this project is intended to be a first functional prototype, the Pentaho ecosystem, presented in Section 3.1.1, with MySQL as the underlying DBMS, was chosen to create and manage the data warehouse of this solution.

The essential tools of this system are free, open-source, and work in all popular operating systems, making it a perfect choice to allow future developers to use this prototype as a baseline for future improvements. This is possible because developers can use they environment of choice to reproduce
this functional prototype, and add features on top of it by using other tools that constitute the Pentaho ecosystem.

4.8.1 Populating the Data Warehouse – Implementation

The process of populating the data warehouse is managed by a PDI job (represented in full in Figure B.1). A PDI job aggregates individual pieces of functionality sequentially with the end goal of implementing an entire process. These pieces of functionality are usually PDI transformations, which define data flows, executing the various steps in them in parallel when possible\(^5\).

This is the case with the ETL stage of the process defined in Section 4.6. Each subprocess we have mentioned previously, is implemented as a PDI transformation, a small piece of functionality that is part of a bigger process that manages these transformations, the PDI job. This job can be executed in two different ways: graphically, or in the command line by running a script.

The data warehouse can be created by opening the PDI job using Spoon, introduced in Section 3.1.1, and running the job. The process supports some input parameters related to database connection details, paths to external SQL scripts, the path to the directory containing the input Excel files, and the quantity of messages to log. The files do not need to be in the same directory, as all subdirectories of the provided path will also be recursively scanned for those files. Rows failing the implemented validation checks are logged and not inserted into the data warehouse. These logs are shown in Spoon and can also be directed to an external file. After reading the logged messages, the user may correct the invalid rows and run the process again, as rows already existing in the data warehouse will not be reinserted.

An alternative way to run this process without requiring a graphical user interface was also created via an executable Python script. It supports the same input parameters, but they are introduced in the computer terminal used to run the script. In its initial stage, the input Excel files with information grades are cleared of any non-relevant rows, and then the inserted parameters are directed to Kitchen, the component of PDI that executes jobs from a command line.

This Python script alternative was developed with three considerations in mind: first, it is a necessity to enable automated periodic updates to the data warehouse, by running the command with a system task scheduler; second, Spoon takes several seconds to load, and the graphical elements are not required to execute this process; third, we were unable to skip the leading invalid lines that exist on the automatically generated Grades files of our degree of reference by using PDI itself, so a fix for this outside PDI had to be implemented. Although these lines could be deleted manually by the end users, this system aims to minimize the amount of manual labor and the learning curve as much as possible. Messages logged by PDI are part of this script's output.

\(^5\)https://help.pentaho.com/Documentation/8.0/Products/Data_Integration/Data_Integration_Perspective/010
Taking the aforementioned third consideration into account, the first execution of this process should be performed by the Python script if the initial rows of any sheet in any grades Excel file do not contain data nor headers. Even though PDI’s Excel input step\(^5\) contains a field to ignore a certain number of rows per sheet before processing the actual data, this number must be static and we have found that it varied for different Excel files. In addition, Excel files without these irrelevant rows should be able to be processed in full by PDI, which this would not occur if rows were skipped. Fortunately, the Python script overwrites the original files with the respective clean, valid versions, which allows usage of the graphical alternative in future executions if desired.

4.8.2 Querying the Data Warehouse – Implementation

The approach of querying the data warehouse is different. PDI jobs can be executed by higher level PDI jobs, and this possibility was considered when designing the process presented in Section 4.7.

As such, our top-level PDI job actually executes two different PDI jobs, one for execution of course queries, and another for execution of generational queries, as per Figure 4.11.

The job responsible for the execution of course queries takes advantage of another feature, which is the possibility of running a lower level job for every row in a stream. In this particular case, the job should be executed for every course name ever defined. As such, initially, a PDI transformation runs an SQL query to get all course short names ever defined in a curricular plan associated to the selected degree, and keeps them in the stream, to be retrieved by the external job, thanks to the PDI Copy rows to result\(^7\) step. Then, an inner job, which is the job that finally executes the defined course queries by executing the PDI transformations associated with them, is executed for every row in the stream (with each row containing a distinct course name). This architecture was defined in Figure 4.12.

The process utilized for generational queries is similar, as defined in Figure 4.16.

Execution of the top-level PDI job is possible using one of two different methods: the first uses a graphical user interface, and the other uses a command line. Therefore, the presented alternatives are the same as the usable methods to populate the data warehouse from input Excel files, described in Section 4.8.1.

The first method uses Spoon to open the file containing the top-level PDI job, and then running it. Running this job opens a dialog window by default, where all configurations, such as the custom parameters of our system and the log level, can be set\(^8\). This dialog box is represented in Figure 4.17.

---
\(^5\)[https://wiki.pentaho.com/display/EAI/Excel+Input+(XLS,+XLSX)+including+OpenOffice+Workbooks+(ODS)]
\(^6\)[https://wiki.pentaho.com/display/EAI/Copy+rows+to+result]
\(^7\)[https://help.pentaho.com/Documentation/8.0/Products/Data_Integration/Data_Integration_Perspective/040/010]
The supported custom parameters, which are common to both alternatives of running this process, are as follows:

- Database connection details;
- Degree abbreviation to consider;
- Range of admission years;
- Possible minimum and maximum grades (useful if a grading system is different than the Portuguese, where the minimum grade required to pass a course is 10 and the maximum possible grade is 20);
- Maximum number of semesters since admission;
- Path to the directory containing the Excel files to output.

This wide range of custom parameters increases the flexibility of the system, by tailoring it to more business users, that may use a different grading system (as long as the passing grades are positive...
integers), or by having decreased processing time by only fetching queries for a portion of the available data.

Alternatively, the process may be executed via the command line, by running an executable Python script, setting the parameters that the business user does not wish to use default values for. In addition to the above parameters, the Python script supports setting the Kitchen log level, which is also part of Spoon’s dialog box. As is the case with the Python script for the process described in Section 4.6, messages logged by PDI are also shown in the command line used to execute this script.

4.9 Summary

The first functional prototype, which was implemented as part of this project, was described throughout this chapter. We have discussed the goals of this DSS, the gathered business requirements, and the process used to gather them.

Afterwards, the architecture of the DSS was presented, consisting of three components: the set of data sources, the data warehouse, which was populated from that set of data sources via an ETL process, and an analysis component, which provides the desired data visualizations by performing SQL queries over the data warehouse. This data warehouse was created according to a fact constellation serving as our multidimensional model. This fact constellation contains five dimension tables, referring to the concepts of degree, curricular plan, course, student, and time, and four fact tables: one to store which students have completed a degree and when, another to store student activity details, another to link students to the degrees they were admitted to, and at what time that occurred, and a final fact table to store records on student grades.

The set of data sources used to populate the data warehouse consists of four Excel file types, Degrees, Admissions, Curricular plan, and Grades. The correspondences between the columns of these Excel file types and the elements of the multidimensional model of the data warehouse were presented after defining these file types.

Interaction with the data warehouse is ensured by two processes: one populates it from its data sources, and the other performs all queries supported by the DSS over the populated data warehouse. These processes were presented, and finally, implementation details on the system were explained, including the reasons that led to choosing the Pentaho ecosystem, with MySQL as the underlying DBMS.
5 Evaluation

The evaluation process of the implemented DSS was performed by validating the obtained results against previous reports and Excel files created by the coordinator of our degree of reference (LEIC-T), and by ensuring that all results could be obtained in a reasonable amount of time.

As such, multiple experimental validation rounds were made until the implemented DSS yielded the expected results, a process explained in Section 5.1, and execution times for the desired process with different quantities of input data were measured, with the results being presented in Section 5.2.

5.1 Experimental Validation

To ensure that the system worked as expected, the implemented DSS was loaded with all available information for our degree of reference, LEIC-T. Available information comprised student admissions, student grades, and curricular plans between the school years of 2007/2008 and 2017/2018. Then, all possible queries for this degree were executed, and results were compared with both past reports (an account on Fenix, IST’s campus management system, is required to access them), and private Excel result files created by the coordinator of this degree.

The validation process yielded the expected results, as, whenever comparable, the query results as obtained from the implemented prototype matched exactly or had minor deviations (of 0.1 to 0.2% of average ECTS credits obtained, for instance), which can be mostly blamed on rounding errors on the files used for validation.

However, the validation process was relatively complex, for two distinct reasons: first, there were differences in conditions between implemented queries and the analogous results in the degree coordinator’s files and reports, and second, several errors existed in such files and reports.

Regarding the first difficulty, all queries were specified as part of the process of gathering business requirements (explained in Section 4.1.1), and some criteria were defined differently, making the comparison with existing results impossible.

In particular, direct comparison of any course query (queries Q1 to Q6 in Table 4.1) is not possible, because the subset of considered students differs. In the files and reports created by the coordinator of
LEIC-T, grades of all students in the degree (and even some of other degrees, due to inconsistencies with files generated by Fenix), were considered. However, the implemented system only considers grades of students whose admission information is available – this means that students admitted before 2007, students admitted via special methods, and the rare cases of students not in LEIC-T but being considered by the degree coordinator’s files, were excluded by the DSS during the loading process. The most common difference was regarding the students admitted before 2007, as there were multiple cases of this happening for most courses in most semesters (even including semesters from 2017/2018, more than 10 years after admission), making any results regarding course incomparable. However, if the Excel files used for validation were altered to only consider the same subset of students, the query results matched.

Another example of incompatible conditions is the measurement of the number of students having completed a degree in each semester (Q9 in Table 4.1). As curricular plans changed in LEIC-T, the number of ECTS credits associated with the courses changed. As a result, even though students were expected to have obtained exactly 180 ECTS upon completion of the degree, this was often not the case.

For instance, the curricular plan valid during the 2012/2013 school year stated that the courses “Foundations of Programming” and “Discrete Mathematics” awarded 6.0 and 4.5 ECTS credits, respectively; however, in the next school year, these numbers were changed to 7.5 ECTS credits for each of the aforementioned courses, resulting in a combined difference of 4.5 ECTS between students passing those courses in 2012/2013 and 2013/2014. In the following years, the students passing those courses in 2012/2013 completed courses with reduced numbers of ECTS credits (to compensate the courses associated with more credits from 2013/2014 onward), resulting in their completion of the degree with less than 180 ECTS credits obtained in total. Therefore, the implemented DSS states that 0 students have completed the degree according to this rule in 2012/2013.

Another method to figure out whether a student has completed a degree is by gathering all courses (as they are all mandatory), and ensuring that the student has passed them all. However, as curricular plans changed in LEIC-T, the courses (and their names) were also changed; as a result, this method does not yield the desired results for some generations. An example of this can also be applied to the changes between 2012/2013 and 2013/2014; in 2013/2014, a new course, named “Introduction to Computer Architecture”, replaced two courses in 2012/2013: “Digital Systems” and “Computer Architecture”. As a result, students having completed both courses in 2012/2013 were also considered as having completed “Introduction to Computer Architecture” due to course equivalency, but the implemented DSS only handles student grades, and there were no grades for these students in this course, as they actually had completed other equivalent courses instead. Therefore, the implemented DSS states that no students have completed the degree according to this rule in 2012/2013. As this is also the case with
5.1. EXPERIMENTAL VALIDATION

the rule mentioned previously, this system states that no students admitted in 2012/2013 have completed the degree considering either criteria, which is not true, according to the files and reports used for validation.

Regarding the second difficulty, many errors were found in both the Excel files and the reports used for validation. As it was not possible to track the source of the inconsistencies between these results and the results provided by the implemented DSS, many hours of investigation were wasted investigating and attempting to debug the various components of the DSS unnecessarily. The coordinator of LEIC-T stated that errors in these files and reports were likely due to the complexity of the utilized formulas and difficulties in adapting them to changes in curricular plans, which is why they were investigated as well. Unfortunately, these concerns were indeed justified. Additionally, the usage of links to external files (in particular, the degree coordinator’s global output Excel file depended on all student grades for the 6 semesters following admission – each semester associated with a Grades file as introduced in Section 4.4) also added a degree of complexity and a source for eventual errors.

The errors in these files affected validation with all generational queries. For instance, the degree coordinator’s global output Excel file included a compilation of grades in a given semester for each student, and this compilation was transmitted via an Excel formula related to the linked Excel files. However, issues with this formula led to grades being associated with the wrong students, and even to some students have the impossible passing grade of "0". This led to many validation inconsistencies, such as active students being incorrectly considered as inactive (affecting validation for queries Q7, Q8, and Q17 of Table 4.1), students being associated with incorrect ECTS credit numbers (affecting validation for queries Q9, Q15, and Q16), and students being associated with incorrect grades (affecting validation for queries Q10 and Q11).

The validation process did not just consist of issues that could not be directly blamed on the DSS however. Two distinct sources of errors arose during the various iterations of this prototype: the first source of errors was the populator process (defined in Section 4.6), in which issues with validation of the input Excel files led to some students and grades not being considered, even though they should have been considered; second, some of SQL queries in the query executor process (defined in Section 4.7) were defined incorrectly in earlier stages of this prototype, leading to unexpected results.

Issues related to the populator process were easy to verify, as there was no need to rely on the files and reports used for validation. However, this was not the case with the other issues, as those files and reports were necessary and were incorrect in earlier stages; they were eventually fixed in cooperator with the coordinator of LEIC-T, which made the process of debugging the incorrectly defined SQL queries possible.
5.2 Performance

The acceptance of a system usually relies on performance, as it will not be used by anyone if it takes an unreasonable amount of time to execute tasks. As such, as part of the process of gathering business requirements, which is described in Section 4.1.1, two performance requirements were set:

- Maximum time elapsed to populate the data warehouse must not be higher than 30 minutes, with all 11 available generations of LEIC-T serving as input. Execution time should increase linearly with the number of generations of this degree;
- Maximum time elapsed to execute all supported queries (as per Table 4.1) must not be higher than 60 minutes, with all 11 available generations of LEIC-T serving as input. Execution time should also increase linearly with the number of generations.

Execution time was gathered by executing each process via Spoon, with the option to gather performance metrics (which can be seen on Figure 4.17) checked. After execution, a Gantt chart with the execution time of each component in milliseconds was displayed.

To verify how the execution time of the processes increased with an increased number of generations, a script was developed to iterate all data sources (excluding files of type Degrees) related to the 11 generations of LEIC-T for which we had data for, and multiply them by 2, 4, or 8 times. This was possible by changing the institutional IDs associated with the students, and by increasing the years associated with the existing curricular plans, student admissions, and student grades by 11, for each iteration.

By performing the process above, input files for 11, 22, 44, and 88 generations of LEIC-T were generated. For each of these four sets of input files, we have populated the data warehouse of the implemented DSS, using a newly created database, and executed all queries on this populated data warehouse, and measured the execution times of the processes and their main components. More specific details on the numbers related to the generation of these four sets of input files can be found in Table C.3.

Performance results of the process that populates the data warehouse are presented in Section 5.2.1, and results regarding the process that queries the data warehouse are shown in Section 5.2.2. In addition, attempts at optimizing performance are presented in Section 5.2.3.

5.2.1 Populating the Data Warehouse – Performance

The business requirement for this process was ensuring an execution time of 30 minutes at most, considering all 11 available generations of LEIC-T, with a near linear increase in execution time when the number of available generations is increased.
5.2. PERFORMANCE

The results of the time elapsed to load each of the input data sets (considering 11, 22, 44, and 88 generations of LEIC-T on a newly created database are displayed in Figure 5.1. The dark blue plot represents the sum of the execution times of each of the three components of this process: the verification stage (Section 4.6.1), the ETL subprocesses that directly handle the Excel input files (Sections 4.6.3 to 4.6.6), and the ETL subprocesses that populate the remaining fields and tables of the data warehouse (Section 4.6.7). Results with more detail can be seen in Table C.1.

![Figure 5.1: Execution time of the populator process and its components, by number of input generations of LEIC-T](image)

By analyzing the plot corresponding to execution time of the entire process, we can see that the part of the business requirement regarding loading time for 11 generations of LEIC-T was met, as the entire process took less than two minutes to complete. In addition, it can be also seen that this requirement would be met, even if the maximum number of 88 generated years were being considered with the same time limit, as loading time for 88 generations totalled 18.6 minutes, still well under the 30 specified minutes.

It can also be seen that the dark blue plot represents a near straight line, representing a near linear increase of execution time, as required. The 18.6 minutes required for loading 88 generations are satisfactory, as this represents an increase in execution time of 969% for an increase in number of generations of 700%, proving that the process will scale well over time. The increase in execution time when a high number of generations is inserted at once can be blamed mostly on the ETL subprocesses than populate additional fields, which is expected, as the most complex calculations are made at this stage, and these are the processes responsible for populating the tables that increase exponentially in size with the number of generations.
Further analysis of Figure 5.1 allows us to conclude that the execution time of the verification stage is not relevant, as the steps performed at this stage are simple and their complexity increases slowly— all that is done is creating the data warehouse from an SQL script, and traversing the provided input directory and its subdirectories to ensure that valid Excel input files exist. The highest recorded execution time for this stage was approximately 8 seconds, for 88 input generations.

The execution time for the ETL subprocesses directly handling Excel files increased as expected. For each input data set, the number of generations doubled, which meant that the number of input Excel files and input rows also approximately doubled. As such, an increase of execution time by around 100% was expected by generation. An analysis of Table C.1 allows us to conclude that the execution times increased by 49% when loading 22 generations, 167% (compared to the previous result) when loading 44 generations, and 54% when loading 88 generations, which proves that these subprocesses scaled well.

### 5.2.2 Querying the Data Warehouse – Performance

For the query executor process, the business requirement involved an execution time of 60 minutes at most, considering a data warehouse containing information for all 11 available generations of LEIC-T, with a near linear increase in execution time, for increases in the number of available generations.

A chart of the elapsed execution time for this process, split into three subcomponents, for each of the inserted data sets (11, 22, 44, and 88 generations) can be seen in Figure 5.2.

![Figure 5.2: Execution time of the query executor process and its components, by number of input generations of LEIC-T](image-url)
The three aforementioned subcomponents are the executor of course queries (Section 4.7.1), the executor of generational queries (Section 4.7.2), which excluded Q13 (in Table 4.1), and the excluded Q13. The dark blue plot represents the sum of the execution times for all three subcomponents, for each number of input generations of LEIC-T. Precise execution times can be found in Table C.2.

Two conclusions can be quickly drawn from analyzing this chart: first, execution time for 11 input generations was satisfactory, being well under the specified 60 minutes; second, the execution time increased exponentially, rather than linearly.

Regarding the first conclusion, the execution time for 11 input generations was approximately 175 seconds, less than 5% of the maximum allowed time. Execution times of earlier versions of the prototype were much higher or even infinite, because many optimizations had not been implemented yet. The implemented optimizations will be explained in Section 5.2.3.

However, the execution time did not scale well for subsequent executions with higher numbers of input generations. The plot representing the execution of generational queries excluded Q13 to allow easy identification of the culprit. Each execution of query Q13, considering 88 generations as input, lasted 13.1 seconds, which may not seem very high at first. Unfortunately, this query, like all generational queries of this type, was executed once for every possible combination of admission year and semester number since admission, and this number was 860, considering 88 input generations (as per Table C.3), leading to a total execution time of over three hours just for this query. This represents 91% of the execution time of the entire process. Attempts at optimizing it, such as insertion of indexes on fields that serve as join keys, and splitting the query into more smaller queries did not decrease the execution time significantly. Therefore, this is a crucial limitation that should be worked on by future prototypes.

To ensure that the supported query Q13 was indeed the only culprit, another chart, disregarding Q13 altogether, was created, and is presented in Figure 5.3. In this case, execution times increased linearly, and the plot lines are similar to the ones seen in Figure 5.1. The execution time of this process for 88 generations totalled 21 minutes approximately, which is satisfactory, given that this number of generations would only be reached in 2099.

By analyzing the two main components of this new process, we can conclude that course queries contribute much less to total execution time than generational queries. This is to be expected, as course queries are simpler (they involve joining less tables, and usually smaller tables), are executed less often (they run once for each course name, which is 35 times in total; generational queries run from 11 times to 860 times in total), and only six course queries exist, against thirteen generational queries.
5.2.3 Performance Optimization

In earlier stages of the prototype, the process used to populate the data warehouse that constitutes the central point of the implemented DSS ran indefinitely when higher numbers of generations were applied, to the point where MySQL would be unresponsive for several hours until manually killed. In later stages, the query executor process would be too slow and the maximum execution time of 60 minutes was not met. The results presented above were only possible due to three optimizations: revamping the PDI transformation that populates the student_degree table (presented in Section 4.6.7), applying indexes on fields commonly used to join tables, and revamping the PDI transformation responsible for retrieving all possible admission year/semester number combinations, to be provided as input to generational queries. Optimizations involving database tuning were avoided, as they would not be easily replicated in other environments.

When attempting to populate the data warehouse with 22 generations, the entire process ran indefinitely. The culprit was identifiable via Spoon, as a waiting clock was displayed over the “Set degree completion” step of Figure B.1, and MySQL would no longer respond after that step, in what is believed to be a memory issue. Initially, the process performed a large MySQL query, responsible for populating the entire student_degree table at once. An attempt to optimize this was to retrieve all admitted students first by querying the student_admission table, and then perform a simpler query for each admitted student. These queries used less memory, and the process would finally complete, even for the maximum number of 88 generations.
The second optimization was implementing indexes on fields commonly used as join keys. Implementing indexes meant that insertions and updates on the various tables of the data warehouse would be slower, but that retrieving results by performing SQL queries would be a significantly faster operation. Thanks to this optimization, all generational queries, except Q13 (in Table 4.1), scale linearly and have acceptable execution times.

The final optimization was revamping the method used to retrieve all possible combinations of admission years and semester numbers since admission. The output of this method is provided to some generational queries, as seen in Figure 4.16. The bottleneck in this method was a nested query including an aggregation function, used to retrieve the latest semester for which there is data for. This was calculated once for every possible combination, even though the result did not depend on said combination. With 860 possible combinations with default values for 88 input generations, this ended up taking minutes to execute. The solution to this issue was separating this nested query, performing this calculation only once, leading to a reasonable execution time of 1.4 seconds for the 88 generations, the maximum number tested for this prototype.

5.3 Summary

In this chapter, we have compared the implemented prototype to the business requirements that were gathered. The results were generally positive: all required queries were implemented and validated against the files and reports that the coordinator of our degree of reference (LEIC-T) possessed, the execution times of both processes when all of the 11 available generations used as input were satisfactory, each totalling less than 3 minutes, while the business requirements allowed up to 30 minutes of execution time for the process used to populate the data warehouse of this prototype, and 60 minutes for the process used to execute queries over the data warehouse, and execution times scaled linearly for the first of these processes.

However, this was not the case for the query executor process, which scaled exponentially with an increase in the number of input generations. Multiple optimizations were introduced in the meantime, but the only implemented optimization targeting this process, the introduction of indexes, failed to significantly decrease the execution time of query Q13 of Table 4.1. To ensure that Q13 was the only culprit in this process, it was excluded from a second run, and the results were as expected – the process finally scaled linearly, with acceptable execution times for any tested number of input generations.
Conclusion

This document presented the first functional prototype of a DSS applied to degree coordination. This system was created with the intention of automating the currently manual process of extraction of data related to students, courses, and student performance, manipulating them, and obtaining data visualizations, which can be used for an eventual creation of reports by degree coordinators.

For that purpose, an initial study on concepts related to DSSs, data warehousing and data analysis was performed, and some notions more closely related to the domain of degree coordination were defined. We have investigated a set of ecosystems that could be used to implement this system, and chose a free, open-source set of tools. Some existing applications of DSSs in the context of educations, starting with a more theoretical approach that proposed a general framework to build data warehouses for information systems related to higher education, and ending with a quick overview of an implementation of a data warehouse for university accreditation.

The conclusion of the investigation performed above led to the beginning of the implementation stage, starting with a design phase. The goals of the DSS to implement were defined, business requirements were gathered with the help of our primary end user (the coordinator of LEIC-T, the degree we have used as reference throughout this project), and the architecture of the system to implement was defined as a result. In addition, a multidimensional model was defined for the data warehouse, and it has been changed throughout the various iterations of the implemented system, to work around limitations that were eventually found, and to facilitate eventual queries performed over the data warehouse. This data warehouse was populated from a set of data sources also defined at this design stage, with the schema mappings between data sources and multidimensional model fields being presented as well.

Interaction with the data warehouse was ensured by two processes: a populator process, which is responsible for initial population of the data warehouse, and subsequent updates including new input data, and a query executor process, responsible for executing the set of supported queries over the data warehouse, with the results being written to Excel files.

This prototype was evaluated via two methods: validation against existing results, and performance measurement, taking the gathered business requirements into account. The results were generally positive, as all results output by the prototype were successfully validated whenever comparison was possible, and the execution times met expectations in most cases. The exception to this was the query
executor process, which did not handle large amounts of data well, as execution times increased ex-
ponentially. A specific query was found as the culprit, but the attempted optimization methods did not
significantly decrease its execution time.

As this project represents a first functional prototype, there is still a lot of work to be done. Limi-
tations of the prototype and features that can be applied on top of it will be discussed in Section 6.1.
Nevertheless, the ecosystem selected for this prototype allows any future developer to reproduce it in
their favorite development environment, enabling reusability of the currently implemented DSS.

6.1 Future Work

Many details and features still remain unexplored in this application. Some assumptions that facilitate
the execution of some of the supported queries are made. These assumptions are reasonable in the
context of LEIC-T and can be extrapolated to other degrees, but not to all. The most limiting assumption
is that there is no distinction between mandatory and elective courses. Two methods are used to verify
if a student has completed a degree: obtention of the minimum requirement of ECTS credits, and
conclusion of all courses in the curricular plan valid at the time. Neither method makes a distinction
between mandatory and elective courses, which makes some parts of this system not applicable to
many degrees.

Another limitation is also related to degree completion, and is related to the latter of the aforemen-
tioned methods. Curricular plans often change in degrees, and with them, courses taught in the context
of a degree change, and course equivalency is often offered in these cases. This prototype does not
handle course equivalency, as information is directly gathered from student grades. Therefore, changes
in the multidimensional model or a different method to compute degree completion by a student could
be explored.

Other than the above, some features that were not explored at all in the context of this prototype
could be eventually explored. One of those features is the introduction of an OLAP server, allowing
dimensional analysis of data, via a drag and drop interface, or via MDX queries. Another unexplored
feature, also related to the field of data analysis, is the implementation of dashboards, with more filtering
options, allowing the visualization of more tailored KPIs in a simple graphical interface. A third unex-
plored feature is automation of reports, ensuring that the process between inserting data to the DSS
and creating a report does not require any human interaction. For increased flexibility, options between
preformatted reports, parameter-driven predefined reports, and manual development of reports could
be provided. All three of the suggested features can be implemented by using tools available for free as
part of the selected ecosystem.


Kurniawan, Y. and E. Halim (2013). Use Data Warehouse and Data Mining to Predict Student Academic Performance in Schools: A Case Study (Perspective Application and Benefits). In *IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*.


Appendices
A Full graphical representation of the multidimensional model

Figure A.1: Multidimensional model represented in the form of a diagram, including field data types
Figure A.2: Multidimensional model as a diagram, including example field values
PDI representations of ETL subprocesses

Figure B.1: Main PDI job used to build the data warehouse
Figure B.2: PDI transformation that handles Excel files containing data on degrees
Figure B.3: PDI transformation that handles Excel files containing data on admissions (split vertically)
Figure B.4: PDI transformation that handles Excel files containing data on curricular plans (split vertically)
APPENDIX B. PDI REPRESENTATIONS OF ETL SUBPROCESSES

Figure B.5: PDI transformation that handles Excel files containing data on grades

Figure B.6: PDI transformation that populates the num_enrollments attribute
Figure B.7: PDI transformation that populates the enrolled\_semester1 attribute

Figure B.8: PDI transformation that populates the f\_student\_degree table
APPENDIX B. PDI REPRESENTATIONS OF ETL SUBPROCESSES

Figure B.9: PDI transformation that populates the `f_student_activity` table

Figure B.10: PDI transformation that populates the `first_passed` attribute

Figure B.11: PDI transformation that populates the `is_new` attribute
## Performance results

Table C.1: Number of seconds required for the execution of the components of the populator process, by number of years of LEIC-T input data

<table>
<thead>
<tr>
<th>Process type / Number of years</th>
<th>11 years</th>
<th>22 years</th>
<th>44 years</th>
<th>88 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification stage</td>
<td>5.993</td>
<td>5.958</td>
<td>6.782</td>
<td>8.027</td>
</tr>
<tr>
<td>ETL stage - Excel subprocesses</td>
<td>18.475</td>
<td>27.455</td>
<td>73.307</td>
<td>112.548</td>
</tr>
<tr>
<td>ETL stage - remaining subprocesses</td>
<td>80.144</td>
<td>154.207</td>
<td>438.577</td>
<td>998.176</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>104.612</strong></td>
<td><strong>187.620</strong></td>
<td><strong>518.666</strong></td>
<td><strong>1118.751</strong></td>
</tr>
</tbody>
</table>

Table C.2: Number of seconds required for the execution of the components of the query executor process, by number of years of LEIC-T input data

<table>
<thead>
<tr>
<th>Process type / Number of years</th>
<th>11 years</th>
<th>22 years</th>
<th>44 years</th>
<th>88 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch course names</td>
<td>0.020</td>
<td>0.021</td>
<td>0.019</td>
<td>0.016</td>
</tr>
<tr>
<td>Q1</td>
<td>4.451</td>
<td>3.555</td>
<td>7.048</td>
<td>12.923</td>
</tr>
<tr>
<td>Q2</td>
<td>4.315</td>
<td>4.263</td>
<td>9.211</td>
<td>15.634</td>
</tr>
<tr>
<td>Q3</td>
<td>4.654</td>
<td>4.857</td>
<td>10.246</td>
<td>17.102</td>
</tr>
<tr>
<td>Q4</td>
<td>4.952</td>
<td>5.196</td>
<td>11.077</td>
<td>19.318</td>
</tr>
<tr>
<td>Q5</td>
<td>5.624</td>
<td>5.204</td>
<td>12.437</td>
<td>23.006</td>
</tr>
<tr>
<td>Q6</td>
<td>6.655</td>
<td>5.936</td>
<td>11.235</td>
<td>20.599</td>
</tr>
<tr>
<td><strong>Course queries – subtotal</strong></td>
<td><strong>175.283</strong></td>
<td><strong>392.650</strong></td>
<td><strong>438.577</strong></td>
<td><strong>998.176</strong></td>
</tr>
<tr>
<td>Fetch admission years</td>
<td>0.016</td>
<td>0.032</td>
<td>0.025</td>
<td>0.014</td>
</tr>
<tr>
<td>Q7</td>
<td>0.816</td>
<td>1.136</td>
<td>4.287</td>
<td>11.045</td>
</tr>
<tr>
<td>Q8</td>
<td>0.763</td>
<td>1.499</td>
<td>4.806</td>
<td>16.020</td>
</tr>
<tr>
<td>Q9</td>
<td>0.529</td>
<td>0.867</td>
<td>2.504</td>
<td>8.219</td>
</tr>
<tr>
<td>Q12</td>
<td>0.805</td>
<td>1.474</td>
<td>4.520</td>
<td>15.609</td>
</tr>
<tr>
<td><strong>By admission year – subtotal</strong></td>
<td><strong>2.929</strong></td>
<td><strong>5.008</strong></td>
<td><strong>16.142</strong></td>
<td><strong>50.907</strong></td>
</tr>
<tr>
<td>Fetch years / semester numbers</td>
<td>0.345</td>
<td>0.368</td>
<td>0.463</td>
<td>1.375</td>
</tr>
<tr>
<td>Q10</td>
<td>24.017</td>
<td>53.861</td>
<td>106.924</td>
<td>220.153</td>
</tr>
<tr>
<td>Q11</td>
<td>23.607</td>
<td>52.134</td>
<td>108.341</td>
<td>223.061</td>
</tr>
<tr>
<td>Q13</td>
<td>25.003</td>
<td>106.239</td>
<td>1017.413</td>
<td>11286.970</td>
</tr>
<tr>
<td>Q14</td>
<td>4.839</td>
<td>8.937</td>
<td>19.642</td>
<td>42.798</td>
</tr>
<tr>
<td>Q15</td>
<td>25.572</td>
<td>54.954</td>
<td>117.314</td>
<td>237.724</td>
</tr>
<tr>
<td>Q16</td>
<td>25.373</td>
<td>55.815</td>
<td>119.801</td>
<td>244.141</td>
</tr>
<tr>
<td>Q17</td>
<td>3.963</td>
<td>8.728</td>
<td>20.405</td>
<td>51.900</td>
</tr>
<tr>
<td>Q18</td>
<td>4.150</td>
<td>8.796</td>
<td>19.541</td>
<td>43.988</td>
</tr>
<tr>
<td>Q19</td>
<td>4.814</td>
<td>8.778</td>
<td>20.417</td>
<td>44.099</td>
</tr>
<tr>
<td><strong>By year / semester number – subtotal</strong></td>
<td><strong>141.683</strong></td>
<td><strong>358.610</strong></td>
<td><strong>1550.261</strong></td>
<td><strong>12396.189</strong></td>
</tr>
<tr>
<td>Generational queries – subtotal</td>
<td><strong>144.612</strong></td>
<td><strong>363.618</strong></td>
<td><strong>1566.403</strong></td>
<td><strong>12447.096</strong></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>175.283</strong></td>
<td><strong>392.650</strong></td>
<td><strong>1627.676</strong></td>
<td><strong>12555.694</strong></td>
</tr>
</tbody>
</table>
Table C.3: Statistics related to the generated sets of input files by number of years of LEIC-T input data

<table>
<thead>
<tr>
<th>Measure / Number of years</th>
<th>11 years</th>
<th>22 years</th>
<th>44 years</th>
<th>88 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of admitted students</td>
<td>1019</td>
<td>2038</td>
<td>4076</td>
<td>8152</td>
</tr>
<tr>
<td>Number of course names</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Number of curricular plans</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48</td>
</tr>
<tr>
<td>Number of combinations of year/semester number for generational queries (considering that max. semesters is set to 10)</td>
<td>90</td>
<td>200</td>
<td>420</td>
<td>860</td>
</tr>
<tr>
<td>Number of input files</td>
<td>40</td>
<td>79</td>
<td>157</td>
<td>313</td>
</tr>
<tr>
<td>Number of rows in data warehouse</td>
<td>46 199</td>
<td>110 414</td>
<td>296 916</td>
<td>902 264</td>
</tr>
</tbody>
</table>
Installation

Installation instructions are provided for Ubuntu and Mac OS.

Java

Pentaho Data Integration (PDI) requires Java. A 64-bit version of Java 8 is preferred, and both the Java Development Kit (JDK) and the Java Runtime Environment (JRE) will work.

On Ubuntu, OpenJDK can be installed with:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

You will need to set the `PENTAHO_JAVA_HOME` environment variable afterwards, so that PDI knows where to find Java. The value should correspond to the absolute path of the java executable file, minus the final "/bin/java" portion. So, if OpenJDK was installed, the full absolute path would be "/usr/bin/java", and the variable’s value would be "/usr". Therefore, using this example, `PENTAHO_JAVA_HOME` could be permanently set with the following commands:

```
export PENTAHO_JAVA_HOME=/usr
source ~/.bashrc
```

On Mac OS, the following commands can be used to install Java:

```
brew update
brew tap caskroom/versions
brew cask install java8
```

If Homebrew is not installed, install it before trying to install Java on Mac OS:

```
/usr/bin/ruby -e "/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Ensure that Java 8 is the default version of Java by running

```
java -version
```
MySQL

On Ubuntu, install MySQL with:

```
sudo apt-get install mysql-server
```

The root user is created during the install process and the MySQL service is automatically started.

On Mac OS, install MySQL and start the service with:

```
brew install mysql
brew tap homebrew/services
brew services start mysql
```

Also on Mac OS, set the password for the root user after installing MySQL. Assuming that the password to set is `rootroot`:

```
mysqladmin -u root password 'rootroot'
```

Create the `deg_coordination` database

Enter the MySQL monitor as root:

```
mysql -u root -p
```

After entering the MySQL console, create the database:

```
CREATE DATABASE deg_coordination;
exit;
```

Create a MySQL user (optional)

Enter the MySQL monitor as root, and then create a user to manage the database created above:

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON deg_coordination.* TO 'user'@'localhost';
exit;
```

Pentaho Data Integration

Download the latest version of PDI: https://sourceforge.net/projects/pentaho/files/Pentaho%208.1/client-tools/pdi-ce-8.1.0.0-365.zip/download

After downloading, unzip the file to an appropriate location. This is automatically done by default if the download was performed via Safari.

Make sure all dependencies are installed by running PDI:

```
./data-integration/spoon.sh
```
**Visualize Excel files**

On Ubuntu, LibreOffice is a default application in most distributions. If this is not the case with yours, it is available in the default application packager; thus, it can be installed with:

```bash
sudo apt-get install libreoffice
```

On Mac OS, Microsoft Excel for Mac can be used, or, for a free alternative, LibreOffice is also available: https://www.libreoffice.org/get-help/install-howto/os-x/

**Python**

On Ubuntu, install Python 3 and `pip`\(^1\) with:

```bash
sudo apt-get install python3 python3-pip
```

On Mac OS, the command to run is:

```
brew install python
```

`openpyxl`\(^2\) is required to parse the Excel files. It can be installed with `pip` for Python 3:

```bash
pip3 install openpyxl
```

**Git and project software**

Git is pre-installed on Mac OS. On Ubuntu, install it with:

```bash
sudo apt-get install git
```

Project software can be retrieved by running the following command after navigating to the desired parent directory:

```bash
git clone https://github.com/rubenanaganu/degree_coordination_dss
```

Afterwards, open the settings Python file and change the `DEFAULT_KITCHEN_PATH` variable to the absolute path of the `kitchen.sh` file, which is inside the `data-integration` folder where PDI is installed. You can edit this file with `nano`\(^3\) (pre-installed with both Ubuntu and Mac OS), for instance:

```
nano degree_coordination_dss/implementation/settings.py
```

The default value of the `DEFAULT_KITCHEN_PATH` variable is "/home/current_user/Pentaho/data-integration/kitchen.sh". You can exit `nano` by pressing Ctrl + X on Ubuntu, or Cmd + X on Mac OS.

---

1 [https://pypi.org/project/pip/](https://pypi.org/project/pip/)
3 [https://www.nano-editor.org/](https://www.nano-editor.org/)
File structure

Input Excel files

The Excel files used for evaluation of this prototype are available in the /input/ directory.

Four different Excel file types exist, and they must respect a regular expression in order to be considered, as seen in the table below:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>File name regex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees</td>
<td>Degrees to consider</td>
<td>degrees.*?.</td>
</tr>
<tr>
<td>Admissions</td>
<td>Admission information of students</td>
<td>admissions([-])+.(19[789][0-9][0-9]—20[0-9][0-9]).*?.</td>
</tr>
<tr>
<td>Curricular plan</td>
<td>Curricular plans to consider</td>
<td>plan([-])+.(19[789][0-9][0-9]—20[0-9][0-9]).[(12)].*?.</td>
</tr>
<tr>
<td>Grades</td>
<td>Information on grades</td>
<td>grades.(19[789][0-9][0-9]—20[0-9][0-9]).[(12)].*?.</td>
</tr>
</tbody>
</table>

PDI files

The actual implementation of both implemented processes – the populator process and the query executor process – is contained in PDI files. These PDI files are split into two logical types: transformations (.ktr extension) and jobs (.kjb extension). Steps in transformations are executed in parallel whenever possible, and intend to transform data at a lower level. Jobs control flows at a higher level, with sequential steps, and they may call PDI transformations in the process.

Populator process

PDI files regarding the populator process, which is responsible for taking input files and using them to populate the data warehouse, are stored in the /implementation/building/ directory. The top-level file responsible for running the populator process is main.kjb, which runs the PDI transformations stored in the same folder as part of the process.

Query executor process

PDI files for the query executor process, responsible for running the implemented queries over the populated data warehouse and returning Excel files as output, are stored in /implementation/fetching/ and its subdirectories. The top-level file responsible for running the query executor process is named queries.kjb. The query executor process handles two types of queries by executing two PDI jobs, one handling each type of query: course/outer.kjb (course queries) and generation/outer.kjb (generational queries). More query types can be appended to the top-level job in the future.

Course queries are executed once for every course name ever defined in a curricular plan. This is ensured by a transformation initially called by the outer job which fetches all courses via an SQL query,
and uses a Copy rows to result\textsuperscript{4} step to allow the outer job to retrieve such courses. For each course, the PDI job defined in course/inner.kjb is called; this job contains all supported course queries, each associated to a separate transformation.

Generational queries have two different levels of granularity: they may be executed once for each admission year (level 1), or once for each possible combination of admission year and semester number since admission (level 2). These levels of granularity are treated differently: level 1 queries are managed by a subjob defined in generation/by_admission_year_outer.kjb, and level 2 queries are managed by generation/by_admission_year_and_semester_number_outer.kjb.

Each of the aforementioned subjobs follows a structure similar to the populator process’ top-level job: they contain a transformation that obtains all possible input values (every admission year, or every admission year and semester number combination), and then call an inner job for each input value, in which all supported queries of the respective level of granularity are defined.

**SQL scripts**

Two SQL scripts are available in the /implementation/sql/ directory: deg_coordination_procedure.sql contains a stored procedure to populate the time dimension with time IDs for all semesters between the years 1970 and 2099, and deg_coordination_tables.sql contains the definition of the multidimensional model on which the data warehouse relies on. These SQL scripts are used by the populator process if such process is running for the first time since the creation of the deg_coordination database.

**Python scripts**

Three Python scripts are available in the /implementation/ directory. The two most relevant ones are build.py and fetch.py, which are used to execute the populator process and the query executor process, respectively. These scripts were created to avoid making the usage of the graphical user interface of PDI a necessity. In addition, if the input Excel files regarding grades contain invalid rows at the beginning that should be deleted, build.py will delete such lines without requiring user intervention; this is not the case with Spoon (the graphical user interface), and therefore Spoon should not be used if such invalid rows may exist.

build.py and fetch.py are able to avoid usage of Spoon by using Kitchen. Kitchen is the command line equivalent of the “Run” functionality of Spoon, and this is called by each Python script, after the input arguments are parsed and validated.

\textsuperscript{4}https://wiki.pentaho.com/display/EAI/Copy+rows+to+result
The remaining Python script is `data_generator.py`, which was used to generate data for evaluation purposes. Based on an input set of Excel files of sequential years, this script will multiply them by any specified number. This new set of input files can then be used to populate the data warehouse after resetting the database.

Information on the supported arguments in each Python script can be seen by running the respective script with the "-h" option. For instance, the following occurs when running `build.py` with such option:

```bash
$ python3 implementation/data_generator.py -h
usage: data_generator.py [-h] -m MULTIPLIER -i INPUT -o OUTPUT

optional arguments:
  -h, --help             show this help message and exit
  -m MULTIPLIER, --multiplier MULTIPLIER
                         Multiplier over the original data
  -i INPUT, --input INPUT
                         Absolute path to the input dir
  -o OUTPUT, --output OUTPUT
                         Absolute path to the output dir
```

`validators.py` is used for validation of parameters in the aforementioned scripts, in order to isolate this responsibility away from the scripts themselves. `settings.py` defines a set of global variables that serve as support for the Python scripts.

### Running the populator process

The populator process can be executed via Spoon by opening the `/implementation/building/outer.kjb` file, pressing the "Run" button (this will open a dialog box), setting the desired parameters, and pressing the "Run" button in the dialog box. The following parameters are accepted:

<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DB_HOST</code></td>
<td>localhost</td>
<td>MySQL host name</td>
</tr>
<tr>
<td><code>DB_NAME</code></td>
<td>deg_coordination</td>
<td>MySQL database</td>
</tr>
<tr>
<td><code>DB_PASSWORD</code></td>
<td>root</td>
<td>MySQL password</td>
</tr>
<tr>
<td><code>DB_PORT</code></td>
<td>3306</td>
<td>MySQL port number</td>
</tr>
<tr>
<td><code>DB_USER</code></td>
<td>root</td>
<td>MySQL username</td>
</tr>
<tr>
<td><code>DIR_PATH</code></td>
<td></td>
<td>Path to the directory containing the input XLSX files</td>
</tr>
</tbody>
</table>

Alternatively, the process can be executed with Python. This is the only method to run the populator process if any input Excel file of type "grades" contains invalid rows at the beginning, that are not related to the actual data. Information on the supported arguments of `build.py` can always be checked with the "-h" option:

```bash
$ python3 implementation/build.py -h
```
usage: build.py [-h] [--kitchen KITCHEN]
               [--level {Basic,Nothing,Error,Minimal,Detailed,Debug,Rowlevel}]
               --csv_dir CSV_DIR --user USER [--password [PASSWORD]]
               [--db DB] [--host HOST] [--port PORT]

optional arguments:
-h, --help            show this help message and exit
--kitchen KITCHEN     Path to kitchen.sh
--level {Basic,Nothing,Error,Minimal,Detailed,Debug,Rowlevel}
                       Pentaho log level for the job to run
--csv_dir CSV_DIR     Path to the directory containing the input Excel files
--user USER           MySQL username
--password [PASSWORD] Use MySQL password. Requested at runtime if not provided
--db DB               MySQL database name
--host HOST           MySQL hostname
--port PORT           MySQL port

As an example, we will assume that we want to load the files stored in the directory
"/home/user/Downloads/input/" (and its subdirectories), and that we want to increase the log level of
PDI from "Basic" to "Debug":

    python3 implementation/build.py
      --csv_dir /home/user/Downloads/input
      --level Debug

Running the query executor process

The query executor process can be executed by opening the /implementation/fetching/queries.kjb in
Spoon and running it. The following table lists the accepted parameters:

As is the case with the populator process, the query executor process can also be executed with Python.
Information on the usage of fetch.py is as follows:

    $ python3 implementation/fetch.py -h
usage: fetch.py [-h] [--kitchen KITCHEN]
               [--level {Basic,Nothing,Error,Minimal,Detailed,Debug,Rowlevel}]
               --csv_dir CSV_DIR [--degree DEGREE]
               [--max_admission_year MAX_ADMISSION_YEAR]
               [--min_admission_year MIN_ADMISSION_YEAR]
               [--max_year MAX_YEAR] [--max_grade MAX_GRADE]
               [--min_grade MIN_GRADE] [--num_semesters NUM_SEMESTERS] --user
               USER [--password [PASSWORD]] [--db DB] [--host HOST]
               [--port PORT]

optional arguments:
-h, --help            show this help message and exit
--kitchen KITCHEN     Path to kitchen.sh
<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_HOST</td>
<td>localhost</td>
<td>MySQL host name</td>
</tr>
<tr>
<td>DB_NAME</td>
<td>deg_coordination</td>
<td>MySQL database</td>
</tr>
<tr>
<td>DB_PASSWORD</td>
<td>rootroot</td>
<td>MySQL password</td>
</tr>
<tr>
<td>DB_PORT</td>
<td>3306</td>
<td>MySQL port number</td>
</tr>
<tr>
<td>DB_USER</td>
<td>root</td>
<td>MySQL username</td>
</tr>
<tr>
<td>DEGREE</td>
<td>LEIC-T</td>
<td>Short name of the degree to consider</td>
</tr>
<tr>
<td>MAX_ADMISSION_YEAR</td>
<td>2099</td>
<td>Maximum admission year to consider when fetching admission years</td>
</tr>
<tr>
<td>MAX_GRADE</td>
<td>20</td>
<td>Maximum possible grade</td>
</tr>
<tr>
<td>MAX_YEAR</td>
<td>2099</td>
<td>Ignore years higher than the provided (i.e. admission year + number of semesters &lt;= provided school year) when fetching admission year / semester number pairs</td>
</tr>
<tr>
<td>MIN_ADMISSION_YEAR</td>
<td>1970</td>
<td>Minimum admission year to consider when fetching admission years</td>
</tr>
<tr>
<td>MIN_GRADE</td>
<td>10</td>
<td>Minimum possible grade (to pass a course)</td>
</tr>
<tr>
<td>NUM_SEMESTERS</td>
<td>10</td>
<td>Number of semesters since admission to consider</td>
</tr>
<tr>
<td>PATH</td>
<td></td>
<td>Path to the directory containing the output XLSX files</td>
</tr>
</tbody>
</table>

For this example, we will store the output Excel files in a custom directory, analyzing the LEIC-T degree, and using a different MySQL user:

```bash
python3 implementation/build.py
  --csv_dir /home/user/Downloads/output
```
--degree LEIC-T
--user coordinator
--password very_secure