A Decision Support System for Degree Coordination

Rúben Anágua
rubenanagua@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

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.

Keywords: Decision Support System, Data Warehouse, Higher Education, Academic Performance, Degree Coordination

1. Introduction

The use of Decision Support Systems (DSSs) is becoming widespread in areas connected 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 [4].

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, such as the percentage of courses they have passed.

The process of collection and extraction of data is mostly manual, typically resorting to programs such as Microsoft Excel to organize and transform data. The reports for most degrees are built manually, and rely on several worksheets, to store data regarding grades, admission lists, and curricular plans, rather than having such data stored in a single data repository. 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 were translated into actual development. Moreover, some are specific to certain universities or are not directly related to the scope of our project – an academic degree and its students. Thus, there is an open problem in this area, which our DSS aims to solve.

1.1. Objectives

This project aimed to create a DSS to automate the process of extracting data, manipulating it, and obtaining of data visualizations in the form of Excel files, 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 query results regarding indicators related to the evolution of academic performance in a given degree and its courses.

1.2. Document Outline

This paper is organized in five more sections. Section 2 presents the considered tools and the characteristics found more relevant in them. Section 3 presents two case studies of DSSs in education, with a final discussion of common shortcomings. Implementation of the prototype is discussed in Section 4, and its evaluation is made in Section 5. Finally, we conclude and present work that can be performed in the future.

2. 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 (Extract, Transform, Load) process, and the presentation of data stored in the data warehouse. Features slated for future prototypes, such as Online Analytical Processing (OLAP) analysis, must be compatible with the work already performed as part of this initial prototype.

Considered factors for the selection of the ecosystem include the availability or a free edition, flexibility, learning curve, whether the code is open-source or not, and cross-system compatibility.

Three BI stacks were considered: Pentaho, SQL Server Business Intelligence, and Qlik.

2.1. Pentaho

Pentaho's BI suite[1] consists in a wide set of open-source pillars that work independently from each other, but can be combined to form a full ecosystem. Pentaho's free community edition includes a full BI suite of core tools, along with the ability to install several community-driven server plugins on top of the Pentaho platform. These tools were all created with Java, making the suite compatible with all popular operating systems. A popular example is Ctools[2], a set of open-source tools mitigating some limitations of the free edition, such as the creation of dashboards.

The tool to use for the ETL process is Pentaho Data Integration (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 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.

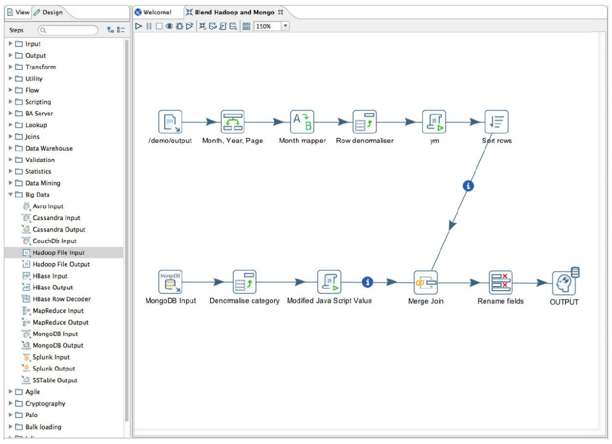


Figure 1: PDI's drag-and-drop interface

OLAP cubes can be created with Pentaho Schema Workbench[3]. For reporting purposes, the subset of dashboard-related tools in Ctools or Pentaho Reporting[4] can be used.

2.2. SQL Server Business Intelligence

Microsoft SQL Server[5] is a popular, closed-source Relational Database Management System(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[6] 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 compatible with Linux.

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). This tool supports relational

[1]http://www.pentaho.com/
[2]https://help.pentaho.com/Documentation/6.1/0R0/CTools/CTools_Overview
[3]https://mondrian.pentaho.com/documentation/workbench.php
[4]https://community.hds.com/docs/DOC-1009856-pentaho-reporting
[5]https://www.microsoft.com/en-us/sql-server/sql-server-2017
[6]https://imagine.microsoft.com/

OLAP, multidimensional OLAP, and hybrid OLAP, unlike Mondrian, Pentaho's analysis engine, which is based on relational OLAP. Reports can be created using SQL Server Reporting Services (SSRS) or Power BI[7]. However, all of these applications are Windows-only (excluding Power BI, which also has a cloud-based version).

2.3. Qlix

Qlik Sense and QlikView are the main products of Qlik[8] 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[9]. 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, and persistent data can be stored using proprietary 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 Multidimensional Expressions (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.

2.4. Selecting the Right Ecosystem

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 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.

[7]https://powerbi.microsoft.com/

[8]https://www.qlik.com/

[9]https://www.qlik.com/us/company/academic-program

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 ecosystem 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. The selected RDBMS was MySQL.

3. Related Work

This section aims to analyze the current state of DSSs in education. We will describe two different implementations of such systems in this context, and discuss their shortcomings.

3.1. Case Study: The Croatian Higher Education Information System (ISVU)

The ISVU[10] 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.* [1] 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.

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

[10]http://www.isvu.hr/javno/hr/index.shtml

2000 Server as the operating system. This multidimensional database is the source for the OLAP server and its OLAP cubes, which interacts with the user via a web-based application.

In 2009, Mekterovic *et al.* [3] presented some improvements to the data warehouse, considering changes in requirements due to the introduction of the Bologna process[11], 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 predefined queries, detailed *ad hoc* queries (where the user could define constraints in a query via text boxes and combo boxes, and this was then transformed to a parameterized SQL query), and summarized *ad hoc* queries (where users could perform queries with some OLAP operations over a drag-and-drop interface).

3.2. Case Study: Data Warehousing for University Accreditation

Sinaga *et al.* [5] 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 Grade Point 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 via PDI, with the extraction of data from multiple sources (relational databases, Excel files, and Microsoft Word files) being considered as the biggest challenge.

3.3. Discussion

The introduced examples, and the remaining examples in the related work related to this field, have many shortcomings. he 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.

4. Implementation

In this section, 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.

4.1. Gathering Business Requirements

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 [2].

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 (Bachelor of Science in Computer Science and Engineering at IST-Taguspark), 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,

[11] http://ec.europa.eu/education/policy/higher-education/bologna-process_en

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 of the conclusion of this four stage process, it was specified that the DSS would support four input Excel file types, process them and populate a data warehouse based a multidimensional model designed as a result, and support a set of 19 predefined queries, with the results being written to a set of Excel files with the specified levels of granularity.

Additionally, 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.

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 2.

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.

The analysis component provides a set of Excel files as output, obtained by performing SQL queries over the data warehouse. 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. This model is presented graphically in Figure 3. Primary keys are represented in bold and foreign keys are italicized. Associations between tables and their respective cardinalities are also represented.

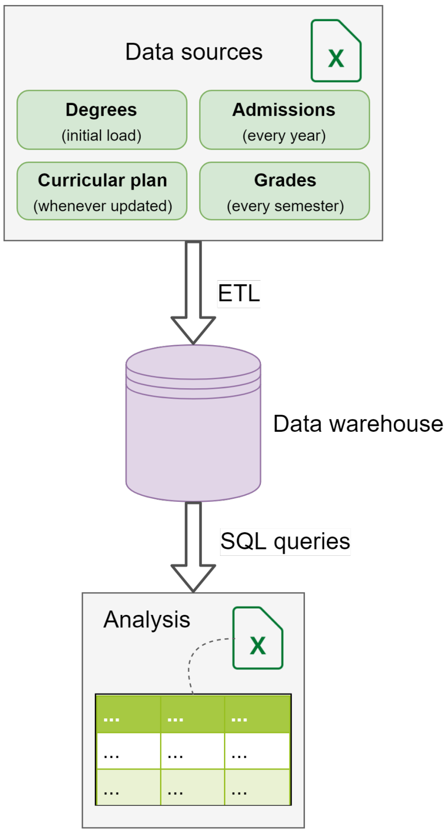


Figure 2: Architecture of the developed DSS

Information in this data warehouse is mainly structured in five different dimensions, which serve as five different perspectives of interest to our use case.

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[12] credits required for completion of the degree, and the number of years for which the degree is designed for (for example, LEIC-T is designed to have a duration of 3 years[13]).

The student dimension is represented by the *d_student* dimension table. Each student is identified by its institutional ID and, optionally, by the stage of admission to the degree.

The time dimension is represented by the *d_time* dimension table. The chosen time granularity is the semester, as a semester is the minimum amount of

[12] https://ec.europa.eu/education/resources/european-credit-transfer-accumulation-system_en

[13] https://fenix.tecnico.ulisboa.pt/cursos/leic-t/curriculo

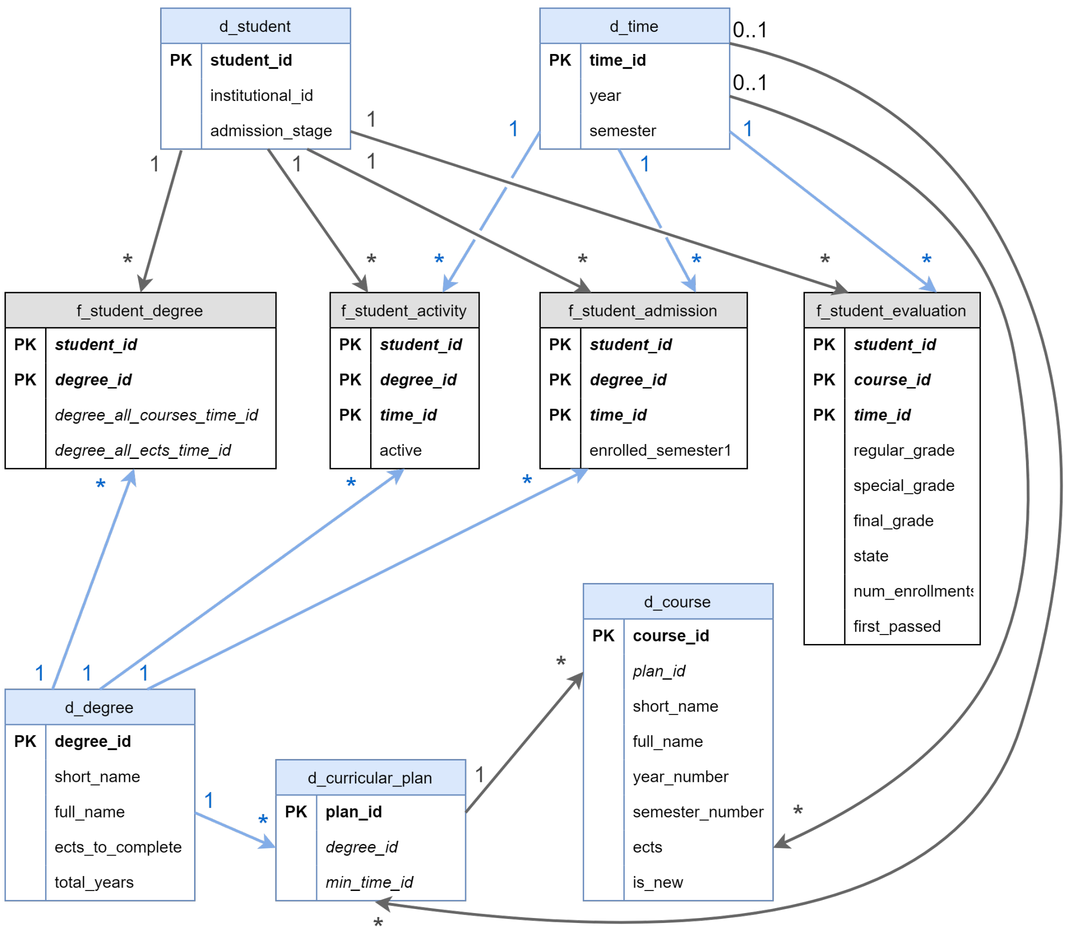


Figure 3: Multidimensional model represented in the form of a diagram

time needed by a student to finish a course and to have an associated grade.

Curricular plans are not necessarily static over time; therefore, they are also stored on the database, thanks to the *d_curricular_plan* dimension table. Each curricular plan is associated with a degree and a minimum semester of validity, via surrogate keys identifying the degree and time dimensions, respectively.

The *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 *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.

Four fact tables are part of this data warehouse, and they contain keys to the aforementioned dimensions, 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 *f_student_activity*. It contains a Boolean field named *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 (students not evaluated in any course).

The *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 *enrolled_semester1* is available to filter out students that did not enroll in all available courses of their first semester.

In the *f_student_degree* fact table, each student and degree is associated with two fields: *degree_all_courses_time_id* and *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_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: *state* determines whether a student has passed a course, failed it, or has not been evaluated, *num_enrollments* is the number of times that a student had enrolled in the associated course until the associated semester, and *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. Populating the Data Warehouse

he process of populating the data warehouse from a set of data sources is composed by two stages: a verification stage and an ETL stage, as Figure 4 shows.

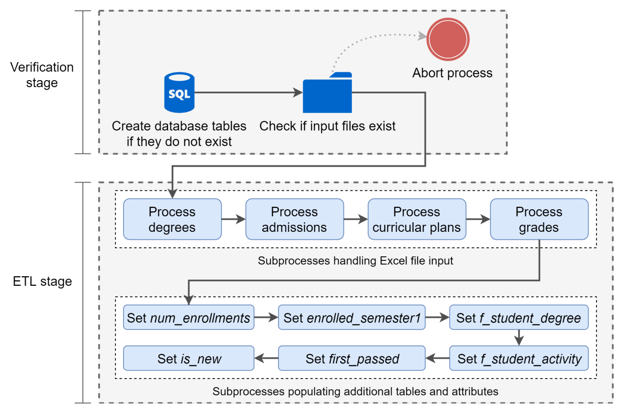


Figure 4: Diagram describing the populator process

The *verification stage* involves two steps: first, creating the database tables if they do not exist yet, which includes calling a stored procedure to populate the time dimension, and second, checking if any valid input files exist.

After this first stage is successful, the more complex *ETL stage* begins. 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. This second group of ETL subprocesses was created with the main objective of facilitating common queries that may be performed over this database. For instance, we can verify if a student is active in a given degree for a given semester by checking if the respective entry exists in *f_student_activity* and if *active* is set to true.

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.

4.5. Querying the Data Warehouse

The process of gathering output data from the warehouse can be split into two: one manages course queries over the data warehouse, and another manages queries regarding generations, as seen in Figure 5. Each subprocess gathers all possible results for all queries in an individual degree, as provided to the DSS when running this process.

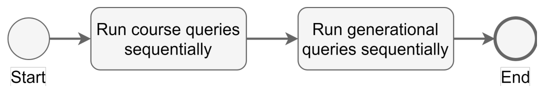

Start
Run course queries sequentially
Run generational queries sequentially
End

Figure 5: Diagram overview of the query executor process

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 queries run for each course. This architecture is represented in Figure 6.

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, 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. This lower level process runs once for every course name retrieved in the first stage.

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 of the aforementioned level 1 of granularity, the two stage process shown in Figure 6 also applies here; however, rather than running each query once for every course name, the query runs once for every generation (year of admission of students to the degree under analysis). An example of

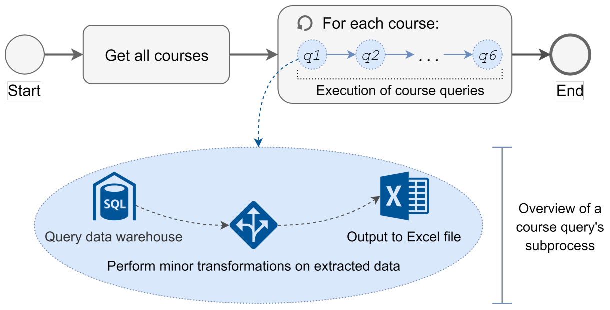
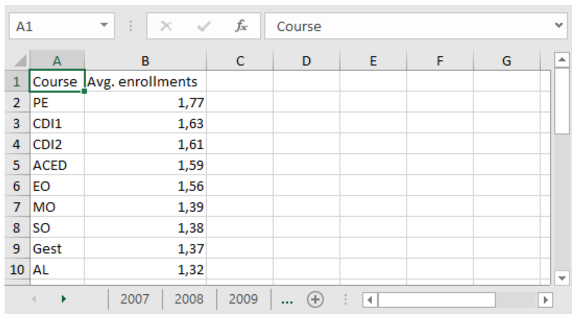

Start
Get all courses
For each course:
q1
q2
q6
Execution of course queries
End
Query data warehouse
Output to Excel file
Perform minor transformations on extracted data
Overview of a course query's subprocess

Figure 6: Diagram overview of the query executor process

an output file for one of these queries is displayed in Figure 7.



	A	B	C	D	E	F	G
1	Course	Avg. enrollments					
2	PE	1,77					
3	CDI1	1,63					
4	CDI2	1,61					
5	ACED	1,59					
6	EO	1,56					
7	MO	1,39					
8	SO	1,38					
9	Gest	1,37					
10	AL	1,32					

Figure 7: Output Excel file for the average number of enrollments needed to pass a course, by course

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.

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 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, and execution times for the desired process with different quantities of input data were measured.

5.1. 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, 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, and some criteria were defined differently, making the comparison with existing results impossible. In particular, direct comparison of any course query 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. However, if the Excel files used for validation were altered to only consider the same subset of students, the query results matched.

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.

These errors 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, and students being associated with incorrect ECTS credit numbers.

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, 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 queries in the query executor process were defined incorrectly in earlier stages of this prototype, leading to unexpected results.

5.2. Performance

The following business requirements were set for acceptance of this system: first, the time elapsed to populate the data warehouse must be 30 minutes or less, with all 11 available generations of LEIC-T serving as input; second, the time elapsed to execute all supported queries must be 60 minutes or less, with all 11 available generations of LEIC-T as input. Execution time should increase linearly with the number of generations in both cases.

To verify how the execution time of the processes increased with the number of generations, a script was developed to iterate all data sources (excluding files listing the degrees to consider) 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.

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.

The results for the populator process can be seen in Figure 8.

By analyzing the plot corresponding to execution

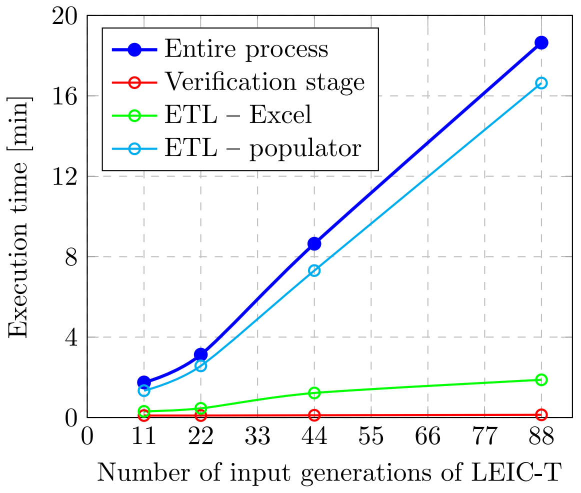


Figure 8: Execution time of the populator process and its components

time of the entire process (which is the sum of the execution times of its three subcomponents), 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.

It can also be seen that the dark blue plot represents a near linear increase of execution time. 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.

Regarding the query executor process, the results are shown in Figure 9.

The dark blue plot corresponds to the execution time of the entire process, and the three subcomponents are: the executor of generational queries excluding one of the queries (in green), the executor of course queries (in red), and the excluded query (in orange).

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

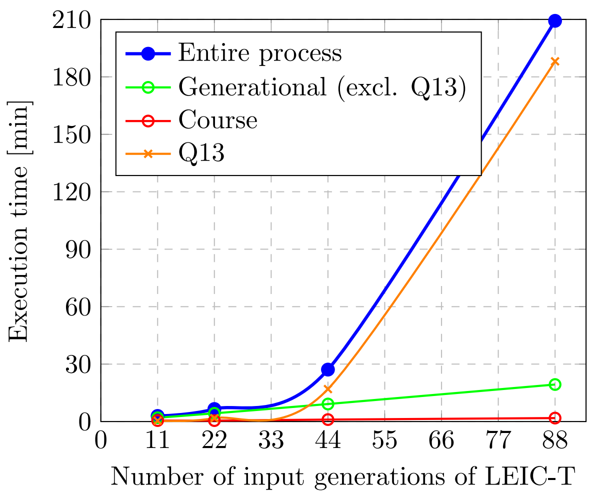


Figure 9: Execution time of the query executor process and its components

were much higher or even infinite, because many optimizations had not been implemented yet.

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 one of the queries to allow easy identification of the culprit. Attempts at optimizing it, such as insertion of indexes on fields that serve as join keys, and splitting the query into smaller queries did not decrease the execution time significantly. Therefore, this is a limitation that should be worked on by future prototypes. Considering the maximum number of input generations, 88, this query alone accounted for 91% of the execution time of the entire process, including 19 queries.

We have ensured that this query was the only culprit, as not considering it in the dark blue plot leads to a linear increase in execution time rather than an exponential increase.

6. Conclusions

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.

As this project represents a first functional prototype, there is still a lot of work to be done. Some assumptions that facilitate the execution of some of the supported queries were made; they 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.

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 unexplored 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.

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.

Acknowledgements

I am very grateful to my thesis advisors Prof. Helena Galhardas and Prof. Nuno Mamede, for their continuous guidance and support during this year.

I would also like to thank my family and friends, as they all played important roles in my life.

References

[1] M. Baranović, M. Madunić, and I. Mekterović. Data Warehouse as a Part of the Higher Education Information System in Croatia. Technical report, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, 2003.

[2] R. Kimball and M. Ross. *The Data Warehouse Toolkit*. Wiley, third edition, 2013.

[3] I. Mekterović, L. Brkić, and M. Baranović. Improving the ETL process of Higher Education Information System Data Warehouse. In *WSEAS International Conference on Applied Informatics and Communications*, 2009.

[4] V. L. Sauter. *Decision Support Systems for Business Intelligence*. Wiley, second edition, 2010.

[5] A. S. Sinaga and A. S. Girsang. University Accreditation using Data Warehouse. *Journal of Physics: Conference Series*, (801), 2016.