Design and Implementation of a Business Intelligence system for Website Visitors Statistics

Author: Daan Wissing

Supervisors: Dr. Toon Calders
ir. Thijs Putman

Eindhoven, April 23, 2012
Abstract

StudyPortals is a website which collects detailed information about study programs and universities from Europe. Since its foundation, the number of visitors and the number of displayed programs have been rising considerably. Analytical insight into the user base is very important for the long-term strategy: these insights can determine on what group to focus, how to improve the website or how to market the website to universities.

Since June 2011, extensive user data is gathered from the website, including what searches are done on the internal search engine, which programs are being viewed, which banners are shown, but also from which country the users come from. Although all of this data is already stored in a relational database, there is no simple method or system to make an overview of the desired data. The system described in this thesis is a first attempt to create a usable system for these kind of overviews.

While the data is already stored in a database, we extract, transform and load the data in another database designed for this exact purpose. For this, we have identified several dimensions that we want to examine: the time, the origin of the user, the search done on the website and the program that was viewed. Two scripts were created to facilitate this: one script to initially populate the database with the data available on that moment, and one script to keep the database up-to-date on a daily basis. The main challenge with these scripts is processing the sheer volume of the data.

On top of the database, a data warehouse is placed. This layer makes it easy for the user of the system to browse through the data overviews by doing typical OLAP operations, such as drill down, pivot and slice and dice. Installation and configuration was done using an open source OLAP server, Mondrian.

Finally, the results from these analyses are visualized using a JSP package called JPivot. This package provides a user interface to the OLAP functionality of Mondrian and is able to generate tables and graphs of the data. This packages is deployed with prepared MDX queries, which can be altered by the end user.

While doing this project, analysis was done on the growth of the database. This came to around 300 MB per month, which can be considered small for a typical OLAP data source. Also, the usage of materialized views was researched, and as a consequence three views were materialized which improved the response time of some queries considerably.

The system is ready to use as-is, but can be easily expanded to analyze more or other types of data. Other possible future developments are the integration in the existing systems of StudyPortals and the configuration to use this system as a Service Oriented Architecture.
Acknowledgements

I would like to thank the following people who have made it possible for me to do this project and for guiding and supporting me throughout my study.

My first thanks will go to the supervisors of my graduation project; Toon Calders from the Department of Mathematics and Computer Science of the Eindhoven University of Technology, for the guidance, positive feedback and suggestions given throughout this project; and Thijs Putman from StudyPortals, for providing support and answers whenever I needed it and for sparking my interest in this field of study in the first place.

Second, I would like to thank the rest of the StudyPortals team for supporting my project in general, providing me with the opportunity to do this project, and for periodically reminding me that there are more things in life beside study and work.

Next, I want to give my thanks to my friends and fellow students with whom I worked, studied, laughed and discussed during my studies in Eindhoven, especially Bart, Daniël, Maarten, Thomas, Rutger, Simon and Wouter. You have been a positive motivator for me all that time.

I would also like to thank my family, especially both of my parents and my brother, for supporting me in my choice of study, for trusting me to make my own choices – even if that choice lead to a delay in my study – and for continuing to believe in me.

Last but not least, I want to thank my girlfriend, for encouraging me during good times and supporting me during hard times. You always kept me motivated.

I could not have done this without any of you!
## Contents

List of Figures iv

List of Tables v

1 Introduction 1
   1.1 Environment 1
   1.2 Data Warehousing 2
   1.3 Project Goal 3
   1.4 Outline of the document 4

2 Existing Infrastructure at StudyPortals 5
   2.1 Existing Database 5
   2.2 Other Statistic Systems 7

3 Design 9
   3.1 Subset of the Existing Database Selected for Analysis 9
   3.2 Design of the database 10
   3.3 Granularity of the data 16
   3.4 Aggregate tables 19
   3.5 OLAP 23

4 Implementation 27
   4.1 MySQL 27
   4.2 Mondrian 28
   4.3 JPivot 32

5 Data Conversion 33
   5.1 Initial Data Loading 33
   5.2 Keeping the data up to date 34
   5.3 Challenges 34

6 Analysis and Conclusion 39
   6.1 Analysis on the Growth of the OLAP Database 39
   6.2 Evaluation of the system 46
   6.3 Goal Analysis 48
   6.4 Future Work 49

A Website screenshots 51
B Actions possible on the website 53
C Database Schemas 55
D Statistics Database Content Examples 57
E OLAP Database Content Examples 59
F Mondrian Screenshots 62
Bibliography 66
List of Figures

1.1 Typical Data Warehousing Architecture .......................... 3
2.1 Screenshot of the Advanced Search page ........................ 6
3.1 The schema of the OLAP database ................................. 11
3.2 Data in the database after a search is done .................... 19
3.3 The lattice of possible views from an example data warehouse .. 20
3.4 Example of a data cube .......................................... 24
3.5 Example of Pivoting a data cube .................................. 25
3.6 Drill Down and Roll Up example ................................ 26
3.7 Slice and Dice example ............................................ 26
5.1 Examples of correct and incorrect strategies of batching ........ 37
   (a) Incorrect batches of identifiers .............................. 37
   (b) Correct batches of identifiers ............................... 37
6.1 Amount of unique visitors of the website ......................... 41
6.2 The total size in disk size ....................................... 44
6.3 The absolute daily grow in number of rows ....................... 44
   (a) Growth of all tables .................................. 44
   (b) Growth of smaller tables ................................ 44
6.4 The absolute daily grow in disk size ............................ 45
   (a) Growth of all tables .................................. 45
   (b) Growth of smaller tables ................................ 45
6.5 Running times of materialized views, before and after aggregation 46
6.6 A report showing popular disciplines on the y-axis and countries on the x-axis ................................ 47
6.7 A report showing popular programs on the y-axis and months on the x-axis ................................ 47
A.1 Result page of a search ..................................... 51
A.2 Example of a program view .................................... 52
C.1 Schema of the OLAP database ................................ 55
C.2 Schema of the Statistics database ............................... 56
D.1 Example of data on stems in the Statistics Database ............ 57
D.2 Example of data on stems in the Statistics Database ............ 58
E.1 Database content after one search .............................. 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.2</td>
<td>Database content after another search</td>
<td>60</td>
</tr>
<tr>
<td>E.3</td>
<td>Database content after viewing a program</td>
<td>60</td>
</tr>
<tr>
<td>E.4</td>
<td>Database content after viewing a program again</td>
<td>61</td>
</tr>
<tr>
<td>F.1</td>
<td>Screenshot of the number of searches done for countries per origin continent</td>
<td>63</td>
</tr>
<tr>
<td>F.2</td>
<td>Screenshot of discipline search percentage per country</td>
<td>64</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Description of the Date table</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Description of the Program view table</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Description of the Search table</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>Description of the Parameter table</td>
<td>15</td>
</tr>
<tr>
<td>3.5</td>
<td>Description of the Keywords table</td>
<td>15</td>
</tr>
<tr>
<td>3.6</td>
<td>Description of the Visit table</td>
<td>15</td>
</tr>
<tr>
<td>3.7</td>
<td>Description of the Fact table</td>
<td>16</td>
</tr>
<tr>
<td>3.8</td>
<td>A test of the granularity of the Date dimension</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>(a) Size with granularity of a day</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>(b) Size with granularity of a month</td>
<td>18</td>
</tr>
<tr>
<td>3.9</td>
<td>Top 5 total benefit during the first round</td>
<td>22</td>
</tr>
<tr>
<td>3.10</td>
<td>Top 5 total benefit during the second round</td>
<td>22</td>
</tr>
<tr>
<td>3.11</td>
<td>Top 5 total benefit during the third and final round</td>
<td>23</td>
</tr>
<tr>
<td>6.1</td>
<td>Growth Results after each month</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(a) One Month</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(b) Two Months</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(c) Three Months</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(d) Four Months</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(e) Five Months</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>(f) Six Months</td>
<td>43</td>
</tr>
<tr>
<td>6.2</td>
<td>Running times of some reports</td>
<td>46</td>
</tr>
<tr>
<td>6.3</td>
<td>Running times of some reports</td>
<td>48</td>
</tr>
<tr>
<td>B.1</td>
<td>Actions stored in the Statistics database</td>
<td>54</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Information is one of the most basic needs in modern organizations. Without knowledge about clients, products, targets or the market in general, no organization can become successful. One of those information pillars is statistical knowledge of your business target. It can give you a big advantage when you know where the interests and goals of your targets lie.

1.1 Environment

StudyPortals [1] is a website concerning studying abroad. It maintains a collection of studies on many levels (bachelors, masters, PhDs, scholarships, short courses) that are available throughout Europe. The company started as a hobby project by three Industrial Engineering students. They had the opinion that finding a study abroad was too difficult and time consuming. Creating a simple portal (named MastersPortal) that provided future students with information about Master studies in other countries and the ability to compare them, proved to be a niche in the market. In 2009, a private enterprise was founded, which quickly became the market leader. At the moment of writing, over 950 universities and institutions are participating in 39 European countries, providing over 18,000 Masters, 5,000 Bachelors, 1,500 PhDs, 1,400 Scholarships and 1,000 Short Courses, divided over their respective portal sites MastersPortal.eu, BachelorsPortal.eu, PhDPortal.eu, ScholarshipPortal.eu and ShortCoursesPortal.eu. These portals are available in several languages: English, German, French and Spanish.

The goal of StudyPortals is both informing potential students about interesting study opportunities and giving universities the possibility to advertise their courses and programs to potential students. Knowing what the potential students are interested in is one of the key challenges within the company. To this end, a logging system has been introduced in June 2011. This system keeps track of what visitors of the website look at, where they click on, where they come from, what they search for and so on. Furthermore, the system keeps track of when and where banners are shown, which programs were promoted extra and much more. The data collected by this system is currently only used for invoicing universities, but the main purpose of logging is to collect data that might be interesting in some point in time.
The data gathered is initially stored in log files at the time that it is produced. These log files are then loaded in a relational database on a daily basis. Detailed information about this database is available in section 2.1. However, this database alone is not enough to analyze data about visitors of the website. To accomplish that, we need to build a Business Intelligence system on top of that. What information we want to analyze exactly is further discussed in section 1.3. How we will realize this will then be described in chapters 3 and 4.

1.2 Data Warehousing

In 1993, Codd et al [2] compiled a mandate about OLAP systems. In this mandate, they made a clear distinction between OLTP (OnLine Transactional Processing) and OLAP (OnLine Analytical Processing). OLTP is the classical transactional database: a structured storage for transactional data, such as personnel records or inventory systems. The requirements for these types of systems are:

- The data is instantaneously accurate (i.e. the data is representative of the reality at that very moment).
- Efficient for all transactions, including insert and update transactions.
- Support data manipulation to update existing records.
- Optimized to perform typically small queries affecting one or two tables.
- Designed to have very low redundancy by normalization of data.

In contrast, OLAP systems have a very different set of requirements. They are mostly used for analyzing data (hence Analytical Processing) and supporting Business Intelligence applications. Requirements of these systems are:

- The data is historically accurate as of a given point in time (i.e. the data is representative of the reality at some point in history).
- Efficient for reading and aggregate operations.
- Support queries joining many tables.
- Insert new data upon changes, to keep historical data intact.
- Allow redundancy to facilitate faster processing of queries.

In 1997, Chaudhuri and Dayal [3] provide an overview of the then-current situation on OLAP Technology and Data Warehousing. They define data warehousing as “a collection of decision support technologies, aimed at enabling the knowledge worker to make better and faster decisions.” Figure 1.1 shows a typical data warehouse architecture.

This architecture is aimed at large, enterprise-wide systems, collecting analytical information about all subjects across the whole organization. When an organization is sufficiently large (say, over 1,000 employees), designing and implementing a data warehouse is “a long and complex process, requiring extensive business modeling and may take many years to succeed”[3]. Smaller versions, concerning only specific data (subsets focused on selected subjects) are called Data Marts. These are easier to develop and roll out than a full scale data warehouse. However, it may be difficult to integrate data marts afterwards if this was not taken into account beforehand.
OLAP systems typically come in two flavors: Multidimensional OnLine Analytical Processing (MOLAP) and Relational OnLine Analytical Processing (ROLAP) [4]. The difference between these systems is the method of storing and querying data.

In MOLAP systems, the data is stored in specialized multidimensional structures, for example a high dimensional matrix. These systems typically have provisions and methods for handling sparse data and advanced indexing. They also generally provide a more space efficient storage and fast query responses. Disadvantages of MOLAP are that some tools have difficulty querying dimensions with a high cardinality (i.e. a high number of members) and data loading and updating can become a lengthy process.[4]

ROLAP systems are built upon a relational database for storing data. These have specialized index structures and typically scale better as more facts are entered in the data warehouse. Also, ROLAP systems are more flexible when the data cube is redefined at a later point in time. Disadvantages of ROLAP are that the query performance is typically slower, management of aggregate tables needs to be done separately and SQL (the standard query language for relational databases) is not optimized for heavy calculations like budgeting reports. [4]

Hybrid Online Analytical Processing (HOLAP) is a technique that combines both of these techniques. Typically, for high-level summary data MOLAP is used, while the storage of detailed data is done by using ROLAP. [4]

1.3 Project Goal

The goal of this project is to build a system to transform and analyze parts of the statistical data present in the current system. After this transformation, the data should be presented in a way that makes it easy to analyze and create overviews. The data used for analysis is a subset of the data stored in the Statistics database and will be completely focused on the actions of users. The specification of this subset was determined in consultation with the StudyPortals
team.

This system will be separated (for now) from other projects at StudyPortals; the only other system it is dependent of is the Statistics database (described in section 2.1). The intention of StudyPortals is to eventually integrate it within the existing administration environment. However, this is something for the future and was not a part of the project.

The goal of the project can be stated in several points:

1. To determine the subset of data over which analysis should be done, in consultation with the StudyPortals team.
2. To design a system that is able to analyze the data collected in the existing database, with respect to the subset mentioned above.
3. To implement this design, conforming to the development environment that is already present in the company.
4. To create a mechanism that keeps the data that is analyzed up to date, if necessary.
5. To design and create a way to visualize the analyzed data.

In section 6.3 we will reflect on the goals stated here and discuss if we have achieved these goals and how they were achieved.

1.4 Outline of the document

In chapter 2 we will describe the infrastructure already present at StudyPortals, and how this may benefit our goals. Chapter 3 gives an outline of the design for the eventual system, including theoretical background. The actual implementation and the choice of components is discussed in chapter 4, while chapter 5 deals with extracting, transforming and loading of data – both to populate the warehouse initially and to keep the warehouse up to date. Finally, in chapter 6 we analyze how the database is expected to grow in the future, show some results from developing the system and we will reflect on the goals stated for this project earlier.
Chapter 2

Existing Infrastructure at StudyPortals

2.1 Existing Database

As mentioned in the introduction, there already exists a database containing statistics of the website. In the remainder of this document, we will call this database the Statistics database. The schema of this database can be seen in figure C.2. It has a schema that closely resembles a star schema[4], with the Facts table as the central table connecting all dimensions.

The data in the Statistics database is updated on a daily basis. The actions are first recorded in a temporary plain text file on the server at the moment the action takes place. For each timeframe a new file is used, timeframes run from 23:00 to 23:00 the next day. These actions are then loaded in the database every night around midnight. After the actions are loaded, the indexing tables (see section 2.1.4) are updated.

In the remainder of this section, the layout and data of this database will be briefly explained.

2.1.1 Actions part in the Statistics Database

This part of the database consists of the tables facts, actions, actions and entities. In this part, the actions that have occurred on the website are stored. Actions include searches, views, impressions (of banners), clicks and submissions. These actions take place at a certain location, such as on a search result page, a page of a country, an institute or a program. With most actions, there is also an entity_id recorded. The meaning of this identifier is dependent on the context of the action. For example, if the action is done on a program, the entity_id is the identifier of that program. If the action is done on a country or university, the entity_id represents the identifier of the country or the university, respectively. If the action is done on a search, the entity_id is the identifier of that search in the same database. For an overview of all possible actions, see Appendix B.
2.1.2 Search part in the Statistics Database

In the Search part of the database the different searches are stored. On the website a search can be done in two ways: a quick search in which keywords and a discipline are entered, or an advanced search in which more parameters can be entered. In this database and throughout the rest of this document, no distinction is made between these two kinds of search. There are roughly three subparts in the Search dimension: keywords, parameters and search domain. Figure 2.1 shows the search page of the website and has the subparts highlighted.

The keywords part stores the keywords used in the search, if any. Besides the keywords, the stem of the keyword used is stored as well. This stem is language dependent. Because of this, it is likely that some keywords are stored double because the stem is different in another language. See also section 3.1 for why this is the case. This part consists of the tables search_keywords, search_stems, keywords and stems.

The parameters part only consists of the search_parameters table. This table contains the parameters that can be used by the user to refine a search. Parameters are: the maximum tuition fee, the degree achieved with the study program, the alternative program schedule (like parttime, online), special programs, minimal duration and maximal duration.

The search domain part captures in which domain the user wants to search for programs; the desired disciplines, countries and regions are stored. Although it is only possible to select one of each category on the website at this time, the database is designed to be able to deal with multiple choices per category. The data for these tables is fixed in the website design, meaning that...
users can only choose them from a list. However, these lists can change over time. This part consists of the tables search_countries, search_regions, search_disciplines, countries, regions and disciplines.

Unlike the table layout suggests, search ids in the facts table do not indicate searches done at that time. Instead, a search id represents the search that preceded the action that was taken. When an actual search is done, it is stored with an action as described in section 2.1.1. So for example: if a search is being refined, the search id in the facts table is the search with the old parameters, keywords and domain, while the entity id in the action part of the table is the search id of the new search action.

2.1.3 Visit part in the Statistics Database

This part stores information about the visiting user. In the table visits the visitors IP address, the ISO 3166 country code [5] and the browser language of the visitor are stored. Every fact in the database is coupled to a visitor.

2.1.4 Indexing tables

The three tables facts_visits, facts_actions and facts_searches are called indexing tables. These tables are actually aggregate tables over their respective dimensions visit, action and search. Aggregation is done per dimensional entity over the course of one hour. The column fact_count counts the number of facts in this hour with the respective id. The column facts is an encoding of the fact ids of the facts table over which the aggregation has taken place. The encoding is in place to minimize the space taken by these tables.

2.2 Other Statistic Systems

Besides the Statistics database, there are also some other basic statistical systems being used at StudyPortals. These systems are used for analyzing the current strategy, to provide statistics in white-papers and for informing potential clients.

StoneSteps

First of all, there is a tool called StoneSteps [6]. This tool reads out server logs and displays a graphical representation of these logs. Statistics gathered by this system include the number of unique visitors, the number of hits, the country of the visitors, number of page requests and so on. The number of unique visitors is the most important measure delivered by this tool. Statistics are viewable per portal or all combined.

Google Analytics

Another tool that is used is Google Analytics\(^1\). Google Analytics is used to monitor traffic coming to the website. The main application for this is to investigate running campaigns (such as advertisements on Facebook or Google

\(^1\)http://www.google.com/analytics/
AdWords) and for Search Engine Optimization. It can track the origin of the visitor, when they visit and how long they stay on the website. Also, it tracks the link that is being used to come to the page. With some smart choice for links, this can be used to track from which website or campaign a visitor comes from.

Both of these systems are not aware of the search engine developed by Study-Portals. Because of this, it is not possible to become completely aware of what the visitors are interested in. For these systems, the website is a black-box, with traffic going in and out, but they are unaware what happens inside of that box. Therefore there is a need for a system that does know the inner workings of the website.
Chapter 3

Design

3.1 Subset of the Existing Database Selected for Analysis

Before creating any design of a database, we need to determine what data we exactly want to analyze. As stated before, analyzing the complete set of data in the Statistics database will be counter-productive and creates a cluttered system in which the desired information will be hard to find, if not impossible. The way this was achieved was by interviewing several stakeholders of the system – the CTO, the CEO and the Director of Student Value.

As explained in section 2.1 there are several dimensions, hierarchies and subdimensions to choose from when analyzing. The following options exist:

- Origin of the visitor
- Search
- Several types of action (see Appendix B for which actions are present)
- Time

The most important and obvious data that should be included is the data directly concerning visitors of the website. The Visits table provides in this information. The country and the continent of the visitor can be derived from the field country. The field language is not very interesting, as it denotes the language option used for the keyboard of the visitor. This language differs from country to country, in which most countries have their own standard (such as France using the fr language), while some countries which have their own standard use some other standard (such as The Netherlands having the nl option, but typically using the en-US option [7]). 40% of the visitors used the en-US option, of 19% the language was not determined and 9% used the option en-GB, confirming that the keyboard language is not a usable property of information.

A straightforward way of finding out about the interest of visitors is to analyze the searches done by visitors. Since all of the parts described in section 2.1.2 may be of equal importance, almost all of it will be included in the OLAP database. The only exception is the stems of the keywords. Stemming is done by the custom search engine and these stems are included in the logging system. A drawback is that the stemming is dependent on the language selected on the website; each language uses its own stemming mechanism. Consequence is that
some words (e.g. “computer”) is stemmed different in the different languages (“computer” becomes “comput” in English, while it becomes “komputer” in German). Because the language used on the website is not logged, the usability of stems is greatly decreased. See appendix D how this is stored in the database.

Not all visitors use the search functionality, some do not even visit the front page. According to Google Analytics, a significant part of the visitors first use a search engine (such as Google, Bing or Yahoo) to search for a study. When a StudyPortals link comes up in the results, it is very likely to be a link to a program. A user’s visit to the program pages reveal a lot of information about the interest of a visitor as well. Therefore, all relevant available information about visitors viewing program pages – i.e. the name of the program, the program_id and the label – are included in the new database.

Last but not least, it is also important to analyze the moment that the actions took place. Since the facts table in the Statistics database stores the timestamp (with precision of a second) of the actions performed, we can make our own choice of the level of granularity. This decision was made in a later stage and is elaborated upon in section 3.3.

So to recap: the following data is included in the subset of data that is to be analyzed:

- The country of origin of the user.
- Searches done by the user:
  - The search phrase entered in the search bar.
  - Several options available while doing an advanced search, such as filters on degrees, alternatives and special programs.
  - Minimal and maximal duration of the study.
  - Maximal annual tuition fee.
  - Discipline of the program.
  - The country and region in which the program is taught.
- Individual programs viewed by the user:
  - The name of the program
  - The unique identifier of the program (program_id)
  - The label of the program
- The time of the user action.

### 3.2 Design of the database

The original, already existing Statistics database (see section 2.1) contains a lot of data, which includes data that is not relevant to this project. This has a big impact on the size of the database, which at the start of this project contained 75 million rows in the facts table and had a disk size of 6.1 Gigabyte. Also, the design of this database is in the hands of the IT team of StudyPortals, meaning that changes to this database will be limited to fit into the rest of the company projects. Therefore, the decision has been made with the supervisors to create a separate database that satisfies the needs of the system. This database will contain only the information that is needed for the system, in such a way that
analysis using OLAP techniques is supported. The final scheme of the OLAP database is shown in figure 3.1.

![Diagram of OLAP database schema](image.png)

**Figure 3.1: The schema of the OLAP database**

### 3.2.1 Snowflake Schema

Chaudhuri and Umeshwar [3] give a short description of database design technologies for data warehouses. Since we implement a ROLAP system, the notion of star schema and snowflake schema are very interesting. These schemas consist of a central Fact table that contains pointers to multiple Dimension tables.

A star schema is a special instance of a snowflake schema, in that it does not support normalized attribute hierarchies while the snowflake schema does. The power of these schemas is that they are not normalized as is usually the case in other ER diagrams. The disadvantage of this is that on one hand, there is a large probability of redundancy in the database. On the other hand, loading and querying data becomes more efficient.

**Snowflake Schema Normal Form**

Levene and Loizou have formalized the concept of a snowflake schema in [8]. The paper defines some theoretical structures and notations of databases, which are then used for a formal definition of the snowflake schema. Informally, a schema R is in Snowflake Schema Normal Form if it satisfies the following conditions:

1. R should be *acyclic*, thus inducing a join tree structure on R, where all edges have a distinct *label* (see below).
2. The intersection of any two relations in R (which labels an edge in the join tree) is a foreign key of one relation referencing the primary key of the other relation.
3. The hierarchical structure of the join tree of $R$ induces its integrity constraints in terms of Functional Dependencies and Inclusion Dependencies.

4. The root of the join tree of $R$ should be in Boyce-Codd Normal Form.

Analysis of the schema

We can see that the schema of the OLAP database is acyclic, thus satisfying condition 1. When analyzing the relations between the tables, we see that in all relations the primary key of one table is used in the other table as a reference, thus satisfying condition 2.

The table $\text{Facts}$ is the root in the Join Tree. The columns in this table are all foreign keys, except from the column $\text{count}$. This column is not a dimension or a foreign key, but a measure. More about measures can be read in section 3.5. The table is thus in Boyce–Codd Normal Form, satisfying condition 4.

Considering the Functional Dependencies and Inclusion Dependencies (condition 3); these can indeed be induced by the schema. The Functional Dependencies are:

- $\text{visit\_id} \rightarrow \{\text{country, continent}\}$
- $\text{view\_id} \rightarrow \{\text{program\_id, description, label}\}$
- $\text{search\_id} \rightarrow \{\text{parameter\_id, country, region, discipline, keyword\_id}\}$
- $\text{date\_id} \rightarrow \{\text{year, quarter, month, month\_name, day}\}$
- $\text{keyword\_id} \rightarrow \text{phrase}$
- $\text{parameter\_id} \rightarrow \{\text{tuition, degrees, alternatives, features, min\_duration, max\_duration}\}$

The Inclusion Dependencies are:

- $\text{Fact}[\text{view\_id}] \rightarrow \text{Program\_view}[\text{view\_id}]$
- $\text{Fact}[\text{visit\_id}] \rightarrow \text{Visit}[\text{visit\_id}]$
- $\text{Fact}[\text{search\_id}] \rightarrow \text{Search}[\text{search\_id}]$
- $\text{Fact}[\text{date\_id}] \rightarrow \text{Date}[\text{date\_id}]$
- $\text{Search}[\text{parameter\_id}] \rightarrow \text{Parameter}[\text{parameter\_id}]$
- $\text{Search}[\text{keyword\_id}] \rightarrow \text{Keywords}[\text{keyword\_id}]$

3.2.2 Tables of the OLAP database

Date

The $\text{Date}$ table (table 3.1) stores the date entities. A large part of the schema of this table is administration for easier usage in the OLAP layer. For example: Quarter and Month\_name are both deduced from the date value in the Statistics database. Quarter is only used for an extra layer of granularity between Month and Year, and Month\_name is used for display purposes only.

An alternative to storing the date could have been using a Unix Timestamp or a Date\_Time type in the $\text{Fact}$ table. When handling fact data, the appropriate date level (month, day etc) could be calculated on the fly. However, this approach is not optimal when a large piece of the data cube is accessed and aggregated over some time level (e.g. a request is to see the number of visitors per continent per month). In that case, the OLAP server is doing the conversion from Timestamp to month and grouping the results instead of the results
<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Content</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date_ID</td>
<td>Integer</td>
<td>Identifier</td>
<td>Primary key</td>
</tr>
<tr>
<td>Year</td>
<td>Integer</td>
<td>The year</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Quarter</td>
<td>Integer</td>
<td>The quarter, denoted with Q1, Q2 etc.</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Month</td>
<td>Integer</td>
<td>The number of the month, 1-based, so January is 1, February is 2 etc.</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Month_name</td>
<td>String</td>
<td>The name of the month (e.g. January)</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Day</td>
<td>Integer</td>
<td>The day of the month</td>
<td>Not NULL</td>
</tr>
</tbody>
</table>

Table 3.1: Description of the Date table

being grouped and queried by the database server. Since the database server is optimized to do the grouping/querying and the conversion of TimeStamp is rather expensive, we chose the solution stated here. This solution is also advised in the official documentation of Mondrian [9]. The consequence is that this table is not in Boyce–Codd Normal Form. Since only the Fact table needs to be in BCNF, this is not directly a problem.

Program

In table 3.2 the schema for individual programs is shown. A program is a study program that is offered by some institution, typically a university. A program is identified by the program id, but the description (most of the time the name) and the label can change over time.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Content</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>View_ID</td>
<td>Integer</td>
<td>Identifier</td>
<td>Primary key</td>
</tr>
<tr>
<td>Program_id</td>
<td>Integer</td>
<td>Identifier of the program</td>
<td>not NULL</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>A description (typically a title) of the program</td>
<td>-</td>
</tr>
<tr>
<td>Label</td>
<td>String</td>
<td>Can be one from (epic, premium, regular, grey) or NULL. Additional labels can be added later.</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.2: Description of the Program_view table

The label is a characteristic of a program that defines its status on the website. Universities have the option to buy extra exposure for their programs. This can come in several packages, each having a different cost for the universities. The label can be used to analyze if the extra exposure has any effect on the popularity of the program.
Search

The Search table (table 3.3) stores the different searches. Search in itself also has leaf tables (or subdimensions), these being Parameter and Keywords.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Content</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search_ID</td>
<td>Integer</td>
<td>Identifier</td>
<td>Primary key</td>
</tr>
<tr>
<td>Parameter_id</td>
<td>Integer</td>
<td>The foreign key pointing to the Parameter table</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Country</td>
<td>String</td>
<td>The country on which is searched</td>
<td>-</td>
</tr>
<tr>
<td>Region</td>
<td>String</td>
<td>The region on which is searched</td>
<td>-</td>
</tr>
<tr>
<td>Discipline</td>
<td>String</td>
<td>The discipline on which is searched</td>
<td>-</td>
</tr>
<tr>
<td>Keyword_id</td>
<td>Integer</td>
<td>The foreign key pointing to the Keywords table</td>
<td>-</td>
</tr>
<tr>
<td>Fingerprint</td>
<td>String</td>
<td>A fingerprint unique to this search. Used for quickly searching while updating the database. Calculated with the md5 algorithm</td>
<td>Unique, not NULL.</td>
</tr>
</tbody>
</table>

Table 3.3: Description of the Search table

The fields Discipline, Region and Country currently only contain one entity (e.g. “France” for Country, or “Computer Science & IT” for Discipline). However, it is possible that in the future you can search for multiple entities at once on the website. When this happens, the values in this field will be the concatenated string of entities, e.g. “France,Germany” for Country. For a discussion on the usefulness of this solution, see section 4.1.1.

Parameter

In the Parameter table (table 3.4) some common parameters used in a search are stored. This table is exactly the same as the table with the same name in the Statistics database, which is the main reason for this layout. It is safe to assume that the combination of all values except the identifier is unique in this table.

One particular thing to note is the choice of storing the fields Degrees, Alternatives and Features as a set. This has to do with the eventual implementation of this database and is further explained in section 4.1.1.

Keywords

This table (table 3.5) stores the full phrase that is used in a search. At the time of writing, the keywords as stored in the Statistics database are wrong; for some reason, most (over 80%) of the keywords contain the word “management”. This is a bug in the import script that populates the Statistics database. Although it is outside of the project, it influences the data used in our system. There
is not much that could have been done within this project. The procedures and implemented code will continue to exist so that in a future date, when the original bug is fixed, the OLAP database can be repopulated with correct information.

Visit

In table Visit the countries of origin are stored. The schema of this table is shown in table 3.6.

Since the country of the visitor is determined by the ip address, determining the country is sometimes impossible. For example, the country code A2 is used to designate an ip address assigned to a satellite provider. In the same vein, the country code EU is assigned to providers within the European Union, but from which the country is not known. Also, sometimes it proves to be impossible to get any country code at all from an ip address. In the cases that the country code cannot be resolved, a NULL value is used.
**Fact**

The **Fact** table (table 3.7) is where all the facts are collected. Every fact has at least a Visit id, because all facts are initiated by a visitor of the web site. Also, every fact has a Date id, because every fact has occurred at a certain time or timeframe. Furthermore, either a Search id or a Visit id is present, but never both.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Content</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact_id</td>
<td>Integer</td>
<td>Identifier</td>
<td>Primary key</td>
</tr>
<tr>
<td>Visit_id</td>
<td>Integer</td>
<td>The foreign key pointing to the Visit table</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Search_id</td>
<td>Integer</td>
<td>The foreign key pointing to the Search table</td>
<td>If NULL, this fact does not concern a search</td>
</tr>
<tr>
<td>View_id</td>
<td>Integer</td>
<td>The foreign key pointing to the View table</td>
<td>If NULL, this fact does not concern a view</td>
</tr>
<tr>
<td>Date_id</td>
<td>Integer</td>
<td>The foreign key pointing to the Date table</td>
<td>Not NULL</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>The number of times this fact has occurred</td>
<td>Not NULL</td>
</tr>
</tbody>
</table>

Table 3.7: Description of the **Fact** table

### 3.3 Granularity of the data

One important aspect when designing the snowflake schema, is to decide on the granularity of the stored data. Choosing the granularity is a consideration between either a larger level of detail, or a more compact database. Granularity can be done on several levels and choosing the right level is an important design decision. The most obvious dimension to decide on granularity is the date dimension, but other dimensions are possible candidates as well. In this database, the date and the visit dimensions are both candidates to be analyzed with respect to granularity.

#### 3.3.1 Date Granularity

When storing the facts in the **Fact** table, a date is connected to this fact. When similar facts happen on the same date, the **count** field indicates how many of these facts happen at that date. Choosing the granularity of the date has a large impact on the eventual number of rows in the **Fact** table.

Considering the data, there are several options on the level of granularity. From finest to coarsest these are:

- **Hour** Tracking the facts by the hour is probably a bit overkill. Since the website visitors come from all over the world, separating the hours is of limited
added value because of the different time zones visitors are in. This is because the time that is logged is the time of the server, which uses the Central Europe Time. Any relation to the time of day (e.g. more visitors in the evening) is very hard to discover, since in that case we need to calculate the time zone of the country the visitor is from. In some cases, like Russia or the USA, the country has multiple time zones, making this even more difficult.

**Day** Tracking facts on a day-to-day basis has a good level of detail: not much information is lost and very common facts are still grouped together. However, it is possible that this option will generate too much data, while using the lowest level of granularity might be very rare.

**Week** Facts on a weekly basis is a bit tricky. Weeks can overlap months (i.e. a week has facts of two different months) and even years. Therefore, when aggregating over weeks, we can no longer reliably calculate information about a single month or year, both of which are very likely to be desired in the end.

**Month** Monthly facts are expected to have the minimal granularity desired for analysis, since trend analysis is mostly done on a monthly basis. As compensation, the space needed to store the facts is smaller than when a finer granularity is used. How much smaller is not yet known, experiments will have to show this.

**Quarter** Facts calculated over a quarter are desired, but is too coarse. When analyzing facts over a time, information over individual months is lost. Although the storage per quarter will be significantly less that lower levels of granularity, it does not compensate the loss of information.

**Year** Even worse than with quarters, yearly fact collection is effectively useless for analyzing trends over time, especially now that not much information is available.

In the end, the best candidates for granularity are either per day or per month. An experiment (see section 3.3.3) will have to decide which should be used in this system.

### 3.3.2 Visit Granularity

Just like Date, choosing the level of granularity of visitors is a consideration between a large level of detail or a compact database.

There are several possible layers of granularity. From finest to coarsest:

**IP address** This is essentially tracing facts on an individual level. When using this layer it is possible to do some datamining on individuals (e.g. users that have searched for X also visit program Y). With the large number of visitors however, this would result in a very large database. Also, currently datamining on an individual level is not one of the goals of this system, so collecting facts at this level is probably overkill.

**Country** One popular dimension over which analysis is done is on the country level. For example, sometimes analysis is done to see which disciplines or
programs are popular in a particular country. Besides that, there are only a limited number of countries (at the moment 234 distinct countries are present in the database), so the size of the Fact table will not explode too much when choosing this level.

Continent In our system, 7 continents are identified. This means that choosing this level will mean a very compact Fact table concerning the visitors origins. However, a lot of information about individual countries is lost, and since these are explicitly mentioned in the analyses desired this level is not detailed enough to be very useful.

Global This level of granularity is essentially leaving out the visit table altogether. As discussed, analysis based on origin country is desired, meaning that this is not an option.

In this table the granularity level picked is on the level of Country, because it both satisfies the wishes of the system of analyzing data on a per-country basis, while the impact of choosing this level on the size of the Fact table is acceptable.

3.3.3 Date Granularity Experiment and Outcome

Because we have not yet made a decision on the granularity level of Date, a short experiment is done to make the right choice.

The two options available are on the level of daily facts, or of monthly facts. Keeping in line with the main size experiment (see section 6.1), we will see how big the database is after 6 full months of data. We will consider both number of rows and disk size, and look at the size of the Fact table and the size of the whole database. The result of this experiment can be seen in table 3.8.

<table>
<thead>
<tr>
<th># Rows</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>2,876,673</td>
</tr>
<tr>
<td>Database</td>
<td>3,325,454</td>
</tr>
</tbody>
</table>

(a) Size with granularity of a day

<table>
<thead>
<tr>
<th># Rows</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>1,685,940</td>
</tr>
<tr>
<td>Database</td>
<td>2,134,544</td>
</tr>
</tbody>
</table>

(b) Size with granularity of a month

Table 3.8: A test of the granularity of the Date dimension

As we can see when comparing table 3.8a to table 3.8b, choosing daily facts over monthly facts hardly doubles the size of the Fact table, both in number of rows and in disk size, while in theory it could have been multiplied by 30. An explanation why the daily facts are not 30 times as big, is that not all facts happen every day.

What can be concluded from this experiment is that the granularity of a day is the best choice: it offers detailed information, while the size of the database does not increase dramatically.

3.3.4 Example of a visitor generating data

To visualize what exactly happens when a user does some action on the website, figure 3.2 shows what rows are inserted in the OLAP database when a search is done. In this example, a user from The Netherlands is doing a search. This
search is for programs in Germany within the discipline of Computer Science & IT. The program must yield a Master of Science degree, is a fulltime study but shorter than 24 months.

<table>
<thead>
<tr>
<th>parameter_id</th>
<th>tuition</th>
<th>degrees</th>
<th>alternatives</th>
<th>features</th>
<th>min_duration</th>
<th>max_duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>msc</td>
<td>NULL</td>
<td>Fulltime</td>
<td>NULL</td>
<td>24</td>
</tr>
</tbody>
</table>

Figure 3.2: Data in the database after a search is done

In Appendix E there are some more examples of content in the OLAP database.

### 3.4 Aggregate tables

#### 3.4.1 What are aggregate tables?

Aggregate tables are tables that are essentially summaries of the fact table. Those summaries “lose” information of the facts, making them smaller than the fact table. For example, when an aggregate table does not take the time dimension into account, it summarizes over all the facts and grouping facts together which are equivalent except for the time. Listing 3.1 gives an example of how such an aggregation would look like when materialized by an SQL query.

The database on which this example runs is a simplified version of our database, having three dimensions: visit, time and discipline.

```sql
SELECT v.country, s.discipline, SUM(f.fact_count) FROM fact AS f INNER JOIN visit AS v ON v.id = f.visit_id INNER JOIN search AS s ON s.id = f.search_id GROUP BY v.country, s.discipline;
```

Listing 3.1: Example query of materializing a view aggregated over time

When an aggregate table aggregates over a certain dimension, we say that it is a *materialization* of a view. This view (or query) consists of the dimensions which were not used in the aggregation process. So in the example, the view (visit, discipline) was materialized. The purpose of aggregate tables is to decrease the time for processing common or complex queries. For example, say that the original fact table contains 1 million rows, and the view we just materialized contains 5,000 rows. This means that, disregarding overhead, displaying the view (visit, discipline) will cost only 0.5% of the time it used to. Moreover,
aggregating this view also means that other views (like the view \((\text{visit})\)) can be displayed faster.

Harinarayan et al. [10] provide an interesting and useful way to determine which views are the best candidates to be aggregated. In the first part of their paper, the lattice framework (notated as \(< L, \preceq >\)) is defined. This framework is used to create a lattice over all possible views in a data cube. The lattice of our example can be seen in figure 3.3.

In the lattice, every node denotes the dimensions and the used hierarchical level that are present in that view. So in \((\text{visit}, \text{day})\) the dimensions visit and time are present, while using the day level in the hierarchy of time. Thus, this view is an aggregation over discipline. The number in the nodes displays the number of records there are present in that view. So for example, \((\text{visit})\) shows us that there are 200 distinct visits in the system. With this lattice, it is easy to see which views are dependent on other views and how much. When materializing for example the view \((\text{Visit, Discipline})\), the views \((\text{Visit})\) and \((\text{Discipline})\) both benefit of this as well. This is because when calculating these views, the materialized view \((\text{Visit, Discipline})\) can be used, which is much smaller that the original fact table.

This brings us to the second part of the paper, namely how to determine which views to materialize. Some notations are introduced, such as \(C(v)\) being the (space) cost of view \(v\). Typically, we express this in number of rows. Also, the dependency relation \(\preceq\) is introduced between two views. When \(a \preceq b\), then \(b\) is an ancestor of \(a\). In other words, \(a\) can be calculated from \(b\). For example, the view \((\text{visit})\) can be calculated from the view \((\text{visit, time})\), thus \((\text{visit}) \preceq (\text{visit, time})\). This also counts for hierarchies, so e.g. in a Time dimension, \((\text{year}) \preceq (\text{month}) \preceq (\text{day})\).

The goal of the algorithm is making the total benefit as high as possible, while materializing a static number of views. This is done by selecting those views which benefit of materializing is most advantageous. This is done by calculating the total benefit of view \(v\) over a set of views \(S\), notated with \(B(v, S)\). The benefit is defined as follows:

1. For each \(w \preceq v\), define the quantity \(B_w\) by:
   - Let \(u\) be the view of least cost in \(S\) such that \(w \preceq v\)
   - If \(C(v) < C(u)\), then \(B_w = C(v) - C(u)\). Otherwise, \(B_w = 0\).
2. \(B(v, S) = \sum_{w \preceq v} B_w\)

The method uses a greedy algorithm to determine which views should be materialized. First, the total number of views to be materialized is determined.

Figure 3.3: The lattice of possible views from an example data warehouse

<table>
<thead>
<tr>
<th>Visit, Discipline, Day (1M)</th>
<th>Visit (200)</th>
<th>Discipline (25)</th>
<th>Day (5K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Next, choose the view of which $B(v, S)$ is maximal and add it to the set of views to be materialized. Repeat this until the maximal number of views is reached.

The benefit of this algorithm is proven to be at least 63% (e to be exact) of the benefit of the optimal solution.

### 3.4.2 Using Aggregate tables in the OLAP Database

Before applying these to our database we need to determine if it is indeed necessary. If our most intensive queries can be executed within a reasonable time, it will probably cost more effort to implement and maintain the aggregation of table than it is worth. According to [10] the accepted maximum query execution time lies around “a few seconds or a few minutes at the most”. Since this definition is a bit vague, we set our maximum query processing time at 10 seconds. That means, if there is a view that takes more that 10 seconds to process, it is worth it to materialize some views. Note that this does not mean that eventually all views should take less than 10 seconds to process, because this might be impossible when no aggregation is done and the benefits of meeting this standard may not weigh up to the extra size the aggregated tables add to the database.

A quick experiment shows us that the view (visit.country, program.program_id) currently takes 14.77 seconds to process, meaning that it is worthwhile to materialize some views in our system. First off, we need to investigate which views are in our system. What we know is that there are several interesting dimensions with their respecting hierarchy in our data warehouse:

- **Time:**
  - year
  - quarter
  - month
  - day
- **Visit:**
  - Continent
  - Country
- **Program:**
  - Program
  - label
  - description
  - program_id
- **Search:**
  - Discipline
  - Country
  - Features
  - Degrees
  - Alternatives

Note that not all hierarchies are listed: for example from Search, the hierarchy tuition is not listed. These hierarchies are not shown because the chance that they will be used are very low as the information they give is of low value. Another restriction we make is that a materialized view may consist of at most 3 dimensions. We do this because the chance that more than 3 dimensions are used at the same time is very low. This way we can eliminate some combinations beforehand, reducing the number of views we need to evaluate.

To determine the size of a view, we use a representative subset of facts. The subset will be the facts collected during the month November 2011. Using this we are able to determine the expected size in the long run. Views that consider the Time dimension will of course be properly calculated to determine the eventual size. Considering the nature of the system and the company, It is decided to calculate towards a usage time of 5 years. So in views where (day)
or (month) is included, we have to multiply the found value with 60 (5 years \* 12 months/year) to get the expected cost of the view.

A template query determining the size is shown in listing 3.2.

```
SELECT COUNT(*)
FROM (  
    SELECT f.id
    FROM fact AS f
    INNER JOIN visit AS v ON v.id = f.visit_id
    LEFT JOIN search AS s ON s.id = f.search_id
    LEFT JOIN parameter AS ps ON ps.id = s.paramter_id
    INNER JOIN date AS d ON d.id = f.date_id
    LEFT JOIN program_view AS p ON p.id = f.view_id
    GROUP BY <view>
) AS view_materialization
```

**Listing 3.2: Template query to determine the size of a view**

### 3.4.3 Views evaluation

With respect to the dimensions and hierarchies mentioned above, we come to a total of 82 candidate views to materialize. Of these views, table 3.9 shows the top 5 views that will generate the most benefit when materialized.

<table>
<thead>
<tr>
<th>View</th>
<th>Size of the view</th>
<th>Total Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit.country, day, features</td>
<td>279,060</td>
<td>599,901,780</td>
</tr>
<tr>
<td>visit.country, day, alternatives</td>
<td>399,960</td>
<td>597,846,480</td>
</tr>
<tr>
<td>visit.country, day, degrees</td>
<td>484,080</td>
<td>596,416,440</td>
</tr>
<tr>
<td>visit.country, day, label</td>
<td>495,300</td>
<td>596,225,700</td>
</tr>
<tr>
<td>visit.country, day, s.country</td>
<td>1,128,780</td>
<td>585,456,540</td>
</tr>
</tbody>
</table>

**Table 3.9: Top 5 total benefit during the first round**

The best view to materialize is (v.country, day, features), with a total benefit of 599,901,780. Interesting to see is that all views in the top 5 incorporate (v.country, day). Both of these are on the lowest layer of their respected hierarchy (Time and Visit) and are used in a sizable subset of all views. Consequence of this is that no less than 17 views benefit from the materialization of this view.

<table>
<thead>
<tr>
<th>View</th>
<th>Size of the view</th>
<th>Total Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit.country, day, alternatives</td>
<td>399,960</td>
<td>316,506,960</td>
</tr>
<tr>
<td>visit.country, day, degrees</td>
<td>484,080</td>
<td>315,749,880</td>
</tr>
<tr>
<td>visit.country, day, label</td>
<td>495,300</td>
<td>315,648,900</td>
</tr>
<tr>
<td>visit.country, day, s.country</td>
<td>1,128,780</td>
<td>309,947,580</td>
</tr>
<tr>
<td>visit.country, day, discipline</td>
<td>1,727,880</td>
<td>304,555,680</td>
</tr>
</tbody>
</table>

**Table 3.10: Top 5 total benefit during the second round**

After materializing this view, table 3.10 gives us the new top 5 views. As expected, the total benefit of the previous top 5 goes down, but the ranking
itself has not changed. This round, the view (visit.country, day, alternatives) has the highest total benefit.

<table>
<thead>
<tr>
<th>View</th>
<th>Size of the view</th>
<th>Total Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit.country, day, degrees</td>
<td>484,080</td>
<td>315,749,880</td>
</tr>
<tr>
<td>visit.country, day, label</td>
<td>495,300</td>
<td>315,648,900</td>
</tr>
<tr>
<td>visit.country, day, s.country</td>
<td>1,128,780</td>
<td>309,947,580</td>
</tr>
<tr>
<td>visit.country, day, discipline</td>
<td>1,727,880</td>
<td>304,555,680</td>
</tr>
<tr>
<td>visit.country, month, features</td>
<td>32,040</td>
<td>284,282,880</td>
</tr>
</tbody>
</table>

Table 3.11: Top 5 total benefit during the third and final round

Table 3.11 gives the result after materializing the view (visit.country, day, alternatives). Not much has changed in the top 5 of views; the view (visit.country, day, alternatives) is obviously absent and the view (visit.country, month, features) has joined the list.

After a discussion with some of the potential users of the system, it is decided that it will probably be more effective to materialize the view (visit.country, day, discipline) instead of (visit.country, day, degrees), since Discipline is an attribute that will be analyzed on a regular basis. This could have been avoided if we also calculated the probability of a query of being requested as described in [10]. However, for this to be effective an extra survey among the user group should be needed and probably some educated guesses about the probabilities.

3.5 OLAP

As described in section 1.2, OLAP is an extra layer on top of some data source, in this case the OLAP database. The function of this layer is to fetch and compute analytical data, such as averages, totals and outliers from a more low-level data source. While this can be theoretically done by a relational database and some smart SQL queries, an OLAP layer can do this more efficient and is easier to use. More important however is that there exist several tools (either free or commercial) that already implement all of these functionalities. The only thing needed is to configure such an OLAP system so that it can handle the data in the database.

The most important configuration of an OLAP system is the OLAP schema. In this schema, the data cubes, dimensions, hierarchies, levels and measures are defined. In section 3.1 the dimensions are defined and the hierarchies are indirectly determined in section 3.2.2 and section 3.3.

3.5.1 Data Cube

The data structure of OLAP is referred to as a Data Cube. The reason for this is that when the data is visualized, it is often done by showing a three-dimensional cube like in figure 3.4 – although most OLAP systems typically have more than three dimensions.
3.5.2 Dimensions, Hierarchies and Levels

The location of the cell in the Data Cube is determined by the values of its dimensions. These dimensions are units like time, geographical location, size, manufacturer etcetera. In the data cube of figure 3.4, the dimensions are Time, Origin and Discipline. Dimensions have at least one hierarchy – most of the time it is exactly one. A hierarchy defines the order of the levels in the dimension. Levels define the granularity of the dimension, for example one year, one week or one day. It is possible for multiple hierarchies to exist for the same dimension. The time dimension, for instance, has the levels all, year, month, week and day. One possible hierarchy is all → year → month → day. However, another hierarchy which does include the week level, is all → week → day. These two hierarchies can co-exist.

The dimensions are already decided upon in section 3.1. The levels of most dimensions were self-explanatory and could be deduced from the dimensions itself. For the dimensions time and visitor origin, see section 3.3.
3.5.3 Measures

Measures are a very important part of OLAP – they denote the value of a cell. The simplest form of this value is the number of facts within that cell (e.g. the number of searches for "Business Information Systems" on 01-01-2012). But other forms exist as well; for example the sum of the measured values within a range, the maximum, the minimum or the number of distinct values are some basic measures.

In our case there is one clear measure that is the most important: the sum of actions done. This can reflect on a search or a program view. This measure is easily extracted from the database by summing the count values from the fact table.

However, there are some other measures that can be of interest, like percentages. For example, the percentage of views that was done on a particular program. Or the percentage of searches done on a particular discipline with regards to the origin. These measures are called calculated measures.

3.5.4 OLAP operations

There exist several typical OLAP operations that can be executed on the data cube.

Pivot

The pivot operation (sometimes also referred to as rotating) is essentially reorientating the data cube. This is done so that the user of the OLAP system can get a different overview of the data. If, for example, the user is looking at a matrix overview of the visitors and the visit date, he may wish to pivot the view so he can see an overview of the visitors and the programs they have viewed. It is called pivoting because the point of view with respect to the data cube is changed. A pivot example is shown in figure 3.5.

Drill down / Roll up

Many times, the dimensions in a data cube have multiple levels, arranged in a hierarchy. When the user wants to have a more detailed view of some dimension, he can do a drill down operation. For example, if the user is looking at an
overview of the searches done in the first quarter of 2012, he can drill-down to the Month level, showing the searches done in January, February and March. The extreme variant of this is a drill-through. This operation drills the dimension down to the lowest level. The inverse of drill down is called roll-up. This is done when a user wants to have a more general overview of the data. An example for drilling down and rolling up can be seen in figure 3.6.

Slice and Dice

When analyzing data, the user sometimes would like to look at one specific point in the data. This is called slicing the data cube. For example, the user only wants to see the data affiliated with users that come from India. In this case, there is a slice done on India through the data cube.

Extending this, a dice is done when more than two dimensions are sliced. So in the previous example, the user can use a dice operation to only view the data affiliated with users from India in the last quarter of 2011. Figure 3.7 gives an example of slice and dice.
Chapter 4

Implementation

4.1 MySQL

Implementation of the OLAP database is straightforward: the design is detailed enough to make the decisions about the implementation easy. Choosing the Database Management System is done by discovering what is in use within the company and see if it can be useful. Since the Statistics database is stored in a MySQL database[11], it is only logical to create another database on the same MySQL server. The benefit of this is that no other system needs to be installed, and transactions between the two databases can be done efficiently. Also, only one database access permission is needed to read and update the OLAP database.

4.1.1 Implementation of sets

Special attention should go to the implementation of some sets in MySQL. Although MySQL supports sets\(^1\) in a table, these work more primitive than expected. For instance, when values are retrieved from a set column, the result will be a string of all values separated by commas. There are no high-level functions on sets in MySQL, such as unions, intersections or complements. This is understandable, as these functions, as well as sets in general, can be modeled using standard database design.

This problem is most prevalent with the three set attributes (degrees, alternatives and features) in the parameter table. These are already stored as a set in the Statistics database. Storing them so the OLAP server is able to recognize it as a proper set would require the addition of two extra tables per column, one for storing the single value and one to store the (many-to-many) relation. Although this implementation is not very complex, it would add two extra joins to any query concerning these attributes. Using this solution will most likely result in an increase of query time.

Ultimately, the chosen solution is to use the set type as provided by MySQL. The reason is that implementing these sets using proper database modeling techniques would cost a lot of extra time for research and testing, although that\(^1\)

\(^1\)http://dev.mysql.com/doc/refman/5.6/en/set.html
solution would ultimately have more use for analysis. This is something that can be investigated further in the future.

This solution is also chosen to store the domain part of the search (i.e. the disciplines, countries and regions). The problem is not as prevalent though, since as of yet is it not possible to search for multiple disciplines, countries or regions on the website. If this at some point does become possible, then the database is prepared for it.

4.2 Mondrian

Numerous systems offering OLAP functionality are available. Some well-known commercial OLAP products are SQL Server Analysis Services (Microsoft), Business Objects (SAP), Cognos Business Intelligence (IBM) and Hyperion (Oracle). Freeware products exist as well, such as Mondrian OLAP server (Pentaho), Palo (Jedox AG) and icCube OLAP Server (icCube). Funds for purchasing a commercial product were not available, while the free alternatives promised to be good enough for the usage in this project.

Since we already designed our data storage in a relational database, we are looking for a system that supports ROLAP. While Palo and icCube only support MOLAP, Mondrian [12] supports ROLAP and is thus the most obvious choice. Mondrian features the option to connect to multiple types of databases (MySQL, Oracle, PostgreSQL and many more), uses MDX (MultiDimensional eXpressions) as a query language and is developed in Java, so it can be run on virtually any Operating System. There are several reporting interfaces that can be connected to Mondrian; JPivot, Saiku and Pentaho Data Analysis.

4.2.1 Mondrian Schema

The Mondrian schema is the OLAP schema that defines the measures, dimensions, hierarchies and levels of the multi-dimensional database. It is constructed using XML, in which also the mapping to the relational database is made.

For an example of a dimension definition, see listing 4.1. In this definition, the [Time] dimension is defined, with one hierarchy. The base table for this dimension is the date table, with date_id as the primary key. The foreign key in the fact table is also called date_id. All levels of the hierarchy are mapped to the correct columns in the date table.

Now that this dimension has been defined, we can use MDX queries to fetch a set of date entities. For example, the expression [Time].[2012].[Q1].[January] defines the month January, which can be used to calculate facts over this period of time. Also notice that the level month is captioned by the name of the month (e.g. January) but is ordered by the number of the month (in this case, 1). MDX also offers a set of functions to manipulate these levels, members and sets. For example, the expression [Time].[2012].[Q2].Children() returns all the months in Quarter 2 of 2012.

Defining a dimension which consists of multiple tables need to define the join as well. An example is in listing 4.2.

The dimension defined here is the Search Phrase used in a Search. The Fact table does not directly reference this table, but it does reference it indirectly via
the **Search** table. Therefore, we needed to explicitly define this relation in the OLAP schema.

Measures are equally important to define. In listing 4.3 the measure for program views is defined. The design of the database schema groups together the views and searches, meaning that we can not directly calculate this measure by simply summing the values from the `count` column. Instead, we sum the values of those rows in which the `view_id` is not `NULL`.

```xml
<Measure name="Views" aggregator="sum" datatype="Integer">
  <MeasureExpression>
    <SQL dialect="mysql">
      (case when 'fact'.view_id IS NULL then 0 else 'fact'.count end)
    </SQL>
  </MeasureExpression>
</Measure>
```

Listing 4.3: Views measure

The measure for searches is similar to the measure for views, with the only difference that the check for a non-`NULL` value is done on the `search_id` column. Aside from these relatively simple constructs, Calculated Members exist as
These constructs do not calculate their value directly from a value in the fact table, but use an MDX formula. These can be used in two different ways: in an MDX query using the WITH MEMBER clause, or by defining a Calculated-Member in the schema.

Listing 4.4 is an example of a CalculatedMember as a Measure. What it does, is slicing over the number of searches done on a Discipline and dividing this over the total number of searches. It then displays the value as a percentage with two significant figures. This measure comes in handy when one wants to make an overview on what disciplines users search for with respect to where the user comes from.

Lastly, there are also Named Sets. These constructs are very similar to Calculated Members, except that they define a Set instead of a Member. Again, there are two ways of defining a Named Set. The first option is to use the WITH SET clause in an MDX expression. The second option is to declare a NamedSet in the schema.

Listing 4.5 defines a set called “Last 6 Months”. What it does is first calculating which month it currently is (using the MDX function CurrentDateMember()) and then creating a set which contains this month and the 5 months that precede it. A good usage for this set is when an overview of only the last six months is desired, instead of the whole period of monitoring. It is also useful for ordering entities being popular in the last 6 months, such as programs, disciplines or host countries.

### 4.2.2 Implementation and results of aggregate tables

The implementation of the aggregate tables is straightforward; however we are bound by the limitations of our OLAP server, in this case Mondrian. One of the limitations is that Mondrian does not offer any service to maintain or manage these tables, it can only detect and read out aggregate tables. That means that we have to update these tables manually, or at least in some other way. Also, we need to define the desired aggregate tables beforehand, while changes
to these tables or the dimensions they aggregate over need to be accompanied
with similar changes in the OLAP schema.

Another limitation is the naming of the tables and columns; Mondrian has
a strict set of rules for recognizing and handling aggregate tables. For example,
the name of an aggregate table needs to be of the form `agg_{name}_{name of
the fact table}`). So for our view `visit.country, day, features`, the table
name would be `agg_features_fact`.

Columns are subject to rules as well. There has to be a column named
`fact_count`, which counts the number of facts in the original `fact` table were ag-
gregated in this entry. Furthermore, names for foreign ids in the aggregate table
must match their equivalent name in the original `fact` table. Names for columns
depicting a dimensional level need to be of the form `{hierarchy name}_{level
name}`, while names of measures need to be of the form `{measure name}`. So
for our view `visit.country, day, features`, the column for the dimensional
level `Features` needs to have the name `parameter_features_features`, while
the measure `Searches` needs to have the name `searches`.

Now that all the aggregate tables are properly named and ready to be dis-
covered by Mondrian, all we need to do is to expand the Mondrian schema with
the aggregate tables. This is done by inserting the code from listing 4.6 in the
schema.

```xml
<AggName name="agg_features_fact">
  <AggFactCount column="fact_count" />
  <AggForeignKey factColumn="date_id" aggColumn="date_id" />
  <AggForeignKey factColumn="visit_id" aggColumn="visit_id" />
  <AggMeasure name="[Measures].[Searches]" column="searches" />
  <AggLevel name="[Parameter.Featur..features]" column="parame...features_features" />
</AggName>
```

Listing 4.6: Defining aggregate table of Features

Now, when an MDX query is performed, Mondrian automatically uses an
aggregate table when the measure and the level in the query match (or can be
calculated from) the materialized view.

The schema of the aggregate tables can be found in figure C.1.

4.2.3 Caching of Mondrian

While Mondrian does not directly stores the data, it uses a caching mechanism
to store previously executed queries. This mechanism is in particular useful
when a query is executed multiple times within a short period of time. The
documentation that describes this mechanism states that updating the data in
the database is not automatically detected by Mondrian. The advice is that a
special function of Mondrian needs to be called to clear the cache. However,
after some experimenting, it turns out that when doing only daily insertions in
the database the cache does not need to be cleared to show the new data. It is
not clear what the reason is for this behavior, but an undocumented feature or
out-of-date documents is the most likely.
4.3 JPivot

As already mentioned in section 4.2, there are several interfaces available for usage by Mondrian: JPivot, Saiku and Pentaho Data Analyzer. Pentaho Data Analysis is part of a large Business Intelligence package developed by Pentaho and uses a so-called subscription model. Since it is not possible to solely use the Data Analysis tool and installing the full Business Intelligence package is a bit overkill, this option is not very desirable.

Saiku is a “modular open-source analysis suite offering lightweight OLAP” developed by Analytical Labs. It features a rich interface that allows the user to create and save their own MDX queries with relative ease. However, after trying out this system with both the default dataset that was included in the installation and the already (partly) designed Mondrian schema, it gave the impression that the product was not totally finished yet. Some bugs made it very difficult to connect to the SQL database. Furthermore, it came already installed in its own version of Apache Tomcat. This restricts the system administrator of StudyPortals in installing the system on their server. Therefore, Saiku could not be used in this context.

JPivot is an older, open-source framework written in JSP. While the visuals of this front-end may look a bit outdated, the functionality is impressive. When viewing a query, it is possible for the user to do a manual drill-down/roll-up, swap the axes, export the results to a PDF or Excel file, create a graph of the data and so on. Furthermore, JPivot is automatically provided when downloading the Mondrian package from the official website. Finally, the only requirements to deploy a page using JPivot are a Tomcat installation and the installation of the Java Runtime Environment on the server.

Creating pages with JPivot is relatively easy. In the standard JPivot query page, the only thing that needs to be changed is the `<jp:mondrianQuery>` element. This element defines both the connection to the right data source – in our case, to the OLAP database – and the MDX query to display.

```xml
<jp:mondrianQuery id="query01"
    jdbcDriver="com.mysql.jdbc.Driver"
    jdbcUrl="jdbc:mysql://s183.studyportals/?user=*****&password=*****"
    catalogUri="/WEB-INF/queries/stats.xml">
    SELECT [Origin].[All Origins] ON COLUMNS,
    ORDER([CountrySearch].Members, [Measures].[Searches], BDESC)
    ON ROWS
    FROM [Hits]
    WHERE [Measures].[Searches]
</jp:mondrianQuery>
```

Listing 4.7: JPivot query for searched Host Countries

In listing 4.7 is an example of the `<jp:mondrianQuery>` element. The connection to the database is specified in the `jdbcUrl` attribute, while the MDX query to display is stated between the start- and end tag.
Chapter 5

Data Conversion

The most important aspect of a data warehouse is of course the data itself. Since this system uses its own data storage instead of an already existing one, populating this storage is not a trivial matter. In general, there are two approaches to this: converting and loading all known data from the existing data sources, and periodically updating the data storage with new data. Initially loading the data storage is discussed in section 5.1, while periodic updating is discussed in section 5.2.

PHP [13], a server-side scripting language was chosen as the language to write scripts for facilitating data loading. The choice to use PHP is not an obvious one, since there are a number of limitations to PHP that might have a negative impact on overall performance and functionality:

1. In a standard PHP installation, the memory usage of a PHP script is limited to 128 MegaByte\(^1\).
2. Since PHP is a weakly typed language, overhead tends to become large for variables [14].
3. The script needs to be run in a PHP context, meaning that the script cannot be run on a machine without a PHP installment.

The reason that PHP was chosen despite these limitations, is primarily because PHP is the de facto language used within the company. This makes sense, since PHP is a popular language for designing dynamic web pages. Consequence is that there are also several tools developed and maintained specific for PHP. One of them is a library that deals with connecting to a MySQL database, structuring queries in a way that prevents SQL-injections and retrieving results.

5.1 Initial Data Loading

When the system is initially rolled out, the data storage is empty. This gives us some advantages. For example, there is no need to take into account which values for dimensions have already been stored. Instead, we can just populate each dimension in one go. The approach to filling the database is:

1. Per dimension:

\(^1\)http://php.net/manual/en/ini.core.php#ini.sect.resource-limits
(a) Load the data from the existing Statistics database into memory.
(b) Convert the data to a format that conforms to the schema of the empty database.
(c) Store the data on the OLAP database.
(d) Save the primary keys of the data on the OLAP database in a mapping from old keys to new keys.

2. Load the relevant fact rows from the fact table.
3. Convert and aggregate the facts.
4. Store the facts in the OLAP database.
5. Populate the aggregate tables.

5.2 Keeping the data up to date

During the lifetime of the system, more data is stored in the Statistics database. Loading this data into our OLAP database is of course an important aspect of the system. The existing architecture of systems consists of a batch of scripts that is executed every day, including a script that updates the Statistics database on a daily basis. Therefore, it is logical to create a similar script that updates our database on a daily basis, based on the Statistics database. The approach to keep the data up to date is:

1. Calculate up to which date the database was updated
2. Pre-fetch all of the possible countries
3. Go over all facts between that last date and yesterday:
   (a) Check if the dimensional entities concerning this fact are already in the database:
      i. If not: insert the entity and store the new identifier.
      ii. Else: retrieve the identifier of the entity.
   (b) Update the corresponding (in-memory) fact row for the OLAP database.
4. Send all of the facts to the OLAP database.
5. Update the aggregate tables.

5.3 Challenges

Writing both of the scripts, several challenges occurred. Some challenges were easy to solve, others required large portions of the code to be rewritten. This section will describe some notable major challenges while implementing these scripts and the solutions chosen to deal with these challenges.

5.3.1 SQL result size

It became clear very quickly that loading all of the data at once would give OutOfMemory exceptions, due to limitation 1. After some debugging, it turns out that the home-brewed library that handles the database connection is doing some caching, which takes up a lot of memory. There exists an option to use an iterator over the query results. However, when testing this functionality it appeared that the results were the same as when not using an iterator – an
OutOfMemory exception. It appears that this functionality is currently broken and will not be fixed in the near future.

A workaround is to run the SQL query in parts using the LIMIT statement. With this statement, we collect the results in parts of, say, 30,000 rows. For example, for the first part of the results we use LIMIT 0, 30000, indicating that we want 30,000 results, starting with row 0. After processing this first batch of results, we can retrieve the next part of the result. For the second part we use LIMIT 30000, 30000, indicating that we want 30,000 results, starting with row 30,000. This continues, until the part that is processed contains less than 30,000 rows. In that case, we know that all results have been retrieved and we can continue with the next step.

5.3.2 Inserting rows in the OLAP database

There were some issues with sending large amounts of data to the OLAP database. At first, the rows were sent one by one, like in listing 5.1.

```
1 INSERT INTO visit (country, continent) VALUES
  (The Netherlands, Europe);
```

Listing 5.1: Inserting one row at a time

This proved to work slower than needed; when inserting a row in a table, the database management system does some overhead tasks, such as locking the table, checking constraints, updating indexes et cetera. When inserting several thousand rows in sequence, this overhead will cause the script to slow down significantly. The overhead can be reduced by inserting multiple rows at once, for instance like in listing 5.2. In these cases, the overhead tasks will be done only one time (locking the database once, checking constraints after all rows are inserted, updating indexes after all rows are inserted).

```
2 INSERT INTO visit (country, continent) VALUES
  (The Netherlands, Europe),
  (Germany, Europe),
  (United States, North-America),
  (India, Asia)
```

Listing 5.2: Inserting four rows at a time

When testing the new method of sending all rows to be inserted together, an error occurred. As it turns out, MySQL has a (standard) maximum package size of 1024 KB\(^2\). This is avoided by partitioning the packet in smaller INSERT statements of maximal 10,000 rows per statement. After implementing this feature, the rows were inserted without a problem and in a timely fashion.

A possible alternative to this solution could have been using database transactions. However, the library used does not support transactions of any kind but can only run separate SQL queries. Also, no interface for bulk inserts is provided, so in this context this solution is the best we can do.

\(^2\)http://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html#sysvar_max_allowed_packet
5.3.3 Mappings size

During the insertion of dimensions, a mapping is maintained from identifiers in the Statistics database to identifiers in the OLAP database. The reason for this is that, when processing the Fact table, there is an easy lookup from the old dimension identifier (e.g. the search id) to the equivalent dimension identifier in the OLAP database. For example, the mapping from search ids may look like in listing 5.3.

```php
$search_map = array(
    1 => 1,
    2 => 1,
    3 => 2,
    ...;
); // Listing 5.3: Example of a mapping of search ids
```

In this example, the searches in the Statistics database with identifiers 1 and 2 are mapped to the search entry with identifier 1 in the OLAP database.

Since the size of some dimensions (such as search) have the potential to become very large (as of 1-3-2012, the search dimension has over 300,000 rows), the mapping of these dimensions also grow in size. The size can become so problematic that there is a real chance that the memory limit of 128 MB is exceeded. There are two possible solutions to this problem. One solution used within the company is to use a form of writing to memory through some native PHP methods\(^3\). By directly storing the integers in memory without any unneeded overhead, the memory usage will go down significantly. Disadvantage is that writing directly to memory is less structured than variables, making it more prone to bugs.

Another possible solution is to make use of a relatively new construct in PHP: SplFixedArray\(^{15}\). This class is much like a slimmed down version of the standard array in PHP, with some changes:

- Only integers are allowed to use for indexing.
- The size of the array needs to be allocated explicitly.
- Indexes always go from 0 to the length of the array minus 1.

Several online articles provide some insight in the differences between array and SplFixedArray regarding performance and memory footprint \([16, 17, 18]\). The overall consensus is that SplFixedArray reduces the memory usage significantly, as long as the limitations are taken into account. Especially longer arrays benefit from this.

Eventually we chose to use the solution with SplFixedArray. The main reason for this was that this construct is a properly documented feature and did not require us to rewrite large pieces of code. The alternative of writing the mapping directly to memory appeared error-prone and there was no experience with it.

When implemented, the solution was applied to the mapping of search ids, parameter ids and phrase ids. Mapping to identifiers of program views proved to

be counter-effective, since this produced a sparse array with minimal 10 million entries, due to the way the views are stored in the Statistics database. Since the number of distinct views is not extremely high (57,000 as of 2-3-2012) using the standard array is sufficient in this case.

5.3.4 Matching the correct Original Facts

In the Facts table in the Statistics database, there are many types of facts stored (see section 2.1). To match the right facts in this table, first the desired action identifiers were calculated. However, in most cases the number of action ids that need to be searched became so big that the SQL query sent to the database became too large, similar to the problem in section 5.3.2. The same solution was applied (partitioning the query) but this introduced a new challenge, since several action ids in the Statistics database can map to the same dimensional entity in the OLAP database. This means that splitting up the query needs to be done carefully as not to separate actions that should be counted together. If these actions are split up, there is a chance that duplicate facts will end up in the OLAP database.

A solution for this is to partition the action ids in a smart way. The pseudocode for this solution is shown in Algorithm 1. In short: the action ids are divided in batches of maximal $MAX_LENGTH$ ids, unless an entity maps to a set of action ids longer than $MAX_LENGTH$. In that case, the set of action ids will be in one separate batch. Figure 5.1 gives an example of a good and a bad strategy of creating these batches. Here, the colored nodes all represent an action id from the Statistics database. If two nodes have the same color, they map to the same entity in the OLAP database. The size limit of batches in this example is 4, while the maximum number of entities that can be sent is 6. Figure 5.1a shows what happens when the entities are simply divided in batches of equal size: the green entities are split in two batches, with the consequence that a duplicate fact will appear in the database. Figure 5.1b shows that with the improved method, while more batches are needed, no duplicates will appear in the fact table.

A possible issue can arise when a set of ids is so large, that the resulting SQL query is still too large. With the current data this has not happened yet, but it is possible that it will be a problem in the future.
Algorithm 1 Pseudocode of dividing the action ids

\[
N = \text{new Mapping()}
\]

for all action\_ids = a\_id do

Create a mapping M from a\_id to the entity\_id

\[
N(\text{entity\_id}) \leftarrow N(\text{entity\_id}) \cup \{a\_id\}
\]
end for

Entity\_Group ← \text{PARTITIONIDS}(N)

function \text{PARTITIONIDS}(N)

\[\text{Merged} = \text{new array()}
\]

\[i \leftarrow 0\]

for all \(N = \text{action\_ids}\) do

if \(\text{Merged}[i] = \emptyset\) then

\(\text{Merged}[i] \leftarrow \text{action\_ids}\);

else if \((\|\text{Merged}[i]\| + ||\text{action\_ids}||) \leq \text{MAX\_LENGTH}\) then

\(\text{Merged}[i] \leftarrow \text{Merged}[i] \cup \text{action\_ids}\);

else

\[i \leftarrow i + 1\]

\(\text{Merged}[i] \leftarrow \text{action\_ids}\);

end if

end for

return \text{Merged}

end function

5.3.5 Reducing the Number of Facts per Batch

This challenge was mainly present in the script that initially populates the OLAP table. When collecting the facts, all facts associated with a group of certain program views or a group of certain searches are handled together (see section 5.3.4). As the Statistics database grows, the number of facts that was loaded and handled per batch also grew, together with the memory footprint. After a while, this might become a problem, because it is possible that in the future the database needs to be repopulated.

The solution for this problem is to retrieve and handle the fact rows on a per-month basis. This means that each batch of new fact rows only contains facts from one month. Side effect of this solution is that the time to import all of the facts is dramatically increased, from around 5 hours to at least 15 hours as of 1-3-2012. Because this script is not intended to be used on a regular basis, this is not a problem.
Chapter 6

Analysis and Conclusion

6.1 Analysis on the Growth of the OLAP Database

In this short study we will examine in what way the data is expected to grow over time. The goal is to find out what the (eventual) growth rate is of the data, whether this might become a problem and what recommendations can be made if any problems arise.

6.1.1 Setup of the Study

This experiment will analyze the growth of the OLAP database over six months. First we will look at the size after importing one month, then after two months, after three months and so on. We will analyze every table separately and will also look at the growth of the database as a whole. Two measures for size will be used:

- **Number of rows**: The number of rows in the table.
- **Disk size**: The disk size (in KB, MB or whatever is appropriate at the time).

Measuring will be done using the phpMyAdmin dashboard, which has direct access to the OLAP database. This dashboard displays a reliable estimation of all measures, while exact values can be calculated using some simple SQL queries.

6.1.2 Theoretical expectations of the Growth

Most of the tables are designed in such a way that, over time, their growth will decline. These tables are:

- Keywords
- Parameter
- Program_view
- Search
- Visit

Some of the table are of such nature that their growth will not decline, but are expected to grow linearly over time. These are:
• Date
• Fact

A detailed expectation on growth will be stated per table in the next sections.

Expectations on Date

It is a given that the size of the Date table will grow linearly, because every day exactly one new entry will be made in this table. However, since it is only one row per day, the impact on the overall size of the database will be minimal.

Expectations on Search

The Search table represents all unique searches, including which phrase is used, which parameters are entered etcetera. Since this table is dependent on the tables Parameter and Keywords, its growth is expected to be at least as large as these two. In worst case, the number of rows will be the number of Parameter rows times the number of Keywords rows, which is not even including the distinct regions, countries and disciplines searched. However, most of the searches will be very general, e.g. only searching for a discipline and a country. Therefore, the expectation is that the growth of searches will slowly decline over time.

Expectations on Keywords

There are several keyword phrases that are very popular and are used very often (for example “International Relations”). However, there will also be misspellings (“Intarnational Relations”), different combinations of phrases (“Relations Internationals”), searches in different languages (“Relaciones Internacionales”), or phrases that do not make any sense at all (“0344408567addisu”).

The expectation of this table is that in the first few months it will grow explosively, until all common phrases are in the database. After this, the table will continue to grow, as there is a theoretically infinite number of search phrases, but the growth rate will continue to shrink.

Expectations on Parameter

Parameters are extra options that can be attached to a search, e.g. searching for only B.Sc studies or for a maximal tuition fee of €5,000. A parameter exists of 6 options. Of these 6 options, there are three (degrees, alternatives and features) with a finite number of possibilities. It is also the case that most visitors only use a limited number of options, mostly just one or two. Therefore, the expectations is that the size of this table will stabilize after a quick growth in the first few months.

Expectations on Program view

Program views are views by visitors on a certain program id with a certain description. The program_ids are mostly static (i.e. a program will always have the same id) and not many programs will go away, although new programs are expected to be added over time. More interesting is that the description with the program can be changed by the universities. This means, every time
that a description has been changed a new entry will be entered in this table. Regardless, it is expected that the growth rate over time will be quite small.

**Expectations on Visit**

Entries in the `Visit` table are only distinct by country. Because the number of countries is variable, but limited (currently around 260 distinct country codes are known), it is expected that this table will not (or hardly) grow. Also, because of the limited data, it will take up very little disk size.

**Expectations on Fact**

This table is the center of the database design. It stores the actual facts, i.e. how many times something occurred in the system. Therefore, it is also the prime suspect of taking up much space in the database. Assuming that the number of visitors stays equal (which is not entirely realistic, see below), it can be expected that this table will grow linearly with time. Since it is dependent on all other tables, the amount of possible combinations is sheer endless. Without testing it is very hard to guess how large this table will become, but it is expected that its size will be very significant in the long run [4]. The expectation is that the size of this table will increase linearly.

An important fact in the grow of this table is the number of visitors on the website, as mentioned before. Figure 6.1 shows us (with the yellow line) that the number of visitors is somewhat affected by the season. Moreover, there seems to be an increase in the number of visitors since October 2011. Why this is the case is uncertain, but better publicity and attainability (the site is translated in English, German, French and Spanish) might be factors in this. These numbers were taken using Stone Steps Webalyzer [6] on February 3rd.

![Figure 6.1: The amount of unique visitors, yellow line. Green bars represent hits, dark blue bars represent visits, light blue bars represent pages, orange line represents hosts.](image)

**6.1.3 Execution of the Study**

The most challenging part of this test was to find a way to include the dimension tables (`Search`, `Visit`, essentially everything except `Fact`) in such a way that only the rows relevant for that time period are imported. With the current
import script, all leaf tables are imported in one go, regardless of at what time
they actually occur. The solution for this is to add an extra condition to the
query to only include the months needed. For example, on searches it would
become like in listing 6.1.

| INNER JOIN facts_searches AS f ON s.search_id = f.search_id |
| ... AND DATE(f.hour) >= '2011-06-01' |
| AND DATE(f.hour) < '2011-07-01' |

Listing 6.1: Importing only the searches done in June 2011

The result of this is that the import queries typically are somewhat slower,
because most of the time an extra join needs to be made.

A new problem with this approach is when using GROUP_CONCAT operations,
used in importing phrases and searches. Because an extra join is introduced,
an attribute can be included more than once. For example in phrases, when
“Management” has been used as a keyword in two months, the GROUP_CONCAT
result will become “Management Management”. For this analysis this would
not have been a problem if it were not for the fact that keywords are stored
multiple times (once per language) in the original database. The result is that
phrases that are normally identical while stored as a different language, could
be stored different and thus storing “duplicates”. The solution for this is to
use the DISTINCT clause within GROUP_CONCAT. This removes all duplicates in
GROUP_CONCAT. A quick test shows that this hardly has any impact on phrases
that use the same keyword more than once (13 out of 51,803).

An exception has been made to the table Parameter. There is no easy way to
import this without using multiple joins. Because it is expected that parameters
will not be significant in the growth and its data size is already small to begin
with (2.5 MB in the original database), it is decided not to take this table into
consideration in the overall analysis.

### 6.1.4 Results of the Study

Table 6.1 give the result of the experiment with the additional choices made in
section 3.3.3 and the setup described in section 6.1.1.

### 6.1.5 Analysis of the Results

As expected, the tables Date and Visit have a negligible growth in data size.
The growth in row numbers in Date is misleading, as every day one row is added
to the table, so the impact of this is much lower than at first glance. The Visit
table quickly becomes stable, as the number of possible countries is limited.

The growth of Search, both in rows as in data size, is more of interest.
As was expected, the growth is quite large at first and then slowly declined.
However, the total size of this table grows with about the same (absolute)
amount of data every month (∼3-4 MB per month). As of now, this is not a
very bad thing (with linear growth, in five years the table will be ∼200 MB
large, which is quite large but still acceptable). Another issue is the disk size;
<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>30</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>11,911</td>
<td>0.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>22,034</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>74,803</td>
<td>4.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>422,945</td>
<td>16.5 MB</td>
</tr>
<tr>
<td>Total</td>
<td>553,757</td>
<td>23 MB</td>
</tr>
</tbody>
</table>

(a) One Month

<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>61</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>20,077</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>23,639</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>131,517</td>
<td>7.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>828,286</td>
<td>32.5 MB</td>
</tr>
<tr>
<td>Total</td>
<td>1,027,219</td>
<td>43 MB</td>
</tr>
</tbody>
</table>

(b) Two Months

<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>92</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>27,368</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>24,199</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>187,517</td>
<td>10.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>1,260,392</td>
<td>49 MB</td>
</tr>
<tr>
<td>Total</td>
<td>1,523,767</td>
<td>62.5 MB</td>
</tr>
</tbody>
</table>

(c) Three Months

<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>122</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>34,809</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>25,045</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>247,111</td>
<td>13.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>1,734,702</td>
<td>67.6 MB</td>
</tr>
<tr>
<td>Total</td>
<td>2,066,834</td>
<td>85.1 MB</td>
</tr>
</tbody>
</table>

(d) Four Months

<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>153</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>42,962</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>26,192</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>308,675</td>
<td>17.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>2,283,883</td>
<td>89 MB</td>
</tr>
<tr>
<td>Total</td>
<td>2,688,057</td>
<td>111.5 MB</td>
</tr>
</tbody>
</table>

(e) Five Months

<table>
<thead>
<tr>
<th>Table</th>
<th>Row Size</th>
<th>Disk Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>184</td>
<td>16 KB</td>
</tr>
<tr>
<td>Keywords</td>
<td>43,268</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>Program_view</td>
<td>27,789</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>Search</td>
<td>349,752</td>
<td>19.5 MB</td>
</tr>
<tr>
<td>Visit</td>
<td>234</td>
<td>16 KB</td>
</tr>
<tr>
<td>Fact</td>
<td>2,876,673</td>
<td>112 MB</td>
</tr>
<tr>
<td>Total</td>
<td>3,325,454</td>
<td>136.5 MB</td>
</tr>
</tbody>
</table>

(f) Six Months

Table 6.1: Growth Results after each month
since the rows in the `Search` table contain several fields with strings, the size per row is also a significant contribution to the overall disk size.

The table `Program_view` grows very little, again as expected. The initial set of programs in the first month contains 22,000 programs, which slowly grows with 500 - 1,500 programs per month over time.

Figure 6.3 and 6.4 show the absolute growth in number of rows and disk size respectively. These figures give a good picture of how the tables will grow over time. As we see, the table `Fact` has the most impact on the growth of the database. After 6 full months, the `Fact` table takes up over 85% of the disk size.

Also in figure 6.4a we can see that the absolute disk growth is approximately 25 MB per month. Although slightly increasing, it is not expected that it will become very large (say 80 MB per month). Extrapolating from this, the yearly growth of the database comes to ~300 MB per year, or almost 3 GB over 10 years.

![Figure 6.2: The total size in disk size](image1.png)

![Figure 6.3: The absolute daily grow in number of rows](image2.png)

### 6.1.6 Conclusions

Most of the tables grow as expected. A first-month sprint has occurred as expected at the tables `Keywords`, `Visit` and `Search`. However, `Search` continued
to grow faster than expected, having almost linear growth. This is mainly caused by changes in the combination of Keywords, Parameter, and countries, regions and disciplines searched for. It is still expected that the growth of Search will go down. The reason this is not observed yet is because the time frame over which was measured (6 months) is apparently not large enough. This could not be changed however, since more data was not available at the moment.

Also, Fact grows mostly linear, as expected. We can even see a slight increase in growth. This is probably caused by an increase of visitors to the website. What is slightly more concerning is the fact that the Fact table takes up most of the database size. Since the growth of the Fact table is not expected to decrease, it will be difficult to come up with a solution to decrease the size of this table. Normalization might be a solution to reduce the overall size of the database, but in general this decreases query processing [8]. The Fact table itself however cannot be realistically further normalized. Currently it increases with 23 MB per month. Extrapolating this, the table will grow with 276 MB per year.

As noted before, the database as a whole grows with approximately 300 MB per year, of which most is coming from the Fact table. This growth is acceptable for a database with this many rows and amount of data stored. Therefore, we can conclude that the growth of the database is not an aspect of the system to worry about.

Note that there are some aspects that are not taken into account. One of them is indexes. It is possible that, if the system proves to be too slow, one or more indices need to be added to the database, which in itself takes up disk space. Depending on which table and column(s) a new index is set, the growth of the database will most likely increase.

Another aspect that has not been taken into account is aggregate tables. At the time of the research these were not taken into account because it was not clear yet how these would be implemented and which views would be materialized. Now that they are present, it is estimated that they will grow linearly (for the same reasons as why the Fact table grows linearly) with approximately 4.5 MB per month. Since this is only a fraction of the total growth per month, the aggregate tables will not have a significant impact on the size of the database.
6.2 Evaluation of the system

6.2.1 Benefit of aggregate tables

In section 4.2.2 we discussed the implementation of aggregate tables in our OLAP system. Here we will analyze if this implementation indeed has the desired effect. Figure 6.5 shows the running times of various views that make use of an aggregate table. These running times are taken using my personal laptop, which uses an Intel Core2Duo processor at 2.80GHz.

<table>
<thead>
<tr>
<th>View</th>
<th>Not Materialized</th>
<th>Materialized</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features, Month</td>
<td>23.9 seconds</td>
<td>6.4 seconds</td>
<td>3.73 x</td>
</tr>
<tr>
<td>Features, Continent</td>
<td>22.2 seconds</td>
<td>7.5 seconds</td>
<td>2.96 x</td>
</tr>
<tr>
<td>Alternatives, Month</td>
<td>23.4 seconds</td>
<td>8.4 seconds</td>
<td>2.78 x</td>
</tr>
<tr>
<td>Alternatives, Continent</td>
<td>21.5 seconds</td>
<td>13.7 seconds</td>
<td>1.57 x</td>
</tr>
<tr>
<td>Discipline, Month</td>
<td>23.8 seconds</td>
<td>4.7 seconds</td>
<td>5.06 x</td>
</tr>
<tr>
<td>Discipline, Continent</td>
<td>24.3 seconds</td>
<td>10.6 seconds</td>
<td>2.29 x</td>
</tr>
</tbody>
</table>

Table 6.2: Running times of some reports

Figure 6.5: Running times of materialized views, before and after aggregation

As was expected, the aggregate tables have a positive effect on the query time of some views. This benefit ranges from around 150% to more than 5 times the original query time. The differences can be explained by a fluctuating load of the database server.

6.2.2 Reports

When the system was implemented, some standard reports were defined. In Appendix F, several screenshots from these reports are displayed. Notable in most reports is that the chosen level of granularity on both axes contain many entities: be it months, countries or disciplines. So for some reports, only the top
20 or top 25 items are displayed, because the less popular items are probably of less interest and there is always an option to see these other items.

Figure 6.6 shows a part of the report in which the most popular disciplines searched for are held against the most visiting countries.

Figure 6.6: A report showing popular disciplines on the y-axis and countries on the x-axis.

Figure 6.7: A report showing popular programs on the y-axis and countries on the x-axis.

Figure 6.7 is a screenshot of a report that shows popular programs. The popularity of these programs is calculated from the number of times they were viewed by a visitor in the last six months. This is done using a Calculated Set as shown in listing 4.5 and aggregating over this set.

6.2.3 Running Times

Since the system makes use of a large (and still growing) dataset, an evaluation of the running time is appropriate. These times were taken when the system was deployed on the actual server, so these response times may be different from the response times in section 6.2.1. Table 6.3 gives us the running times of some reports.

As can be seen, the response time for some queries can be much longer than other queries, which is mainly dependent on the type of data that is required to execute the query. In general, the response times are short enough to be
 usable. Also, the data cache of Mondrian (not utilized here) can have a positive influence in the query response time.

### 6.3 Goal Analysis

In this section we will reflect on the goals that were stated in section 1.3. For each point we will discuss whether the goal has been met and what has been done to meet the goal.

**Goal 1: To determine the subset of data over which analysis should be done**

In section 3.1 the set of data over which the analysis should be done was determined. The selection was done on a logical basis of what should be included in an analytical system of users, in consultation with the StudyPortals team. The set was kept deliberately small, so no extraneous data is analyzed. Should the set be expanded in the future, then several aspects of the system need to be adapted, as described in section 6.4.

**Goal 2: To design a system that is able to analyze the data collected in the existing database, with respect to the subset**

In chapter 3 the overall design and design decisions were made. This design was specified for the subset of the data to be analyzed, but it is suitable for extension should the need be there. The design of the database was the most important part of the design process, the design of the OLAP system and the visualization were built onto this.
Goal 3: To implement this design, conforming to the development environment that is already present in the company

Chapter 4 describes how the design described in chapter 3 was implemented. Chapter 5 in particular describes how we initially load the database, how we keep the database up-to-date and which challenges were met while implementing these functions.

Goal 4: To create a mechanism that keeps the data that is analyzed up to date, if necessary

Because the decision was made to create and maintain a separate database for the system, it was indeed necessary to create a mechanism to keep this database up to date. In chapter 5 we described how this was achieved. In hindsight, more effort had to be put in this than was expected. The result however is an autonomous process that keeps updating the database with the relevant data, which should be easy to modify if other types of data need to be analyzed.

Goal 5: To design and create a way to visualize the analyzed data

The choice to use JPivot for the visualization of the data made this goal very easy. The only things that needed to be done for this were to define the exact MDX queries to be used, make sure that the right database was used and to create a portal-like index page that links to all reports.

An unmentioned goal of the system was also that these reports would become easy to define, even when the system as a whole was already up and running and the original developer is no longer involved. This is possible by recreating an HTML page with the structure like one of the existing pages, change the MDX query in the \texttt{<jp:mondrianQuery>} tag and add the link to the page to the index page.

6.4 Future Work

6.4.1 Expand Field of Analysis

It is conceivable that in the future, the field of analysis needs to be expanded. For example, the views on countries and universities might be interesting to analyze as well, as well as clicking on banners or submitting inquiries. To support this, changes need to be made on several levels:

- First of all, the database schema needs to be expanded. For every new field of analysis a new dimension needs to be created. This can range from a new table linking to the \texttt{Fact} table to several subdimensions integrated in an existing dimension.
- Following this, the import scripts need to be expanded as well.
- To be able to analyze the new fields of analysis, the Mondrian OLAP schema needs to be updated as well to incorporate the new dimensions and/or subdimensions. Also, possibly new Measures need to be defined.
- Finally, the web pages need to be updated or new pages need to be created to actually visualize the new information.
6.4.2 Enriching dimensions

The program_view dimension currently only has the program_id, label and description as attributes. This could be expanded by enriching this data using the programs database. This way, extra information on the programs, like location, university, duration and inclusion in special projects (like Erasmus Mundus) are available to be used in reports.

6.4.3 Clustering of search phrases

Currently, the search phrases extracted from the Statistics Database are not usable, due to a bug in the original statistics system. When this problem has been fixed, a clustering algorithm is a possibility to gain more insight in the search phrases. Clusters can be defined with the data that is currently available, while new data is classified during updates to the OLAP database.

6.4.4 Integration in the existing environment

The next step for this project in the future is integrating it in the existing working environment of StudyPortals. This means that the design of web pages displaying the results need to be overhauled and the main page needs to be placed somewhere on the StudyPortals working portal. Another challenge is that the webpages used by JPivot are made using Java Server Pages (JSP). The usage of these within the existing environment of PHP might be a challenge.

6.4.5 Service Oriented System

Mondrian has the ability to serve its functionality as a service to other systems, using XMLA [19]. XMLA (or XML for Analysis) is an industry standard used for data access in analytical systems. It uses SOAP methods to either send an execution command, which typically contains an MDX query, or a discovery command, which is used to discover properties of a particular schema, such as rowsets, properties and keywords.

This ability of Mondrian makes the possible integration in the existing working environment even more interesting, but it also means that the current configuration needs some adaption. First off, the Mondrian installation needs to be configured in such a way that it indeed can be used as an XMLA service. Second, when integrating the service into the existing environment, a framework or library that supports XMLA is necessary to be used. Since no PHP library that supports XMLA exists as of yet, this may be a bit of a challenge.
Appendix A

Website screenshots

Figure A.1: Screenshot of the search result page. The search was done with the parameters of figure 2.1. Also visible are some banner impressions (on the right).
Figure A.2: Example of a program view on the StudyPortals website. In this case, the program *ICT Innovation, Distributed Systems and Services (DSS)*—(M.Sc.), offered by the *TU Berlin*. 
Appendix B

Actions possible on the website

**Action**: The name of the action.
**Type**: the type of action that was taken.
**Subject**: the subject on which the action was taken (abbreviated).
**Label**: The label associated with this action. Meaning is context dependent.
**Location**: On what kind of page the action was done.

<table>
<thead>
<tr>
<th>Action</th>
<th>Type</th>
<th>Subject</th>
<th>Label</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program View</td>
<td>view</td>
<td>prog</td>
<td>grey, regular, premium, epic</td>
<td>NULL</td>
</tr>
<tr>
<td>University View</td>
<td>view</td>
<td>uni</td>
<td>regular, premium</td>
<td>NULL</td>
</tr>
<tr>
<td>Scholarship View</td>
<td>view</td>
<td>schol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Provider View</td>
<td>view</td>
<td>prvdr</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Discipline View</td>
<td>view</td>
<td>disc</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>File view</td>
<td>view</td>
<td>file</td>
<td>NULL</td>
<td>prog, uni, cntry, disc, rslts, srch, other</td>
</tr>
<tr>
<td>Country view page</td>
<td>view</td>
<td>cntry</td>
<td>regular, premium, popular</td>
<td>NULL</td>
</tr>
<tr>
<td>Banner Click</td>
<td>clic</td>
<td>bnnr</td>
<td>NULL</td>
<td>prog, uni, cntry, disc, rslts, srch, other</td>
</tr>
<tr>
<td>Non-revenue link click</td>
<td>clic</td>
<td>link</td>
<td>NULL</td>
<td>prog, uni, cntry, disc, rslts, srch, other</td>
</tr>
<tr>
<td>Revenue link click</td>
<td>clic</td>
<td>link</td>
<td>revenue</td>
<td>prog, uni, cntry, disc, rslts, srch, other</td>
</tr>
<tr>
<td>Program link click</td>
<td>clic</td>
<td>link</td>
<td>revenue, prog, rss</td>
<td>prog</td>
</tr>
<tr>
<td>University link click</td>
<td>clic</td>
<td>link</td>
<td>revenue, uni, rss</td>
<td>uni</td>
</tr>
<tr>
<td>Country link click</td>
<td>clic</td>
<td>link</td>
<td>revenue, cntry, rss</td>
<td>cntry</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Action</th>
<th>Type</th>
<th>Subject</th>
<th>Label</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>University Spotlight click</td>
<td>clic</td>
<td>uni</td>
<td>regular, premium</td>
<td>spot</td>
</tr>
<tr>
<td>Inquiry Submit</td>
<td>subm</td>
<td>lead</td>
<td>NULL</td>
<td>prog</td>
</tr>
<tr>
<td>Question Submit</td>
<td>subm</td>
<td>cntct</td>
<td>NULL</td>
<td>prog, uni</td>
</tr>
<tr>
<td>Newsletter Subscription</td>
<td>subm</td>
<td>news</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Program Impression in related programs</td>
<td>impr</td>
<td>prog</td>
<td>grey, regular, premium, epic</td>
<td>prog</td>
</tr>
<tr>
<td>Program Impression in search results</td>
<td>impr</td>
<td>prog</td>
<td>grey, regular, premium, epic</td>
<td>rslts</td>
</tr>
<tr>
<td>Program Impression in landing-page</td>
<td>impr</td>
<td>prog</td>
<td>grey, regular, premium, epic</td>
<td>lndpg</td>
</tr>
<tr>
<td>University Impression on nearby universities</td>
<td>impr</td>
<td>uni</td>
<td>regular, premium</td>
<td>uni</td>
</tr>
<tr>
<td>University Spotlight Impression</td>
<td>impr</td>
<td>uni</td>
<td>regular, premium</td>
<td>spot</td>
</tr>
<tr>
<td>Banner Impression</td>
<td>impr</td>
<td>bnnr</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Basic Search</td>
<td>bsc_s</td>
<td>srch</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Advanced Search</td>
<td>adv_s</td>
<td>srch</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Refine Search</td>
<td>ref_s</td>
<td>srch</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Quick Search</td>
<td>qck_s</td>
<td>srch</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Table B.1: Actions stored in the Statistics database
Appendix C

Database Schemas

Figure C.1: The definitive schema of the OLAP database. Tables in yellow are aggregate tables.
Figure C.2: Schema of the Statistics database
Appendix D

Statistics Database Content Examples

Figure D.1: Example of data on stems in the Statistics Database. The search string that was entered was “computer management”.

STEMS

<table>
<thead>
<tr>
<th>stem_id</th>
<th>stem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>comput</td>
</tr>
<tr>
<td>2</td>
<td>manag</td>
</tr>
</tbody>
</table>

SEARCH_STEMS

<table>
<thead>
<tr>
<th>search_id</th>
<th>stem_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

SEARCH

<table>
<thead>
<tr>
<th>search_id</th>
<th>parameter_id</th>
<th>fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>###</td>
</tr>
</tbody>
</table>

SEARCH_KEYWORDS

<table>
<thead>
<tr>
<th>search_id</th>
<th>keyword_id</th>
<th>order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

KEYWORDS

<table>
<thead>
<tr>
<th>keyword_id</th>
<th>stem_id</th>
<th>keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>computer</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>management</td>
</tr>
</tbody>
</table>
Figure D.2: In this example, the search string that was entered was again “computer management”. Both keywords are stemmed different than in the previous example. This is because in the previous example, the website language was English, while in this example the website language is German.
Appendix E

OLAP Database Content Examples

![Database Diagram]

Figure E.1: Database content after a search. In this search, a visitor from *The Netherlands* is searching for a *Computer Science & IT* program for a *msc* degree in *Germany*, which is *fulltime* and takes at most *24 months*. The search was done on *March 1st, 2012*. This database is the result of the search done as seen on screenshot 2.1.
Figure E.2: Database content after another search. This search was done by a visitor from India, searching for a Human Medicine program with the phrase surgery. The search was done on March 1st, 2012. Rows in green are new rows.

Figure E.3: Database content after viewing a program. A visitor from The Netherlands has viewed a program with program_id 5, description Marketing and label regular. The program was viewed on March 2nd, 2012. Rows in green are new rows.
Figure E.4: Database content after viewing a program again, by someone from the same country on the same day. Rows in yellow are “altered”. Note however that the rows are not actually altered in the database itself, since facts are on a daily basis. Instead, this row is only altered during the execution of the Periodic import script.
Appendix F

Mondrian Screenshots
Searches on Countries

The country that was searched for, calculated against the country of origin.

<table>
<thead>
<tr>
<th>Country</th>
<th>All Origins</th>
<th>null</th>
<th>Africa</th>
<th>Asia</th>
<th>Europe</th>
<th>North America</th>
<th>Oceania</th>
<th>South America</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Countries</td>
<td>1,629,450</td>
<td>59,703</td>
<td>123,560</td>
<td>426,277</td>
<td>822,726</td>
<td>143,489</td>
<td>15,137</td>
<td>38,558</td>
</tr>
<tr>
<td>null</td>
<td>1,042,004</td>
<td>37,712</td>
<td>83,714</td>
<td>263,299</td>
<td>526,443</td>
<td>95,646</td>
<td>10,252</td>
<td>25,678</td>
</tr>
<tr>
<td>Germany</td>
<td>127,636</td>
<td>5,199</td>
<td>7,984</td>
<td>47,820</td>
<td>93,411</td>
<td>9,273</td>
<td>1,073</td>
<td>2,897</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>76,467</td>
<td>2,599</td>
<td>3,911</td>
<td>16,955</td>
<td>44,287</td>
<td>6,399</td>
<td>569</td>
<td>1,687</td>
</tr>
<tr>
<td>Netherlands</td>
<td>56,928</td>
<td>2,221</td>
<td>2,883</td>
<td>11,038</td>
<td>35,099</td>
<td>4,081</td>
<td>535</td>
<td>1,071</td>
</tr>
<tr>
<td>France</td>
<td>44,246</td>
<td>1,529</td>
<td>2,968</td>
<td>12,315</td>
<td>21,095</td>
<td>4,620</td>
<td>468</td>
<td>1,252</td>
</tr>
<tr>
<td>Spain</td>
<td>31,888</td>
<td>946</td>
<td>1,107</td>
<td>6,708</td>
<td>17,627</td>
<td>3,902</td>
<td>375</td>
<td>1,333</td>
</tr>
<tr>
<td>Sweden</td>
<td>30,501</td>
<td>1,150</td>
<td>1,669</td>
<td>6,570</td>
<td>18,383</td>
<td>2,104</td>
<td>310</td>
<td>475</td>
</tr>
<tr>
<td>Belgium</td>
<td>28,134</td>
<td>949</td>
<td>3,517</td>
<td>6,244</td>
<td>14,748</td>
<td>1,906</td>
<td>154</td>
<td>556</td>
</tr>
<tr>
<td>Austria</td>
<td>27,175</td>
<td>977</td>
<td>1,864</td>
<td>7,652</td>
<td>14,182</td>
<td>1,963</td>
<td>167</td>
<td>361</td>
</tr>
<tr>
<td>Italy</td>
<td>26,494</td>
<td>887</td>
<td>1,185</td>
<td>7,177</td>
<td>11,800</td>
<td>2,528</td>
<td>180</td>
<td>677</td>
</tr>
<tr>
<td>Switzerland</td>
<td>24,367</td>
<td>776</td>
<td>1,948</td>
<td>6,652</td>
<td>12,631</td>
<td>2,236</td>
<td>224</td>
<td>500</td>
</tr>
<tr>
<td>Denmark</td>
<td>20,421</td>
<td>801</td>
<td>1,272</td>
<td>4,099</td>
<td>12,426</td>
<td>1,247</td>
<td>217</td>
<td>355</td>
</tr>
<tr>
<td>Finland</td>
<td>18,455</td>
<td>915</td>
<td>2,134</td>
<td>6,669</td>
<td>7,207</td>
<td>1,186</td>
<td>135</td>
<td>248</td>
</tr>
<tr>
<td>Norway</td>
<td>16,548</td>
<td>805</td>
<td>1,796</td>
<td>5,788</td>
<td>6,536</td>
<td>1,247</td>
<td>153</td>
<td>219</td>
</tr>
<tr>
<td>Ireland</td>
<td>10,529</td>
<td>360</td>
<td>681</td>
<td>2,466</td>
<td>5,711</td>
<td>1,033</td>
<td>89</td>
<td>189</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>6,746</td>
<td>282</td>
<td>519</td>
<td>2,084</td>
<td>3,170</td>
<td>535</td>
<td>54</td>
<td>122</td>
</tr>
<tr>
<td>Poland</td>
<td>5,443</td>
<td>241</td>
<td>573</td>
<td>1,784</td>
<td>2,300</td>
<td>466</td>
<td>33</td>
<td>46</td>
</tr>
<tr>
<td>Portugal</td>
<td>5,398</td>
<td>182</td>
<td>206</td>
<td>1,128</td>
<td>3,093</td>
<td>434</td>
<td>39</td>
<td>284</td>
</tr>
<tr>
<td>Turkey</td>
<td>3,411</td>
<td>191</td>
<td>354</td>
<td>2,051</td>
<td>1,480</td>
<td>289</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>3,572</td>
<td>131</td>
<td>534</td>
<td>848</td>
<td>1,760</td>
<td>228</td>
<td>13</td>
<td>56</td>
</tr>
<tr>
<td>Hungary</td>
<td>3,517</td>
<td>139</td>
<td>309</td>
<td>1,115</td>
<td>1,430</td>
<td>295</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>Greece</td>
<td>3,479</td>
<td>209</td>
<td>184</td>
<td>618</td>
<td>2,205</td>
<td>216</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>Russia</td>
<td>2,654</td>
<td>78</td>
<td>167</td>
<td>879</td>
<td>1,210</td>
<td>256</td>
<td>14</td>
<td>50</td>
</tr>
<tr>
<td>Cyprus</td>
<td>2,233</td>
<td>125</td>
<td>323</td>
<td>673</td>
<td>946</td>
<td>133</td>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>Iceland</td>
<td>1,848</td>
<td>94</td>
<td>180</td>
<td>515</td>
<td>868</td>
<td>153</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>Estonia</td>
<td>1,867</td>
<td>88</td>
<td>169</td>
<td>563</td>
<td>808</td>
<td>144</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>Romania</td>
<td>1,752</td>
<td>68</td>
<td>219</td>
<td>503</td>
<td>846</td>
<td>136</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Slovenia</td>
<td>1,621</td>
<td>68</td>
<td>40</td>
<td>347</td>
<td>881</td>
<td>110</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Ukraine</td>
<td>1,360</td>
<td>60</td>
<td>352</td>
<td>433</td>
<td>337</td>
<td>107</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Lithuania</td>
<td>1,000</td>
<td>36</td>
<td>114</td>
<td>411</td>
<td>622</td>
<td>85</td>
<td>9</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure F.1: Screenshot of the number of searches done on a host country, plotted against the continent of the visitor. Clicking the + and – symbols on the axes expands/contracts the members, drilling down/rolling up in the cube.
## Disciplines over Continents

The total percentage of sub-disciplines are not counted toward the main disciplines as well. This is because there is no distinction made between these two in the database. When you want to calculate the totals for the main disciplines, just add the results from all of the sub-disciplines to the result of the main disciplines.

Colors indicate outliers: green means that the percentage is at least 50% higher than average over all countries, red means that the percentage is at least 50% lower than average over all countries.

![Table of Discipline Search Percentage per Country](Figure_F_2_Table.png)

### Figure F.2: Screenshot of discipline search percentage per country. Red cells are percentages that are at least 50% lower than average, green cells are percentages that are at least 50% higher than average.)
Bibliography


