Improving the ETL process and maintenance of Higher Education Information System Data Warehouse

Igor Mekterović, Ljiljana Brkić, Mirta Baranović

Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, HR-10000 Zagreb
CROATIA
{Igor.Mekterovic, Ljiljana.Brkic, Mirta.Baranovic}@fer.hr

Abstract: HEIS (Higher Education Information System) is a project funded by the Croatian Ministry of Science, Education and Sports started in the year 2001. HEIS is a comprehensive information system that provides support for education related processes taking place within a higher education institution. As a part of the project, a data warehouse was developed to provide reporting and analytical features. This paper presents the HEIS data warehouse architecture, comments on the data model and addresses issues (and our solutions) that arose during the seven-year development and maintenance period. Foremost, we address improvements in the ETL process and maintenance process.

Key-Words: Data warehouse, Higher education information system, ETL, Maintenance, Dimensional model.

1 Introduction

A data warehouse is a repository of integrated information, available for querying and analysis [1]. The basic idea is to extract relevant information (usually from relational databases), transform it, and load into the data warehouse where it is structured in a way that facilitates querying and analysis.

While the business world recognized benefits of data warehousing for decision making, it is not used extensively in higher education neither in Croatia nor in the rest of the higher education community.

Higher education has a lower level of acceptance and a lower rate of adoption of data warehousing for decision making. Although some efforts in implementing data warehouse and data mining in higher education institutions have been noted [10][11], in many institutions it is still in the emerging or developing stage [4][6].

Guan, Nunez, and Welsh [5] stated that only a fraction of colleges' and universities' data are captured, processed and stored in their information system. Therefore academic deans and rectors often complain about the lack of valid and reliable information about their finances, personnel and students.

Very little research has been reported dealing with the data warehousing in higher education especially about successfully implementing data warehousing in this environment and about the benefits that they bring to such institutions.

Although the development of the data warehouse system for Croatian higher education institutions [2] has started 7 years ago (within the Higher Education Information System project), it is still used for operational activities rather than for strategic purposes.

In the last few years, with the advent of the Bologna Process which aims to create the European Higher Education Area [8], series of reforms have been in progress in Croatian higher education. Legislation harmonization in the higher education area in Croatia should have preceded these rapid changes, but that didn't happen. Even when the new legislation came into effect it was inconsistently interpreted i.e. according to foregoing practice. This often resulted in different practices in higher education institutions and caused difficulties in supporting such inconsistencies in HEIS.

Furthermore, HEIS is a highly complex system with numerous business processes and business rules. The fact that HEIS DW is used for operational purposes on a daily basis and provides a means for attaining various influential reports (e.g. students' scholarships, study fees, etc) speak to its importance.

Even though there is only one data source (i.e. the relational database of the Higher Education Information System) with fairly clean data [2] data quality problem is not eliminated. Data quality is one of the biggest issues within ETL (Extract, transform and load) process which is the core

process [7] in data warehousing. It is a process that extracts data from given sources, transforms it, and loads transformed data into destination data structures.

Seemingly processes taking place at the HE institutions (enrolment of freshmen and sophomores, scholarship payment, examinations, graduation, …) that cause changes in the source data are few and are occurring only several times a year (thus simplifying ETL process) but that is hardly true. For instance, Fig 1 shows the number of records in the fExam fact table from 26.3.2009 until 29.7.2009. On average, 5295.93 records are added every day. Having in mind that these records provide a foundation for determining, for instance - who will be granted a scholarship or not, it is understandable why it is of the great importance for the data in the HEIS DW to be up-to-date.

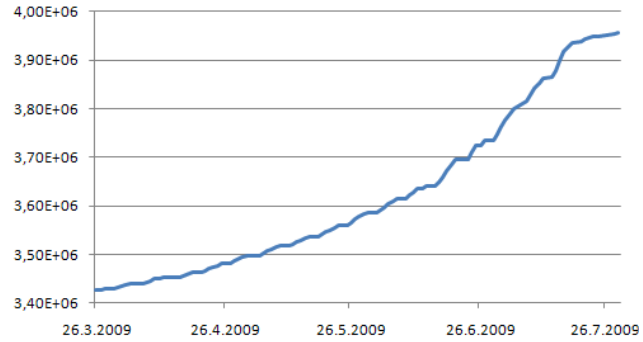

Fig. 1 Number of records in the fExam fact table from 26.3.2009 until 29.7.2009.

In this paper we emphasize on the ETL process. We propose and formally define ETL methodology that was developed and implemented under the HEIS DW project. Further on, we describe the HEIS architecture and comment on the data model. Due to financial constraints we weren't in position to use existing commercial BI data analysis and presentational tools but were forced to develop our own. Within our developed framework (web application) we've applied data warehousing methodology to track users' behavior and thus facilitate maintenance and increase the quality of service. Chapter 5 comments on that and brings few maintenance use cases.

2 HEIS architecture

Fig. 2 depicts the simplified architecture of the HEIS.

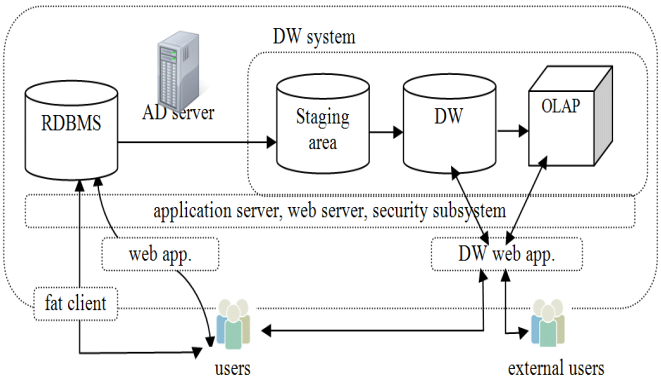

Fig. 2 Simplified HEIS architecture

Users access relational database via fat client application (e.g. registrar's office, teachers) and web application (e.g. students, teachers). Apart from regular users, the ones with access rights to the transactional system, HEIS provides data warehouse access to a small group of external users (intended for university rectors, minister of science and his staff).

The system is governed by a single active directory server (accessed using LDAP protocol by the non-windows servers). Every night, data is copied from the relational system to the staging area where it is transformed and propagated to the data warehouse and OLAP server. Users access the data warehouse only through the data warehouse web application. This was the obvious solution due to the physical and administrative dislocation of our clients (the user can be any lecturer or administrative employee of a higher education institution anywhere in Croatia using arbitrary operating system). Using their web browser users can query data via graphical interface. Data is interfaced through three categories of queries:

- predefined queries
- detailed ad hoc queries
- summary ad hoc queries

Predefined query is a set of hard-coded queries (SQL or MDX) stored in the database that cover a certain topic. For instance, one such query (set of queries) covers enrollment process. Of course, every user is able to retrieve only the subset of data that corresponds to his or hers permissions. This option is particularly useful for standard reports requested by the ministry of science or the university.

Detailed ad hoc queries are parameterized SQL queries that yield detailed listings. For instance, these are used to get the list of students ordered by some excellence criteria to be used when granting scholarships. A user can choose from various different areas of interest (usually corresponding to the underlying fact tables) and then parameterize and execute queries. These queries have proven very useful, especially for the administrative staff that seeks to find various lists of students, courses etc.
Finally, summary ad hoc queries are queries against OLAP system, that allow for custom, user-generated queries. Using a graphical drag and drop interface, users are able to tailor their own queries and analyze data using standard OLAP functionalities like drill-up, drill-down, cross joins, etc.

3 Data model

The dimensional model is a logical design technique that seeks to make data available to the user in an intuitive framework that is intended to facilitate querying [3]. The dimensional model is composed of fact and dimensional tables where fact tables are normalized tables that represent the very process being tracked. Dimension tables represent parties involved in that process. In business areas like banking or retail the tracked processes are those with easily established measures (e.g. units ordered or sold, money spent, etc.) and typical dimensions are Customer, Product, etc. In educational institutions, apart from the measurable processes, there is a fair share of factless processes, i.e. event tracking processes (e.g. a courses being attended, exams being applied for, etc.).
HEIS DW employs dimensional model and consists of 15 fact tables and 61 dimension tables (Table 1). 430 users use HEIS actively while 5613 users have accounts, but have not yet used DW. HEIS dimension and fact table statistics is shown in Tables 2 and 3.

Table 1 HEIS table and users statistics

Object	*count*
Fact tables	15
Dimension tables	61
Users	430-5613*

**5613 users have accounts, but have not yet used DW application*

Table 2 HEIS dimension table statistics

Dimension table	*Row count*	*Col. count*
dStudent	338,881	61
mdDemogrStudent	119,590	31
dCourseComponent	45,070	20
dCourse	40,806	15
dExamAppliedAfter	32,768	3
mdStudentParent	28,953	19
dResultCount	20,001	2
dStaff	15,968	22
dTime	12,694	12
dWebAppUser	5,613	6
dAdmtExamInterval	3,002	7
mdStaffAcademic	2,061	15
dOrgUnit	2,003	10
dCourseOfStudy	1,931	28
dTimeByDay	1,443	6
(…) 46 more		

Fact tables cover many of the processes taking place in a higher education institution, from the admittance exams, yearly (course, course of study) enrollments, exam attempts and graduation process. However, these fact tables cover only a portion of relational system and new fact tables and dimensions are being developed continuously, as the demand for them emerges.

Table 3 HEIS fact table statistics

Fact table	*Row count (*1M)*	*Dim. count*	*Measure count*
fCourseEnrollment	4,573	18	14
fExamCourseOfStudy	3,489	27	11
fSatisfiedCourses	3,448	12	10
fExam	3,432	24	9
fExamApplication	2,104	23	8
fYearEnrollment	0,417	23	19
fExamPassedInAcdmYear	0,311	7	5
fSumStudentCourseOfStudy	0,200	35	3
fScholarship	0,155	24	11
fStudentCourseOfStudy	0,136	16	3
fAdmittanceExam	0,075	20	14
fQueryLog *(last month data)*	0,034	8	5
fCourseOfStudyCompletion	0,012	21	6
fHEITransfer	0,007	8	5
fSpecialStatus	0,005	4	4

4 ETL process

ETL process is the most time-consuming part of building a data warehouse. Data has to be collected, usually from various sources, integrated and transformed prior to loading it into the data warehouse. In our case, the situation is somewhat simplified, since there is only one, high-quality data

source (HEIS relational database). On the other hand, HEIS is also a very complex data source in terms of business rules and data relationships. One has to keep in mind that HEIS encompasses roughly one hundred different higher education institutions having different (to some extent) curriculums, business rules and so forth. For instance, some higher education institutions follow a "hard-coded" curriculum, while others employ a prerequisite model where students have much more freedom in crafting their own study, as they pick courses from a wider set of courses, as long as they've passed courses defined as prerequisites. Even though data integrity is checked in both application and data layer, inconsistent data still gets into the database. With a large number of users (several hundred active users, not counting more than 100k active students) erroneous data is entered on the daily basis (e.g. exam dates in the distant future). Sometimes the ever-changing business rules are initially partially implemented and thus can allow insertion of bad data. HEIS is also an ever expanding system. Not only do the business rules evolve (i.e. introduction of Bologna process), but also new institutions join in. Sometimes, though rarely, institutions that are joining HEIS load data from their legacy information systems, bypassing application layer. Such actions facilitate errors and inconsistencies in the data.

On the other hand, it is required that HEIS DW contains complete and accurate data. In other applications it is unlikely that one erroneous row (e.g. phone call in a telecom DW) will make a difference, but in HEIS DW one row sometimes can make a difference. HEIS DW is used for operational purposes on a daily basis. For example, HEIS DW is used to generate scholarship listings and one missing row can make a difference between paying for education and free education. Thus, it is required that every fact table has an error threshold (e.g. 0). This enables every institution to define how many erroneous records per fact table they are willing to tolerate. If the threshold is exceeded, users are provided with the last valid data snapshot. Furthermore, fact tables contain data for various institutions and one institution's data must not influence the other. Otherwise, erroneous data from one institution could prevent other institutions' users from having an up-to-date version of the data, since all records are stored within a single fact table. Therefore, fact table data has to be additionally horizontally segmented. This way, different institutions (can) have different data versions: one can have an up-to-date view of the data, while other can have an older version of the data, both residing in the same fact table. With these problems in mind, we have set to develop a robust ETL procedure that will handle the aforementioned challenges.

In this chapter we present a formal definition of our data warehouse objects and define the ETL process.

Definition

The set of fact tables is denoted $\mathcal{F}$, $|\mathcal{F}| = n$, $\mathcal{F} = \{F_1, ..., F_n\}$ where each F_i is defined as follows:

$F_i = <Fid_i, Fname_i, FPK_i, \mathcal{FFK}_i, \mathcal{FI}_i, \mathcal{FS}_i, \mathcal{FR}_i, Fload_i>$; these elements being respectively unique identifier, fact table name, primary key constraint, a set of foreign key constraints, a set of indexes, set of attributes (from table schema), a rowset (set of tuples) and an ETL function that loads data into this fact table.

The set of dimension tables is denoted $\mathcal{D}$, $|\mathcal{D}| = m$, $\mathcal{D} = \{D_1, ..., D_m\}$ where each D_i is defined as follows:

$D_i = <Did_i, Dname_i, DPK_i, \mathcal{DFK}_i, \mathcal{DI}_i, \mathcal{DS}_i, \mathcal{DR}_i, Dload_i>$ these attributes being respectively unique identifier, dimension table name, primary key constraint, a set of foreign key constraints, a set of indexes, set of attributes, a rowset (set of tuples) and an ETL function that loads data into this dimension.

Further on we introduce concept of prime tables (dimensional and fact ones). Prime tables are introduced to provide a behind the scene loading and testing platform that, after the constraints are checked, can be quickly batch copied into the production tables.

Based on that, an analogous set (denoted with prime) of fact and dimension tables is defined:

$\mathcal{F}'$, $\mathcal{F}' = \{F'_1, ..., F'_n\}$ where each F'_i is defined as follows:

$F'_i = <Fid'_i, Fname'_i, FPK'_i, \mathcal{FFK}'_i, \mathcal{F}\mathcal{I}'_i, \mathcal{FS}'_i, \mathcal{FR}'_i, Fload'_i>$. Elements are analogous to previous definition but with following constraints:

- $F'name_i = '\_' + Fname_i$
- FPK'_i is analogous to FPK_i (composed of the same attributes)
- $\mathcal{FFK}'_i$ is analogous to $\mathcal{FFK}_i$ (same number of keys, composed of the same attributes)
- $\mathcal{FI}'_i$ is analogous to $\mathcal{FI}_i$ (same number of indexes, composed of the same attributes)
- $\mathcal{FS}'_i = \mathcal{FS}_i$
- $Fload'_i$ is analogous to $Fload_i$

Also: $\mathcal{D}'$, $\mathcal{D}' = \{D'_1, ..., D'_m\}$ where each D'_i is defined as follows:

D_i' = <Did'$_i$, Dname'$_i$, DPK'$_i$, $\mathcal{DFK}'_i$, $\mathcal{DJ}'_i$, $\mathcal{DS}'_i$, $\mathcal{DR}'_i$, Dload'$_i$ > with elements analogous to the previous definitions and following constraints:

- Dname' i = '_' + Dname i
- DPK'i is analogous to DPKi (composed of the same attributes)
- $\mathcal{DFK}$'i is analogous to $\mathcal{DFK}$i (same number of keys, composed of the same attributes)
- $\mathcal{DJ}$ 'i is analogous to $\mathcal{DJ}$i (same number of indexes, composed of the same attributes)
- $\mathcal{DS}$ 'i = $\mathcal{DS}$i
- Dload'i is analogous to Dloadi

In summary, these prime tables form a parallel set of tables to the production tables having the same structure and analogous constraints; e.g. if a fact table F(a, b, c) references dimension table D(b, d, e) via attribute b, then F'(a, b, c) references D'(b, d, e) via attribute b.

Based on the set $\mathcal{F}$, we define a set of error tables $\mathcal{E}$, $\mathcal{E}$ = {E_1, ..., E_n} where each E_i is defined as follows: E_i = <Eid$_i$, Ename$_i$, $\mathcal{ES}_i$, $\mathcal{ER}_i$, $\mathcal{ET}_i$> ; these elements being respectively unique identifier, error table name, set of attributes (from table schema), rowset (set of tuples) and a set of constraints that define horizontal segmentation of the fact tables: $\mathcal{ET}_i$ = { ET_1 , …, ET_k } where each ET_i is defined as follows ET_i = <EF_i, Et_i> with elements being filter statement and error threshold respectively. These are used to assess whether a horizontal segment is valid or not. i.e. whether it will be used to update the production segment.

Also, following rules apply:

- Ename i = 'err' + Fname i
- $\mathcal{ES}$i = $\mathcal{FS}$i U {timeStamp, comment}

Excluding all columns from corresponding fact tables, error tables contain timestamp column and description column indicating which constraints were not satisfied.

Analogously, a set of dimension error tables is defined: $\mathcal{ED}$ = {ED_1, ..., ED_m} where each ED_i is defined as ED_i = <EDid$_i$, EDname$_i$, $\mathcal{EDS}_i$, $\mathcal{EDR}_i$, $\mathcal{EDT}_i$ > with:

- EDname i = 'err' + Dname i
- $\mathcal{EDS}$i = $\mathcal{DS}$i U {timeStamp, comment}

For instance, if an institution has two erroneous records having e.g. date of exam set ten years in the future, these two rows will be detected and moved from the prime table into the according error table.

Finally, we define three agents :

***Changer* ($\mathcal{M}$)** whose argument is domain or working set.

Changer is able to drop or create keys or indexes for a given domain. His role is only to speed up data loading - before loading data into (any kind of) tables, foreign keys, indexes and primary keys are dropped. Afterwards, keys and indexes are created again. It is considerably easier to develop and maintain ETL process when this process is performed programmatically than manually.

```
Changer (M) {
 if      (M == D) {
           J :=U_{i=1}^{m} DJ_i ;
           FK:=U_{i=1}^{m} DFK_i ;
           PK:=U_{i=1}^{m} DPK_i;}
 else if (M == D') {
           J :=U_{i=1}^{m} DJ'_i;
           FK:=U_{i=1}^{m} DFK'_i ;
           PK:=U_{i=1}^{m} DPK'_i;}
 else if (M == F) {
           J :=U_{i=1}^{n} FJ_i ;
           FK:=U_{i=1}^{n} FFK_i ;
           PK:=U_{i=1}^{n} FPK_i;}
 else if (M == F') {
           J :=U_{i=1}^{n} FJ'_i;
           FK:=U_{i=1}^{n} FFK'_i ;
           PK:=U_{i=1}^{n} FPK'_i;}
 dropAll () {
     ∀ idx_i ∈ J   drop idx_i;
     ∀ fk_i ∈ FK  drop fk_i;
     ∀ pk_i ∈ PK  drop pk_i; s
   }
 raiseAll () {
      ∀ idx_i ∈ J   raise idx_i;
      ∀ fk_i ∈ FK  raise fk_i;
      ∀ pk_i ∈ PK  raise pk_i;
   }
}
```

***Cleaner* ($\mathcal{PK}$, $\mathcal{FK}$, $\mathcal{J}$, $\mathcal{R}$, $\mathcal{ER}$)** with arguments being respectively: set of primary key constraints, set of foreign key constraints, set of indexes, rowset and error rowset. Using given arguments *Cleaner* moves tuples from $\mathcal{R}$ that do not satisfy given constraints to $\mathcal{ER}$ and attaches timestamp and comment to each invalid tuple.

$Cleaner.clean(\mathcal{PK},\ \mathcal{FK},\ \mathcal{J},\ \mathcal{R},\ \mathcal{ER})$ {
  $\forall\ pk_i \in \mathcal{PK}'$
    $errRowSet := tupleSet \subseteq \mathcal{R} \mid \forall\ tuple \in tupleSet:\ tuple\ doesn't\ satisfy\ pk_i$
    $\mathcal{R} := \mathcal{R} \setminus errRowSet$
    $\mathcal{ER} := \mathcal{ER} \cup errRowSet$
  $\forall\ idx_i \in \{\mathcal{J} \mid idx_i\ is\ unique\}$}
    $errRowSet := tupleSet \subseteq \mathcal{R} \mid \forall\ tuple \in tupleSet:\ tuple\ doesn't\ satisfy\ idx_i$
    $\mathcal{R} := \mathcal{R} \setminus errRowSet$
    $\mathcal{ER} := \mathcal{ER} \cup errRowSet$
  $\forall\ fk_i \in \mathcal{FK}$
    $errRowSet := tupleSet \subseteq \mathcal{R} \mid \forall\ tuple \in tupleSet:\ tuple\ doesn't\ satisfy\ fk_i$
    $\mathcal{R} := \mathcal{R} \setminus errRowSet$
    $\mathcal{ER} := \mathcal{ER} \cup errRowSet$
}

Loader prepares the tuples from source rowset so that they satisfy given threshold for each tuple set that is defined via filter statements. Once source data is prepared to adhere to the given constraints it can be loaded quickly into the production tables.

$Loader$ (){
$prepareAllDimensions$(){
$\forall\ i \in \{1..\mathrm{m}\}$
    $prepare(\mathcal{DR}'_i,\ \mathcal{DR}_i,\ \mathcal{DER}_i,\ \mathcal{DET}_i)$;
}
$prepareAllFactTables$() {
    $\forall\ i \in \{1..\mathrm{n}\}$
    $prepare(\mathcal{FR}'_i,\ \mathcal{FR}_i,\ \mathcal{ER}_i,\ \mathcal{ET}_i)$;
}
  $prepare(\mathcal{R}',\ \mathcal{R},\ \mathcal{ER},\ \mathcal{ET})$ {
$\forall\ \mathrm{ET}_i \in \mathcal{ET}$
$if\ (|\mathrm{EF}_i(\mathcal{ER})| \geq Et_i)$ {
$\mathcal{R}' := \mathcal{R}' \setminus \mathrm{EF}_i(\mathcal{R}')$;
}
  }
  $loadAll()$ {
    $\forall\ i \in \{1..\mathrm{m}\}\ merge(\mathcal{DR},\ \mathcal{DR}')$;
    $\forall\ i \in \{1..\mathrm{n}\}\ merge(\mathcal{FR},\ \mathcal{FR}')$;
  }
}

When implemented, agents rely on a set of metadata tables describing the aforementioned structures. Metadata tables are mostly programmatically populated using information extracted from the system tables, but some parts are manually populated (e.g. which tables are dimensions and which are fact tables). That way, we could make an easy upgrade from the existing system by programmatically populating metadata tables. When developing a new fact table, the required number of changes in the ETL process is significantly reduced. Instead of writing a number of alter table statements, it is only required to populate metadata tables using existing procedures.
Finally, we propose (and implement) the subsequent ETL process:

$heisETL()$ {
$Changer\ \mathrm{CD}' = new\ Changer(\mathcal{D}')$
$\mathrm{CD}'.dropAll()$
$\forall\ i \in \{1..\mathrm{m}\}\ \mathcal{DR}'_i = \emptyset$
$\forall\ i \in \{1..\mathrm{m}\}\ execute\ \mathrm{Dload}'_i$
$\forall\ i \in \{1..\mathrm{m}\}\ Cleaner.clean$
  $(\mathcal{DPK}'_i, \mathcal{DFK}'_i, \mathcal{DJ}'_i, \mathcal{DR}'_i, \mathcal{EDR}_i)$
$\mathrm{CD}'.raiseAll()$
$Changer\ CF' = new\ Changer(\mathcal{F}')$
$CF'.dropAll()$
$\forall\ i \in \{1..\mathrm{n}\}\ \mathcal{FR}'_i = \emptyset$
$\forall\ i \in \{1..\mathrm{n}\}\ execute\ \mathrm{Fload}'_i$
$\forall\ i \in \{1..\mathrm{n}\}\ Cleaner.clean$
  $(\mathcal{FPK}'_i, \mathcal{FFK}'_i, \mathcal{FJ}'_i, \mathcal{FR}'_i, \mathcal{ER}_i)$
$CF'.raiseAll()$
$Loader.prepareAllDimensions()$
$Loader.prepareAllFactTables()$
$Changer\ \mathrm{CD} = new\ Changer(\mathcal{D})$
$Changer\ \mathrm{CF} = new\ Changer(\mathcal{F})$
$\mathrm{CF}.dropAll()$
$\mathrm{CD}.dropAll()$
$Loader.loadAll()$
$\mathrm{CD}.raiseAll()$
$\mathrm{CF}.raiseAll()$
$\mathrm{OLAP}.\mathrm{processCubes}()$
}

The last six statements are the critical ones and should pass uninterrupted. Before they're executed the website is closed and when they're finished, the website is opened again. Our ETL process was designed with this in mind and the main purpose of the prime tables is to minimize that downtime [9].
It should be noted that, in the end of the ETL process, error tables contain valuable information about the erroneous data. They are used to rectify the data. When analyzing data, every user is presented with version (date) of the fact table and the number of erroneous rows. This way, users are much more eager to rectify their data. Although the data warehouse should not be the cleansing tool for the relational database, it often is, and inconsistencies in the data often become obvious only when one begins to analyze the data.

Fig. 3 shows simplified scheme of the implemented ETL process ("empty records" are indicated with white background, last good data snapshot is indicated with dotted background, latest version (new records) is indicated with shaded background).

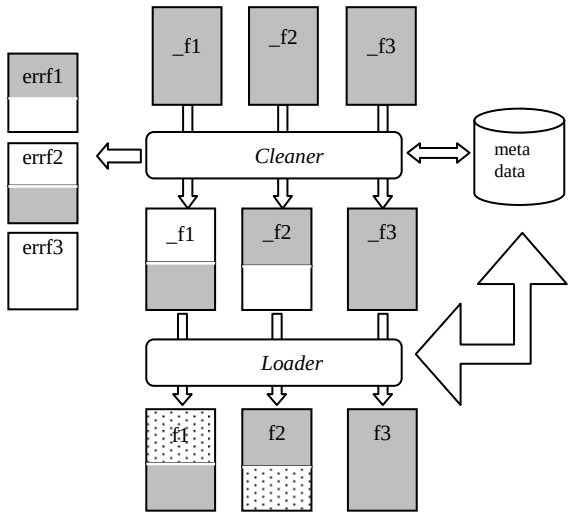

Fig. 3 Simplified scheme of Cleaner and Loader actions

5 Query log

HEIS DW is interfaced using the web application and all user activity on the website is logged. To increase the quality of service, we've decided to apply the business intelligence methodology to our log data. Queries, in particular, are logged in three steps: before execution, when executed and when displayed. Based on the gathered data we have defined *fQueryLog* fact table and included it in the ETL process along with other fact tables in HEIS DW (Fig. 4).

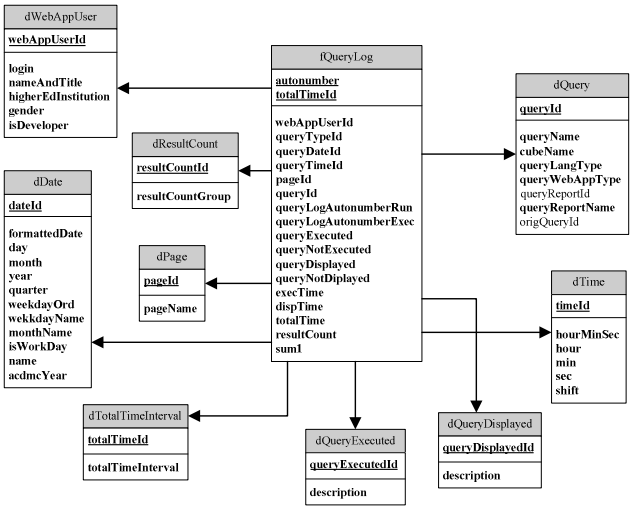

Fig. 4 *fQuerlyLog* fact table star schema

Integrated query logging options supported by some RDMBS manufacturers would not suffice in our case because we're tracking queries executed both on RDBMS and OLAP server, and are matching those queries with our predefined queries and categories. For instance, one non-parameterized query consist of a number of MDX and SQL queries – we are able to track the execution time of every query and the overall execution time of the non-parameterized query, and we're able to distinguish when a single query is being executed within a non-parameterized query or by itself. We've defined 9 dimensions and 9 measures. Measures *queryExecuted* and *queryDisplayed* (can be either 0 or 1) are used both as measures and dimension keys. *queryLogAutonumberRun* and *queryLogAutonumberExec* are degenerate dimensions, used for debugging purposes and to maintain direct reference to log entries. Dimension *dQuery* provides a catalog of predefined queries. This catalog is derived from MDX or SQL queries stored in the database. As for custom queries (*ad hoc* queries generated by a user), since they're not available in the catalog (database), they are represented as generic "custom query" entry in the dimension *dQuery*.

This fact table allows for all kinds of analysis that allow for a proactive approach to maintenance.

Fig. 5 shows total time distribution for the past 15 working days. The x-axis shows query execution time intervals (in seconds) grouped by query type (MDX or SQL). The y-axis shows the number of queries (for instance, approximately 2000 MDX queries were executed in 0-1 second interval). It is apparent that most of the queries execute in less than a second, but there is a small fraction of queries that took longer to execute, some even more than 5 minutes. We've examined the data and found that the query optimizer didn't work well with certain users (related to the way user access policy has been implemented) on a detailed query that was referencing the *fCourseEnrollment* table. Using the query optimizer hints we've reduced that time to below 30 seconds in the worst case and to below 10 seconds on average. This problem was fixed even before helpdesk received a complaints from users.

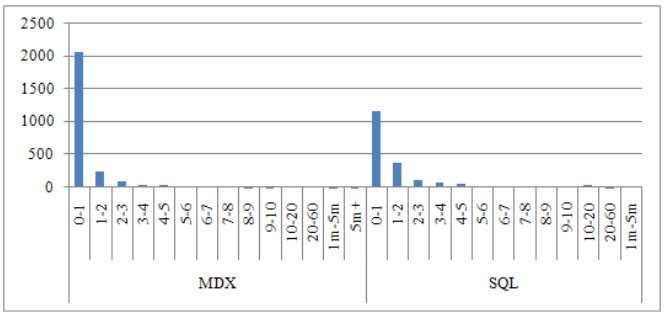

Fig. 5 Total time distribution by query type for past 15 working days

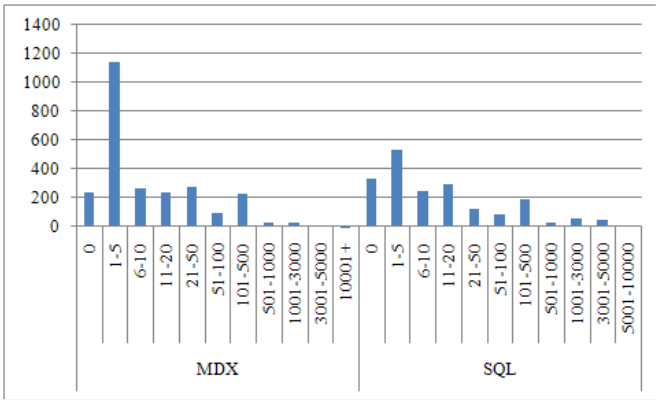

Fig. 6 Result count distribution by query type for past 15 working days

Fig. 6 shows result count distribution for the past 15 days by query type – we've noticed a number of queries with more than 5k rows (every detailed query is limited to the top N rows, usually 5k or 10k). It turned out that certain users were (mis)using DW to unload data from the HEIS.

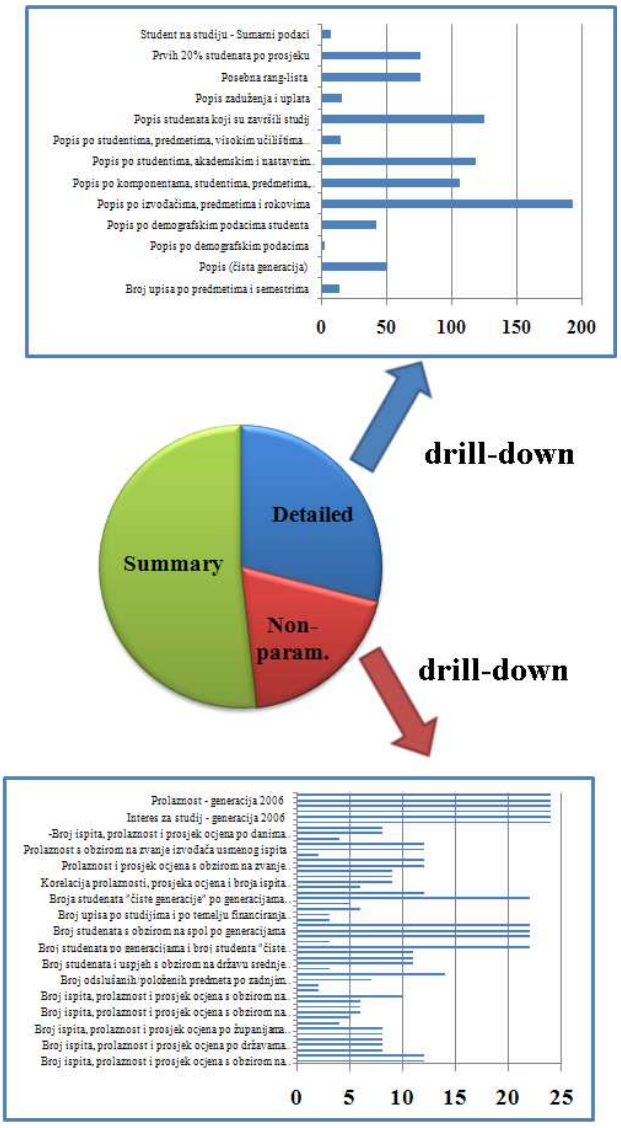

Fig. 7 Number of queries by application type in the past 15 days. Query names in Croatian are not meant to be readable

Fig 7 shows the number of queries by application type for the past 15 days. It is apparent that all types are used. From there, one can drill down (except for the summary queries) to see which of the queries within categories are used more often or not used at all, etc.

In conclusion, this fact table, along with other logs we're maintaining, allows one to have a clear picture of users' behavior and to detect bad query performance as soon as possible.

6 HEIS DW portal

Ultimately, enhancements to our ETL process affect user's experience. Users are notified in a simplified way of the current state of their data. Fig. 8 shows the initial page of the HEIS DW portal – user is informed that one fact table has two erroneous rows and, since that institution has a 0 threshold policy, last valid snapshot is used. Three frames below show a shortened list of detailed ad hoc queries, summary ad hoc queries and predefined queries (from left to right).

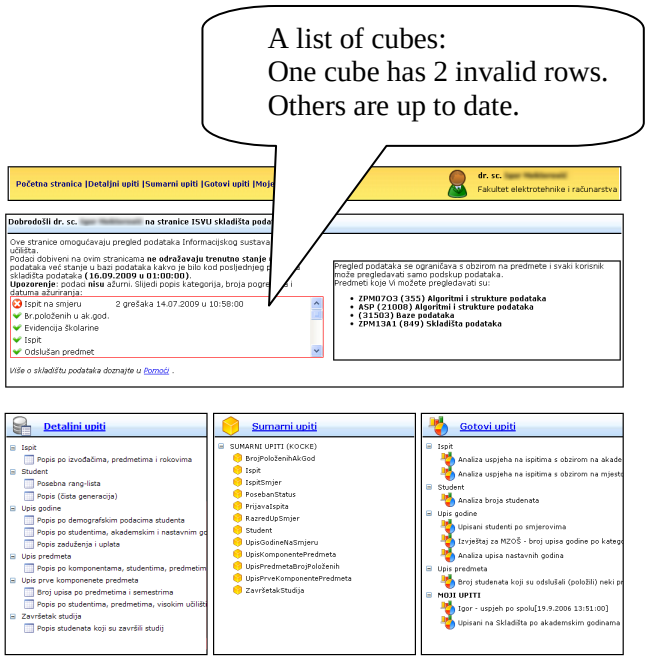

Fig. 8 DW portal – a user's perspective

The HEIS DW portal also has a restricted-access administrator's pages. Those pages are used by the help desk to facilitate daily maintenance and support operations. For instance, help desk personnel can inspect various logs (cleaner log, loader log,

changer log, overall etl log, etc.), see a list of currently active users and their permissions, analyze various reports on queries being executed by users, etc.

Fig. 9 shows the administrator's perspective on problems with the ETL process. There are 4 erroneous rows in total, equally distributed over two higher education institutions.

Administrator can easily see what institution has problem with which fact table and, notably – why. With this information at hand, administrator can take appropriate actions.

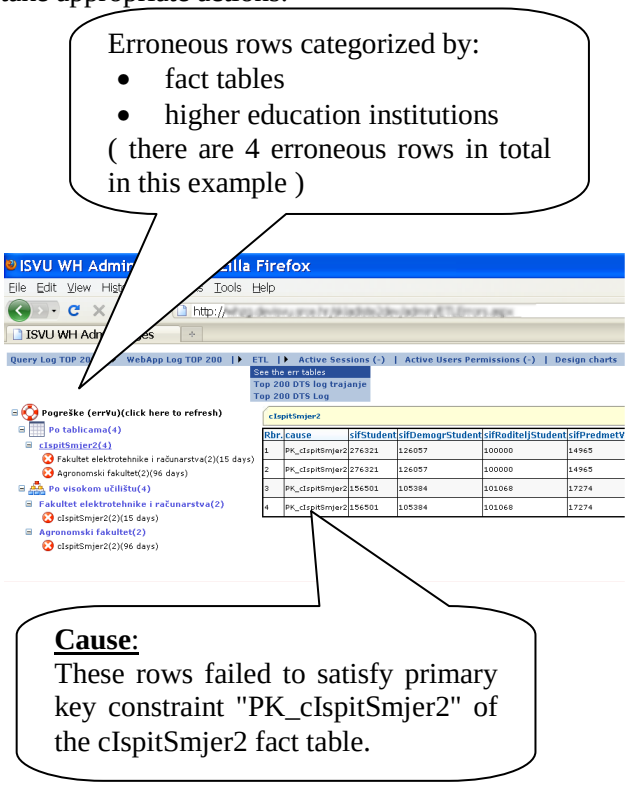

Fig. 9 DW portal – an administrator's perspective

7 Conclusion

In this paper Data Warehouse for the Higher Education Information System in Croatia is presented. The system enables fast and easy-to-use, web based way of querying and analyzing data in order to facilitate operational work and provide support for decision making in Croatian higher education institutions. It is explained that HEIS DW has to be accurate and up-to-date as it is being used on daily basis for operational purposes. For instance, it is used to generate all kinds of scholarship grants on various higher educational institutions, often using different criteria of excellence. Since a possibility of erroneous data entering relational database always exists, it is very important to prevent one higher education institution's errors from affecting another. Naturally, only one fact table per certain topic (e.g. course enrollment process) is used to store all higher education institutions data. We explain the necessity for horizontal data segmentation (in our case by higher education institutions) and versioning, thus isolating one institution data (and errors) from another.

As a solution to this problem, robust and efficient ETL process that allows for horizontal segmentation is proposed and formally described.

It is often stated that data warehousing is a process, not a project. Principal, if not the most important aspect of that process is maintenance and tending to the user's experiences and requirements. Proposed ETL process enhancements affect user experience in a positive way. Users are more informed and in a position to rectify erroneous data in collaboration with the help desk. Further on, it is described how data warehousing methodology has been used to facilitate maintenance and allow for a proactive approach to maintenance that increases the quality of service.

In the future, we would like to see HEIS DW used more as a strategic tool, as it provides that functionality. To facilitate that process, our plans involve new ways of data visualization and data mining.

References:

[1] Inmon WH. Building the Data Bridge: The Ten Critical Success Factors of Building a Data Warehouse. *Database Programming & Design* (1992)

[2] Baranović, M., Madunić M., Mekterović I.: Data Warehouse as a Part of the High Education Information System in Croatia, *Proceedings of the 25th International Conference on Information Technology Interfaces*, 121-126, (2003)

[3] Kimball R. *The Data Warehouse Toolkit*. New York: John Wiley And Sons; (1996)

[4] Heise, D. L.: Data Warehousing and Decision Making in Higher Education in the United States, Doctoral thesis (2006)

[5] Guan, J., Nunez, W., Welsh, J.F.: Institutional strategy and information support: the role of data warehousing in higher education, *Capus-Wide Information Systems*, 19(5), 168-174, (2002)

[6] Macfarlane, B.: Business and management studies in higher education: the challenge of academic legitimacy; *International Journal of Educational Management*, Vol. 9 No. 5, 1995,

pp. 4-9 © MCB University Press Limited, 0951-354X (1995)

[7] Sherman, R.: Beyond ETL and Data Warehousing; *InfoManagement Direct*, February 19, 2009

[8] European Commission Education & Training, http://ec.europa.eu/education/policies/educ/bologna/bologna_en.html

[9] M. Miličević, M. Baranović, V. Batoš: A Dynamic QoS Control Approach Based on Query Response Time Prediction. *WSEAS transactions on Computers*. Issue 8, Volume 4, (August 2005), p. 882-889

[10] C. dell'Aquila, F. Di Tria, E. Lefons, F. Tangorra: Business Intelligence Applications for University Decision Makers, *WSEAS Transactions on Computers*, Issue 7, Volume 7, (July 2008), p. 1010-1019.

[11] G. Kulvietis, J. Mamcenko, I. Sileikiene: Data Mining Application for Distance education Information System. *WSEAS Transactions on Information Science and Applications*, Issue 8, Volume 3, (August 2006), p.1482-1488,.