

**University of New Mexico**  
**UNM Digital Repository**

---

Electrical and Computer Engineering ETDs

Engineering ETDs

---

7-12-2014

# Designing and Implementing a Data Warehouse using Dimensional Modeling

VINAYA GANAPAVARAPU

Follow this and additional works at: [https://digitalrepository.unm.edu/ece\\_etds](https://digitalrepository.unm.edu/ece_etds)

---

## Recommended Citation

GANAPAVARAPU, VINAYA. "Designing and Implementing a Data Warehouse using Dimensional Modeling." (2014).  
[https://digitalrepository.unm.edu/ece\\_etds/92](https://digitalrepository.unm.edu/ece_etds/92)

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact [disc@unm.edu](mailto:disc@unm.edu).

VINAYA BHARADWAJ GANAPAVARAPU

*Candidate*

COMPUTER ENGINEERING (ECE)

*Department*

This thesis is approved, and it is acceptable in quality and form for publication:

*Approved by the Thesis Committee:*

GREGORY L. HEILEMAN

, Chairperson

TERRY J. TURNER

CHRISTOPHER C. LAMB

---

---

---

---

---

---

---

---

---

---

# Designing and Implementing a Data Warehouse using Dimensional Modeling

by

**Vinaya Bharadwaj Ganapavarapu**

B.Tech., Jawaharlal Nehru Technological University, 2008

THESIS

Submitted in Partial Fulfillment of the  
Requirements for the Degree of

Master of Science  
Computer Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2014

©2014, Vinaya Bharadwaj Ganapavarapu

# Dedication

*To my family.*

# Acknowledgements

I would like to thank my advisor, Professor Gregory Heileman and committee members Terry Turner and Christopher Lamb for their patience, help and support. I would also like to thank Rita Abeyta from the Office of Institutional Analytics, UNM for helping me in understanding higher education data.

# Designing and Implementing a Data Warehouse using Dimensional Modeling

by

**Vinaya Bharadwaj Ganapavarapu**

B.Tech., Jawaharlal Nehru Technological University, 2008

M.S., Computer Engineering, University of New Mexico, 2014

## **Abstract**

As a part of the business intelligence activities initiated at the University of New Mexico (UNM) in the Office of Institutional Analytics, a need for a data warehouse was established. The goal of the data warehouse is to host data related to students, faculty, staff, finance data and research and make it readily available for the purposes of university analytics. In addition, this data warehouse will be used to generate required reports and help the university better analyze student success activities.

In order to build real-time reports, it is essential that the massive amounts of transactional data related to university activities be structured in a way that is optimal for querying and reporting. This transactional data is stored in relational databases in an Operational Data Store (ODS) at UNM. But for reporting purposes, this design currently requires scores of database join operations between relational database views in order to answer even simple questions. Apart from affecting performance, i.e., the time taken to run these reports, development time is also a factor,

as it is very difficult to comprehend the complex data models associated with the ODS in order to generate the appropriate queries.

Dimensional modeling was employed to address this issue. Dimensional modeling was developed by two pioneers in the field, Bill Inmon and Ralph Kimball. This thesis explores both methods and implements Kimball's method of dimensional modeling leading to a dimensional data mart based on a star schema design that was implemented using a high performance commercial database. In addition, a data integration tool was used for performing extract-transform-load (ETL) operations necessary to develop jobs and design work flows and to automate the loading of data into the data mart.

HTML reports were developed from the data mart using a reporting tool and performance was evaluated relative to reports generated directly from the ODS. On average, the reports developed on top of the data mart were at least 65% faster than those generated from directly from the ODS. One of the reason for this is because the number of joins between tables were drastically reduced. Another reason is that in the ODS, reports were built against views which when queried are slower to perform as compared to reports developed against tables.



# Contents

<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	1
1.2 Proposed Solution . . . . .	3
<b>2 Background</b>	<b>5</b>
2.1 Existing philosophies on data warehouses . . . . .	5
2.2 Differences between Inmon and Kimball methods of approach to data warehousing . . . . .	8
2.3 Definitions . . . . .	9
<b>3 Design and Implementation</b>	<b>12</b>
3.1 Process of building a Kimball data warehouse . . . . .	12
3.2 Business Matrix . . . . .	14
3.3 Grain . . . . .	14

## Contents

3.4	Logical and physical design . . . . .	15
3.5	Solutions provided by the data mart design . . . . .	17
3.6	Further optimization of the data mart for reports . . . . .	19
3.6.1	Building summary fact tables . . . . .	19
3.7	Design issues . . . . .	20
3.7.1	Resolving many-to-many relationships . . . . .	20
3.7.2	Implementing slowly changing dimensions (SCD) . . . . .	21
<b>4</b>	<b>Extract Transform Load (ETL)</b>	<b>30</b>
4.1	Populating the data mart . . . . .	30
4.2	Logical steps in ETL . . . . .	30
4.3	Refreshing the data mart . . . . .	35
4.4	ETL performance and data validation . . . . .	36
<b>5</b>	<b>Reporting from the data mart</b>	<b>42</b>
5.1	Data Mart Performance . . . . .	42
5.2	Reports . . . . .	44
<b>6</b>	<b>Future Work</b>	<b>48</b>
6.1	Student data mart table structure (DDL) . . . . .	53
	<b>Appendices</b>	<b>72</b>

*Contents*

**References**

**73**

# List of Figures

1.1	Student credit hours generated by faculty. . . . .	3
1.2	ODS versus data mart performance. . . . .	4
2.1	Inmon data warehouse Architecture [2]. . . . .	7
2.2	Kimball data warehouse Architecture [1]. . . . .	8
3.1	Kimball data warehouse life cycle [1]. . . . .	13
3.2	Business Matrix. . . . .	15
3.3	Granularity of fact tables. . . . .	15
3.4	Granularity of dimension tables. . . . .	16
3.5	Student data mart schema. . . . .	17
3.6	Student Enrollment Star Schema. . . . .	24
3.7	Course Enrollment Star Schema. . . . .	25
3.8	Instructor Course Assignment Fact Schema. . . . .	26
3.9	Summary Fact Tables. . . . .	27
3.10	Instructor Bridge Table. . . . .	28

*List of Figures*

3.11	Slowly Changing Dimension Type 1. . . . .	28
3.12	Slowly Changing Dimension Type 2. . . . .	29
3.13	Slowly Changing Dimension Type 3. . . . .	29
3.14	Hybrid Slowly Changing Dimension. . . . .	29
4.1	Source to Target mapping in SAS <sup>®</sup> DI Studio. . . . .	32
4.2	Populating a staging table using SQL transformations in SAS <sup>®</sup> DI Studio. . . . .	33
4.3	Populating a dimension table. . . . .	33
4.4	Populating a fact table - Finance Fact. . . . .	37
4.5	Joins and where conditions in SAS <sup>®</sup> DI Studio. . . . .	38
4.6	Work flow, parallel jobs and scheduling in SAS <sup>®</sup> DI Studio. . . . .	38
4.7	Job scheduling triggers in SAS <sup>®</sup> DI Studio. . . . .	39
4.8	SQL Extract transform to run custom SQL. . . . .	39
4.9	Proc-SQL example to run custom queries. . . . .	40
4.10	Control flow of transformations in a job. . . . .	40
4.11	Job status. . . . .	41
4.12	Job statistics. . . . .	41
5.1	Average ODS versus data mart performance. . . . .	44
5.2	SCH generated by faculty by academic period, college. . . . .	46
5.3	Student hours enrolled by academic period, college. . . . .	47

*List of Figures*

6.1	Sankey Diagram for 2008 Arts and Sciences Chemistry College Flow.	50
6.2	Sankey Diagram node highlight example. . . . .	51
6.3	Sankey diagram statistics display. . . . .	52

# Chapter 1

## Introduction

### 1.1 Overview

The University of New Mexico (UNM) had nearly 30,000 students enrolled in 2011 and enrollment has steadily increased by 24% as compared to the year 2001. With a wide range of programs and courses, increase in enrollment each year and sometimes more than one instructor per course, one can imagine the complexity and volume of data that would have been accumulated over the years.

To ensure success of students, it is important to track their progress each semester, i.e., to track if a student has reached a certain achievement level or not, meeting the academic needs of students, establishing and accomplishing short-term and long-term goals. Currently, only quantitative questions like number of students served or number of students receiving a passing grade in a course and so on determine measures taken to ensure progress of students. Such reports however do not enable deeper understanding, for instance, growth of a student in a given time period, comparing growth over years and factors contributing to this.

## *Chapter 1. Introduction*

With sufficient historical data, it is now the time to make proper use of it. It is very important to make data-based decisions. With a proper understanding of the needs of business users, decision makers and policy makers, the historical data and also the new data can be logically modeled to support decision-making or business intelligence.

There are two steps to the business intelligence process. One, is to build a data model to support the reporting requirements of users which will help them track progress and success and the second is to build a predictive data model that will help the policy and decision makers by providing a better vision for the purpose of decision making.

The University of New Mexico currently has an operational data store (ODS) which is being used to serve as the back-end to generate many reports or to cater to many applications that were developed for the academic or the executive offices. But with changing needs, the database design of ODS is not efficient enough to support analytics that need to be developed.

Finally, the difficulty associated with generating the reports necessary to answer specific questions needs to be addressed. This is very difficult to comprehend the ODS table structure given the way it is defined and many users utilize the views in ODS to build their reports; a very time consuming process. Reports built on top of ODS views take a long time to run as views are not physically populated in the database, rather they are just a logical representation of data in tables.

Consider an example as in Figure 1.1 where a report is generated for student credit hours (sch) generated by faculty filtered by college in a given academic period. This report requires a drop down on the academic period and when one of them is selected, the ODS view needs to be queried and calculations need to be performed on the fly. This process of querying an un-indexed view with hundreds of columns and



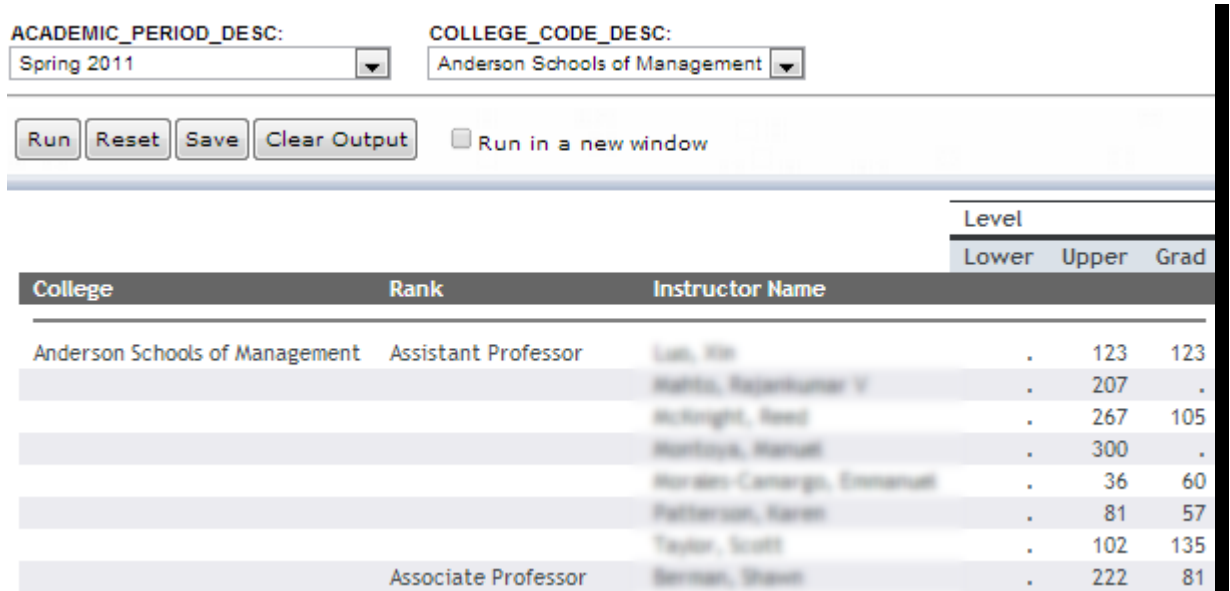


Figure 1.1: Student credit hours generated by faculty.

performing calculations while generating the report consumes an undesirable amount of time.

## 1.2 Proposed Solution

A current need, therefore involves the need to design a database that supports reporting, analytical and decision-making capabilities for executive offices at UNM. An efficient way to solve the problem mentioned in the previous section, is to use one of the industry standard methods in order to design a data warehouse that will provide a platform for running reports and to develop analytics. Bill Inmon and Ralph Kimball are the two pioneers in the theory of building data warehouses. Kimball's method of approach to developing data warehouse was chosen over Inmon's method. The design and implementation of the data warehouse using Kimball's approach as described in [1] will be the main focus of this thesis. The data warehouse will contain

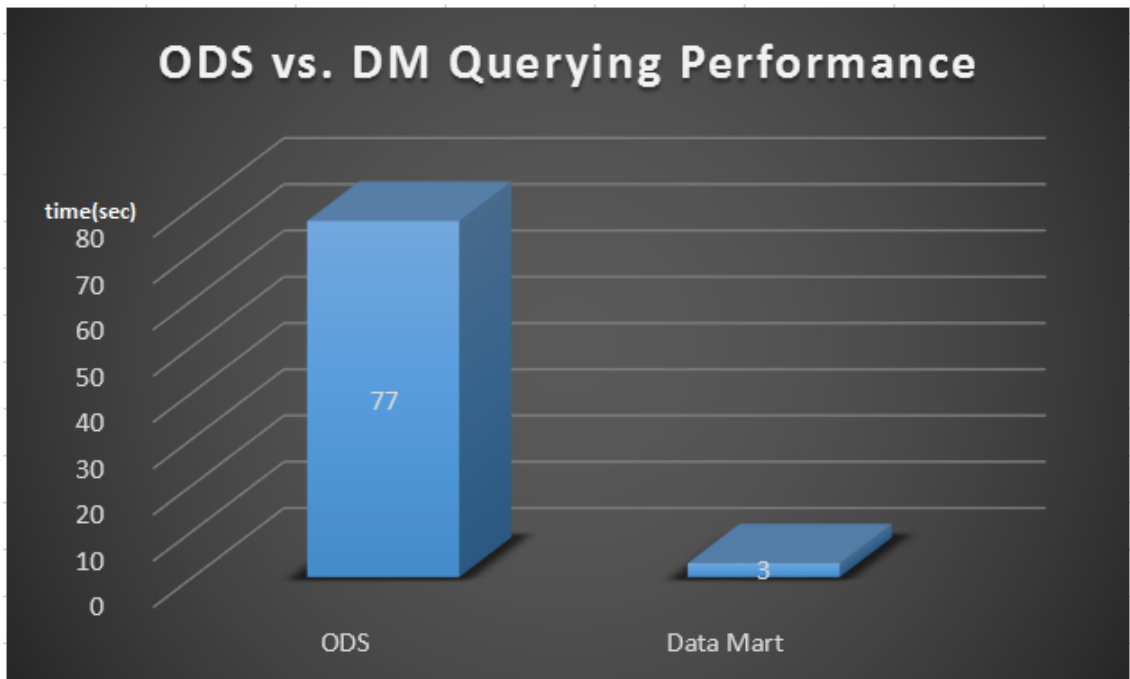


Figure 1.2: ODS versus data mart performance.

data related to students, faculty, staff, finance, research, space, academic outcomes to list a few.

It is evident from the example shown in Figure 1.2 which demonstrates the performance of the data mart against ODS. When a SQL query involving joins between two tables and group by and order by clauses, the data mart returns the results in 3 seconds where as the ODS takes around 77 seconds to return the same results.

# Chapter 2

## Background

### 2.1 Existing philosophies on data warehouses

With exponential growth in data and with increasing interest to analyze and understand it to derive some knowledge out of it, it is absolutely necessary that the historical data be stored in a manner that can be easily analyzed. It is also important to derive some important statistics and various performance metrics. But this cannot be effectively implemented using the ODS. The data needs to be reorganized into a dimensional model.

A third normal form (3NF) is an entity relationship (ER) model where the tables are normalized to a state where there is no data redundancy, attributes in an entity are solely dependent on whole of the primary key of the entity. A primary key can be just one attribute or a composite key consisting of two or more attributes.

The 3NF was initially defined by E.F.Codd and was later expressed differently by Carlo Zaniolo. According to [5], it is defined as follows.

A table is in third normal form (3NF) if and only if for every nontrivial

## Chapter 2. Background

functional dependency  $X \rightarrow A$ , where  $X$  and  $A$  are either simple or composite attributes, one of two conditions must hold. Either attribute  $X$  is a superkey, or attribute  $A$  is a member of a candidate key. If attribute  $A$  is a member of a candidate key,  $A$  is called a prime attribute. Note: a trivial FD is of the form  $YZ \rightarrow Z$ .

A 3NF ER model is good design to handle transactional data. Examples for transactional data are courses taken by students in each term, proposals filed by each principal investigator each year and so on. This type of design is good for performing quick inserts, updates and deletes because the tables usually have a small number of fields with foreign keys to other tables. The reason behind this is that, to generate a simple report one would end up joining scores of tables in a 3NF environment which ends up being very time and resource (hardware) intensive. Apart from this, the model becomes very complex very quickly as a result of which it becomes very difficult to understand and navigate the model even for a developer let alone a business user.

To address this problem, Bill Inmon in Figure 2.1 proposed an idea with a top-down approach and named the system as corporate information factory (CIF) defined in [2]. The components of a CIF include a data warehouse which is built in 3NF and individual de-normalized data marts which are populated from the data warehouse. These data marts cater to individual business process needs. Reporting cubes are built as required on top of the data marts. The data warehouse as defined by Bill Inmon, contains enterprise data without any redundancy at the lowest level of detail i.e., transactional data in 3NF.

According to Inmon, a data warehouse is subject-oriented, non-volatile, integrated, time-variant and has no virtualization.

Another pioneer, Ralph Kimball proposed his own version of a data warehouse

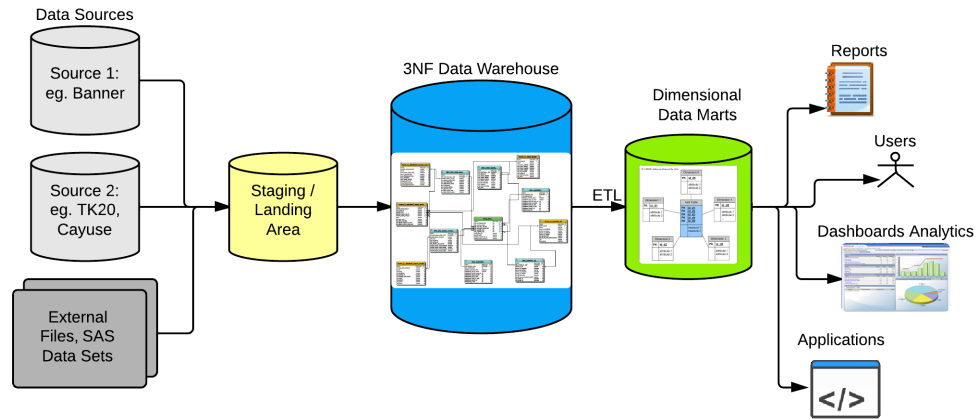


Figure 2.1: Inmon data warehouse Architecture [2].

Figure 2.2 which is termed as a bottom-up approach. A data warehouse as defined by Kimball in [1] is the conglomeration of all individual data marts. These data marts are built using dimensional modeling. Each data mart contains data specific to a business process. This method was chosen as part of business intelligence architecture that has been implemented at UNM.

**The benefits of dimensional modeling technique are:**

**Better data navigation and presentation:** The logical data model is designed in a way that is easy to understand and navigate even for business users. This enables them to build their own reports.

**Easy and low-cost maintenance:** The data is stored in the same way as it is presented unlike in the case of relational databases where in most cases views are built in order to build any reports. This increases the maintenance cost in the case of relational databases or ODS.

**Better performance:** Most reports require summarized data which results

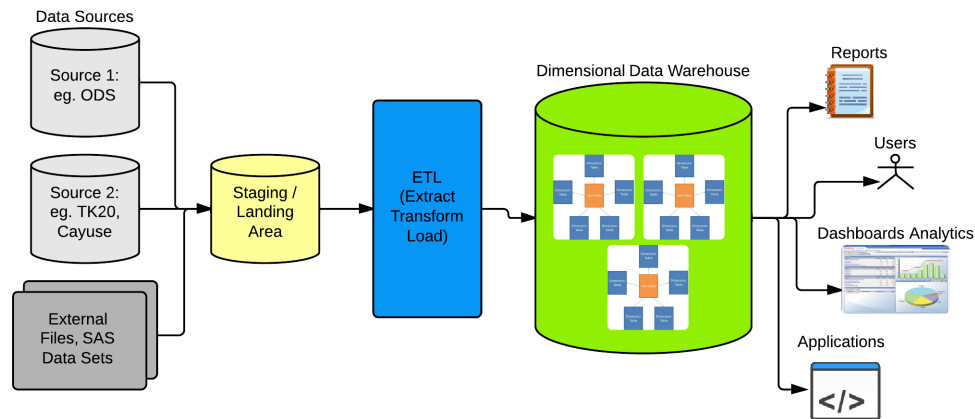


Figure 2.2: Kimball data warehouse Architecture [1].

in a slower performance due to on-the-fly calculations in the case of non-indexed views in ODS. In the case of dimensional modeling, summarized tables are built as required. It also allows the ability to store data history in a manner that is easy to query and build reports on. Such a design delivers faster query performance and to drill down and drill across hierarchies.

## 2.2 Differences between Inmon and Kimball methods of approach to data warehousing

Data warehousing is the process of building a data warehouse. Reports are currently being built on top of the views in ODS. The ODS is not actually optimized for reporting or analytics.

Now in the case of Inmon's approach of data warehousing, the architecture suggests that a 3NF data warehouse be built as the next step, which would contain

## Chapter 2. Background

all the data in the organization, and then build a data marts layer to support the reporting layer. In our case, with ODS and a staging layer in place, it is not optimal in terms of time and money to build another layer of 3NF data warehouse which can be seen as nothing but another staging layer for the data. This data warehouse has no independent deliverable of its own.

In the case of Kimball's approach however, the idea is to build the data marts layer right after the staging layer. These data marts cater to individual business processes identified. All the data marts together then form the data warehouse as defined by Kimball. The common dimensions between the business processes are however shared between them, without building a separate version of it, to maintain a single version of truth and make it simple to update. These are called conformed dimensions. Using this approach, we do not need a second staging layer and since the data marts are specific to a business process, reports can be generated out of it, without waiting for rest of data marts to be designed and implemented.

Therefore, considering reasons like cost-effectiveness and the ability to deliver reports quickly, it has been decided that the Kimball methodology would be used for designing a data warehouse to address our problem.

### 2.3 Definitions

A few of the common terms of data warehousing used in this thesis are explained below as defined by Kimball in [1], [4] and [6].

A **Fact table** consists of the foreign keys to all the dimension tables in the schema and facts that are numerical business measurements. It is a transaction based table.

A **Dimension table** is an explicitly defined subject area in a business. it is a non-transaction based table.

## Chapter 2. Background

**Conformed dimensions** are standardized tables modeled once and shared across multiple fact tables in the same schema or even a different data mart. This is determined by the Bus Matrix. These tables support the ability to drill across and integrate data from multiple business processes. The main advantage of using conformed dimensions is to save storage space. It is also easier to maintain and refresh one table versus multiple versions of the same table.

An **enterprise data warehouse bus matrix** is the architectural blueprint providing the top-down strategic perspective to ensure that data can be integrated across the enterprise.

The **grain** is defined as the lowest level of detail in a table. The grain of a fact/dimension table is the definition of what constitutes a unique fact/dimension table record.

A **factless fact table** is a fact table that contains no facts but captures certain many-to-many relationships between the dimension keys. Most often it is used to represent events or provide coverage information that does not appear in other fact tables. Some examples include tracking student attendance or registration events, identification numbers of building or facility.

The **business key or natural key** identifies a business entity. Examples include student\_id, course\_id and program\_id.

The **primary key** uniquely identifies a record in a table. A primary key can consist of a single field or multiple fields and cannot be a NULL value.

The **Foreign key** is a single field or multiple fields which uniquely identifies a record in another table.

The **Surrogate Key** uniquely identifies a record in a dimension table. It is usually ETL generated and provides the means to maintain data warehouse information



## Chapter 2. Background

when dimensions change. One simple way improve performance of queries is to use surrogate keys. Surrogate keys can be derived from the existing natural keys or it can be a simple integer. As an example, a surrogate key can be a composite key, being the combination, `student_id + academic_period` or just an integer value generated by the ETL program while a record is being inserted the table. Using integer surrogate keys means a thinner fact table and the thinner the fact table, the better the performance.

The **star schema** is a dimensional design for a relational database. In a star schema, related dimensions are grouped as columns in dimension tables, and the facts are stored as columns in a fact table. The star schema gets its name from its appearance: when drawn with the fact table in the center, it looks like a star or asterisk.

The **snowflake schema** is a variation on the star schema. When principles of normalization are applied to a dimension table, the result is called a snowflake schema.

A degenerate dimension is present in the fact table. For example this may be a transaction number, invoice number, ticket number, or bill-of-lading number, that has no attributes and hence does not join to an actual dimension table. A junk dimension is present in the fact table. Examples include boolean indicator or flag fields such as `enrolled_flag`, ethnicity indicators, `etl_timestamp` and so on.

# Chapter 3

## Design and Implementation

Kimball's approach was chosen as the base architecture for supporting business intelligence at UNM. The conceptual model and logical model were designed based on the requirements analysis and source data analysis.

### 3.1 Process of building a Kimball data warehouse

Various steps involved building a Kimball data warehouse include business/user requirements gathering, requirements analysis, source data analysis, target database logical model design, target database physical design, source data cleansing, extract transform load (ETL) process design, data validation, report development and performance analysis. All design principles are based on Kimball's approach to data warehousing as defined in [1].

Different approaches were used for gathering requirements including phone interviews, personal interviews with individual end users and stake holders and also group meetings. A requirements document was prepared in agreement with the business.

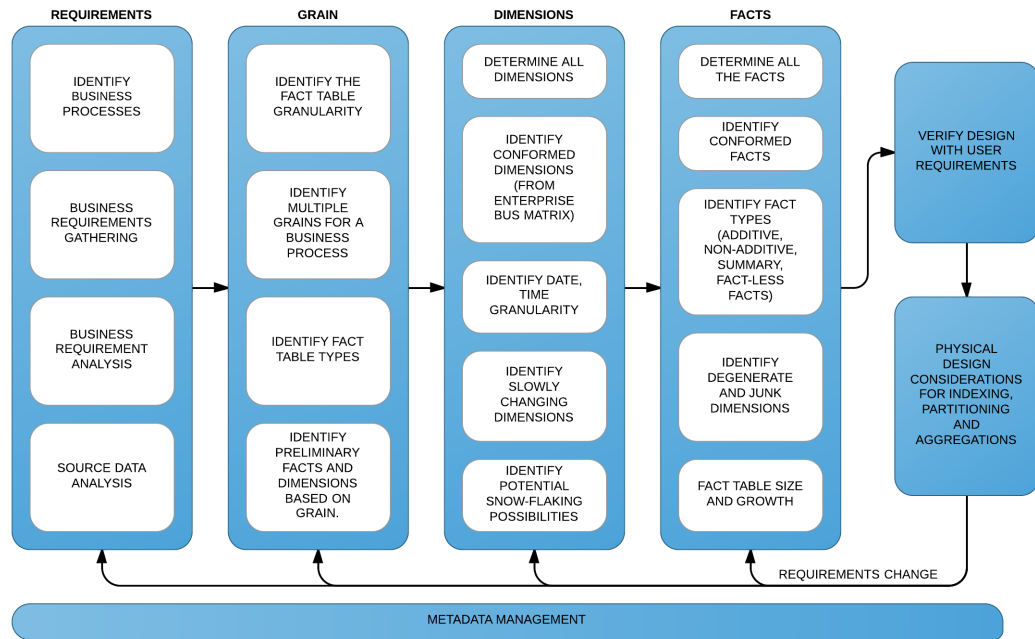


Figure 3.1: Kimball data warehouse life cycle [1].

Requirements analysis was done keeping the structure of the source data in mind. A conceptual data model was prepared which laid out a high level structure of the entities and the relationships between them after identifying separate business needs. Facts and dimensions were identified. Grain of the tables was determined. Conformed dimensions are identified using the business matrix. A data mart may have multiple star schemas but dimensions can be shared between different fact tables or business processes. For example, the student data mart has multiple star schemas. One star schema is dedicated to the academic progress of the student, student detail, program, department, courses, instructors and person data. The second schema captures financial aid. Another star schema is dedicated to applications and admissions.

Next, the logical data model and the physical data model (for Oracle 11g environment) were designed using licensed software, Toad data modeler. There were some

### *Chapter 3. Design and Implementation*

design issues which needed special handling including many-to-many relationships and slowly changing dimensions which are explained later in this thesis.

A document was prepared which contained the source table fields mapped to the target table fields. This is called the source-to-target mapping document. Preparing this mapping document is very essential as it is the base for the extract transform and load (ETL) development process.

The next step in the data warehousing process is the ETL design. SAS<sup>®</sup> data integration software was used to perform the ETL tasks.

Using the data mart as the back end, HTML reports as required by various business units were designed using Web Focus which is the primary reporting tool currently being used at UNM.

## **3.2 Business Matrix**

The business matrix Figure 3.2 is tabular representation of the business process versus the dimension tables. This is an agile modeling method and is done so as to identify the Conformed dimensions. In the figure, we can see that the dimensions academic calendar, course, department and person are shared between multiple business processes. These are called conformed dimensions.

## **3.3 Grain**

The grain of a table is defined as its lowest level of detail. The tables in Figure 3.3 and Figure 3.4 show the grain of the fact and dimension tables in the student data mart. For example, the student enrollment fact table has one row per student per

BUSINESS PROCESSES	IDENTIFIED DIMENSIONS								
	Acad Calendar	Course	Department	Student	Program	Instructor	Financial Aid	Person	
Student Enrollment	X	X		X				X	
Course Enrollment	X	X							
Department Enrollment	X		X						
Program Enrollment	X				X				
Instructional Assignment	X	X	X			X		X	
Financial Aid	X						X		

Figure 3.2: Business Matrix.

FACT TABLES	ATOMIC GRANULARITY	FACTS
Student_Enrollment_Fact	1 row per student, per course, per term	Enrollment,Grade
Course_Enrollment_Fact	1 row per course, per term	Enrollment,Grade
Department_Enrollment_Fact	1 row per department, per term	Enrollment,Grade
Program_Enrollment_Fact	1 row per program, per term	Enrollment,Grade
Instructional_Assignment_Fact	1 row per instructor, per course, per term	Primary Indicator, % Responsibility
Financial_Aid_Fact	1 row per student, per term	Amount Awarded

Figure 3.3: Granularity of fact tables.

course per semester which means that this table will have all the records of a student with respect to all the courses the student was ever enrolled in. Also mentioned are the primary facts in that table. Similarly the dimension tables granularity in Figure 3.4 shows a few main fields in that table.

### 3.4 Logical and physical design

After designing the business matrix and determining the granularity of tables, the next step is logical designing of the data mart. This was done using the Toad data

Chapter 3. Design and Implementation

DIMENSIONS	ATOMIC GRANULARITY	DATA FIELDS
<a href="#">Academic_Calendar_Dim</a>	1 row per academic period	Academic Year and Aid Year Hierarchies
<a href="#">Course_Dim</a>	1 row per course, per academic period	Acad. Period, Course No., Sections, CRN
<a href="#">Department_Dim</a>	1 row per department	Department Hierarchy Information
<a href="#">Student_Dim</a>	1 row per student, per academic period	Student program, campus, college, dept.
<a href="#">Program_Dim</a>	1 row per program	Program Information
<a href="#">Person_Dim</a>	1 row per person	Demographic Information
<a href="#">Financial_Aid_Dim</a>	1 row per fund_code	Fund Information
<a href="#">Instructor_Dim</a>	1 row per instructor	Instructor Department, Rank
<a href="#">Instructor_Group_Dim</a>	1 row per instructor group	Instructor Group Key
<a href="#">Instructor_Group_Bridge</a>	1 row per instructor, per instructor group	Instructor Group Key, Instructor Key

Figure 3.4: Granularity of dimension tables.

modeling tool and most of the business processes or subject areas mentioned in the business matrix diagram were designed using the star schema approach. In a star schema, there is a fact table, which has all the foreign keys to the dimension tables in that subject area and the facts. The dimension tables have data related to different areas like person, course, student, department, program and so on. Figure 3.5 shows the student subject areas within the enterprise university data warehouse. Other subject areas include finance, research and so on. In the student data mart, there are different aspects that have been identified like student enrollment, course enrollment, admissions, financial aid and some summary tables. Each of these aspects have their dimension tables and a fact table. For example, the student enrollment schema has a fact table named `student_enrollment_fact` and dimension tables `academic_calendar_dim`, `student_dim`, `course_dim` and `instructor_dim`. The dimension tables have a surrogate key apart from the natural keys. This is done in order to uniquely identify each record in the table. This surrogate key is usually a number and is automatically generated when the table is being loaded. Figure 3.6 shows the student enrollment star schema with the `student_enrollment_fact` table having all the foreign keys to the surrounding dimension tables and then the actual facts like grades and credit hours. Similarly Figure 3.7 shows the

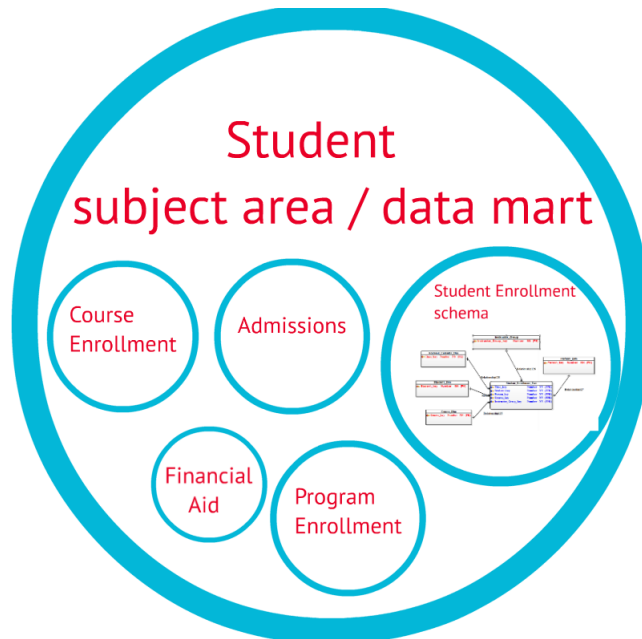


Figure 3.5: Student data mart schema.

course\_enrollement. Figure 3.8 shows the instructor course assignment fact schema. This schema captures all the courses assigned to various instructors over years. It shows a instructional\_assignment\_fact table having facts like percent responsibility of each instructor assigned to a course with multiple instructors. The fact table is connected to the required dimensions. It can also be observed, the table and field names are quite intuitive and are easily understood by anyone.

Indexing the tables appropriately and partitioning the tables if required are part of the physical design. After the physical design, the table definitions are captured into a data definition language (DDL) file and are installed on the database.

### 3.5 Solutions provided by the data mart design

The following is a non-comprehensive list of solutions offered by this Data Mart:

### *Chapter 3. Design and Implementation*

**Student level facts and statistics:** The data mart can answer questions at the most transactional level about students like courses registered by a student in a given term, final grades of each student in a course, enrolled date and drop date of a course and bio-demographic information.

**Course level facts and statistics:** These include total active enrollment in a term in a course, total drop outs out of a course, instructor details of a course including cases multiple instructors per course, total enrollment in each course and enrollment categorized by ethnic groups, average grade of all students in each course and average grade by ethnicity.

**Instructor level facts and statistics:** These statistics include all courses taught by an instructor in a given term, percentage contribution of each instructor in a course with multiple instructors, bio-demographic information about instructors and HR and departmental information about instructors.

**Department level facts and statistics:** Examples for department level statistics that are provided by the data mart include courses offered by a department in a given term and their corresponding facts and summarized reports include enrollment in each course offered by a department over many semesters and grade distribution, i.e., number of students in each grade, in each course offered by a department in a semester.

**Program level facts and statistics:** The data mart provides program level statistics like courses in a program in a given term and their corresponding facts. Some of the summarized level of information includes count of students registered in a program over a period of many semesters.

**Financial aid facts and statistics:** These are statistics on students who have been awarded financial aid in a given term. The data mart can also be used to see how many students received a Pell grant or submitted the free application for Federal



Student Aid (FAFSA) in a semester. This data can be analyzed together with the grades data resulting in interesting information.

**Applications and admissions statistics:** These include account of those who have submitted their test scores and prior GPAs to UNM, count of applicants to each program in each academic period versus count of admitted students, average test scores of applicants who were admitted into each department or program, distribution of admitted students with various test scores and so on.

## 3.6 Further optimization of the data mart for reports

Sometimes, it is required to see reports that need aggregation of the fact tables. Examples include reports needed by some of the executive offices at UNM which require summarized data from the `student_enrollment_fact` table rolled up to the department, program, degrees awarded or student data summarized by term and level. To generate the appropriate reports according to these requirements, the reporting tool needs to perform aggregation on the fly. This may take a lot of time depending on the complexity.

### 3.6.1 Building summary fact tables

To overcome this issue, summary fact tables have been designed to readily address such report requirements. Figure 3.9 shows a few summary fact tables that have been designed as a part of the student data mart. The summary fact tables contain rolled up version of the actual transaction level data. For instance, the `program_enrollment_fact` table has one record per program per semester which states

facts like total enrollment in the program and enrollment distribution by ethnicity. Similarly, the grain of `student_level_summary_fact` table is one record per student and contains cumulative data of the student until that point of time since inception. The table `student_term_summary_fact` has one record per student per semester and keeps track of student GPA, credit hours and such for each semester. The `degree_sawarded_fact` table has degrees awarded to all the students at UNM. These tables enable quicker reporting as on-the-fly calculations can be avoided. In spite of it being true that more storage is required, the creation of these summary tables can be justified considering the facts that storage is not very expensive and the tremendous performance benefit the creation of these tables has to offer.

## **3.7 Design issues**

### **3.7.1 Resolving many-to-many relationships**

It is impossible to implement many-to-many relationships between tables, on a physical database except by having another table between them which splits the many-to-many relationship to many-to-one and one-to-many relationships. In Kimball's dimensional modeling, this is called the bridge table [1].

In this design, the `student_enrollment_fact` table takes care of the many-to-many relationship between student-dimension and course-dimension tables. However, at UNM, we also have multiple instructors for some courses. To implement this, an `instructor_group` dimension table was introduced which has an `instructor_group_key` attribute. There is a one-to-many relationship between `instructor_group` dimension table and `student_enrollment_fact` table.

An `instructor_group_bridge` table is introduced which has an `instructor_group_key` and `instructor_key` (from the `instructor_dimension`) having many-to-one relationship

to both instructor\_group dimension and instructor\_dim tables. Thus we are breaking up a many-to-many relationship into two many-to-one relationships.

### 3.7.2 Implementing slowly changing dimensions (SCD)

The fact tables see frequent insertion of new records in sync with the transactions taking place. Thus, fact table grows quickly over time, but updates to these records are rare. In the case of Dimension tables, the data growth is relatively slow, however updates may be necessary to those records with changes to the business rules, or student data, personal data and so on.

These changes may or may not be tracked based on business requirements. The following are some of the different ways to do so. These methods are based on Kimball's approach to slowly changing dimensions in a data warehouse in [1].

#### **SCD TYPE 1: Overwrite existing record**

In slowly changing dimension Type 1 [1], the existing attribute is updated, i.e., it is over-written with the new value. This type of SCD is used when the business requirements state that no history of data is required for any analysis.

In the example Figure 3.11 shown below, the address is replaced with the new value.

#### **SCD TYPE 2: Add a New Record**

In this type of slowly changing dimension, a new record is added whenever there is a change, instead of updating the existing record with the new value. This is done with the help of two date fields, effective\_start\_date and effective\_end\_date. These

dates are manipulated, whenever there is a change [1].

This type of SCD, is the most commonly used and preferred because of its ability to store unlimited history as well as to store the time of that change.

In the example Figure 3.12, The `effective_end_date` field is initially populated with a random future date value, preferably a few years away, keeping the life cycle of the data warehouse in mind. Now, whenever there is a change to address, the `effective_end_date` is changed to the when the address effectively ends, and a new record is added with the new address value along with the respective `effective_start_date`. Also, there is another field called `current_flag` which is a boolean value indicating whether the record is active or not. This column is included in order to further optimize the query performance on this table when finding active or inactive records.

For example, to pull the active/current records, one may just look for `current_flag = 1` instead of looking at (or joining on) the `effective_start_date` and `effective_end_date` values.

### **SCD TYPE 3: Add new column**

In slowly changing dimension type 3, a new field is introduced to keep limited track of the changes [1].

This type of SCD is used when the business requirements state that only limited history needs to be stored in the data mart. If there is a pre-determined value for the number of versions of history being tracked, this type of SCD handling is helpful in reducing redundancy and thus save storage space.

In the example Figure 3.13, assuming that we are keeping history of two values for social security number, one field is the current social security number called SSN and the previous or expired value is under the field name Prior SSN. Whenever a

change is reported, the prior SSN field is updated with the existing value in SSN field and the SSN field is updated with the new value. We could also have dates to keep track of when the change has occurred.

### **Hybrid SCD (SCD Type 6): SCD Type 1 + Type 2**

There are many types of hybrid slowly changing dimensions, one of which is a combination of SCD types 1 and 2. This is also termed as SCD type 6 [1].

In this type of SCD, as in Figure 3.14, one field in the table can be a Type 2 SCD (field: Address) and another is handled as a type 1 SCD (field: SSN). Initially we have a record with an address and SSN. When there is a change in the address, it is updated according the rules of SCD type 2 i.e., updating the `effective_start_date`, `effective_end_date` and `current_flag` fields. However when there is a change in the SSN, it is updated according to the SCD type 1 rules which is to just replace or over-write the old value.

Chapter 3. Design and Implementation

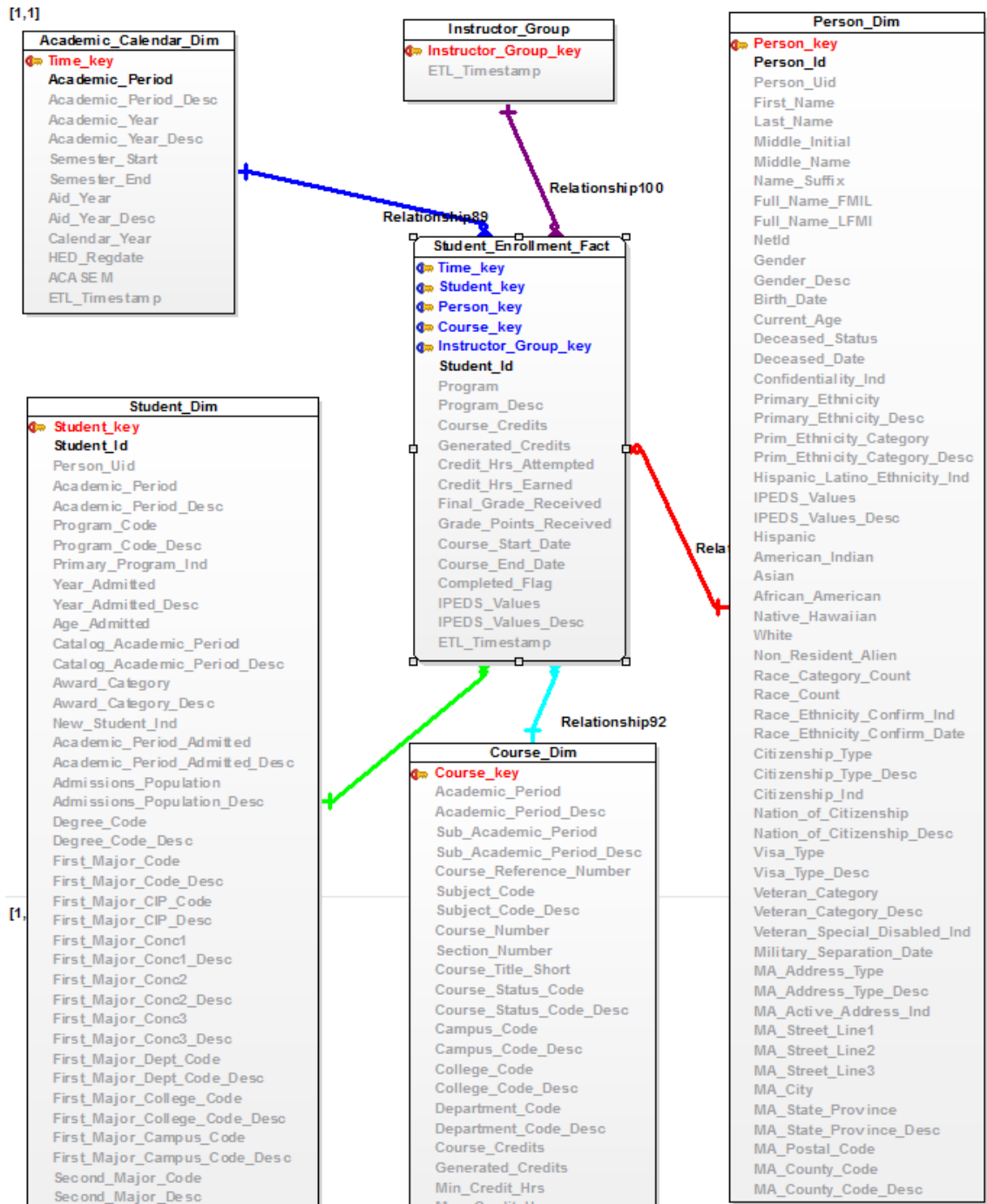


Figure 3.6: Student Enrollment Star Schema.

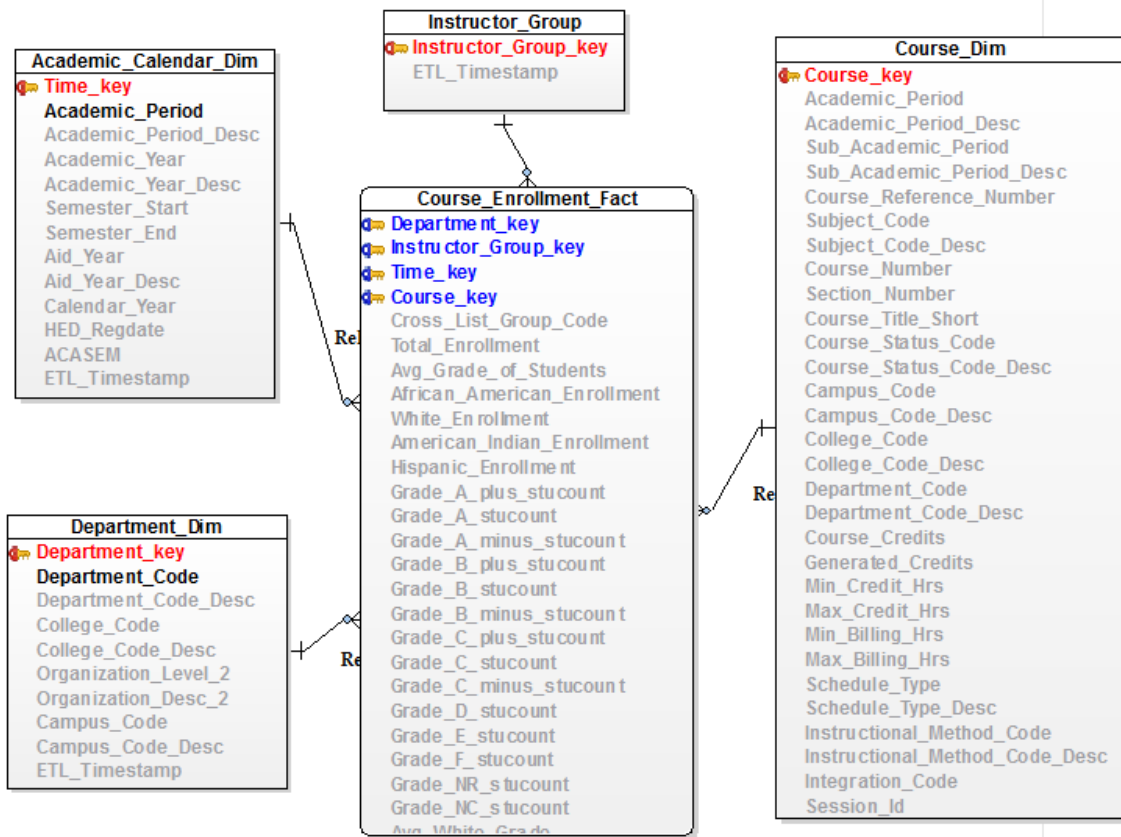


Figure 3.7: Course Enrollment Star Schema.

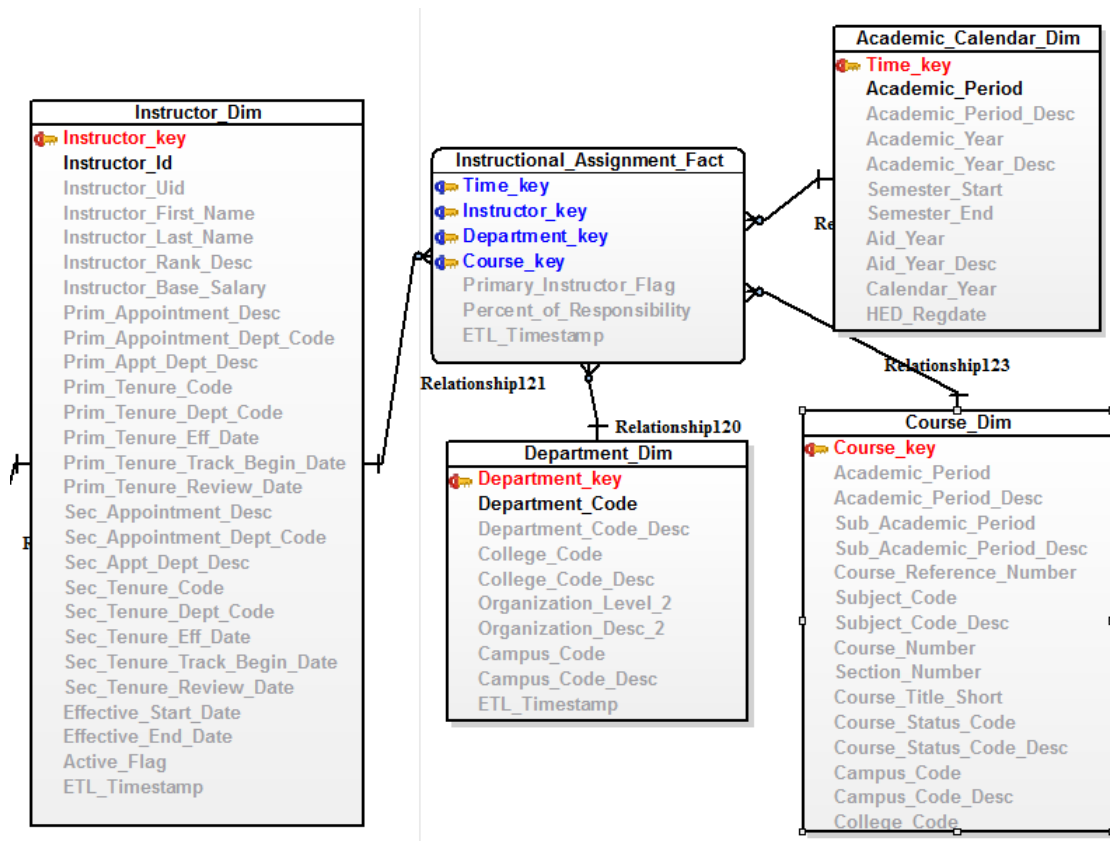


Figure 3.8: Instructor Course Assignment Fact Schema.



Chapter 3. Design and Implementation

Program_Enrollment_Fact	
Time_key	Number
Program	Varchar2(
Term	Varchar2(
Total_Enrollment	Number
Hispanic_Enrollment	Number
African_American_Enrollment	Number
American_Indian_Enrollment	Number
White_Enrollment	Number
ETL_Timestamp	Timestamp

STUDENT_LEVEL_SUMMARY_FACT	
PERSON_KEY	Number
STUDENT_ID	Varchar2(9 )
PERSON_UID	Number
NAME	Varchar2(255)
NO_OF_COURSES_ENROLLED	Number
GPA_TYPE	Varchar2(1 )
GPA_TYPE_DESC	Varchar2(11 )
ACADEMIC_STUDY_VALUE	Varchar2(63 )
ACADEMIC_STUDY_VALUE_DESC	Varchar2(255)
GPA	Number
GPA_CREDITS	Number
CREDITS_ATTEMPTED	Number
CREDITS_EARNED	Number
CREDITS_PASSED	Number
QUALITY_POINTS	Number
ETL_TIMESTAMP	Timestamp(6)

DegreesAwarded_Fact	
Time_key	Number
Biodemo_key	Number
Student_key	Number
Degree_Awarded	Varchar2(30 )
Final_GPA	Number(7,3)
Grad_Status	Varchar2(4 )
Campus_Code	Varchar2(3 )
College_Code	Varchar2(2 )
Department_Code	Varchar2(4 )
Program	Varchar2(12 )
Advisor_Id	Varchar2(9 )
ETL_Timestamp	Timestamp(6)

STUDENT_TERM_SUMMARY_FACT	
TIME_KEY	Number
PERSON_KEY	Number
STUDENT_KEY	Number
TERM	Varchar2(6 )
STUDENT_ID	Varchar2(9 )
PERSON_UID	Number
NO_OF_COURSES_ENROLLED	Number
GPA_TYPE	Varchar2(1 )
GPA_TYPE_DESC	Varchar2(11 )
ACADEMIC_STUDY_VALUE	Varchar2(63 )
ACADEMIC_STUDY_VALUE_DESC	Varchar2(255)
GPA	Number
GPA_CREDITS	Number
CREDITS_ATTEMPTED	Number
CREDITS_EARNED	Number
CREDITS_PASSED	Number
QUALITY_POINTS	Number
ETL_TIMESTAMP	Timestamp(6)

Figure 3.9: Summary Fact Tables.

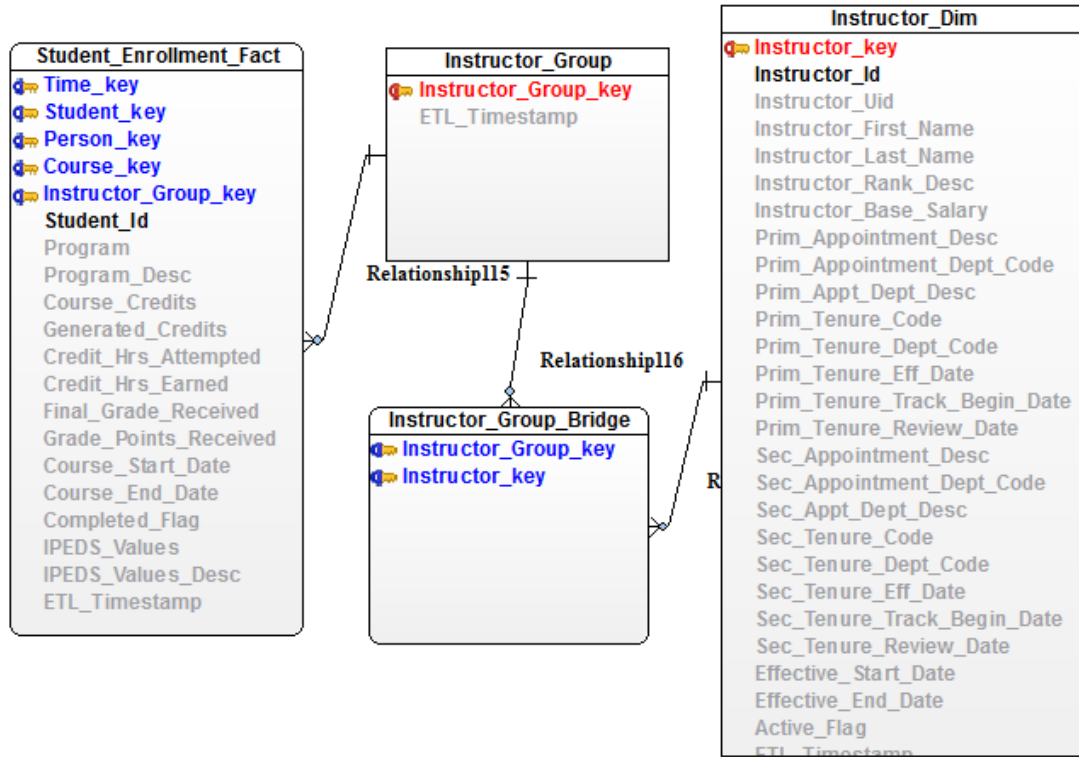


Figure 3.10: Instructor Bridge Table.

ORIGINAL RECORD			
Key	Person_Id	Name	Address
1	101	xyz	1 UNM, ABQ, NM
NEW RECORD			
Key	Person_Id	Name	Address
1	101	xyz	2 Gallup, NM

Figure 3.11: Slowly Changing Dimension Type 1.

Chapter 3. Design and Implementation

ORIGINAL RECORD						
Key	Person_Id	Name	Address	Eff.Start Date	Eff.End Date	Current_Flag
1	101	xyz	1 UNM, ABQ, NM	1/1/2010	9/9/2099	1
NEW RECORDS						
Key	Person_Id	Name	Address	Eff.Start Date	Eff.End Date	Current_Flag
1	101	xyz	1 UNM, ABQ, NM	1/1/2010	1/1/2013	0
2	101	xyz	2 Gallup, NM	1/2/2013	9/9/2099	1

Figure 3.12: Slowly Changing Dimension Type 2.

ORIGINAL RECORD				
Key	Person_Id	Name	SSN	
1	101	xyz	1000001	
NEW RECORD				
Key	Person_Id	Name	SSN	Prior SSN
1	101	xyz	123456789	1000001

Figure 3.13: Slowly Changing Dimension Type 3.

ORIGINAL RECORD							
Key	Person_Id	Name	Address	Eff.Start Date	Eff.End Date	SSN	Current_Flag
1	101	xyz	1 UNM, ABQ, NM	1/1/2010	9/9/2099	1000001	1
NEW RECORDS - Address change - SCD Type 2							
Key	Person_Id	Name	Address	Eff.Start Date	Eff.End Date	SSN	Current_Flag
1	101	xyz	1 UNM, ABQ, NM	1/1/2010	1/1/2013	1000001	0
2	101	xyz	2 Gallup, NM	2/1/2013	9/9/2099	1000001	1
NEW RECORDS - SSN change - SCD Type 1							
Key	Person_Id	Name	Address	Eff.Start Date	Eff.End Date	SSN	Current_Flag
1	101	xyz	1 UNM, ABQ, NM	1/1/2010	1/1/2013	1000001	0
2	101	xyz	2 Gallup, NM	2/1/2013	9/9/2099	123456789	1

Figure 3.14: Hybrid Slowly Changing Dimension.

# Chapter 4

## Extract Transform Load (ETL)

### 4.1 Populating the data mart

Extract transform and load (ETL) process is implemented after source tables have been analyzed and target tables have been designed and installed in the database. The SAS<sup>®</sup> data integration (DI) studio is the ETL software piece of the SAS<sup>®</sup> suite of products. It has a more complex learning curve compared to some other ETL tools like Pentaho or Talend; however, it offers more flexibility in scheduling parallel jobs and work flows.

### 4.2 Logical steps in ETL

The method used for the ETL process is based on the principles from [3] and the technical details of ETL implementation is based on methods defined in [7]. After the data mart has been implemented on the target database, staging tables are designed. Data from various sources including ODS, pre-award research data stored

## Chapter 4. Extract Transform Load (ETL)

in a database system called cayuse, space data stored in a database system called famis is brought into a staging area using ETL jobs. Here the data is cleansed and validated along with other quality checking and improvement processes as required. From the staging tables, data is brought into the data mart tables using SAS<sup>®</sup> data integration studio. First, records are inserted into the dimension tables from the respective staging tables along with ETL generated surrogate keys and then fact tables are loaded by capturing numerical columns from the staging tables and the corresponding surrogate keys from the dimension tables as foreign keys.

The first step in ETL is to define libraries in DI studio that contain connection information to back end database servers. After this, metadata or definitions of tables or views from the source and target environments are imported into the DI studio environment. As a next step, the staging tables are populated using simple transformations in DI studio. Further steps include loading the dimension tables and finally loading the fact tables.

The dimension tables are loaded before the fact tables. The reason behind this is that the fact table contains foreign keys to records from the dimension tables and needs to pick it up while being loaded. This helps to maintain referential integrity of the fact table i.e., a record exists in a fact table if and only if all the corresponding key records exist in the respective dimension tables.

**Source to target mapping:** In the DI studio, within some transformations like ‘table loader’, and ‘splitter’, we have an option to select what input columns we need and then map it to the respective output column. This is done automatically if the source and target column names are same, otherwise the columns need to be mapped manually using point-click and drag method. Figure 4.1 shows such an example.

**Populating a staging table:** Figure 4.2 demonstrates use of simple transformations like ‘extract’ and ‘insert rows’ to populate a staging table called

## Chapter 4. Extract Transform Load (ETL)

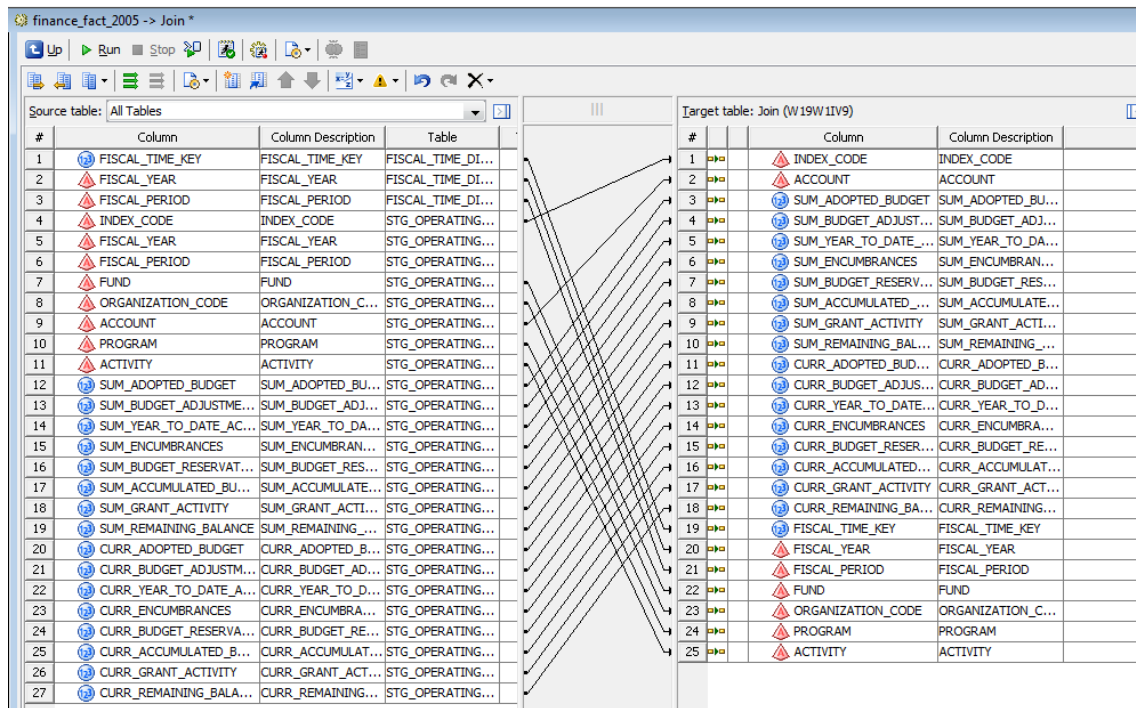


Figure 4.1: Source to Target mapping in SAS<sup>®</sup> DI Studio.

stg\_operating\_ledger\_cum using and ODS source table operating\_ledger\_cum.

Within the extract transformation there is an option to filter the rows extracted according our needs. This operation is similar to ‘select’ used along with a ‘where’ condition in plain structured query language (SQL).

**Loading a dimension table:** Figure 4.3 shows how a dimension table in a data mart is loaded. The source table is year\_type\_definition from the ODS and the target dimension table is academic\_calendar\_dim. In the first step, some simple SQL is run which truncates the dimension table, then the next steps show data extraction from the source table based on some filter conditions mentioned in the extract transformation. The SCD type 1 transformation is then used to generate a value for time\_key which is the surrogate key for the target table. This generates an integer incremented by 1 for each new record inserted into the table. The consequent

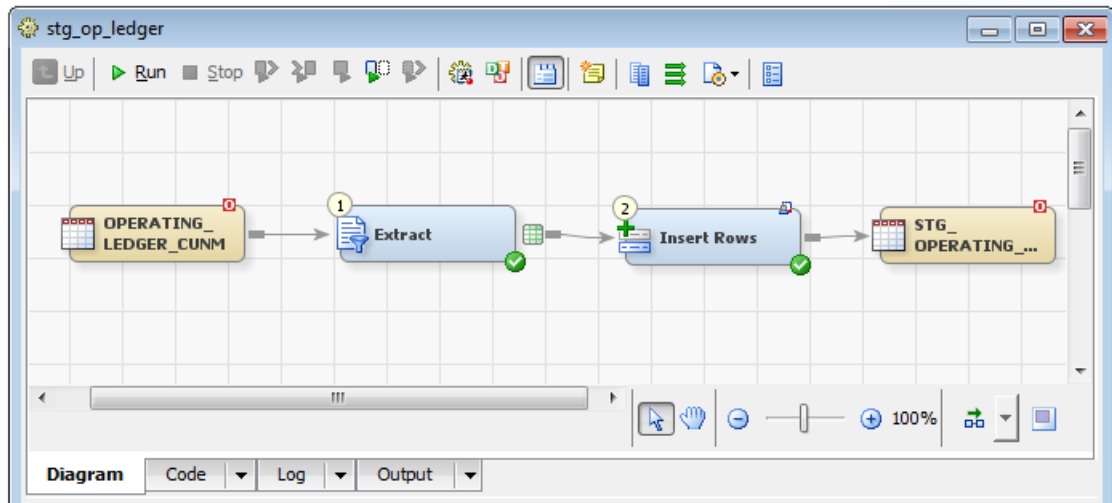


Figure 4.2: Populating a staging table using SQL transformations in SAS® DI Studio.

steps show execution of SQL for some calculated or derived fields in the target table.

**Loading a fact table:** The fact table contains the foreign keys to the respective

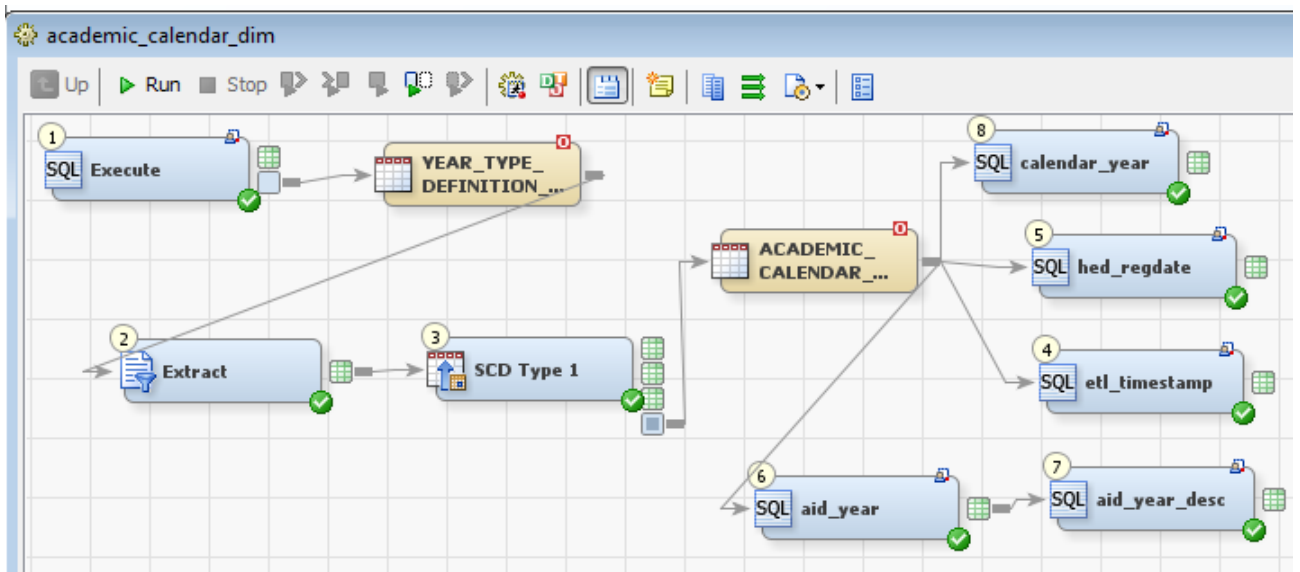


Figure 4.3: Populating a dimension table.

## Chapter 4. Extract Transform Load (ETL)

dimensions which is the reason why fact table needs to be loaded after all the dimension tables have been loaded. In Figure 4.4 an ETL job to populate a fact table is demonstrated. The target table is `finance_fact`. It can be noticed that the facts are coming from the staging table while the keys come from the dimension tables and are pulled using join or lookup transformations.

Joins and where conditions can be specified with in the DI studio, and Figure 4.5 shows such an example. Join types like left, right, full, inner, cross or union can be specified and clauses like where, group by, having, order by and sub-query can be given with in this GUI tool which makes complicated SQL easier to implement.

Generating a work flow in the DI studio is very intuitive. Once jobs are created, they can be run directly from the DI studio interface or can be deployed to a scheduling server. These jobs generate a `.sas` file which is the actual executable program run on the server. Once jobs are created in the DI studio, they are deployed to the scheduling server. Here a work flow is created and jobs are selected to run as part of the flow. In Figure 4.6 a complete flow of loading a star schema tables is shown. First all the staging tables are refreshed. When only all of them are successful, which is represented by the 'and' gate, the work flow moves on to the next jobs, which is to load the dimension tables. These jobs can also be run simultaneously, which can be seen in the comments `'start(index_dim)'` which means the next job kicks off as soon as the current one starts. And finally the fact table is loaded after all dimension tables have been loaded seen as `'done(account_dim)'` in the comments.

In DI studio, a work flow can be scheduled or run based on many conditions. Figure 4.7 shows the wide range of options available. It can be run manually any time, or it can be scheduled to run multiple times at specified points of time. Triggers can be programmed to run the flow at any time or file event including conditions like arrival of a file and increase in a file size.



The DI studio allows enough flexibility to run custom SAS code. This option is present in many transformations including execute, table-loader, splitter, data-transfer and surrogate-key-generator. Figure 4.8 shows a simple job containing an SQL extract transformation which contains SQL code wrapped around by proc-SQL which is a function used by SAS<sup>®</sup> to interpret SQL code. Figure 4.9 shows sample proc-SQL code. This code starts with a proc-SQL command. The option nosymbolgen specifies that log messages about macro variable references will not be displayed. Within the connect string, details like database, path which is defined in a SAS<sup>®</sup> library, user name and password are provided. The password can be encoded by using SAS<sup>®</sup> function called pwencode. The actual SQL statement is an execute statement. Control flow of transformations in a job in Figure 4.10 shows how order by which transformations are run can be changed by dragging them and placing them where desired. The control flow can then be validated with the press of a button. Job status shows progress of a job can be checked for any errors or warnings or successful completion at each stage or transformation. Figure 4.11 shows the status window in the DI studio GUI for the academic\_calendar\_dim job. The engine looks for any precode first and runs it if present and then follows through all the steps and checks for any postcode after the final step. Job statistics as shown in Figure 4.12 play an important role in optimization of jobs. It helps in understanding duration of each job, CPU time, memory occupied and threads on the server. Depending on the necessity, parameters in the job or on the server can be changed to allow more threads or more memory.

### 4.3 Refreshing the data mart

Currently SAS<sup>®</sup> scheduler within the management console is being used to schedule jobs and work flows depending on the refresh needs. There are two ways of refreshing

## Chapter 4. *Extract Transform Load (ETL)*

the data mart tables. Complete reload and incremental load.

The refresh strategy for each table may vary depending on how the source table data is refreshed. For example, in the case of person dimension table, we have one record per person and any change in the detail, is replicated by replacing old data. In such a situation, it is better to opt for a complete reload, instead of tracking and updating changes. Even in the case of a fact table for instance, it is more practical to re-build it each time. This will ensure proper referential integrity of the fact table and that the keys are pointing back to the correct record in the dimension table.

However in the case of student or course dimensions, the data in the respective source tables in the ODS is not manipulated. Only new records are added to them. Therefore an incremental load process may suffice here.

Data mart maintenance is relatively easier and cheaper as compared to a 3NF database because of fewer number of tables involved. A proper naming convention was also followed when naming fact, dimension or staging tables and indexes.

### **4.4 ETL performance and data validation**

The SAS<sup>®</sup> DI Studio was able to perform row inserts into Oracle tables at a rate of one million records per minute. This was achieved while running multiple jobs in parallel. After the data mart is populated, simple tests are run like record counts. More complex queries are also run to check performance and also data validation. Sample results are sent to business users for validation before moving jobs and data to production environment. After this stage, data is automatically refreshed by the time event or trigger events in the scheduling server.

Chapter 4. Extract Transform Load (ETL)

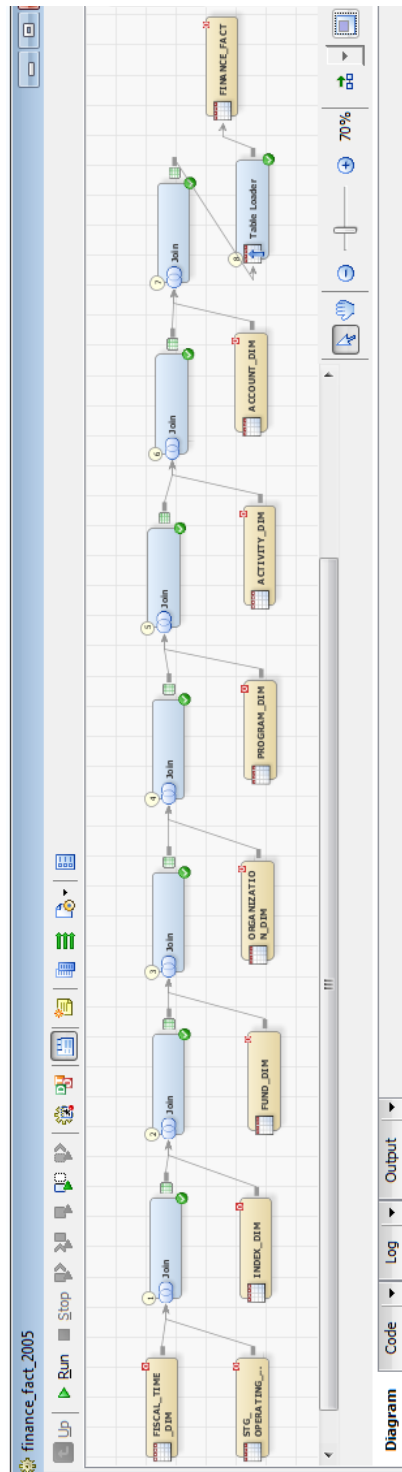


Figure 4.4: Populating a fact table - Finance Fact.

Chapter 4. Extract Transform Load (ETL)

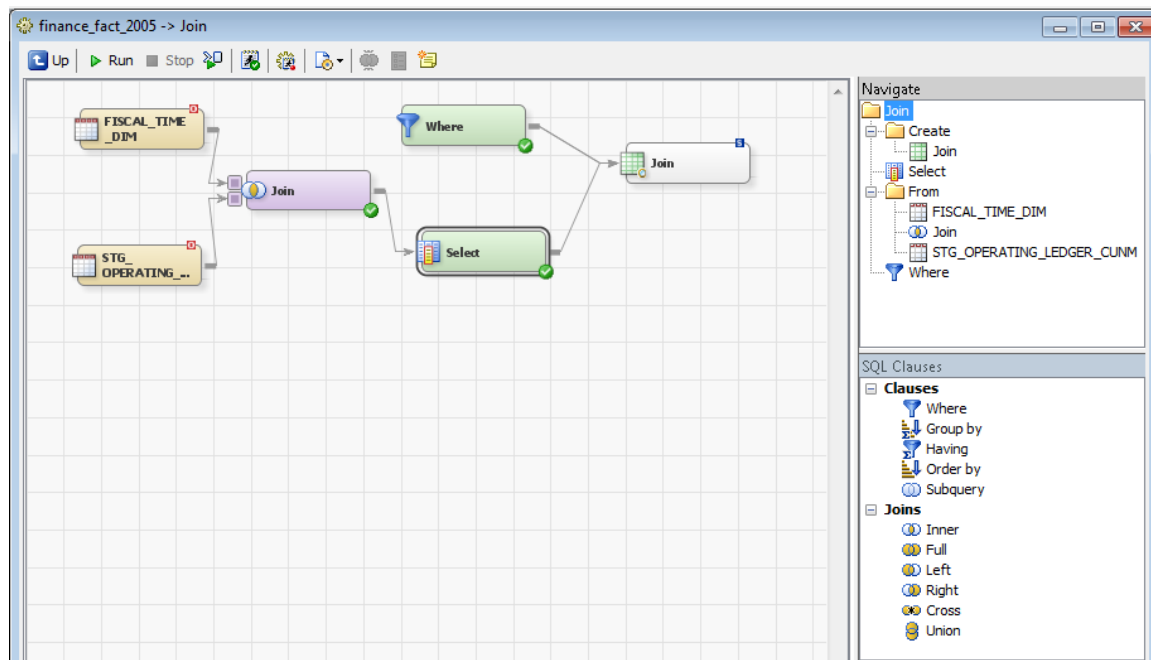


Figure 4.5: Joins and where conditions in SAS<sup>®</sup> DI Studio.

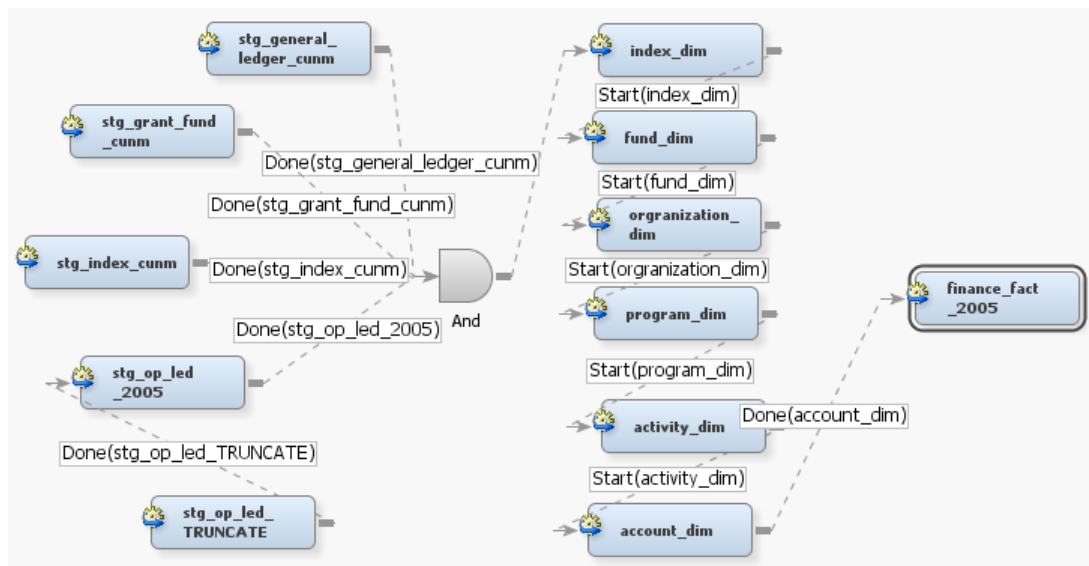


Figure 4.6: Work flow, parallel jobs and scheduling in SAS<sup>®</sup> DI Studio.

Chapter 4. Extract Transform Load (ETL)

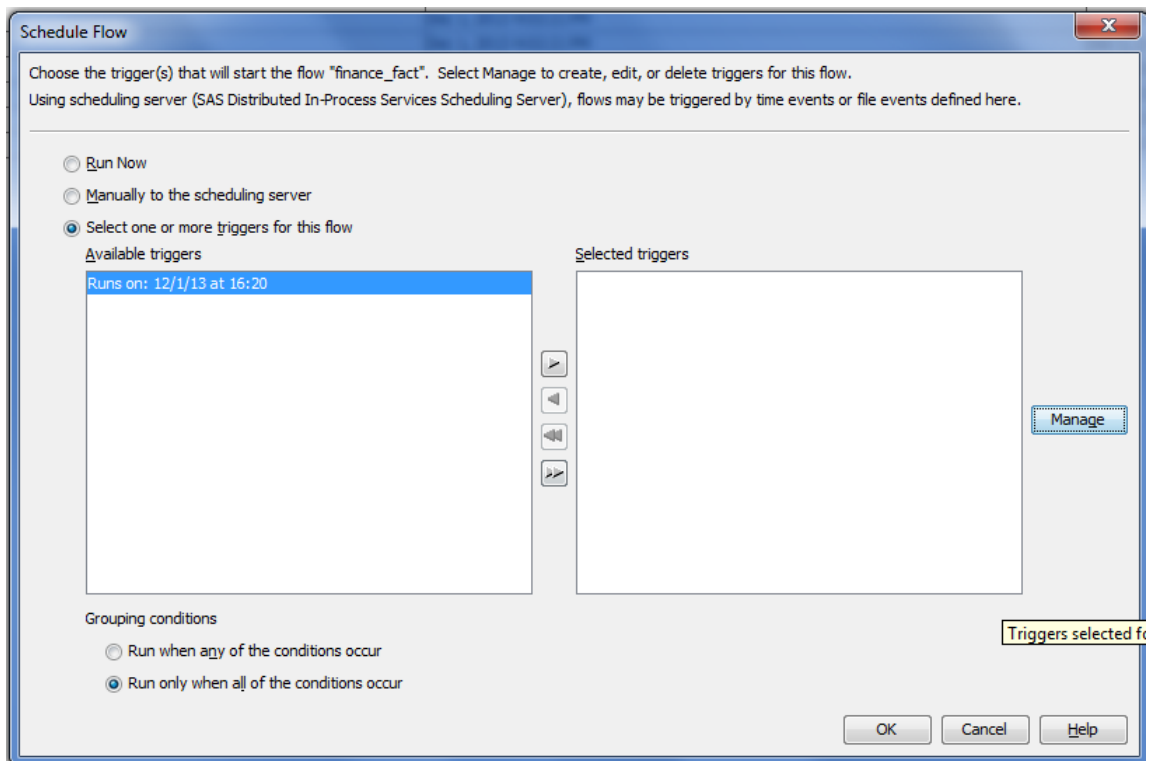


Figure 4.7: Job scheduling triggers in SAS® DI Studio.

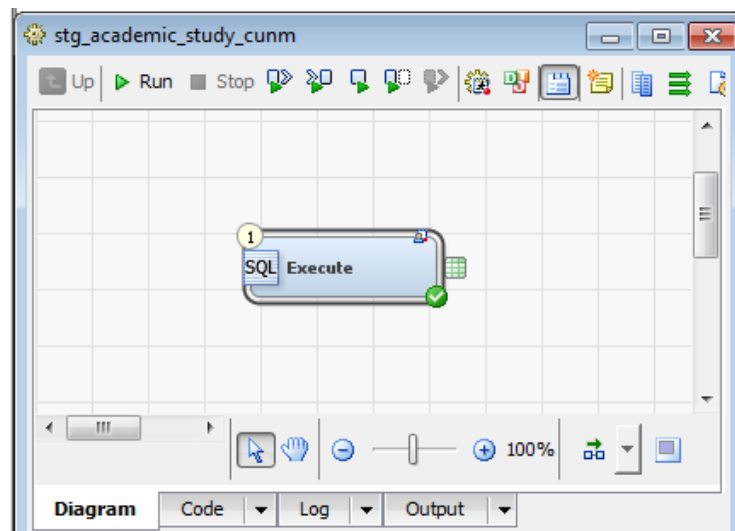


Figure 4.8: SQL Extract transform to run custom SQL.

## Chapter 4. Extract Transform Load (ETL)

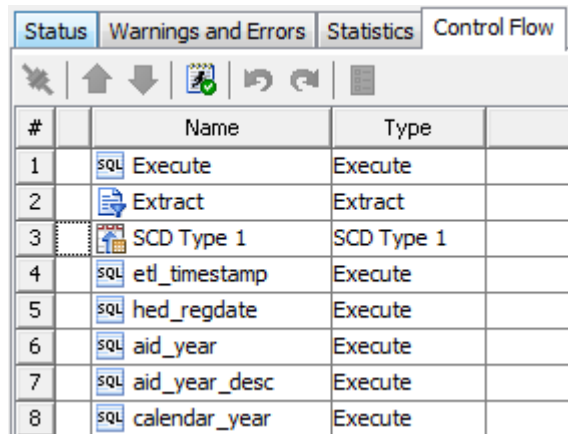
```
proc sql;
  options nosymbolgen;
  connect to oracle (user=unm_oia_staging
    password="{SAS002}B38A003D0E37DB4F3E619F7838AD52CD" PATH=odst);

  execute (
    DROP TABLE unm_oia_staging.STG_ACADEMIC_STUDY_CUNM
  ) by oracle;

  execute (
  CREATE TABLE unm_oia_staging.STG_ACADEMIC_STUDY_CUNM
  ( Student_Id,
    Person_Uid,
    NAME,
    ACADEMIC_YEAR,
    ACADEMIC_PERIOD, , , , . )
  AS SELECT id,
    person_uid,
    name,
    academic_year,
    academic_period, , , , .
    FROM ODSMGR.ACADEMIC_STUDY_CUNM@OIA_ODST_ODSP.UNM.EDU
    WHERE ACADEMIC_PERIOD >= '200680' ) by oracle;

  disconnect from oracle;
quit;
```

Figure 4.9: Proc-SQL example to run custom queries.



The screenshot shows the 'Control Flow' tab in SAS. It displays a sequence of 8 transformations in a table. The table has columns for '#', 'Name', and 'Type'. The transformations are: 1. Execute (SQL), 2. Extract, 3. SCD Type 1, 4. etl\_timestamp (SQL), 5. hed\_regdate (SQL), 6. aid\_year (SQL), 7. aid\_year\_desc (SQL), and 8. calendar\_year (SQL). Each transformation has a small icon to its left representing its type.

#	Name	Type
1	Execute	Execute
2	Extract	Extract
3	SCD Type 1	SCD Type 1
4	etl_timestamp	Execute
5	hed_regdate	Execute
6	aid_year	Execute
7	aid_year_desc	Execute
8	calendar_year	Execute

Figure 4.10: Control flow of transformations in a job.

Chapter 4. Extract Transform Load (ETL)

Order	Name	Status
1	Precode	Completed successfully
2	SQL Execute	Completed successfully
3	Extract	Completed successfully
4	SCD Type 1	Completed successfully
5	SQL etl_timestamp	Completed successfully
6	SQL hed_regdate	Completed successfully
7	SQL aid_year	Completed successfully
8	SQL aid_year_desc	Completed successfully
9	SQL calendar_year	Completed successfully
10	Postcode	Completed successfully
	academic_calendar_dim	Completed successfully

Figure 4.11: Job status.

Duration	CPU Time	Current Memory	System Memory	Current I/O	System I/O	Server	Threads
0.022	0.03	13721.6	12476.416	32	2520	bansasd	11
0.593	0.28	13983.744	12738.56	1264	3784	bansasd	11
0.819	0.31	0	12738.56	0	3784		11

Figure 4.12: Job statistics.

# Chapter 5

## Reporting from the data mart

### 5.1 Data Mart Performance

The data mart was designed and implemented using a dimensional modeling method namely star schema. A star schema is essentially easy to understand, implement and maintain. As seen in the previous chapters, a star schema has a fact table and a few dimension tables. The student data mart has several subject areas like student enrollment, course enrollment, admissions, financial aid and so on. Each subject area has a fact table with dimension tables shared between these multiple subject areas. Such tables are called conformed dimensions. For example the student enrollment star schema has one fact table `student_enrollment_fact` and dimension tables like `academic_calendar_dim`, `student_dim`, `course_dim` etc.. The naming convention has been chosen in such a way that it is apparent as to what data is present in that table. A dimension table contains comprehensive information about that aspect of the subject area. For example, `course_dim` table contains all information regarding a course like college, department, course reference number, academic period in which it was offered, section number and so on. The fact table contains transaction level



## *Chapter 5. Reporting from the data mart*

data capturing all events regarding that subject area.

As each subject area is defined and the tables designed, it becomes very clear as to what tables need to be used for a given query to that subject area. Reports can be generated very easily based on a proper understanding of the data mart structure.

A fact table may contain many records, but because it contains a very small number of columns querying the fact table is much faster. This is one of the main reasons why the data mart should be able to perform better than the ODS in response to report generation requests. More reasons are mentioned below.

**Logical design:** The star schema design requires fewer number of joins to build a report as compared to the number of joins required to build the same report from ODS tables.

**Indexing appropriate columns:** Columns in the tables of this star schema are indexed optimally according the query or report requirements. Whereas in ODS, the tables if they are indexed it is not done optimally. Also most of the time, views rather than tables in ODS are used to build reports by most users at UNM, which are slower to access.

**Using surrogate keys:** We use just this one numerical field to join a fact and a dimension table as compared to using multiple fields to join tables in ODS. For instance, when querying course level statistics, the fact table used is `course_enrollment_fact` which is joined to dimension table `academic_calendar_dim` on `time_key` and to `course_dim` on `course_key`. `Time_key` and `course_key` are numerical fields generated by the ETL job having unique values for each record in a dimension table.

**Using summary fact tables:** In reports which require the use of aggregate functions, summary fact tables are built with the aggregate fields pre-populated so that the reports can directly read from these tables avoiding calculations on-the-fly

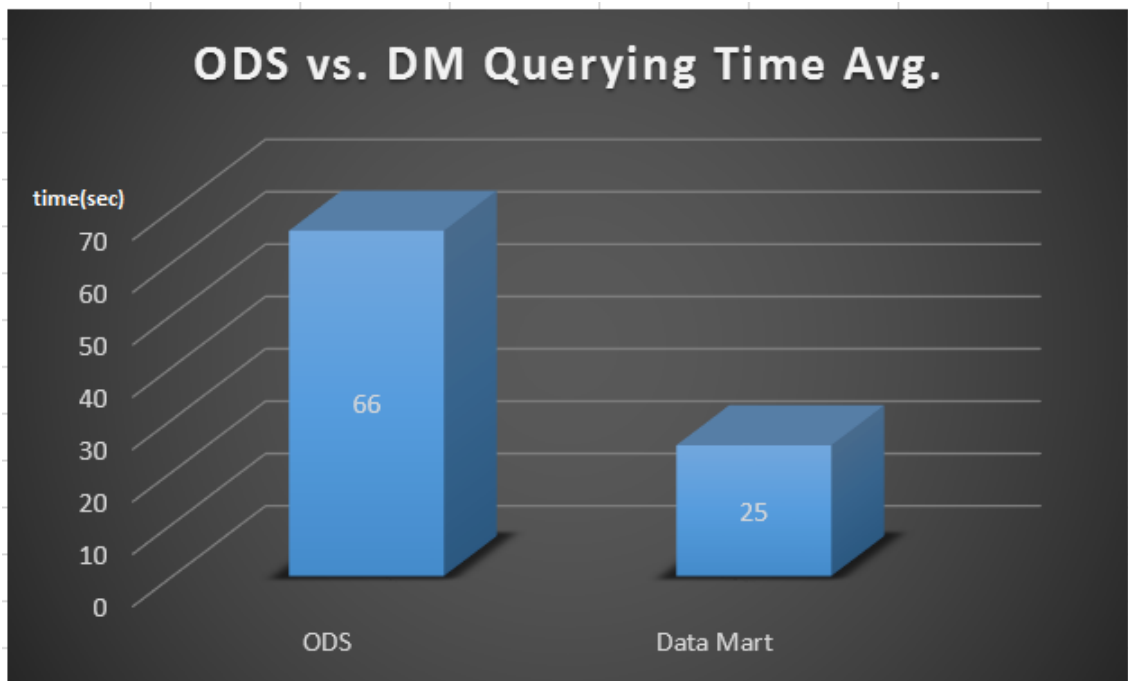


Figure 5.1: Average ODS versus data mart performance.

thus ensuring faster delivery of reports.

I received response the times for the SQL queries against both ODS and the data mart. These queries were based the grade distribution report and also a few other reports which gives grade distributions and pass rates for courses, section, ethnicity. Figure 5.1 shows an averaged performance of data mart versus ODS over different queries. The data mart performed better than the ODS. On an average, the data mart is more than 50% faster than the ODS.

## 5.2 Reports

The purpose building a data mart using star schema is to enable the development of a wide variety of reports and predictive analytics. This is possible because of

## *Chapter 5. Reporting from the data mart*

the flexible way in which the data is organized. It is the same data as in ODS but organized in a different, more logical way. It is also relatively easy to develop reports out of the data mart as compared to ODS because of the fewer number of tables involved.

Several specific reports were proposed to be generated for a project based on this student mart. Some have been built while others are still in development. Below is a non-exhaustive description of a few of them. The WebFOCUS reporting tool is being used to generate these reports.

**General ad-hoc class performance report:** For a given semester, this report allows one to get grade distributions, withdrawal rates and pass rates for selected courses and sections. Courses can be selected by instructor, course reference number, or by selecting a series of courses and sections. This report will also need to have the ability to restrict its output to various sub-populations based on ethnicity, financial aid, or other demographic conditions. Inputs can be course reference number, instructor, academic period, subject code, course number, section number, ethnicity, financial aid status, category of financial aid received, status as a first generation college student (if possible) Outputs are grade distribution, withdrawal rate and pass rate. As an example of demonstrating the simplicity of building a report from the data mart, in this report, dimension tables used are `course_dim`, `academic_calendar_dim`, `student_dim` and the fact table used is `course_enrollment_fact` joined on the surrogate keys `time_key`, `student_key` and `course_key`. Therefore with relatively simple SQL joins and select operations, this complex report was built enabling drill-down and drill-across capabilities which allow dynamic reporting based on selecting academic period, department, course and section.

**Two course grade comparison report:** This report will compare how students in one course during one semester performed in another course during a subsequent semester (e.g. Physics I during Fall of 2012 and Physics II during Spring of

## *Chapter 5. Reporting from the data mart*

2013). This report will also be able to look at sub-populations in a similar manner as the previous report.

A few other reports that are supported by the data mart are mentioned below:

- A report that shows the transitions to and the transitions from a given major. For a given time period, it shows the number and majors of students that transferred into a given major. Also for a given time period, it shows the number and majors of students who transferred from a given major.
- A report that gives the percentage of STEM classes taken by STEM majors, broken down by major for a given academic period.
- A report that gives the number of degrees granted, broken down by major and level of degree including average GPA for a given semester.
- A report that gives the number and percentage of non-STEM majors in STEM courses broken down by course.

Figure 5.2 is a report which shows student credit hours (sch) generated by faculty in each academic period by college. The report in Figure 5.3 shows student enrolled hours in an academic period by college and by course level, i.e., lower undergraduate, upper undergraduate or graduate.

Chapter 5. Reporting from the data mart

ACADEMIC\_PERIOD\_DESC: Spring 2011 COLLEGE\_CODE\_DESC: Anderson Schools of Management

Run Reset Save Clear Output  Run in a new window

College	Rank	Instructor Name	Level		
			Lower	Upper	Grad
Anderson Schools of Management	Assistant Professor	Lee, Kim	.	123	123
		Mathis, Rajarajkumar V	.	207	.
		McKnight, Reed	.	267	105
		Montoya, Manuel	.	300	.
		Horales-Camargo, Emmanuel	.	36	60
		Patterson, Karen	.	81	57
		Taylor, Scott	.	102	135
	Associate Professor	Berman, Shari	.	222	81

Figure 5.2: SCH generated by faculty by academic period, college.

Academic\_Period: Spring 2008 Campus: Main Run

College	Department	Course Level		
		Lower	Upper	Graduate
Anderson Schools of Management ASM	Anderson Schools of Management ASM	1,248	12,304	3,814
<b>Subtotal: Anderson Schools of Management ASM</b>		<b>1,248</b>	<b>12,304</b>	<b>3,814</b>
Associate VP Student Services	Air Force ROTC	71	70	.
	Army ROTC	86	151	.
	Naval ROTC	21	81	.
<b>Subtotal: Associate VP Student Services</b>		<b>178</b>	<b>302</b>	<b>.</b>
College of Arts Sciences A S	*Interdisciplinary: A.S.	99	891	47
	African American Studies	99	177	.
	American Studies Department	1,146	954	290

Figure 5.3: Student hours enrolled by academic period, college.

# Chapter 6

## Future Work

The student data mart being discussed in this thesis currently has a star schema that incorporates fact tables for applications and admissions, student enrollment, course enrollment, semester performance and financial aid. In the near future, we also look to incorporate curriculum data, student assessment data, career services and alumni data. Having this kind of data from high school GPA and admission test scores, academic performance at UNM and how well students are able to move onto a job or higher studies helps us to better understand and measure student success.

A finance data mart is also being built to analyze the flow of money through various organizations for various activities. The finance reports and applications for the president's and the provost's office as well as individual departments will be supported by this data mart.

In the near future, the University of New Mexico plans to build a research space data mart which will have information about the research space allotted to each principal investigator bringing in funding to UNM. This data mart includes the investigator data, research proposal, grant, fund, organization, program, account and activity data. It would also contain the space data which includes what space

## *Chapter 6. Future Work*

was allotted to which researcher and how it was used during a certain period of time. This data mart would give UNM scope to do analytics on how efficiently space is being used on campus, improve space allotment and also predict future usage.

The ETL process will be automated further to support nightly updates of the data mart and automatic report generation and publication. There is scope to take leverage of the built-in transformations in the SAS<sup>®</sup> data integration studio to support dimensional modeling, handle many-to-many relationships and slowly changing dimensions. Apart from that it includes top of the line tools which will help build interesting analytic models including forecasting, what-if analysis and predictive analytic models. Reports will be built using SAS<sup>®</sup> visual analytics (VA).

Sankey diagrams shown below were developed as a part of another analytics initiative and are currently run on Amazon EC2 cloud with a Postgres database as the back end. The data was loaded into this database using Excel spreadsheets. In future, these Sankey diagrams will be supported by the data marts.

The Sankey diagrams can be viewed at [www.provostcloud.unm.edu](http://www.provostcloud.unm.edu). The first example shown in Figure 6.1 is the college flow of majors by semester in 2008, Arts and Sciences college, Chemistry department. There are a bunch of filters and statistics provided at the bottom of the page to drill down to the user's needs. The second sankey diagram in Figure 6.2 highlights part of the flow when one of the nodes is hovered upon by the mouse. This uses Data-Driven Documents (D3) Javascript programming. At the bottom of the web page there are some filters and the corresponding statistics displayed. This is shown in Figure 6.3.

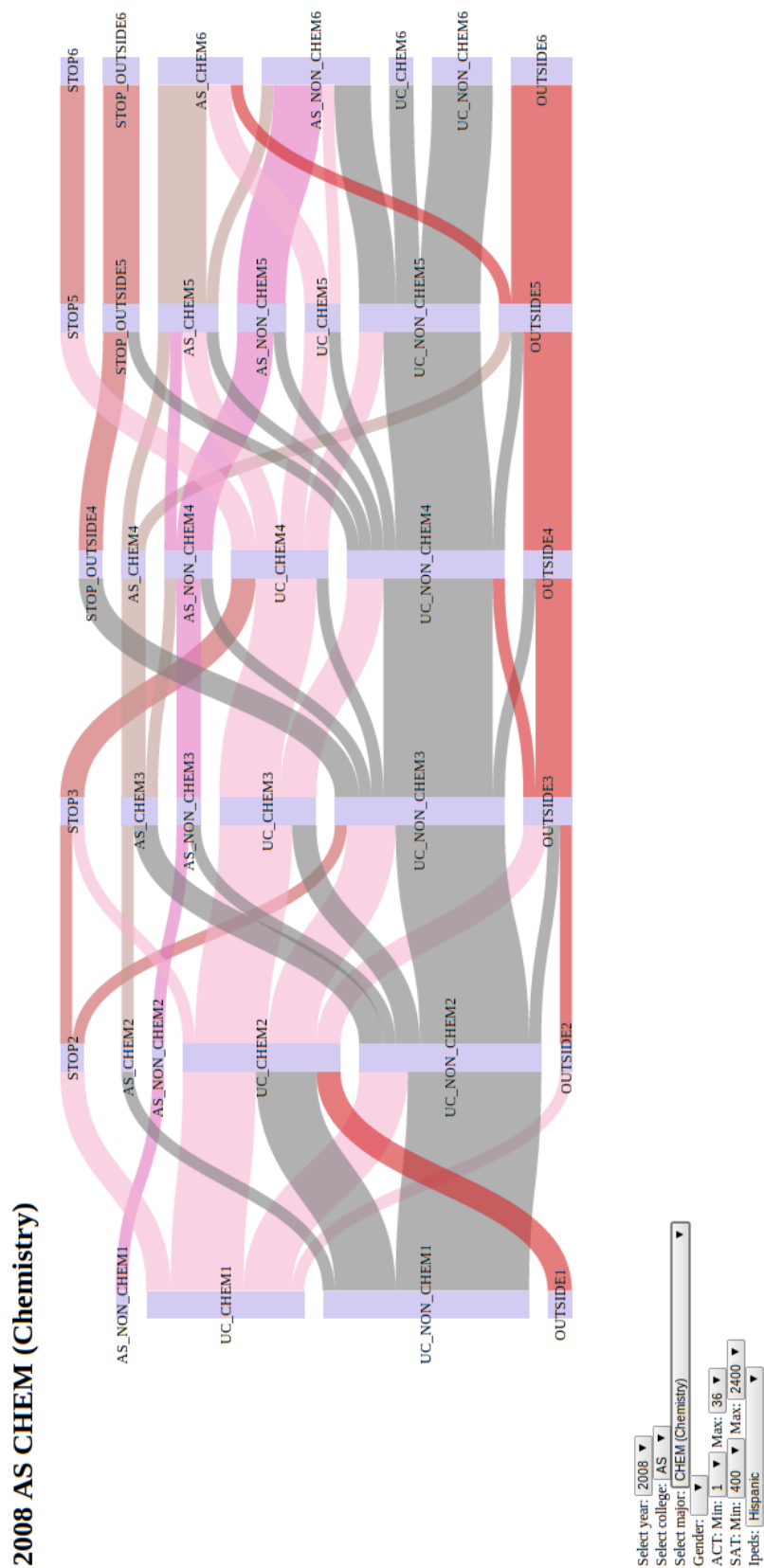
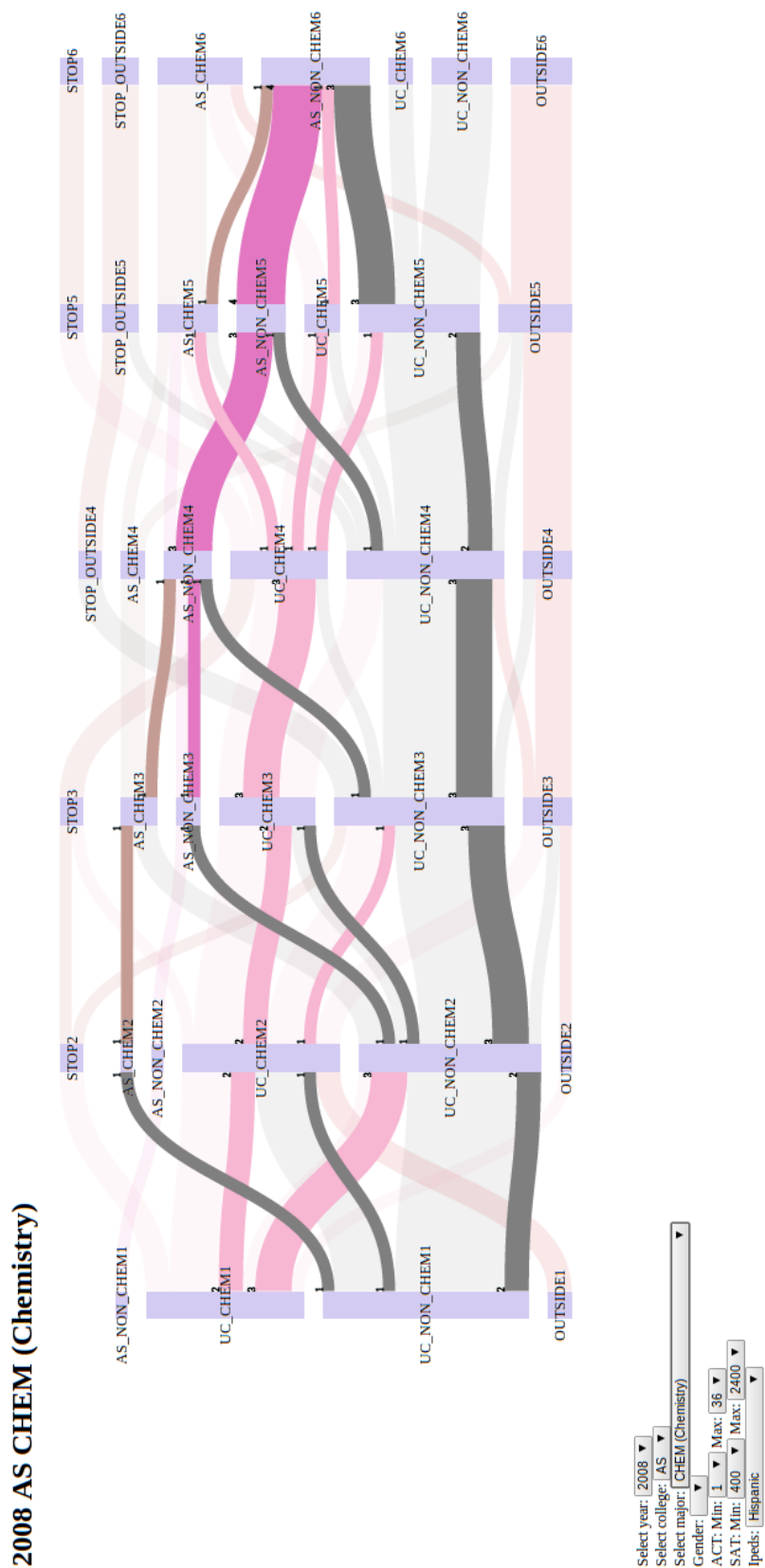


Figure 6.1: Sankey Diagram for 2008 Arts and Sciences Chemistry College Flow.





51  
Figure 6.2: Sankey Diagram node highlight example.

Chapter 6. Future Work

Select year: 2008 ▼  
 Select college: AS ▼  
 Select major: CHEM (Chemistry) ▼  
 Gender: ▼  
 ACT: Min: 1 ▼ Max: 36 ▼  
 SAT: Min: 400 ▼ Max: 2400 ▼  
 Iped: Hispanic ▼

	Semester						
Year grad rate							
Type	semester 1	semester 2	semester 3	semester 4	semester 5	semester 6	total
STOP	0 (0.0%)	2 (6.1%)	2 (6.1%)	0 (0.0%)	2 (6.1%)	2 (6.1%)	2
STOP_OUTSIDE	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (6.1%)	3 (9.1%)	3 (9.1%)	3
AS_CHEM	0	1	3	2	5	7	7
AS_NON_CHEM	1	1	2	4	4	9	9
UC_CHEM	13	13	8	8	3	2	2
UC_NON_CHEM	17	15	14	13	10	5	5
OUTSIDE	2	1	4	4	6	5	5
GRAD_OUTSIDE	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0
GRAD	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0

Figure 6.3: Sankey diagram statistics display.

# Appendices

## 6.1 Student data mart table structure (DDL)

The table structure of the student data mart designed, implemented and in production environment, is given below.

```
CREATE TABLE ACADEMIC_CALENDAR_DIM
(
    TIME_KEY                NUMBER,
    ACADEMIC_PERIOD         VARCHAR2(63 CHAR) ,
    ACADEMIC_PERIOD_DESC   VARCHAR2(255 CHAR) ,
    ACADEMIC_YEAR          VARCHAR2(63 CHAR) ,
    ACADEMIC_YEAR_DESC     VARCHAR2(255 CHAR) ,
    SEMESTER_START         DATE,
    SEMESTER_END           DATE,
    AID_YEAR               VARCHAR2(63 CHAR) ,
    AID_YEAR_DESC         VARCHAR2(255 CHAR) ,
    CALENDAR_YEAR          NUMBER(4) ,
    HED_REGDATE            VARCHAR2(5 CHAR) ,
    ACASEM                 VARCHAR2(30 CHAR) ,
    ETL_TIMESTAMP          TIMESTAMP(6)
```

Chapter 6. Future Work

);

CREATE TABLE COURSE\_DIM

(

COURSE_KEY	NUMBER,
ACADEMIC_PERIOD	VARCHAR2(64 CHAR) ,
ACADEMIC_PERIOD_DESC	VARCHAR2(255 CHAR) ,
SUB_ACADEMIC_PERIOD	VARCHAR2(12 CHAR) ,
SUB_ACADEMIC_PERIOD_DESC	VARCHAR2(50 CHAR) ,
COURSE_REFERENCE_NUMBER	VARCHAR2(5 CHAR) ,
SUBJECT_CODE	VARCHAR2(4 CHAR) ,
SUBJECT_CODE_DESC	VARCHAR2(30 CHAR) ,
COURSE_NUMBER	VARCHAR2(5 CHAR) ,
SECTION_NUMBER	VARCHAR2(3 CHAR) ,
COURSE_TITLE_SHORT	VARCHAR2(30 CHAR) ,
COURSE_STATUS_CODE	VARCHAR2(63 CHAR) ,
COURSE_STATUS_CODE_DESC	VARCHAR2(255 CHAR) ,
CAMPUS_CODE	VARCHAR2(63 CHAR) ,
CAMPUS_CODE_DESC	VARCHAR2(255 CHAR) ,
COLLEGE_CODE	VARCHAR2(63 CHAR) ,
COLLEGE_CODE_DESC	VARCHAR2(255 CHAR) ,
DEPARTMENT_CODE	VARCHAR2(63 CHAR) ,
DEPARTMENT_CODE_DESC	VARCHAR2(255 CHAR) ,
GENERATED_CREDITS	NUMBER(9 ,3) ,
MIN_CREDIT_HRS	NUMBER(7 ,3) ,
MAX_CREDIT_HRS	NUMBER(7 ,3) ,
MIN_BILLING_HRS	NUMBER(7 ,3) ,
MAX_BILLING_HRS	NUMBER(7 ,3) ,

Chapter 6. Future Work

```
SCHEDULE_TYPE                VARCHAR2(3 CHAR) ,
SCHEDULE_TYPE_DESC           VARCHAR2(30 CHAR) ,
INSTRUCTIONAL_METHOD_CODE     VARCHAR2(8 CHAR) ,
INSTRUCTIONAL_METHOD_CODE_DESC VARCHAR2(255 CHAR) ,
INTEGRATION_CODE             VARCHAR2(5 CHAR) ,
SESSION_ID                   VARCHAR2(8 CHAR) ,
COURSE_LEVEL                 VARCHAR2(2 CHAR) ,
COURSE_LEVEL_DESC            VARCHAR2(120 CHAR) ,
CROSS_LIST_GROUP_CODE        VARCHAR2(2 CHAR) ,
ACTUAL_ENROLLMENT            NUMBER(4) ,
PREV_ENROLLMENT              NUMBER(4) ,
COURSE_FEES                  NUMBER(12,2) ,
COURSE_START_DATE            DATE,
COURSE_END_DATE              DATE,
CENSUS_ENROLLMENT_DATE1      DATE,
CENSUS_ENROLLMENT1          NUMBER(4) ,
CENUS_ENROLLMENT_DATE2      DATE,
CENSUS_ENROLLMENT2          NUMBER(4) ,
EFFECTIVE_START_DATE         DATE,
EFFECTIVE_END_DATE           DATE,
GRADE_TYPE                  CHAR(20 CHAR) ,
GRADE_TYPE_DESC              CHAR(40 CHAR) ,
ACTIVE_FLAG                  NUMBER,
ETL_TIMESTAMP                TIMESTAMP(6)
);

CREATE TABLE COURSE_ENROLLMENT_FACT
(
```

Chapter 6. Future Work

TIME_KEY	NUMBER,
COURSE_KEY	NUMBER,
DEPARTMENT_KEY	NUMBER,
INSTRUCTOR_GROUP_KEY	NUMBER,
CROSS_LIST_GROUP_CODE	VARCHAR2(2 CHAR) ,
TOTAL_ENROLLMENT	NUMBER,
A_PLUS_COUNT	NUMBER,
A_COUNT	NUMBER,
A_MINUS_COUNT	NUMBER,
B_PLUS_COUNT	NUMBER,
B_COUNT	NUMBER,
B_MINUS_COUNT	NUMBER,
C_PLUS_COUNT	NUMBER,
C_COUNT	NUMBER,
C_MINUS_COUNT	NUMBER,
D_PLUS_COUNT	NUMBER,
D_COUNT	NUMBER,
D_MINUS_COUNT	NUMBER,
F_COUNT	NUMBER,
CR_COUNT	NUMBER,
NR_COUNT	NUMBER,
NC_COUNT	NUMBER,
W_COUNT	NUMBER,
WP_COUNT	NUMBER,
WF_COUNT	NUMBER,
WNC_COUNT	NUMBER,
W_TOTAL_COUNT	NUMBER,
AUD_COUNT	NUMBER,

Chapter 6. Future Work

```
WHITE_COUNT          NUMBER,
AFRICAN_AMERICAN_COUNT  NUMBER,
AMERICAN_INDIAN_COUNT  NUMBER,
HISPANIC_COUNT        NUMBER,
ASIAN_COUNT           NUMBER,
NATIVE_HAWAIIAN_COUNT  NUMBER,
TWO_MORE_RACES_COUNT  NUMBER,
RACE_ETHN_UNKOWN_COUNT NUMBER,
NON_RES_COUNT         NUMBER,
ETL_TIMESTAMP         TIMESTAMP(6)
);
```

CREATE TABLE DEGREESAWARDED.FACT

```
(
  TIME_KEY          NUMBER,
  STUDENT_KEY       NUMBER,
  PERSON_KEY        NUMBER,
  PERSON_UID        NUMBER,
  STUDENT_ID        VARCHAR2(9 CHAR) ,
  DEGREE_CODE       VARCHAR2(63 CHAR) ,
  DEGREE_CODE_DESC  VARCHAR2(255 CHAR) ,
  AWARD_CATEGORY    VARCHAR2(63 CHAR) ,
  AWARD_CATEGORY_DESC VARCHAR2(255 CHAR) ,
  STATUS_CODE       VARCHAR2(63 CHAR) ,
  STATUS_CODE_DESC  VARCHAR2(255 CHAR) ,
  OUTCOMEAWARDED_IND VARCHAR2(1 CHAR) ,
  GRADUATED_IND     VARCHAR2(1 CHAR) ,
  TRANSFER_WORK_EXISTS_IND VARCHAR2(1 CHAR) ,
```

Chapter 6. Future Work

```
    OUTCOME_GRADUATION_DATE    DATE,
    ACADEMIC_PERIOD            VARCHAR2(63 CHAR) ,
    ACADEMIC_PERIOD_GRADUATION VARCHAR2(63 CHAR) ,
    GRADUATION_STATUS          VARCHAR2(3 CHAR) ,
    GRADUATION_STATUS_DESC     VARCHAR2(255 CHAR) ,
    CREDITS_ATTEMPTED          NUMBER,
    CREDITS_EARNED              NUMBER,
    GPA_CREDITS                 NUMBER,
    QUALITY_POINTS              NUMBER,
    CREDITS_PASSED              NUMBER,
    GPA                          NUMBER,
    ETL_TIMESTAMP               TIMESTAMP(6)
);
```

```
CREATE TABLE DEPARTMENT_DIM
(
    DEPARTMENT_KEY              NUMBER,
    DEPARTMENT_CODE             VARCHAR2(63 CHAR) ,
    DEPARTMENT_CODE_DESC       VARCHAR2(255 CHAR) ,
    COLLEGE_CODE                VARCHAR2(63 CHAR) ,
    COLLEGE_CODE_DESC          VARCHAR2(255 CHAR) ,
    CAMPUS_CODE                 VARCHAR2(63 CHAR) ,
    CAMPUS_CODE_DESC           VARCHAR2(255 CHAR) ,
    ORGANIZATION_LEVEL_2       VARCHAR2(63 CHAR) ,
    ORGANIZATION_DESC_2        VARCHAR2(255 CHAR) ,
    ETL_TIMESTAMP               TIMESTAMP(6)
);
```



Chapter 6. Future Work

```
CREATE TABLE FINANCIAL_AID_FACT
(
    TIME_KEY                NUMBER,
    STUDENT_KEY              NUMBER,
    FINANCIAL_AID_KEY        NUMBER,
    STUDENT_ID               VARCHAR2(9 CHAR) ,
    PERSON_UID               NUMBER,
    AID_YEAR                  VARCHAR2(63 CHAR) ,
    CAMPUS_CODE              VARCHAR2(63 CHAR) ,
    AID_YEAR_DESC            VARCHAR2(255 CHAR) ,
    AWARD_STATUS             VARCHAR2(63 CHAR) ,
    AWARD_STATUS_DESC        VARCHAR2(255 CHAR) ,
    AWARD_STATUS_DATE        DATE,
    AWARD_OFFER_IND          VARCHAR2(63 CHAR) ,
    AWARD_ACCEPT_IND         VARCHAR2(63 CHAR) ,
    AWARD_DECLINE_IND        VARCHAR2(63 CHAR) ,
    AWARD_CANCEL_IND         VARCHAR2(63 CHAR) ,
    AWARD_OFFER_AMOUNT       NUMBER,
    AWARD_ACCEPT_AMOUNT      NUMBER,
    AWARD_PAID_AMOUNT        NUMBER,
    PELL_GRANT_RECEIVED_FLAG NUMBER,
    PELL_ELIGIBLE_FLAG       NUMBER,
    ETL_TIMESTAMP            TIMESTAMP(6)
);
```

```
CREATE TABLE FINANCIAL_AID_FUND_DIM
(
    FINANCIAL_AID_FUND_KEY    NUMBER,
```

Chapter 6. Future Work

```
AID_YEAR                VARCHAR2(63 CHAR) ,
AID_YEAR_DESC           VARCHAR2(255 CHAR) ,
FUND                    VARCHAR2(63 CHAR) ,
FUND_TITLE              VARCHAR2(255 CHAR) ,
FUND.SOURCE             VARCHAR2(63 CHAR) ,
FUND.SOURCE_DESC       VARCHAR2(255 CHAR) ,
FUND.TYPE_CODE         VARCHAR2(63 CHAR) ,
FUND.TYPE_DESC         VARCHAR2(255 CHAR) ,
GIFT_OR_SELF_HELP_AID  VARCHAR2(63 CHAR) ,
GIFT_OR_SELF_HELP_AID_DESC VARCHAR2(255 CHAR) ,
FUND_DETAIL_CODE       VARCHAR2(63 CHAR) ,
FEDERAL_FUND_ID        VARCHAR2(63 CHAR) ,
ETL_TIMESTAMP          TIMESTAMP(6)
);
```

```
CREATE TABLE INSTRUCTIONAL_ASSIGNMENT_FACT
(
    TIME_KEY              NUMBER,
    INSTRUCTOR_KEY       NUMBER,
    COURSE_KEY           NUMBER,
    SUB_ACADEMIC_PERIOD  VARCHAR2(63 CHAR) ,
    SUB_ACADEMIC_PERIOD_DESC VARCHAR2(255 CHAR) ,
    CATEGORY             VARCHAR2(2 CHAR) ,
    PRIMARY_IND         VARCHAR2(1 CHAR) ,
    PERCENT_RESPONSIBILITY NUMBER,
    SESSION_PERCENTAGE  NUMBER
);
```

Chapter 6. Future Work

```
CREATE TABLE INSTRUCTOR_DIM
(
    INSTRUCTOR_KEY          NUMBER,
    INSTRUCTOR_ID           VARCHAR2(9 CHAR) ,
    INSTRUCTOR_UID          NUMBER,
    INSTRUCTOR_FIRST_NAME   VARCHAR2(60 CHAR) ,
    INSTRUCTOR_LAST_NAME    VARCHAR2(60 CHAR) ,
    INSTRUCTOR_RANK_CODE    VARCHAR2(63 CHAR) ,
    INSTRUCTOR_RANK_CODE_DESC VARCHAR2(255 CHAR) ,
    INSTRUCTOR_BASE_SALARY  CHAR(20 CHAR) ,
    PRIM_APPOINTMENT_DESC   VARCHAR2(30 CHAR) ,
    PRIM_APPOINTMENT_DEPT_CODE VARCHAR2(4 CHAR) ,
    PRIM_APPT_DEPT_DESC     VARCHAR2(30 CHAR) ,
    PRIM_EFFECTIVE_START_DT DATE,
    PRIM_EFFECTIVE_END_DT   DATE,
    ACADEMIC_TITLE          CHAR(20 CHAR) ,
    PRIM_TENURE_CODE        VARCHAR2(2 CHAR) ,
    PRIM_TENURE_DEPT_CODE   VARCHAR2(4 CHAR) ,
    PRIM_TENURE_EFF_DATE    DATE,
    PRIM_TENURE_TRACK_BEGIN_DATE DATE,
    PRIM_TENURE_REVIEW_DATE DATE,
    SEC_APPOINTMENT_DESC    VARCHAR2(30 CHAR) ,
    SEC_APPOINTMENT_DEPT_CODE VARCHAR2(30 CHAR) ,
    SEC_APPT_DEPT_DESC     VARCHAR2(30 CHAR) ,
    SEC_TENURE_CODE         VARCHAR2(2 CHAR) ,
    SEC_TENURE_DEPT_CODE   VARCHAR2(4 CHAR) ,
    SEC_TENURE_EFF_DATE    DATE,
    SEC_TENURE_TRACK_BEGIN_DATE DATE,
```

*Chapter 6. Future Work*

```
SEC_TENURE_REVIEW_DATE      DATE,
RANK_EFFECTIVE_START_DT     DATE,
RANK_EFFECTIVE_END_DT       DATE,
ACTIVE_FLAG                  NUMBER,
EFFECTIVE_START_DATE        DATE,
EFFECTIVE_END_DATE          DATE,
ETL_TIMESTAMP                TIMESTAMP(6)
);
```

```
CREATE TABLE MAJOR_DEPT_XWALK
(
  MAJOR_CODE                 VARCHAR2(10 CHAR) ,
  MAJOR_DESCRIPTION          VARCHAR2(31 CHAR) ,
  ORGANIZATION_LEVEL_5      VARCHAR2(7 CHAR) ,
  ORGANIZATION_DESC_5       VARCHAR2(34 CHAR) ,
  OBSOLETE                   VARCHAR2(8 CHAR) ,
  BANNER_MAJOR_CODE         VARCHAR2(10 CHAR) ,
  BANNER_ORG_CODE           VARCHAR2(15 CHAR) ,
  ORGANIZATION_LEVEL_3      VARCHAR2(7 CHAR) ,
  ORGANIZATION_DESC_3       VARCHAR2(34 CHAR) ,
  ORGANIZATION_LEVEL_2      VARCHAR2(7 CHAR) ,
  ORGANIZATION_DESC_2       VARCHAR2(25 CHAR)
);
```

```
CREATE TABLE PERSON_DIM
(
  PERSON_KEY                 NUMBER,
  PERSON_ID                  VARCHAR2(9 CHAR) ,
```

Chapter 6. Future Work

PERSON_UID	NUMBER,
FIRST_NAME	VARCHAR2(63 CHAR) ,
LAST_NAME	VARCHAR2(63 CHAR) ,
MIDDLE_INITIAL	VARCHAR2(15 CHAR) ,
MIDDLE_NAME	VARCHAR2(63 CHAR) ,
NAME_SUFFIX	VARCHAR2(20 CHAR) ,
FULL_NAME_FMIL	VARCHAR2(255 CHAR) ,
FULL_NAME_LFMI	VARCHAR2(255 CHAR) ,
NETID	VARCHAR2(30 CHAR) ,
GENDER	VARCHAR2(63 CHAR) ,
GENDER_DESC	VARCHAR2(63 CHAR) ,
BIRTH_DATE	DATE,
CURRENT_AGE	NUMBER,
DECEASED_STATUS	VARCHAR2(1 CHAR) ,
DECEASED_DATE	DATE,
CONFIDENTIALITY_IND	VARCHAR2(1 CHAR) ,
PRIMARY_ETHNICITY	VARCHAR2(63 CHAR) ,
PRIMARY_ETHNICITY_DESC	VARCHAR2(255 CHAR) ,
PRIM_ETHNICITY_CATEGORY	VARCHAR2(63 CHAR) ,
PRIM_ETHNICITY_CATEGORY_DESC	VARCHAR2(255 CHAR) ,
HISPANIC_LATINO_ETHNICITY_IND	VARCHAR2(1 CHAR) ,
IPEDS_VALUES	NUMBER,
IPEDS_VALUES_DESC	VARCHAR2(63 CHAR) ,
HISPANIC	NUMBER,
AMERICAN_INDIAN	NUMBER,
ASIAN	NUMBER,
AFRICAN_AMERICAN	NUMBER,
NATIVE_HAWAIIAN	NUMBER,

Chapter 6. Future Work

WHITE	NUMBER,
NON_RESIDENT_ALIEN	NUMBER,
RACE.CATEGORY_COUNT	NUMBER,
RACE.COUNT	NUMBER,
RACE.ETHNICITY_CONFIRM_IND	VARCHAR2(1 CHAR) ,
RACE.ETHNICITY_CONFIRM_DATE	DATE,
CITIZENSHIP_TYPE	VARCHAR2(63 CHAR) ,
CITIZENSHIP_TYPE_DESC	VARCHAR2(255 CHAR) ,
CITIZENSHIP_IND	VARCHAR2(1 CHAR) ,
NATION_OF_CITIZENSHIP	VARCHAR2(63 CHAR) ,
NATION_OF_CITIZENSHIP_DESC	VARCHAR2(255 CHAR) ,
VISA_TYPE	VARCHAR2(63 CHAR) ,
VISA_TYPE_DESC	VARCHAR2(255 CHAR) ,
VETERAN_CATEGORY	VARCHAR2(1 CHAR) ,
VETERAN_CATEGORY_DESC	VARCHAR2(45 CHAR) ,
VETERAN_SPECIAL_DISABLED_IND	VARCHAR2(1 CHAR) ,
MILITARY_SEPARATION_DATE	DATE,
MA_ADDRESS_TYPE	VARCHAR2(2 CHAR) ,
MA_ADDRESS_TYPE_DESC	VARCHAR2(50 CHAR) ,
MA_ACTIVE_ADDRESS_IND	VARCHAR2(2 CHAR) ,
MA_STREET_LINE1	VARCHAR2(100 CHAR) ,
MA_STREET_LINE2	VARCHAR2(100 CHAR) ,
MA_STREET_LINE3	VARCHAR2(100 CHAR) ,
MA_CITY	VARCHAR2(63 CHAR) ,
MA_STATE_PROVINCE	VARCHAR2(3 CHAR) ,
MA_STATE_PROVINCE_DESC	VARCHAR2(35 CHAR) ,
MA_POSTAL_CODE	VARCHAR2(30 CHAR) ,
MA_COUNTY_CODE	VARCHAR2(6 CHAR) ,

Chapter 6. Future Work

MA_COUNTY_CODE_DESC	VARCHAR2(20 CHAR) ,
MA_NATION_CODE	VARCHAR2(5 CHAR) ,
MA_NATION_CODE_DESC	VARCHAR2(50 CHAR) ,
ADDRESS_COUNT	NUMBER,
PHONE_NUMBER_COMBINED	VARCHAR2(35 CHAR) ,
PHONE_TYPE	VARCHAR2(5 CHAR) ,
PHONE_DESC	VARCHAR2(50 CHAR) ,
PHONE_COUNT	NUMBER,
EMAIL_ADDRESS	VARCHAR2(255 CHAR) ,
EMAIL_TYPE	VARCHAR2(35 CHAR) ,
EMAIL_TYPE_DESC	VARCHAR2(255 CHAR) ,
EMAIL_COMMENT	VARCHAR2(255 CHAR) ,
EMAIL_PREFERRED_ADDRESS	VARCHAR2(255 CHAR) ,
PE_ADDRESS_TYPE	VARCHAR2(63 CHAR) ,
PE_ADDRESS_TYPE_DESC	VARCHAR2(255 CHAR) ,
PE_ACTIVE_ADDRESS_IND	VARCHAR2(2 CHAR) ,
PE_STREET_LINE1	VARCHAR2(255 CHAR) ,
PE_STREET_LINE2	VARCHAR2(255 CHAR) ,
PE_STREET_LINE3	VARCHAR2(255 CHAR) ,
PE_CITY	VARCHAR2(63 CHAR) ,
PE_STATE_PROVINCE	VARCHAR2(63 CHAR) ,
PE_STATE_PROVINCE_DESC	VARCHAR2(255 CHAR) ,
PE_POSTAL_CODE	VARCHAR2(63 CHAR) ,
PE_COUNTY_CODE	VARCHAR2(63 CHAR) ,
PE_COUNTY_CODE_DESC	VARCHAR2(255 CHAR) ,
PE_NATION_CODE	VARCHAR2(63 CHAR) ,
PE_NATION_CODE_DESC	VARCHAR2(255 CHAR) ,
EFFECTIVE_START_DATE	DATE,

Chapter 6. Future Work

```
EFFECTIVE_END_DATE          DATE,
SSN                          VARCHAR2(63 CHAR) ,
STARS_ID                     VARCHAR2(50 CHAR) ,
ACTIVE_FLAG                  NUMBER,
ETL_TIMESTAMP                 TIMESTAMP(6)
);
```

```
CREATE TABLE PROGRAMDIM
(
PROGRAMKEY      INTEGER,
PROGRAMCODE    VARCHAR2(12 CHAR) ,
PROGRAMDESC    VARCHAR2(30 CHAR) ,
LEVEL_CODE     VARCHAR2(2 CHAR) ,
CAMPUS_CODE    VARCHAR2(3 CHAR) ,
COLLEGE_CODE   VARCHAR2(2 CHAR) ,
DEGC_CODE      VARCHAR2(6 CHAR) ,
ETL_TIMESTAMP  DATE
);
```

```
CREATE TABLE PROGRAMENROLLMENT_FACT
(
TIME_KEY          NUMBER,
PROGRAMKEY        NUMBER,
TOTAL_ENROLLMENT NUMBER,
WHITE_COUNT       NUMBER,
AFRICAN_AMERICAN_COUNT NUMBER,
AMERICAN_INDIAN_COUNT NUMBER,
HISPANIC_COUNT    NUMBER,
```



Chapter 6. Future Work

```
ASIAN_COUNT          NUMBER,
NATIVE_HAWAIIAN_COUNT  NUMBER,
TWO_MORE_RACES_COUNT  NUMBER,
RACE_ETHN_UNKOWN_COUNT NUMBER,
NON_RES_COUNT         NUMBER,
ETL_TIMESTAMP        DATE
);

CREATE TABLE STUDENT_DIM
(
  STUDENT_KEY          NUMBER,
  STUDENT_ID          VARCHAR2(15 CHAR) ,
  PERSON_UID          NUMBER,
  ACADEMIC_PERIOD     VARCHAR2(30 CHAR) ,
  ACADEMIC_PERIOD_DESC VARCHAR2(30 CHAR) ,
  PROGRAM_CODE        VARCHAR2(12 CHAR) ,
  PROGRAM_CODE_DESC   VARCHAR2(30 CHAR) ,
  PRIMARY_PROGRAM_IND VARCHAR2(1 CHAR) ,
  YEAR_ADMITTED       VARCHAR2(4 CHAR) ,
  YEAR_ADMITTED_DESC  VARCHAR2(20 CHAR) ,
  AGE_ADMITTED        NUMBER,
  CATALOG_ACADEMIC_PERIOD VARCHAR2(6 CHAR) ,
  CATALOG_ACADEMIC_PERIOD_DESC VARCHAR2(30 CHAR) ,
  AWARD_CATEGORY      VARCHAR2(2 CHAR) ,
  AWARD_CATEGORY_DESC VARCHAR2(30 CHAR) ,
  NEW_STUDENT_IND     VARCHAR2(1 CHAR) ,
  ADMISSIONS_POPULATION VARCHAR2(2 CHAR) ,
  ADMISSIONS_POPULATION_DESC VARCHAR2(30 CHAR) ,
```

Chapter 6. Future Work

DEGREE_CODE	VARCHAR2(6 CHAR) ,
DEGREE_CODE_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CODE	VARCHAR2(4 CHAR) ,
FIRST_MAJOR_CODE_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CIP_CODE	VARCHAR2(6 CHAR) ,
FIRST_MAJOR_CIP_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CONC1	VARCHAR2(4 CHAR) ,
FIRST_MAJOR_CONC1_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CONC2	VARCHAR2(4 CHAR) ,
FIRST_MAJOR_CONC2_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CONC3	VARCHAR2(4 CHAR) ,
FIRST_MAJOR_CONC3_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_DEPT_CODE	VARCHAR2(4 CHAR) ,
FIRST_MAJOR_DEPT_CODE_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_COLLEGE_CODE	VARCHAR2(2 CHAR) ,
FIRST_MAJOR_COLLEGE_CODE_DESC	VARCHAR2(30 CHAR) ,
FIRST_MAJOR_CAMPUS_CODE	VARCHAR2(3 CHAR) ,
FIRST_MAJOR_CAMPUS_CODE_DESC	VARCHAR2(30 CHAR) ,
SECOND_MAJOR_CODE	VARCHAR2(4 CHAR) ,
SECOND_MAJOR_DESC	VARCHAR2(30 CHAR) ,
SECOND_MAJOR_CIP_CODE	VARCHAR2(6 CHAR) ,
SECOND_MAJOR_CIP_DESC	VARCHAR2(30 CHAR) ,
SECOND_MAJOR_CONC1	VARCHAR2(4 CHAR) ,
SECOND_MAJOR_CONC1_DESC	VARCHAR2(30 CHAR) ,
SECOND_MAJOR_CONC2	VARCHAR2(4 CHAR) ,
SECOND_MAJOR_CONC2_DESC	VARCHAR2(30 CHAR) ,
SECOND_MAJOR_CONC3	VARCHAR2(4 CHAR) ,
SECOND_MAJOR_CONC3_DESC	VARCHAR2(30 CHAR) ,

Chapter 6. Future Work

SECOND_MAJOR_DEPT_CODE	VARCHAR2(4 CHAR) ,
SECOND_MAJOR_DEPT_CODE_DESC	VARCHAR2(30 CHAR) ,
FIRST_MINOR_CODE	VARCHAR2(4 CHAR) ,
FIRST_MINOR_CODE_DESC	VARCHAR2(30 CHAR) ,
SECOND_MINOR_CODE	VARCHAR2(4 CHAR) ,
SECOND_MINOR_CODE_DESC	VARCHAR2(30 CHAR) ,
STUDENT_STATUS_CODE	VARCHAR2(2 CHAR) ,
STUDENT_STATUS_CODE_DESC	VARCHAR2(10 CHAR) ,
CURRENT_TIME_STATUS_CODE	VARCHAR2(2 CHAR) ,
CURRENT_TIME_STATUS_CODE_DESC	VARCHAR2(30 CHAR) ,
STUDENT_CLASS_BOAP_CODE	VARCHAR2(2 CHAR) ,
STUDENT_CLASS_BOAP_CODE_DESC	VARCHAR2(30 CHAR) ,
STUDENT_CLASSIFICATION	VARCHAR2(2 CHAR) ,
STUDENT_CLASSIFICATION_DESC	VARCHAR2(30 CHAR) ,
STUDENT_LEVEL_CODE	VARCHAR2(2 CHAR) ,
STUDENT_LEVEL_CODE_DESC	VARCHAR2(30 CHAR) ,
RESIDENCY_CODE	VARCHAR2(2 CHAR) ,
RESIDENCY_DESC	VARCHAR2(30 CHAR) ,
RESIDENCY_IND	VARCHAR2(1 CHAR) ,
FINANCIAL_AID_ELIGIBLE_FLAG	NUMBER,
FINANCIAL_AID_RECEIVED_FLAG	NUMBER,
FINAID_APPLICANT_IND	VARCHAR2(1 CHAR) ,
STUDENT_RATE_CODE	VARCHAR2(2 CHAR) ,
STUDENT_RATE_CODE_DESC	VARCHAR2(21 CHAR) ,
STUDENT_ATTRIBUTE_CODE	VARCHAR2(63 CHAR) ,
STUDENT_ATTRIBUTE_CODE_DESC	VARCHAR2(255 CHAR) ,
ENROLLED_IND	VARCHAR2(1 CHAR) ,
REGISTERED_IND	VARCHAR2(1 CHAR) ,

Chapter 6. Future Work

```
REGISTERED_ABQ                VARCHAR2(1 CHAR) ,
REGISTERED_LOS_ALAMOS         VARCHAR2(1 CHAR) ,
REGISTERED_TAOS               VARCHAR2(1 CHAR) ,
REGISTERED_GALLUP             VARCHAR2(1 CHAR) ,
REGISTERED_VALENCIA           VARCHAR2(1 CHAR) ,
EFFECTIVE_START_DATE          DATE,
EFFECTIVE_END_DATE            DATE,
STUDENT_POPULATION            VARCHAR2(1 CHAR) ,
STUDENT_POPULATION_DESC       VARCHAR2(30 CHAR) ,
ACTIVE_FLAG                   NUMBER,
ETL_TIMESTAMP                  TIMESTAMP(6)
);
```

```
CREATE TABLE STUDENT_ENROLLMENT_FACT
(
  TIME_KEY                     NUMBER,
  STUDENT_KEY                  NUMBER,
  PERSON_KEY                   NUMBER,
  COURSE_KEY                   NUMBER,
  PROGRAM_KEY                  NUMBER,
  INSTRUCTOR_GROUP_KEY        NUMBER,
  STUDENT_ID                   VARCHAR2(9 CHAR) ,
  CREDIT_HRS_ATTEMPTED        NUMBER,
  CREDIT_HRS_EARNED           NUMBER,
  COURSE_CREDITS               NUMBER,
  FINAL_GRADE_RECEIVED        VARCHAR2(63 CHAR) ,
  GRADE_POINTS_RECEIVED       NUMBER,
  REGISTRATION_STATUS         VARCHAR2(63 CHAR) ,
```

Chapter 6. Future Work

```
REGISTRATION_STATUS_DESC  VARCHAR2(255 CHAR) ,
COMPLETED_FLAG           NUMBER,
ETL_TIMESTAMP              TIMESTAMP(6)
);
```

```
CREATE TABLE STUDENT_LEVEL_SUMMARY_FACT
(
PERSON_KEY                 NUMBER,
STUDENT_ID                 VARCHAR2(9 CHAR) ,
PERSON_UID                 NUMBER,
NAME                       VARCHAR2(255 CHAR) ,
NO_OF_COURSES_ENROLLED    NUMBER,
GPA_TYPE                   VARCHAR2(1 CHAR) ,
GPA_TYPE_DESC              VARCHAR2(11 CHAR) ,
ACADEMIC_STUDY_VALUE      VARCHAR2(63 CHAR) ,
ACADEMIC_STUDY_VALUE_DESC VARCHAR2(255 CHAR) ,
GPA                        NUMBER,
GPA_CREDITS                NUMBER,
CREDITS_ATTEMPTED         NUMBER,
CREDITS_EARNED             NUMBER,
CREDITS_PASSED             NUMBER,
QUALITY_POINTS             NUMBER,
ETL_TIMESTAMP              TIMESTAMP(6)
);
```

```
CREATE TABLE STUDENT_TERM_SUMMARY_FACT
(
TIME_KEY                   NUMBER,
```

*Chapter 6. Future Work*

```
PERSON_KEY                NUMBER,
STUDENT_KEY              NUMBER,
ACADEMIC_PERIOD          VARCHAR2(6 CHAR) ,
STUDENT_ID              VARCHAR2(9 CHAR) ,
PERSON_UID              NUMBER,
NO_OF_COURSES_ENROLLED  NUMBER,
GPA_TYPE                VARCHAR2(1 CHAR) ,
GPA_TYPE_DESC          VARCHAR2(11 CHAR) ,
ACADEMIC_STUDY_VALUE    VARCHAR2(63 CHAR) ,
ACADEMIC_STUDY_VALUE_DESC VARCHAR2(255 CHAR) ,
GPA                    NUMBER,
GPA_CREDITS            NUMBER,
CREDITS_ATTEMPTED     NUMBER,
CREDITS_EARNED        NUMBER,
CREDITS_PASSED        NUMBER,
QUALITY_POINTS        NUMBER,
ETL_TIMESTAMP         TIMESTAMP(6)
);
```

# References

- [1] R. Kimball and M. Ross. The Data Warehouse Toolkit. *The definitive guide to dimensional modeling.*
- [2] W. H. Inmon. Building the Data Warehouse. *Getting Started.*
- [3] R. Kimball and J. Caserta. The Data Warehouse ETL Toolkit. *Practical Techniques for Extracting, Cleaning, Conforming, and Delivering data.*
- [4] R. Kimball and M. Ross. The Kimball Group Reader. *Relentlessly Practical Tools for Data Warehousing and Business Intelligence.*
- [5] T. Teorey, S. Lightstone and T.Nadeau. Database Modeling and Design. *Logical Design.*
- [6] C. Adamson. Star Schema. *The complete reference.*
- [7] SAS Institute Inc.. SAS<sup>®</sup> Data Integration Studio 4.7 *User Guide 2013.*