Spring 5-1-2011

A Data Warehouse for Faculty Pay

John Hardebeck
Dakota State University

A DATA WAREHOUSE

FOR FACULTY PAY

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Systems

Spring, 2011

By

John Hardebeck

Project Committee:

Dr. Stephen Krebsbach

Dr. Mark Moran

Dr. Ronghua Shan

PROJECT APPROVAL FORM

We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Master of Science in Information Systems.

Student Name: John Hardebeck

Master's Project Title: A Data Warehouse for Faculty Pay

Faculty supervisor: Dr. Stephen Kresbach Date: 5/4/11

Committee member: Dr. Mark Moran Date: 5/4/11

Committee member: Dr. Ronghua Shan Date: 5/4/11

ACKNOWLEDGMENT

I would like to thank all of my friends and co-workers at Daytona State College for their support and help as I've worked my way through the MSIS program at Dakota State University. Especially, Lisa Mobley who, as an information technology professional, was a good listener during times when I was struggling with the course load and getting near the end of my rope. Eric Urff for giving me access to the MySQL database where his data warehouse resides so that I could create my own, and who, having traveled this road himself, was able to offer insights and advice that helped me avoid some pitfalls. And most of all, Daniel Jones, who never ceases to amaze me with the depth and breadth of his knowledge relating to all things in the Information Technology field, his help and support have been immeasurable as I have worked my way through the MSIS program.

Above and beyond all others, I would like to thank my wife, Mary Jo. With the advantage of hindsight, it occurs to me that I launched into the MSIS program unilaterally, with little or no discussion involving her. Despite that huge error on my part, she has been patient and understanding during the entire journey, offering help wherever possible. Without her love and support, none of this would have even been possible.

ABSTRACT

Administrators at Daytona State College have long sought for a way to document how much it costs to offer a specific course. When an instructor is either an Adjunct or a fulltime instructor teaching an Overload – essentially, working overtime – this information is readily available. This is not the case for fulltime, salaried instructors who are teaching within the bounds of their salary, what is known as "Inload". There is no direct link in the database of Daytona State's ERP system that connects pay with courses and the fact that the institution's payroll accounting is handled by a third-party organization only compounds the problem. The courses taught by instructors are well documented, it is the payroll records, or lack of them, that is the core problem. We only have access to amounts paid with very few details about what the payment was for.

Since salaried staff members at Daytona State, such as myself, are expected to work a 40 hour week, a similar thing is expected of salaried faculty members. However, since most courses require a considerable amount of time outside of the classroom either preparing lessons or grading them, the question arises of how to determine when an instructor has worked the equivalent of a 40 hour week. To resolve this problem "Load" was created. I don't how or when it came it to being or even what documentation may exist, it's just a known fact I've learned about from co-workers over the course of my 11 years at this institution. To get right to the point, Load determines how much time an instructor must spend in the classroom to equal a 40 hour week. For most college credit courses, but not all, the value is 15, meaning that an instructor must teach 15 credit hours per week to "earn" his or her salary.

It is Load that makes it possible to resolve the problem of assigning dollar amounts to specific courses. Since we can determine how much Load an instructor carried for any course and we know how much his or her salary should be, I have simply divided the annual salary for a given calendar year by the total Load carried during that same period of time. The resulting figure, a load "rate", if you will, can then be applied to specific course to determine how much of the instructor's salary was dedicated to it.

Stated salary amounts for fulltime faculty were not used in the "load rate" calculation since they merely document what an individual should be paid and may not reflect reality. I chose instead to pull actual payment records from the general ledger since they reflect amounts that were actually paid out to specific individuals. Attempts to apply this same method to Adjunct and Overload instructors were unsuccessful due to the extremely complex methods used to determine dollar amounts. However, it should be noted that reports are currently in place which provide very satisfactory estimates of Adjunct and Overload expenses, it is the fulltime (or salaried) instruction that presents a problem.

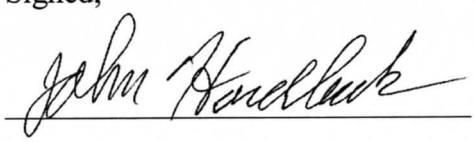
However, the resources needed to perform the operation described above for fulltime faculty would be prohibitive on Daytona State's production database and would interfere with daily operations. To resolve this problem, a data warehouse was constructed along with the necessary scripts and SQL statements to extract the needed data, upload it to the data warehouse, and perform the needed calculations. Reports on the resulting information will be made available to the college community.

DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,



John Hardebeck

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Daytona State College began life in 1957 as Daytona Beach Junior College and over the years evolved into community college and then, in 2006, began offering four-year degrees. While enrollment has increased and decreased at different times of the years, the overall trend has been upward as you can see in the following table.

Table 1. Enrollment numbers for the previous 10 academic years

Catalog Year	Adult Education	College Credit	Continuing Education	Total
CC01	7778	15615	7382	30775
CC02	7669	17061	6488	31218
CC03	7662	17039	5490	30191
CC04	7731	17598	4667	29996
CC05	7466	16971	4358	28795
CC06	7123	16913	3483	27519
CC07	8848	18264	4012	31124
CC08	9808	20059	3444	33311
CC09	9902	21826	3782	35510
CC10	9525	24730	3521	37776

During period from the 2000-2001 academic year through the 2009-2010 period Continuing Education enrollment has declined 52%. But Adult Education enrollment has increased 22% and CC has the greatest increase of all, up 58% during that same period of time. The total overall increase in enrollment from CC01 to CC10 was 22%.

Of course an increasing number of students has resulted in an increasing number of faculty. As you can see in the table shown below, during that same period of time, CC01 through CC10, fulltime faculty numbers have increased by 44%! And the number of instructors teaching overload courses has increased by 38%. Although it interesting to note that during this same period of time, reliance on Adjunct instructors decreased by about 10%. This would seem to indicate that fulltime faculty are increasing in importance, which makes the goal of this data warehouse even more useful.

Table 2. Number of faculty employed over the past decade

Catalog	52001	56001	Total	52101
CC01	216	798	1014	192
CC02	239	747	986	210
CC03	249	734	983	214
CC04	242	712	954	205
CC05	250	679	929	215
CC06	265	654	919	209
CC07	278	654	932	225
CC08	285	724	1009	233
CC09	306	704	1010	250
CC10	312	716	1028	266

Background of the Problem

In the late 1990's Daytona State College, then known as "Daytona Beach Community College", made the decision to purchase an Enterprise Resource Planning system (ERP) to manage the college's business. This system would handle everything from purchasing supplies and paying bills to scheduling courses and registering students, but the Human Resources package (i.e. payroll information) was not included in the new system. The reason for this is unknown, perhaps the decision makers felt the ERP wasn't reliable enough or, since this was new, unexplored territory, they simply didn't want to risk employee pay information to be widely accessible. In any event, that discussion is outside the scope of this project, my goal is to develop a system that will compensate for the lack of detailed pay data available for administrative reporting.

With the decision to not purchase the ERP Human Resources module, all information relating to employees, both staff and faculty, is maintained on a third-party system that is inaccessible to all but a very few employees. As a result of this third-party system, it is quite impossible for the college's administration to perform any type of reporting that relates pay to specific courses taught by faculty members. Unfortunately (and not surprisingly) these types of reports are something the administration would very much like to have, but the structure of the information combined with its sheer volume – just one of the base accounting tables accessed for this data warehouse by itself contains over 32 million records - makes any meaningful reporting impossible. If a useful report could even be constructed, the time it

would take to run and the system resources required would make it very difficult, if not impossible, to use.

The desire then, is for reports that allow Daytona State's administrative decision makers to relate faculty employee expense to specific college courses. For most college staff, whether upper level administrators or rank and file employees (such as those on the front lines interfacing with students or behind the scenes in, say, Information Technology, like myself), this is a simple matter, annual reports suffice since a straight forward, easily defined service is provided "X" number of days per year. With faculty, however, the matter is not that simple as I will explain later.

The economic downturn of recent years that has affected not just Florida, but the entire country and indeed, the world, has only increased Daytona State's need for a way to document and analyze labor expenses as they relate to faculty. Since the salary paid to faculty members for teaching is the single largest cost element of most, if not all, courses, the need for this information is considerable, even more so during times of economic stress and reduced budgets.

Speaking in very broad, general terms, there are two sets of data within the Daytona State College database that must be brought together to achieve the goal of this project, the records relating to how many courses were offered, including their instructors, and those defining how much each individual was paid. Sadly, as mentioned earlier, within the current record structure, nothing exists that connects courses taught directly with salary received. I have devised a method for doing this within the data warehouse, but first I must lay ground work with a discussion of our two separate data sets. We will begin with the courses taught by instructors.

Instructor course load

There are two basic types of faculty at Daytona State College, full-time and adjunct. Full-time instructors are salaried employees who work either a 9-month or 12-month schedule while adjunct faculty are contract workers obligated to teach specific courses for an agreed upon rate. Additionally, it is possible for full-time faculty to teach courses beyond those covered by their salary, essential.ly putting in "overtime" (hereafter known as "Overload").

Of course, any salaried faculty member is expected to work a 40 hour week like other employees, but can hardly be expected to spend 40 hours in the classroom. For every hour spent in the classroom an instructor will spend a considerable amount of time outside of class preparing lessons, grading homework, etc. Since the classes taught by an instructor provide the only clearly defined documentation, by way of the course schedule, as to when he or she is on the job the question then becomes…how many hours must an instructor teach to equal a 40 hour week?

The answer to the above question, at least in the state of Florida, is "Load". Load is method for determining how many hours an instructor must teach to equal a regular 40 hour week. Both instructors and the courses they teach are assigned a load value known as the Faculty Load Type ("FLT"), or simply "load type". The load value itself is deduced by determining how much time an instructor must spend outside the classroom for each hour spent face to face in the classroom with students. For example, if a course had a load type of 15, then for each 15 hours of classroom instruction, 25 hours should be spent in preparation and grading. This would mean that if an instructor with a faculty load type of 15 teaches a course which has a load value of 15, then the instructor must be in the classroom teaching for 15 hours each week to earn his or her salary. While the example where both instructor and course have an FLT of 15 was used above, and it is probably the most common situation, it is possible for both courses and instructors to have load values other than 15. As mentioned earlier, the load value is based on the perceived amount of outside work required by a specific course. Frequently, an instructor with one load type will teach a course with a different load type, in these instances special calculations must be employed to accurately calculate the instructor's load.

Instructor load figures prominently in this project since it is the method by which we determine how much an instructor is working and what is required to earn his or her salary or even if the individual is working an Overload. And as you will see, after a discussion of the accounting records where faculty salaries are stored in Daytona State's database, it also provides the method by which instructor salaries can be matched with individual courses.

Accounting Records

This project is primarily concerned with salaried faculty but will include Adjuncts and Overloads even though a system is currently in place that provides very good estimates of adjunct and overload pay. The reason for this is consistency, I want all of the pay information provided by the data warehouse to be calculated using the same method so that we'll be comparing apples to apples (instead of something else) when analyzing it. Although it should be noted, the existing reports used to estimate adjunct and overload pay will prove useful as a validation tool since their information is calculated independent of the accounting tables.

To begin unraveling some of the mystery surrounding faculty salaries and classes taught, a discussion of our accounting system would be in order, specifically, how the various types of instructors are coded in the ERP system.

First, the accounting records within the ERP system use fairly standard coding that follows National Association of College and University Business Officers (NACUBO) guidelines (Jenzabar, Inc., 2007). Account numbers are a composite of 4 different values, the Fund, Subfund, Object, and Function numbers, in that order. The values that make an account number are discussed below.

Funds are defined in Jenzabar's documentation as "*A collection of assets, liabilities, and equity related to a specific subset of the activities of a nonbusiness organization.*" (Jenzabar, Inc., 2005). The Fund defines a very broad category such as Unrestricted funds, Restricted funds, Scholarship's, etc. A complete listing of currently active Funds is shown below along with descriptions.

Table 3. The Fund table

Fund	Description
10	Current Fund Unrestricted
20	Current Fund Restricted
30	Auxiliary Fund
40	Loans, Endowments, Annuity
50	Scholarship Fund
60	Agency Fund
70	Unexpended Plant Fund
80	Retirement on Debt Fund
90	Investment in Plant Fund

A Subfund is "*a component of the account number that relates to a particular project*" (Jenzabar, Inc., 2005). As they relate to this project, Subfund numbers help define courses, some examples are visible in the table shown below.

Table 4. The Subfund table

Subfund	Course Number	Description
111131500	ENC1101	College Composition
111131500	ENC1102	Literature and Comp.
111310101	MAT0024	Mathematics II
111161700	MAC1105	College Algebra

The Object code is defined as "*A segment of the general ledger account number, designating a classification or expenditure type*" (Jenzabar, Inc., 2005). The Object, or "General Ledger" code is crucial to the success of this project since it defines the different categories of instructors this project intends to study. A complete list of the codes we will be using and related descriptions is shown below.

Table 5. The Object table

Object	Description
52001	Instruct-Fulltime
52101	Instruct-Overload
52102	Inst Overload-Non Fac
52103	Inst-O/L Couns/Lib/Coach
56001	Ops Adjunct Instructor

The Function is "*The portion of an account number that pertains to the purpose of a charge (e.g., the department number)*" (Jenzabar, Inc., 2005) and, to be quite honest, is of no concern whatsoever so far as this project is concerned. But, for completeness sake, a list of Function codes is shown below and it will be included in subsequent discussions.

Table 6. The Function table

Function	Description
85000	Management & Gen Exp
81000	Student Assistance Exp
82000	College Prog Exp
83000	Community Service Exp
84000	Fund Raising Expense

As was mentioned earlier, the Object code defines the various categories of instructors. Collectively the Fund, Subfund, Object, and Function codes define the courses which are taught by instructors. All four of these codes come together in the accounting tables when entries are created to actually pay the instructors for services rendered. The accounting records can tell us that an instructor was paid "X" amount for teaching a certain class, but the dates associated with these entries are pay dates (which occur twice each month) and have no connection with the dates the course (or courses) was/were taught. The question before us is how to take these evenly spaced pay amounts and apply them to courses whose dates rarely, if ever, correspond with them and to divide the money up in the correct proportions.

Statement of the problem

Reports existed within the college's ERP system that detailed courses being taught by each instructor, but these reports couldn't provide any information regarding instructor pay, to supply that information a set of Crystal Reports were developed that could calculate pay amounts. These reports were used all year round to produce "contracts" for Overload and Adjunct faculty detailing pay for each course taught. The reports were reasonably effective, but required a great deal of manual intervention to produce useful information. The total amount to be paid to these instructors would be calculated and then broken down over the number of pay dates that would occur during the period. But specific pay dates weren't stored within the system and had to be hardcoded on each contract! When you factor in the various summer sub-sessions, the fact that college credit courses are paid differently (and sometimes on different dates) than adult education courses, and that in addition to Adjuncts, some 9-month faculty are teaching during a period that fall outside of their contracts, it was necessary to maintain 20 to 40 (or more) of these "contracts". And even then, with all of these contracts it would still be necessary, on occasion, to literally go into the database (through the ERP system) and manually manipulate the data so an accurate contract could be printed and then restore the data to its original state afterwards. In those days, the late 1990's into the early 2000's, there was literally one person at this institution (out of approximately 800 employees) who understood the faculty pay system well enough to make these changes. While this

system worked, the amount of manual intervention required made it very clumsy, a "cludge", if you will.

Due to the not so user friendly interface provided by the college's ERP system a home grown application was developed in Borland C++ Builder called Kaleidoscope. Since Kaleidoscope had proven quite successful and literally supplanted the ERP system in some areas, it was decided to replace yet another part the ERP system, for scheduling courses, with a new system in Kaleidoscope called "Catalog Maintenance". The research and development that went into Catalog Maintenance over a period of several years led to a much greater understanding of how faculty pay is calculated, the addition of several new tables to our ERP database to store additional information, and in the end, a very satisfactory system for scheduling courses and assigning instructors. Additionally, the new database tables provided more detailed information that in turn allowed for the creation of a new, much easier to operate reporting application (also within Kaleidoscope).

In parallel with the development of the new Kaleidoscope Catalog Maintenance application, a new, easier to maintain reporting tool was created that provided accurate reports of faculty pay. The reporting application has been very successful, but despite ease of use and more detailed, accurate data, the information provided is still just an estimate of adjunct and overload pay, no method exists to connect the salaries of full-time faculty with specific courses.

Objectives of the project

A reasonable system is in place for estimating Adjunct and Overload expenses, it is Inload (or salaried) instruction that we are interested in since no method currently exists to link faculty salaries with specific courses. However, Adjuncts and Overloads will be included in this process since they are important for data analysis. And if they are to be included it is critical that the same method be used for extracting and calculating their values, to do otherwise would invalidate all of the data and render comparisons meaningless.

To achieve the above mentioned goal, current methods used for estimating Adjunct and Overload pay will be ignored, all dollar amounts will be pulled from the same database tables for all categories of instruction. It will be impossible to calculate amounts that are precise, but very reasonable, useful estimates should be possible. Toward that end, the goal is

to extract both course and pay information, find a way to consistently relate the two, and to breakdown and apply an individual's annual salary to courses taught. Once that has been achieved, a series of reports will be created for use by the college's management.

LITERATURE REVIEW

Since the problem this data warehouse is attempting to resolve - or at least, mitigate to some degree – is specific to Daytona State College, the only documentation available is that supplied by Jenzabar, Inc., the creators of our ERP system. Additionally, my knowledge regarding the creating of meeting records and assigning instructors is extensive, having worked with the subject for 10+ years I dare say there are only a handful of individuals within this organization who know as much, or more, than I. However, the accounting side of the house is a very different story, it is an area where my knowledge is limited to the "estimates" mentioned earlier which, when all is said and done, have little or no impact on what and instructor receives in his or her paycheck.

Figure 1 is a diagram of the general ledger tables employed by our accounting system, these tables had to be navigated to locate the bits of data crucial to this project. I was hopeful at first when I read…"the system obtains the detail pay amounts for the period from the Subsidiary Transaction record (subtr_rec)" (Jenzabar, Inc., 2005). An inspection of the subtr_rec revealed that while it contained much information regarding expenses, none of the general ledger codes relating to faculty (or any other employees) were present. Evidently, since Daytona State's payroll accounting is done by an outside organization, absolutely no payroll information can be found in the usual places. The search continued.

I eventually located records relating to financial disbursements. The accounting may be done offsite, but when the actual checks are printed, entries had to be made in the general ledger to document the payments. By linking the voucher (vch_rec), general ledger entry (gle_rec), and general ledger transaction (gltr_rec) tables, I was able to acquire the necessary data. The trick now became, how to relate this information to the courses taught by faculty members, that discussion will be saved for the chapter on system design.

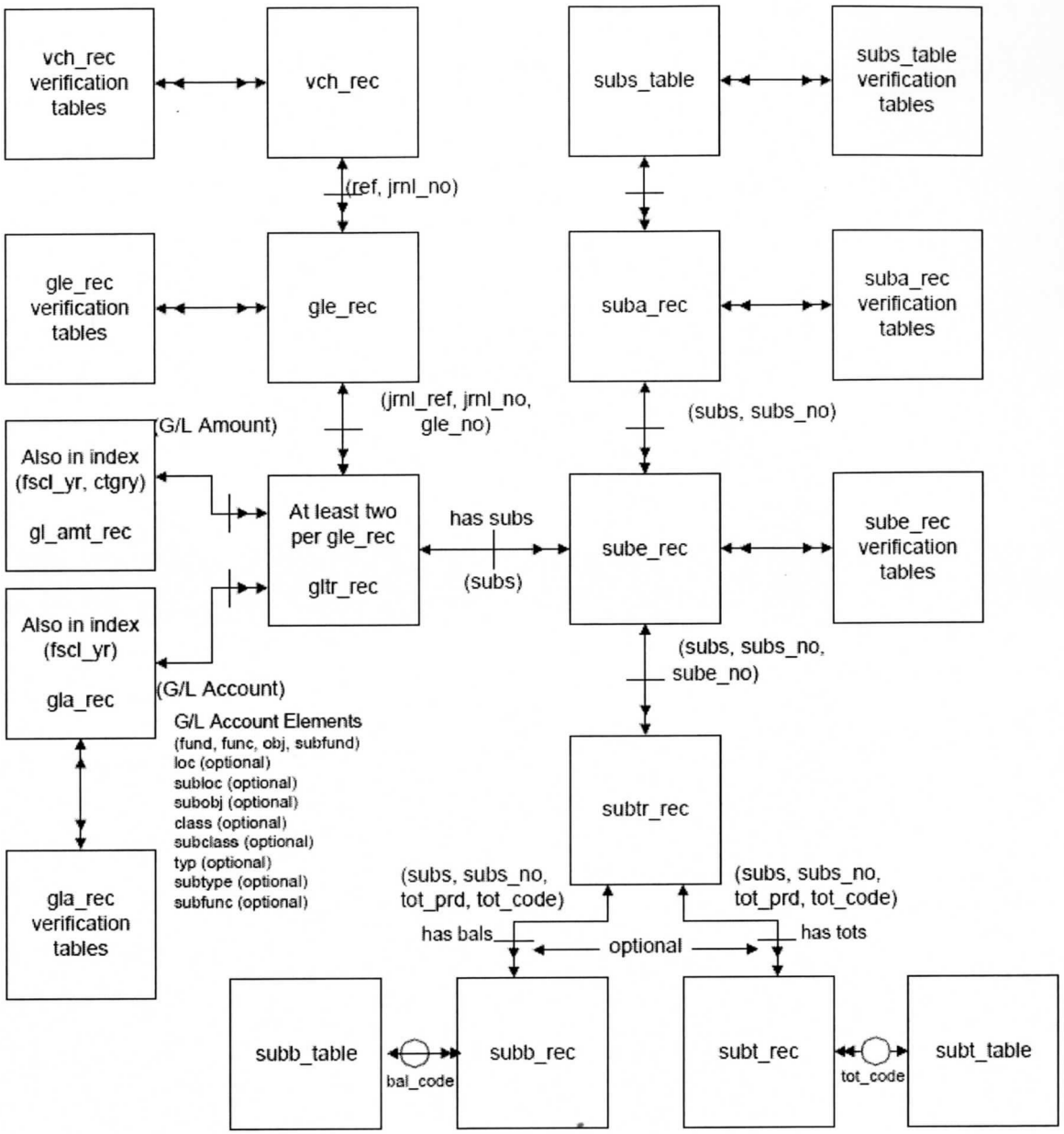


Figure 1. General Ledger table diagram for the ERP system (Jenzabar, Inc., 2005)

SYSTEM DESIGN (RESEARCH METHODOLOGY)

Research

The first step in this project was an analysis of the DSC database to define which tables are needed for this project. A considerable amount of my time is spent resolving issues related to the scheduling of course meetings and instructor load so I am already quite familiar with those tables and several we have added over the years. A diagram of the course, meeting, and instructor tables is shown below.

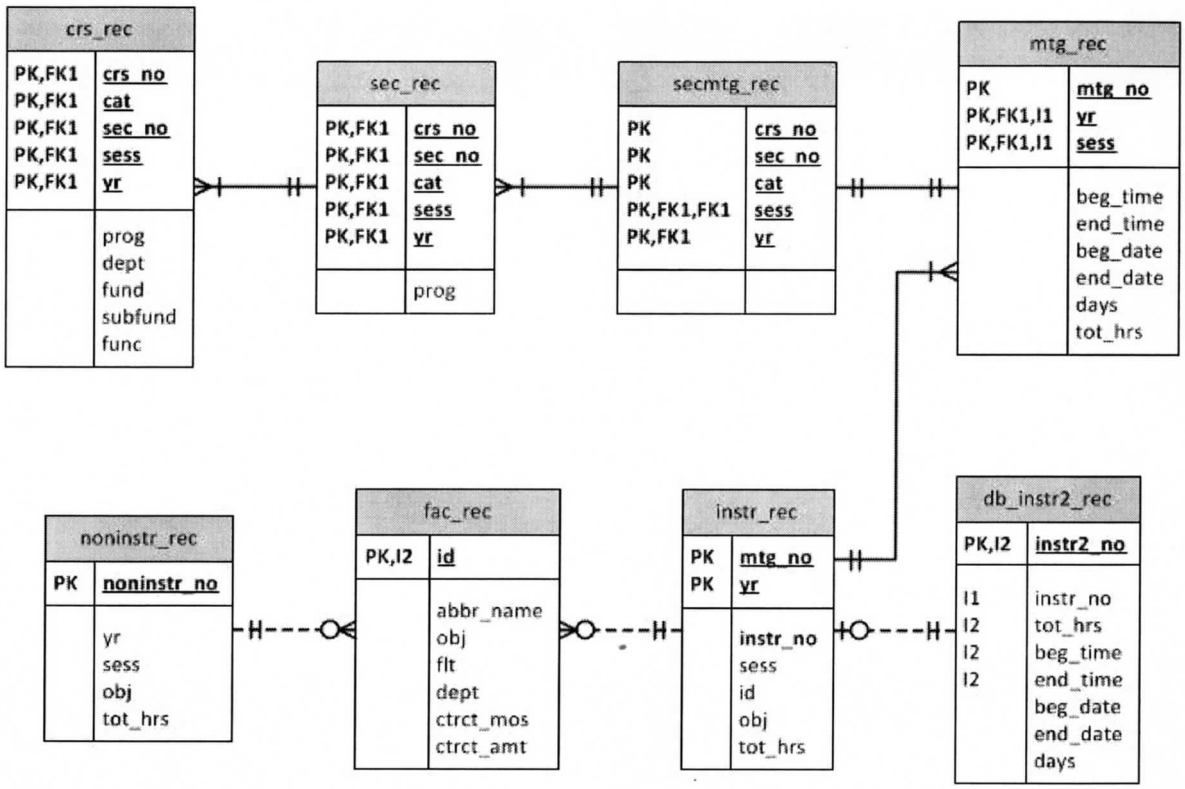


Figure 2. Course, Section, Meeting, Instructor relationships

For the most part the tables shown in figure 2 are relatively simple to understand, crs_rec holds the course masters for each catalog year, sec_rec contains the course sections for each session and year, mtg_rec details specific course meetings, instr_rec lists the instructor(s) for each meeting,

and fac_rec has information specific to individual instructors. But the tables secmtg_rec, noninstr_rec, and db_instr2_rec may require some explanation.

The table secmtg_rec serves as nothing more than a link between sections and meetings, it's nothing more than that.

But the noninstr_rec is a more interesting table, it documents the work load of instructors who are *not* teaching. A very common example of this would be an individual who is conducting research in lieu of actually teaching a course, he or she is on job and being paid by the college, but their duties lie outside the classroom. Sometimes and instructor will have full release, meaning they aren't teaching any courses at all, but usually their schedule will be a mix of teaching and research (for example). So that an instructor's salary is properly divided among the Load hours worked during the year, the Release Load must be included in the Load totals, failure to do so would skew the resulting Load Rate calculations.

The table db_instr2_rec[1] was added by Daytona State to capture information in greater detail than the ERP system allowed and has a strict one-to-one relationship with the instr_rec table. Entries to this table are made by Daytona State's Catalog Maintenance application which has supplanted the ERP system for scheduling courses and instructors. Additionally, all course masters, section, meeting, and instructor records are also created through Catalog Maintenance for the college credit and adult education programs. However, the fact that db_instr2_rec records only go back to 2007 and that our Continuing Education programs still use the ERP means that Load values are stored in two different tables. For CE courses and those CC and AE courses scheduled before 2007, faculty Load will always be in mtg_rec.tot_hrs, but for CC and AE courses scheduled after 2007, the correct value is in db_instr2_rec.tot_hrs. To resolve this problem, load values from both mtg_rec.tot_hrs and db_instr2_rec.tot_hrs are included in the data extraction. The stored procedure "update_fac_load" was then created to copy values from meeting_dim.alt_load, the mtg_rec value, to meeting_dim.fac_load whenever fac_load is 0 and alt_load has a value greater than zero.

As mentioned earlier, the layout and structure of tables related to courses and instructors is well known to me, but those in the financial arena are a different matter entirely, with that being the case, I immediately turned to the ERP documentation for help. A search of the financial documentation revealed that pay amounts could be found in subtr_rec, known as the Subsidiary Transaction record (Jenzabar, Inc., 2005). However, since Daytona State doesn't use the ERP's human resources package, these records were blank where faculty related general

---

[1] The table name includes "db_" to distinguish it from tables original to the ERP system. Also, it should be noted that there are other db_*2_rec tables, but they have no bearing on this project.

ledger codes are concerned. A few questions to those more familiar with the accounting area revealed the answer, while the accounting may be done offsite, when checks are printed, entries have to be made in the general ledger to document the payments. By linking these tables (shown below) I was able to acquire the necessary data.

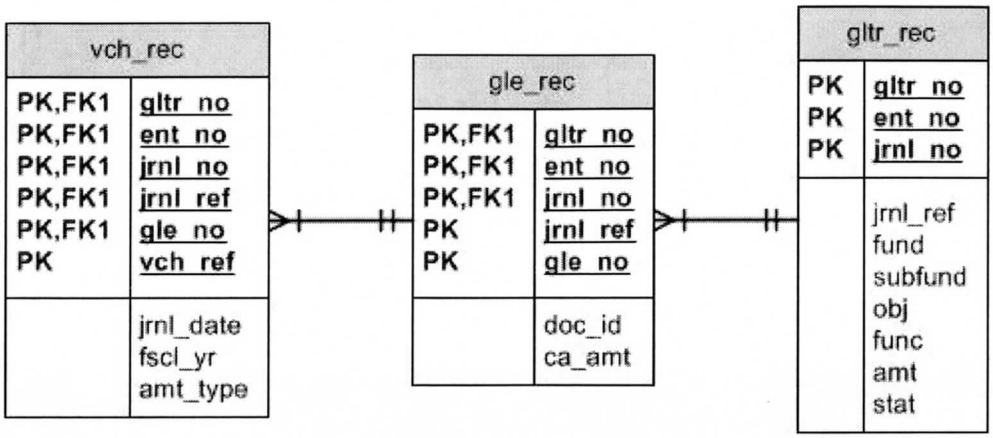


Figure 3. General Ledger tables

I now have all the necessary bits of information, the problem – which will be dealt with in the data warehouse – is how to reliably connect faculty instructional records with faculty pay.

Data Warehouse Design

Three categories of data are being brought into the data warehouse; course, instructor, and pay. To accommodate this data the following tables were created in the data warehouse; course_dim, meeting_dim, and pay_detail. A table (acad_cal_tbl) was also created to hold academic calendar information, it supplies beginning and ending dates for each session and year combination. Figure 4 shows all of the tables in the data warehouse and how they are related. A complete list of tables along with a short description can be found in table 7.

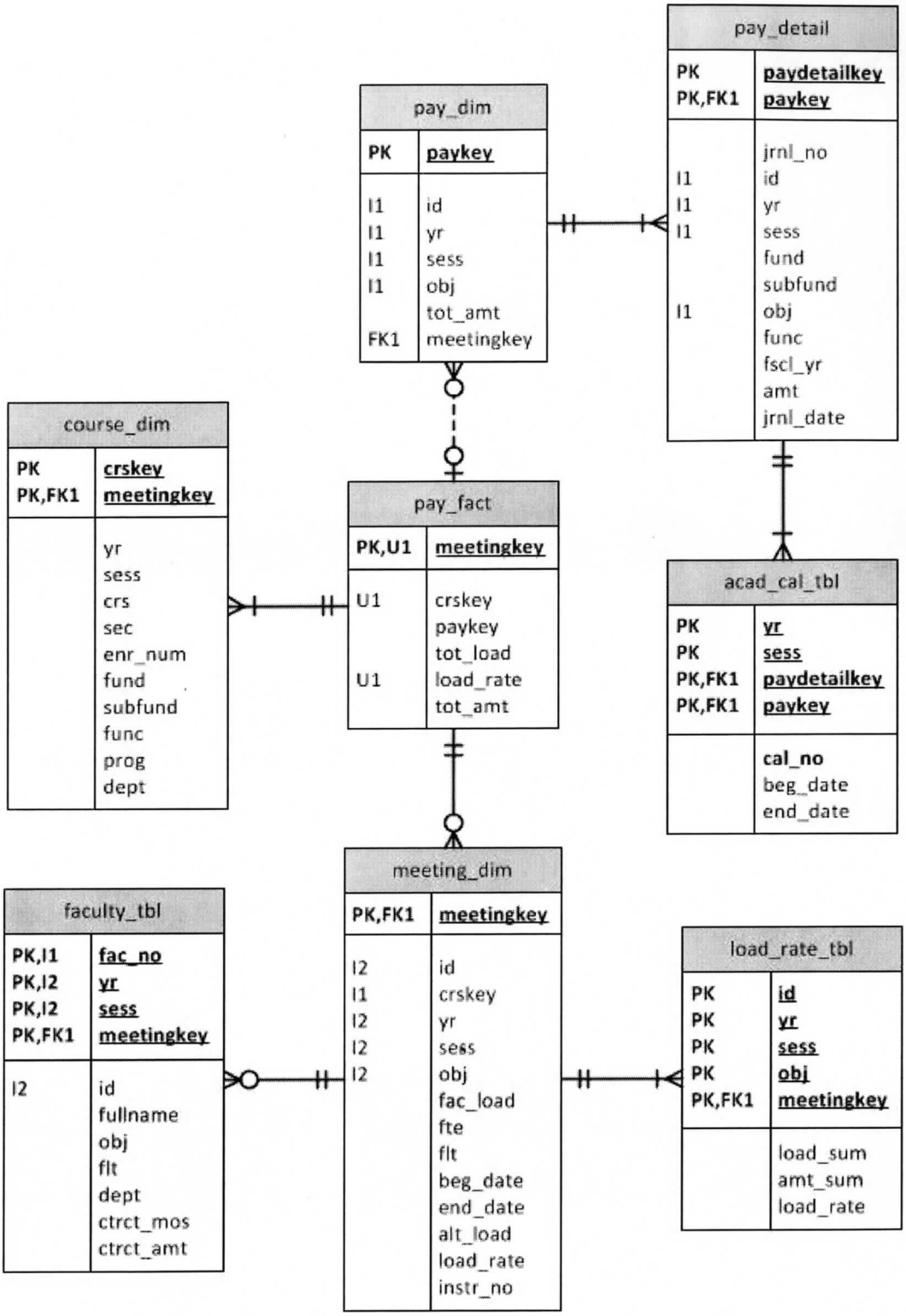


Figure 4. Data Warehouse Diagram

There is considerable overlap between the course and instructor information, but I elected to put course and section information in one table (course_dim) and meeting/instructor in another (meeting_dim). One reason being, an account number within our system requires four parts; fund, subfund, object, and function. The course master supplies the fund, subfund, and function portions of the faculty pay account number and the instructor records supply the object, but I felt it was too early in the process to bring the values together. More importantly, non-instructor (or release time) values have no corresponding section records and having both instructor and non-instructor values in one table would complicate things considerably. Therefore, course and section information is separated into the course_dim table and meeting/instructor/non-instructor information is placed in the meeting_dim table.

Table 7. Tables used in the data warehouse

Table Name	Description
acad_cal_tbl	Beginning and ending dates for every year and session.
course_dim	Course information
faculty_tbl	Information specific to each instructor
load_rate_table	Dollar amounts per load hour for each instructor and G/L code
meeting_dim	Instructors and meetings for each course section.
pay_detail	Pay records for each instructor
pay_dim	Aggregated pay records.
pay_fact	Dollar pay amounts for each course, section, meeting, and instructor.

The pay_detail table holds all relevant pay records for a given year, but lacks a semester value to facilitate matching of pay data with instructor meeting records, to resolve this problem a stored procedure "set_pay_sess" was created. Initially all pay_detail records are created with an arbitrary semester of Summer ("SU"), when the set_pay_sess procedure is run it determines if the journal date indicates whether a specific record was created before Summer started, meaning it occurred in the Spring, or after the Fall term started, giving it a session of "FA". Once this process is complete, the stored procedure "load_pay_dim" is run to sum the pay records for each instructor by year, object, and session and then move the resulting data into pay_dim.

The purpose of this project is to give administrators the ability to run reports that supply the cost of offering a specific course. To accomplish that we need to determine how much the instructor or instructors were paid. My research showed that the best way to

accomplish this was to take the total amount paid to an individual and then divide it by the amount of Load carried during the same period of time. This process is performed by the stored procedure "calc_load_rate" which then puts the resulting information into the table load_rate_tbl.

Once the course_dim, meeting_dim, pay_dim, and load_rate_tbl tables have been populated we have all of the necessary data and are ready to move it into the pay_fact table. This is accomplished by the stored procedure "load_pay_dim" which coordinates records from all four tables in pay_fact records. Once this is accomplished, reports can be run against the data warehouse.

A list of the stored procedures in the order they should be run (along with a short description) is shown in table 8.

Table 8. List of stored procedures

Name	Parameters	Description
upd_mtg_obj	Calendar year	Populate blank fac_obj fields.
upd_fac_load	Calendar year	Populate missing fac_load values.
adjust_load	Calendar year	Recalculates load when the course and instructor FLT values are different.
set_pay_sess_sp	Beginning date for year Ending date for year Calendar year	Assign sessions to pay records for Spring.
set_pay_sess_fa	Beginning date for year Ending date for year Calendar year	Assign sessions to pay records for Fall.
load_pay_dim	Calendar year	Aggregate pay data from pay_detail
calc_load_rate	Calendar year	Assign a dollar value to each load hour.
load_pay_fact	Calendar year	Move data to the fact table and assign a dollar value to each record.

One other table not discussed above is the faculty_tbl which was created to hold data regarding faculty members that might prove useful in analysis, but has little or no bearing on faculty pay.

Uploading information to the course_dim, meeting_dim, pay_dim, acad_cal_tbl, and faculty_tbl tables is accomplished using scripts created especially for this purpose. These

scripts run the appropriate SQL statement against the ERP database, save the resulting data to a text file, and then upload the file to the appropriate data warehouse table. A full list of the scripts with a brief description is shown in table 9, the contents of each script can be viewed in appendix D.

Table 9. Scripts used to extract data for the warehouse

Script Name	Parameters	Description
dwinstrpay.scr	Calendar Year	Calculates adjunct and overload pay on the production database, data is then included by meetingdim.scr.
acadcaltbl.scr	Calendar Year	Defines beginning and ending dates for each session.
coursedim.scr	Calendar Year	Provides details of each course and section combination.
facultytbl.scr	Calendar Year	Supplies general information about each faculty member.
meetingdim.scr	Calendar Year	Details of course meetings, and the instructor assigned.
paydetail.scr	Calendar Year	Amounts paid
runinstrdw.scr	Beginning Date, Ending Date, Calendar Year	A master script that launches all others.

All records relating to sections, meetings, and instructor assignments have a year and session, but corresponding values do not exist in the general ledger, I therefore decided to find a parameter for the scripts that could cross this boundary. The calendar year serves that purpose since all section records have a year component and general ledger records have entry dates. For example, a query of section records for 2010 and general ledger records that fall between 01/01/2010 and 12/31/2010 will capture all instructor records and corresponding pay records for that year. As an added bonus, the two week break at Christmas when almost no activity occurs, acts as a buffer to prevent overlap from one year to the next. Therefore, all scripts except the one for the academic calendar (which is quite small) only require the desired year as a parameter.

Implementation

The data warehouse itself is located in a MySQL database for two reasons. First, as stated earlier, one goal of the project is to allow complex reporting to be undertaken without affecting

performance on the production database, something which is currently impossible. And secondly, a data warehouse currently exists containing a great deal of information relating to courses and student enrollment. Tt is my hope that placing faculty information within the same database will enhance the usefulness of the data collected.

Since the choice of using a MySQL database was essentially made for me and I am unfamiliar that system (at the present time), some study was required before I could create either tables within the database or scripts to populated those tables. Once this had been accomplished, work proceeded on creating the required SQL statements, scripts, tables and related stored procedures discussed in the section labeled "Data Warehouse Design".

One note regarding the importing of data: as has been stated, we will be extracting data from Daytona State's ERP database, which is an Informix system, and bring it into a MySQL database, a completely different animal. Since information can't be moved directly from one system to the other, it was necessary to export data from Informix into a neutral medium (i.e. a text file) and import the resulting file(s) into the MySQL tables.

With the data warehouse created and information safely stored within it, my attention now turned to reporting. We have several reporting tools available at DSC, such as Crystal Reports (which I have used for a number of years) and IBM's Cognos (which I am just beginning to learn). My original intention was to create reports in Cognos, but that is not proving to be expedient at the present time, so the initial set of reports will be created using Crystal Reports.

CASE STUDY (RESULTS AND DISCUSSION)

A fundamental principle of this project was to glean salary information from actual payment records in the general ledger. The following reports from the data warehouse (where identifying information has been replaced with record numbers) validate that this basic goal was achieved.

Record Number	Contract Salary	Estimated Salary
40	51,000.00	52,500.09
94	62,100.00	64,876.46
114	51,800.00	53,350.04
134	56,100.00	57,650.04
188	80,900.00	109,820.37
209	51,000.00	52,110.45
231	62,800.00	82,566.65
275	72,200.00	92,563.32

Figure 5. Comparison of salaries for 9 month faculty documented in the live data base with those calculated from the data warehouse

Contract Months	Record Number	Contract Salary	Estimated Salary
12			
	7,140	107,800.00	82,950.02
	7,190	91,600.00	93,500.17
	7,215	116,000.00	118,149.96
	7,255	89,500.00	91,400.17
	7,305	81,900.00	81,350.00
	7,330	85,100.00	86,949.96
	7,380	66,700.00	62,149.84
	7,430	110,300.00	112,400.09

Figure 6. Comparison of salaries for 12 month faculty documented in the live data base with those calculated from the data warehouse

Most of the amounts from the data warehouse are very close to those in the live database. Several instances where the amounts differed greatly were investigated and it was found that the Load carried by the instructor was correspondingly low, this would indicate the individual could have taken an extended leave, been ill, or was otherwise prevented from carrying a full load and subsequently earing his or her full salary. Cases where the difference is relatively slight could (possibly) be explained by a change in salary. The logic for that being...if an individual's salary were to change, there would be an immediate change in amounts appearing in the general ledger, but the stated salary would no longer be accurate.

Results provided by the initial Crystal Report were interesting for several reasons. First, it was only then that I realized there's really no way to verify the results for fulltime instructors, those teaching under the 52001 general ledger code. We have ample documentation via the course schedules that they taught during a specific period of time. We also know the instructors carried a certain amount of "load" which is documented in the course meeting records. And finally, the general ledger documents that faculty were paid specific amounts of money under specific general codes.

Whatever a specific individual's records might give as his or hers salary, or the official pay rates say they should be paid for teaching as an adjunct, the general ledger says the individual was given a specific amount of money under a specific general ledger code during a specific period of time. The basic logic I used was to total the amount paid during a calendar year, the amount of load carried during that same period of time, and then divide the totals together and come up with a dollar amount per load hour. The resulting load rate could then be used to calculate the cost for specific courses by multiplying an instructor's load amount for the course by the load rate.

Figure 7. Crystal Report linking

Since the basic logic is very simple, the method described above should work very well for instructors teaching under the 52001 general ledger and provide very reasonable, useful estimates. This is especially true when you consider that – at the very least, since moving to the current ERP system – the ability to produce estimates of any kind has never existed, at all.

However, results for the Adjunct and Overload general ledger codes were dismal. In some cases, costs matched very closely those produced by currently used reporting tools, but in other case they were wildly off. The puzzling aspect of this was the general ledger records themselves, payments could be found that tracked very closely with amounts predicted for some course, but in others additional payments were present. At this point in time, I have no idea what those payments could be – perhaps a bonus of some type? – but they are clearly defined in the general ledger as either Adjunct or Overload amounts. In any event, I believe the extremely complex pay structure for these courses is the root of the trouble and make it quite impossible to successfully apply the method used for fulltime instructors. Table 10, shown below, will give you some of idea of the complexities involved.

Table 10. Rates used to calculate Adjunct and Overload pay

Rate Type	Subtype	Description	Rate
AE		Adult Ed hourly	$27.00
AE	52102	Non-instructor	$27.00
AE	ARC	Adults with disabilities	$24.00
AE	AWPR	Adults with disabilities	$24.00
AE	CONC	Adults with disabilities	$24.00
AE	O	AE Overload rate	$32.00
AE	WORC	Adults with disabilities	$24.00
CC		College Credit	$40.00
CC	CO	Co-Op Courses	$40.00
CC	D	Direct Instruction	$40.00
CC	JT	On Job Training	$40.00
CC	ON	Online	$38.00
CC	PL	Private Lessons	$40.00
CC	TBAS	College Credit	$45.00
CC	TBED	College Credit	$45.00
CC	TBET	College Credit	$45.00

The rates contained in table 10 are applied to "contract hours" that have been assigned to each instructor meeting record and as you can see there is a multitude of possibilities. The easy part is the different rates depending on whether the course is either College Credit or Adult Education. But it is the "Subtype" category that makes things more interesting because the individual subtypes don't apply to courses in a consistent way. For example, the 52102 subtype for AE is simply a general ledger code, but the ARC subtype refers to a subprogram code. Likewise for College Credit courses, where the values "CO", "ON" "JT", etc. are instructional methods, "TBAS", TBED" and "TBET" are specific tuition codes.

Contract hours were mentioned earlier. Normally contract hours are created when instructors are assigned to a specific meeting, but this not always true. For example, Online courses have a variable rate, their contract hours calculated by multiplying the course's load value by the number of enrolled students. A similar principle applies to several other instructional methods. Figure 8 contains an excerpt from a reporting application that deals with these complexities.

```
UPDATE db_facload_rec
SET cntr_hrs = reg_num * (7.5*hrs),
crspay = reg_num * (7.5*hrs) * crsrate
WHERE prog = 'CC'
AND im = 'PL'
AND crs_no <> 'MUN1362'
AND cntr_hrs > 0
```

Figure 8. A sample pay calculation SQL statement

Not surprisingly similar complications come into play for calculating load values where rates determined by the ERP (or our home grown Catalog Maintenance) application aren't deemed to be accurate. I will not go into a discussion on this subject, but have included a sample of code from one of our Builder C++ applications to give you some idea.

Because of the delicate manipulations that go into calculating Adjunct and Overload pay as well as load values, the method used for Inload courses could not be applied. I instead reproduced the long, seemingly convoluted series of calculations, an example of which can be seen in Figure 8, in a series of SQL statements, the resulting (and successful) script "DWINSTRPAY.SCR") can be viewed in Appendix F. Excerpts from various reports demonstrating this success can be seen on the following pages. It should be noted that any identifying numbers have been removed and replaced with record numbers that are useless for linking this information to any individual.

course-section	fac load	obj	tot amt
CJK0020-A1	0.53	56001	319.99
CJK0020-A2	0.53	56001	319.99
CJK0020-A2	0.53	56001	319.99
CJK0040-AU	0.27	56001	160.02
CJK0040-AU	0.53	56001	319.99
CJK0040-AU	0.53	56001	319.99
CJK0040-AU	2.13	56001	1,280.01
CJK0040-SP	2.13	56001	1,280.01
CJK0040-SP	1.07	56001	640.03
	Total for G/L code:		**4,960.01**
	Total for Instructor:		**4,960.01**

Figure 9. Report showing instructor pay from the data warehouse for the Adjunct (56001) general ledger code

Payment Schedule

Cost Center	Course		10/15/2010	10/29/2010	11/30/2010
10-111270201	CJK0020-A1		$320.00		
10-111270201	CJK0020-A2		$640.00		
10-111270201	CJK0040-AU			$2,080.00	
10-111270201	CJK0040-SP				$1,920.00
		Totals:	$960.00	$2,080.00	$1,920.00

Payment Schedule Total: $4,960.00

Figure 10. Report showing instructor pay from the live database for the Adjunct (56001) general ledger code

Course/Section Faculty ID	G/L	Load	Load Rate	Totals
CJK0020-A1				
4	52001	3.20	1,184.76	3,791.25
7	56001	2.67	642.88	1,714.23
9	56001	1.60	815.84	1,305.27
12	56001	1.50	2,334.44	3,501.52
13	56001	0.53	600.01	319.99
18	56001	3.20	681.63	2,181.07
19	56001	0.53	685.07	365.35
			Total for this section:	**13,178.67**

Figure 11. Excerpt from course-section expense report for the Adjunct (56001) general ledger code

You can see in Figures 9 and 11 that for course CJK0020, section A1 the amount $319.99 appears and that Figure 10 has a slightly different amount of $320 (due to rounding). All of these values apply to the same person thereby testifying to the consistency and accuracy of the process. Similar results can be seen in Figures 13 through 14 below regarding NUR1423L, section 82.

course-section	fac load	obj	tot amt
NUR1423-01	2.00	52001	1,912.53
NUR1423-03	2.00	52001	1,912.53
NUR1423-80	2.00	52001	1,912.53
NUR1423L-12	4.43	52001	4,232.62
NUR1423L-32	1.57	52001	1,504.97
NUR1423L-81	1.57	52001	1,504.97
NUR1423L-82	3.93	52001	3,759.27
NUR1423L-82	0.92	52001	880.62
NUR1423L-83	1.57	52001	1,504.97
	Total for G/L code:		**19,125.01**
NUR1423L-82	0.65	52101	391.80
	Total for G/L code:		**391.80**
	Total for Instructor:		**19,516.81**

Figure 12. Report showing instructor pay from the data warehouse for the Overload (52101) general ledger code

G/L Code 52101 - FT Faculty Overload

Course	Sec	Ref	Prog	Dept	Cost Center	Rate	Load	Pay Hrs	Pay
NUR1423L	82	2343	CC	NUR	10-111230102	$40.00	0.653	9.795	$391.80
	Dt: 08/30/2010~12/17/2010 Tm: 02:00 pm~03:20 pm Days: ---W---								
	Camp: 0007 Bldg: 0001 Rm: 128A								
					Subtotals:		0.65		$391.80
					Total Load and Released Time:		0.00		
					Grand Totals		**20.64**		**$391.80**

Figure 13. Report showing instructor pay from the live database for the Overload (52101) general ledger code

Course/Section Faculty ID	G/L	Load	Load Rate	Totals		
NUR1423L-82						
2	52001	4.85	956.26	4,639.89		
3	52001	0.50	1,051.88	520.68		
4	52101	0.65	600.00	391.80		
				Total for this section:	**5,552.37**	
				Total for the course:		**5,552.37**
						$5,552.37

Figure 14. Excerpt from course-section expense report for the Overload (52101) general ledger code

CONCLUSIONS

This project was able to achieve its objective of supplying cost information for courses offered by Daytona State College. Because there is no direct link between amounts paid to instructors and the courses they teach, it was necessary to estimate the values, but they are very reasonable estimates that will work nicely for planning and estimating purposes.

So far as deliverables are concerned, it was my original intention to create reports using the Cognos tool that Daytona State is currently implementing, but that isn't possible at this point in time. I have instead elected to use Crystal Reports for the initial set of reports since it is a very powerful reporting that is well established at this college and can be used and understood by a wide number of people. It has one other advantage, we have a home-grown application at Daytona State called "Kaleidoscope" that has the ability to display Crystal Reports. Until such time as reports can be created using Cognos, Crystal Reports provides a very effective means of distributing information gleaned from the new data warehouse. An example of a very simple report is shown in figure 13 (with redacted employee ID numbers, of course).

2,010 FA

Course	Section	Redacted	G/L	Load	Load Rate	Totals		
NUR1423L	11							
		2	56001	6.00	642.65	3,855.93		
						Total for this section:	3,855.93	
NUR1423L	12							
		3	52001	4.43	956.26	4,232.62		
		4	52001	1.57	1,051.88	1,655.44		
						Total for this section:	5,888.06	
NUR1423L	13							
		5	52001	1.57	1,051.88	1,655.44		
		6	56001	4.43	642.65	2,844.52		
						Total for this section:	4,499.96	
NUR1423L	31							
		8	56001	6.00	642.65	3,855.92		
						Total for this section:	3,855.92	
NUR1423L	32							
		9	52001	1.57	956.26	1,504.97		
		10	56001	4.43	586.44	2,595.72		
						Total for this section:	4,100.69	
NUR1423L	33							
		11	52001	1.50	1,051.88	1,577.81		
		12	56001	4.50	560.00	2,520.00		
						Total for this section:	4,097.81	
NUR1423L	56							
		13	52001	0.86	1,051.88	901.88		
		14	56001	5.14	642.65	3,304.91		
						Total for this section:	4,206.79	
NUR1423L	57							
		16	52001	6.00	1,051.88	6,311.25		
						Total for this section:	6,311.25	
NUR1423L	58							
		18	52001	6.00	1,051.88	6,311.25		
						Total for this section:	6,311.25	
NUR1423L	81							
		19	52001	1.57	956.26	1,504.97		
		20	56001	4.43	586.44	2,595.72		
						Total for this section:	4,100.69	
NUR1423L	82							
		22	52001	4.85	956.26	4,639.89		
		23	52001	0.50	1,051.88	520.68		
		24	52101	0.65	600.00	391.80		
						Total for this section:	5,552.37	
NUR1423L	83							
		25	52001	1.57	956.26	1,504.97		
		26	56001	4.43	586.44	2,595.72		
						Total for this section:	4,100.69	
						Total for the course:		56,881.41
								$56,881.41

Figure 13. An excerpt from a Crystal Report detailing course expenses for a nursing lab

One thing I learned from this project is that extracting data from our production database and arranging within the data warehouse in a useful form was trickier than I first thought. The data warehouse itself went through several iterations before settling into the form presented in this paper. I also learned a few syntax things related to the scripts that had

to be created and noticed that SQL doesn't always behave the same way on a MySQL database as it does on the Informix database I use every day.

Once this data warehouse proves itself, in the future I hope to add more detail regarding courses and fulltime instructors. And, perhaps to someday devise a method for reliably calculating Adjunct and Overload expenses. Also, I am now wondering if it would be possible to expand the dataset beyond instructors to include other expenses associated with courses, supplies used in a chemistry or biology class would be good examples of this.

Sadly, compiling data for the Adjunct and Overload instructors using the method applied to fulltime faculty failed, but data was successfully transferred using existing calculations so that information for three classes of faculty will be available.

In closing, let me just say that while the data structure within the ERP system makes it impossible to be 100% accurate in matching faculty salaries with courses taught, my data warehouse provides very reasonable estimates. The simple fact that any estimates are available at all is a huge step forward since no data of any kind was available before the data warehouse was constructed. And finally, this system should provide very useful information to help guide administrative decision making and its importance will only grow as the years go by and more data is added to the system.

REFERENCES

Jenzabar, Inc. (2005). General Ledge Reference, Technical Manual. Cambridge, MA, USA.

Jenzabar, Inc. (2005). Human Resources User Guide. Cambridge, MA, USA.

Jenzabar, Inc. (2005, December 22). Master Glossary. Cambridge, MA, USA.

Jenzabar, Inc. (2007, June 26). General Ledger User Guide. Cambridge, MA, USA.

APPENDICES

APPENDIX A: WORK BREAKDOWN STRUCTURE

1 Analyze production database
1.1 Study CARS/Jenzabar documentation
1.2 Document table structure
1.3 Meet with individual department heads
1.4 Analysis complete
2 Create method to extract data
2.1 Identify needed datafields
2.2 Create SQL's to extract required data
2.2.1 Design SQL statements
2.2.2 Test the statements in a controlled environment
2.3 Develop export files
2.3.1 Format data from existing SQL's
2.3.2 Perform a test run of the modified statements
2.3.3 Check output files for accuracy
2.4 Schedule run times for data extraction
2.5 Data extraction method complete
3 Create data warehouse
3.1 Study MySQL documentation
3.2 Create schemas for data warehouse tables
3.2.1 Design table schemas
3.2.2 Implement table schemas
3.2.3 Correct schema errors
3.2.4 Schemas complete
3.3 Develop upload SQL statements
3.3.1 Create SQL's to upload text files

3.3.2 Run statements in controlled environment with test data

3.3.3 Analyze results

3.3.4 "Correct errors, if any are found"

3.3.5 Upload SQL's complete

3.4 Data warehouse complete

4 Create Reports

4.1 Select reporting tool

4.2 Design reports

4.3 Create reports

4.4 Reports complete

5 Release data warehouse

5.1 Create needed documentation

5.2 Distribute reports

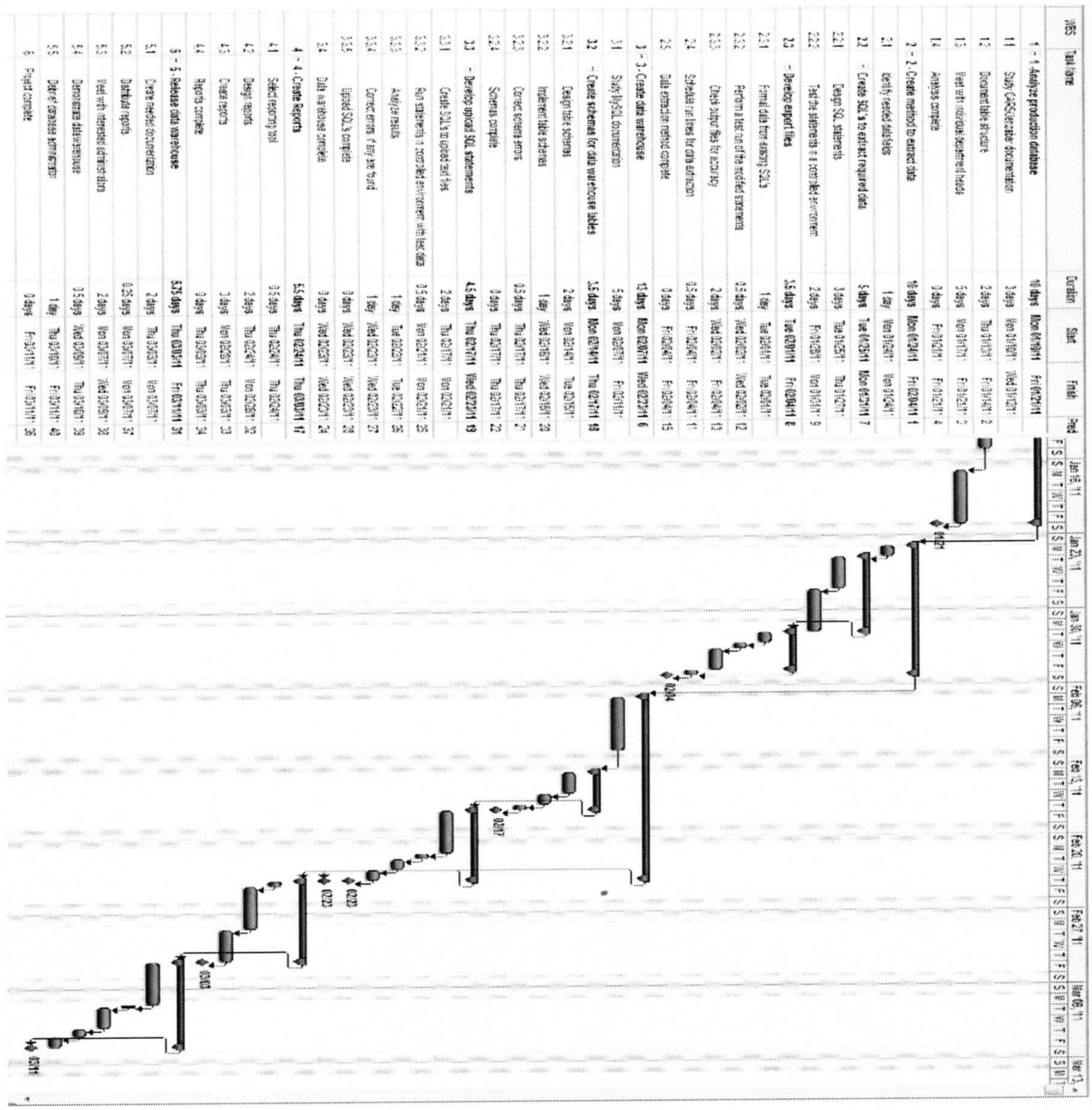
5.3 Meet with interested administrators

5.4 Demonstrate datawarehouse

5.5 Debrief database administrator

6 Project complete

APPENDIX B: GANTT CHART



APPENDIX C: TABLE CREATION STATEMENTS

acad_cal_tbl:

```
CREATE
   TABLE johntest.acad_cal_tbl
   (
      cal_no INT AUTO_INCREMENT,
      yr INT,
      sess CHAR(4),
      beg_date DATE,
      end_date DATE,
      PRIMARY KEY (yr, sess),
      CONSTRAINT ix1 UNIQUE (cal_no),
      CONSTRAINT ix2 UNIQUE (yr, sess)
   )
   ENGINE=MyISAM DEFAULT CHARSET=latin1
```

course_dim:

```
CREATE
  TABLE johntest.course_dim
  (
    crskey INT,
    yr SMALLINT,
    sess CHAR(4),
    crs_no CHAR(12),
    sec_no CHAR(2),
    enr_num INT,
    fund CHAR(2),
    subfund CHAR(9),
    func CHAR(4),
    prog CHAR(4),
    dept CHAR(4),
    PRIMARY KEY (crskey),
    CONSTRAINT ix1 UNIQUE (yr, sess, crs, sec),
    CONSTRAINT ix2 UNIQUE (crskey)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

faculty_tbl:

```
CREATE
  TABLE johntest.faculty_tbl
  (
    fac_no INT AUTO_INCREMENT,
    id INT,
    fullname CHAR(32),
    yr SMALLINT,
    sess CHAR(4),
    obj CHAR(5),
    flt CHAR(2),
    dept CHAR(4),
    ctrct_mos SMALLINT,
    ctrct_amt FLOAT,
    PRIMARY KEY (id, yr, sess),
    CONSTRAINT ix1 UNIQUE (fac_no),
    CONSTRAINT ix2 UNIQUE (id, yr, sess)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

load_rate_tbl:

```
CREATE
   TABLE johntest.load_rate_tbl
   (
      id INT,
      yr SMALLINT,
      sess CHAR(4),
      obj CHAR(5),
      load_sum FLOAT,
      amt_sum FLOAT,
      load_rate FLOAT,
      CONSTRAINT ix1 UNIQUE (id, yr, sess, obj),
      PRIMARY KEY (id, yr, sess, obj)
   )
   ENGINE=MyISAM DEFAULT CHARSET=latin1
```

meeting_dim:

```
CREATE
  TABLE johntest.meeting_dim
  (
    meetingkey INT NOT NULL AUTO_INCREMENT,
    id INT,
    crskey INT,
    yr INT,
    sess CHAR(4),
    obj CHAR(5),
    fac_load FLOAT,
    fte FLOAT,
    flt CHAR(2),
    beg_date DATE,
    end_date DATE,
    alt_load FLOAT,
    load_rate FLOAT,
    instr_no INT,
    crspay FLOAT,
    im CHAR(2),
    crs_no CHAR(12),
    sec_no CHAR(2)
    PRIMARY KEY (meetingkey),
    CONSTRAINT ix1 UNIQUE (meetingkey),
    INDEX ix2 (yr, sess, crs_no, sec_no),
    INDEX ix3 (id, yr, sess, obj)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

noninstr_tbl:

```
CREATE
  TABLE johntest.noninstr_tbl
  (
    noninstr_no INT NOT NULL,
    yr INT,
    sess CHAR(4),
    id INT,
    accrl_meth CHAR(1),
    fte FLOAT,
    fac_load FLOAT,
    fac_dept CHAR(4),
    obj CHAR(5),
    fac_flt CHAR(2),
    PRIMARY KEY (noninstr_no),
    CONSTRAINT ix1 UNIQUE (yr, sess, id, obj),
    CONSTRAINT ix2 UNIQUE (noninstr_no)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

pay_detail:

```
CREATE
  TABLE johntest.pay_detail
  (
    paydetailkey INT NOT NULL AUTO_INCREMENT,
    jrnl_no INT,
    id INT,
    yr SMALLINT,
    sess CHAR(4),
    fund CHAR(2),
    subfund CHAR(9),
    obj CHAR(5),
    func CHAR(4),
    fscl_yr CHAR(4),
    amt FLOAT,
    jrnl_date DATE,
    PRIMARY KEY (paydetailkey),
    CONSTRAINT ix1 UNIQUE (paydetailkey),
    INDEX ix2 (id, yr, sess, obj)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

pay_dim:

```
CREATE
  TABLE johntest.pay_dim
  (
    paykey INT NOT NULL AUTO_INCREMENT,
    id INT,
    yr INT,
    sess CHAR(4),
    obj CHAR(5),
    tot_amt FLOAT,
    PRIMARY KEY (paykey),
    CONSTRAINT ix1 UNIQUE (paykey),
    INDEX ix2 (id, yr, sess, obj)
  )
  ENGINE=MyISAM DEFAULT CHARSET=latin1
```

pay_fact:

```
CREATE
   TABLE johntest.pay_fact
   (
      instr_no INT,
      crskey INT,
      paykey INT,
      tot_load FLOAT,
      load_rate FLOAT,
      tot_amt FLOAT,
      PRIMARY KEY (instr_no),
      CONSTRAINT ix1 UNIQUE (instr_no, crskey, paykey)
   )
   ENGINE=MyISAM DEFAULT CHARSET=latin1
```

APPENDIX D: EXTRACTION SCRIPTS

dwinstrpay.scr

```
#!/bin/csh -f | more
#!/bin/sh
#################################################################
## This script populates dwinstrpay_rec                ##
##                                                    ##
## It then loads the information into the mysql system    ##
#################################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/dwinstrpay.dat

echo "Extracting course, instructor data into " $datfil

isql cars << END_SQL > & dwinstrpay.err

SET ISOLATION TO DIRTY READ;

DELETE FROM dwinstrpay_rec
WHERE dwinstrpay_rec.yr = $in_yr;

INSERT INTO dwinstrpay_rec
(instr_no,
id,
seckey,
mtg_no,
facdept,
ctrct_mos,
db_ctrct,
crsrate,
crspay,
facflt,
ctrct_amt,
facobj,
fullname,
crsobj,
crsflt,
fte,
pct,
cntr_hrs,
accrl_meth,
```

```
instr_days,
instr_beg_time,
instr_end_time,
instr_beg_date,
instr_end_date,
tothrs2,
adjload,
release_load,
release_fte,
campus,
bldg,
room,
im,
calc_tot_hrs,
tot_hrs,
yr,
sess,
subsess,
stat,
reg_num,
enr_num,
max_reg,
hrs,
mtg_beg_date,
mtg_end_date,
sec_locator,
subprog,
sec_beg_date,
sec_end_date,
sec_grp_no,
sec_guaranteed,
prog,
crs_dept,
crsctgry,
tuit_code,
crs_no,
sec_no,
cat,
fund,
subfund,
function,
crs1title,
crs2title)
SELECT
instr_rec.instr_no,
fac_rec.id,
```

dwseckey.seckey,
mtg_rec.mtg_no,
fac_rec.dept,
fac_rec.ctrct_mos,
fac_rec.db_ctrct,
0.0 crsrate,
0.0 crspay,
fac_rec.flt,
fac_rec.ctrct_amt,
fac_rec.obj,
id_rec.fullname,
instr_rec.obj,
instr_rec.flt,
instr_rec.fte,
instr_rec.pct,
instr_rec.cntr_hrs,
instr_rec.accrl_meth,
db_instr2_rec.days,
db_instr2_rec.beg_time,
db_instr2_rec.end_time,
db_instr2_rec.beg_date,
db_instr2_rec.end_date,
db_instr2_rec.tot_hrs,
mtg_rec.tot_hrs,
noninstr_rec.load,
noninstr_rec.fte,
mtg_rec.campus,
mtg_rec.bldg,
mtg_rec.room,
mtg_rec.im,
mtg_rec.calc_tot_hrs,
mtg_rec.tot_hrs,
sec_rec.yr,
sec_rec.sess,
sec_rec.subsess,
sec_rec.stat,
sec_rec.reg_num,
sec_rec.enr_num,
sec_rec.max_reg,
sec_rec.hrs,
sec_rec.beg_date,
sec_rec.end_date,
sec_rec.sec_locator,
sec_rec.subprog,
mtg_rec.beg_date,
mtg_rec.end_date,

```
db_sec2_rec.sec_grp_no,
db_sec2_rec.sec_guaranteed,
crs_rec.prog,
crs_rec.dept,
crs_rec.crsctgry,
crs_rec.tuit_code,
crs_rec.crs_no,
sec_rec.sec_no,
crs_rec.cat,
crs_rec.fund,
crs_rec.subfund,
crs_rec.function,
crs_rec.title1 || ' ' || crs_rec.title2 || ' ' || crs_rec.title3,
db_crs2_rec.title
FROM
 id_rec,
 fac_rec,
 instr_rec,
 mtg_rec,
 secmtg_rec,
 sec_rec,
 crs_rec,
 dwseckey,
 OUTER db_crs2_rec,
 OUTER db_sec2_rec,
 OUTER db_instr2_rec,
 OUTER noninstr_rec
WHERE
 crs_rec.crs_no=db_crs2_rec.crs_no
 AND crs_rec.cat=db_crs2_rec.cat
 AND crs_rec.crs_no = sec_rec.crs_no
 AND crs_rec.cat = sec_rec.cat
 AND sec_rec.yr = $in_yr
 AND sec_rec.cat=db_sec2_rec.cat
 AND sec_rec.yr=db_sec2_rec.yr
 AND sec_rec.sess=db_sec2_rec.sess
 AND sec_rec.crs_no=db_sec2_rec.crs_no
 AND sec_rec.sec_no=db_sec2_rec.sec_no
 AND sec_rec.yr = noninstr_rec.yr
 AND sec_rec.sess = noninstr_rec.sess
 AND sec_rec.crs_no = secmtg_rec.crs_no
 AND sec_rec.cat = secmtg_rec.cat
 AND sec_rec.yr = secmtg_rec.yr
 AND sec_rec.sess = secmtg_rec.sess
 AND sec_rec.sec_no = secmtg_rec.sec_no
 AND dwseckey.crs_no = secmtg_rec.crs_no
```

```
 AND dwseckey.cat = secmtg_rec.cat
 AND dwseckey.yr = secmtg_rec.yr
 AND dwseckey.sess = secmtg_rec.sess
 AND dwseckey.sec_no = secmtg_rec.sec_no
 AND secmtg_rec.mtg_no = mtg_rec.mtg_no
 AND mtg_rec.mtg_no = instr_rec.mtg_no
 AND instr_rec.instr_no = db_instr2_rec.instr_no
 AND instr_rec.id = noninstr_rec.id
 AND instr_rec.id = fac_rec.id
 AND fac_rec.id = id_rec.id;

UPDATE dwinstrpay_rec
 SET
 tothrs2 = ((reg_num * hrs) / 15),
 tot_hrs = ((reg_num * hrs) / 15)
 WHERE
 (im = 'CO' OR im = 'D')
 AND crsctgry IN ('PSV', 'AP')
 AND yr = $in_yr;

 UPDATE dwinstrpay_rec
 SET tothrs2 = ((reg_num * hrs) / 24),
 tot_hrs = ((reg_num * hrs) / 24)
 WHERE (im = 'CO' OR im = 'D')
 AND crsctgry IN ('PSAV', 'APPR')
 AND yr = $in_yr;

 UPDATE dwinstrpay_rec
 SET tothrs2 = (reg_num / 5),
 tot_hrs = (reg_num / 5)
 WHERE im = 'JT'
 AND crs_dept NOT IN ('APP','DAS')
 AND yr = $in_yr;

 UPDATE dwinstrpay_rec
 SET tothrs2 = ((enr_num * hrs) / 15),
 tot_hrs = ((enr_num * hrs) / 15)
 WHERE im = 'ON'
 AND crs_dept = 'APP'
 AND crsctgry IN ('PSV', 'AP')
 AND yr = $in_yr;

 UPDATE dwinstrpay_rec
 SET adjLoad = ((tothrs2/CAST(crsflt as int))*CAST(facflt as int))
 WHERE tot_hrs > 0
 AND (crsflt <> 'PR' and facflt <> 'PR')
```

```
 AND yr = $in_yr;

UPDATE dwinstrpay_rec
SET adjLoad = ((tot_hrs/CAST(crsflt as int))*CAST(facflt as int))
WHERE tot_hrs > 0
 AND (crsflt <> 'PR' and facflt <> 'PR')
 AND yr = $in_yr;

UPDATE dwinstrpay_rec
SET dwinstrpay_rec.crsrate = (select crs_rate
  from db_crsrate_table
  where db_crsrate_table.rate_type = dwinstrpay_rec.prog
  and (db_crsrate_table.sub_type is null or db_crsrate_table.sub_type = '')
  and db_crsrate_table.inactive_date is null)
WHERE dwinstrpay_rec.prog = 'CC'
 AND dwinstrpay_rec.yr = $in_yr
 AND dwinstrpay_rec.crsobj != '52001';

UPDATE dwinstrpay_rec
SET dwinstrpay_rec.crsrate = (select crs_rate
  from db_crsrate_table
  where db_crsrate_table.rate_type = dwinstrpay_rec.prog
  and db_crsrate_table.sub_type = dwinstrpay_rec.im
  and db_crsrate_table.rate_type = dwinstrpay_rec.prog
  and db_crsrate_table.inactive_date is null)
WHERE dwinstrpay_rec.prog = 'CC'
 AND dwinstrpay_rec.crsobj != '52001'
 AND dwinstrpay_rec.yr = $in_yr
 AND dwinstrpay_rec.im = 'ON';

UPDATE dwinstrpay_rec
SET dwinstrpay_rec.crsrate = (select crs_rate
  from db_crsrate_table
  where db_crsrate_table.rate_type = dwinstrpay_rec.prog
  and db_crsrate_table.sub_type = dwinstrpay_rec.tuit_code
  and db_crsrate_table.inactive_date is null
  and db_crsrate_table.rate_type = dwinstrpay_rec.prog)
WHERE dwinstrpay_rec.prog = 'CC'
 AND dwinstrpay_rec.crsobj != '52001'
 AND dwinstrpay_rec.yr = $in_yr
 AND (dwinstrpay_rec.tuit_code = 'TBET'
   OR dwinstrpay_rec.tuit_code = 'TBED'
   OR dwinstrpay_rec.tuit_code = 'TBAS');

UPDATE dwinstrpay_rec
SET dwinstrpay_rec.crsrate = (select crs_rate
```

```
    from db_crsrate_table
    where db_crsrate_table.rate_type = dwinstrpay_rec.prog
    and (db_crsrate_table.sub_type is null or db_crsrate_table.sub_type = '')
    and db_crsrate_table.inactive_date is null)
  WHERE dwinstrpay_rec.prog = 'AE'
   AND dwinstrpay_rec.crsobj != '52001'
   AND dwinstrpay_rec.yr = $in_yr;

  UPDATE dwinstrpay_rec
  SET dwinstrpay_rec.crsrate = (select crs_rate
    from db_crsrate_table
    where db_crsrate_table.rate_type = dwinstrpay_rec.prog
    and db_crsrate_table.sub_type = dwinstrpay_rec.accrl_meth
    and db_crsrate_table.inactive_date is null)
  WHERE dwinstrpay_rec.prog = 'CC'
   AND dwinstrpay_rec.yr = $in_yr
   AND dwinstrpay_rec.crsobj = '52101';

  UPDATE dwinstrpay_rec
  SET dwinstrpay_rec.crsrate = (select crs_rate
    from db_crsrate_table
    where db_crsrate_table.rate_type = dwinstrpay_rec.prog
    and db_crsrate_table.sub_type = dwinstrpay_rec.crsobj
    and db_crsrate_table.inactive_date is null)
  WHERE dwinstrpay_rec.prog = 'CC'
   AND dwinstrpay_rec.yr = $in_yr
   AND dwinstrpay_rec.crsobj = '52102';

UPDATE dwinstrpay_rec
 SET dwinstrpay_rec.crsrate = (select crs_rate
   from db_crsrate_table
   where db_crsrate_table.sub_type = dwinstrpay_rec.subprog
   and db_crsrate_table.rate_type = dwinstrpay_rec.prog
   and db_crsrate_table.inactive_date is null)
 WHERE dwinstrpay_rec.prog = 'CC'
 AND (dwinstrpay_rec.subprog = 'ARC'
    OR dwinstrpay_rec.subprog = 'WORC'
    OR dwinstrpay_rec.subprog = 'AWPR'
    OR dwinstrpay_rec.subprog = 'CONC')
 AND dwinstrpay_rec.crsobj != '52001'
 AND dwinstrpay_rec.yr = $in_yr;

UPDATE dwinstrpay_rec
SET crspay = cntr_hrs * crsrate
WHERE dwinstrpay_rec.yr = $in_yr;
```

```
UPDATE dwinstrpay_rec
SET cntr_hrs = enr_num * hrs,
 crspay = (enr_num * tot_hrs * crsrate * (pct/100))
WHERE prog = 'CC'
 AND im = 'ON'
 AND yr = $in_yr;

UPDATE dwinstrpay_rec
SET cntr_hrs = reg_num * (7.5 * hrs),
 crspay = (reg_num * (7.5 * hrs) * crsrate)
WHERE prog = 'CC'
 AND im = 'PL'
 AND yr = $in_yr
 AND crs_no <> 'MUN1362';

UPDATE dwinstrpay_rec
SET cntr_hrs = (reg_num / 5) * 15,
 crspay = ((reg_num / 5) * 15 * crsrate)
WHERE prog = 'CC'
 AND im = 'JT'
 AND yr = $in_yr
 AND crs_dept <> 'APP';

UPDATE dwinstrpay_rec
SET cntr_hrs = (reg_num * hrs),
 crspay = (reg_num * hrs * crsrate)
WHERE prog = 'CC'
 AND (im = 'CO' OR im = 'D')
 AND yr = $in_yr;

END_SQL

##                                        ##
## End of the extraction sql and execution          ##
############################################################
```

acadcaltbl.scr: academic calendar information

```
#!/bin/csh -f | more
#!/bin/sh
#####################################################################
## This script is used to populate the academic calendar table##
##                                                 ##
## It then loads the information into the mysql system       ##
#####################################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/acadcaltbl.dat

echo "unloading academic calendar information" $datfil
echo "for year " $in_yr


isql cars << END_SQL > & acadcaltbl.err

SET ISOLATION TO DIRTY READ;

unload to '$datfil'
select
 0,
 yr,
 sess,
 to_char(min(beg_date), '%y-%m-%d') as beg_date,
 to_char(max(end_date), '%y-%m-%d') as end_date
from acad_cal_rec
where
 yr = $in_yr
group by yr, sess;

END_SQL

echo "data file created"
##                                        ##
## End of the extraction sql and execution          ##
#############################################################
#############################################################
## Now we load the extracted information into mysql    ##
##                                        ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL
```

```
delete from acad_cal_tbl
where yr = $in_yr;

load data LOCAL infile '$datfil'
 REPLACE into table acad_cal_tbl fields terminated by '|';

ADD_MYSQL

echo "done"
##  End of MYSQL data load                    ##
```

coursedim.scr: Course information

```
#!/bin/csh -f | more
#!/bin/sh
#############################################################
## This script is used to populate course_dim         ##
##                                                   ##
## It then loads the information into the mysql syste  ##
#############################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/coursedim.dat

echo "unloading course information to " $datfil
echo "for year " $in_yr

isql cars << END_SQL > & coursedim.err

SET ISOLATION TO DIRTY READ;

unload to '$datfil'
select
 dwseckey.seckey,
 sec_rec.yr,
 sec_rec.sess,
 sec_rec.crs_no,
 sec_rec.sec_no,
 sec_rec.enr_num,
 crs_rec.fund,
 crs_rec.subfund,
 crs_rec.function,
 crs_rec.prog,
 crs_rec.dept

from
 dwseckey,
 crs_rec,
 sec_rec
where
 dwseckey.yr = $in_yr
 and dwseckey.yr = sec_rec.yr
 and dwseckey.sess = sec_rec.sess
 and dwseckey.crs_no = sec_rec.crs_no
 and dwseckey.sec_no = sec_rec.sec_no
```

```
 and crs_rec.cat = sec_rec.cat
 and crs_rec.crs_no = sec_rec.crs_no;

END_SQL

echo "data file created"
##                                    ##
## End of the extraction sql and execution          ##
############################################################
############################################################
## Now we load the extracted information into mysql     ##
##                                    ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL

 delete from course_dim
 where course_dim.yr = $in_yr;

 load data LOCAL infile '$datfil'
  REPLACE into table course_dim fields terminated by '|';

ADD_MYSQL

echo "done"
##  End of MYSQL data load                    ##
```

meetingdim.scr: Instructor meeting information

```
#!/bin/csh -f | more
#!/bin/sh
#####################################################################
## This script populates meeting_dim
##
## It then loads the information into the mysql system
#####################################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/meetingdim.dat

echo "unloading instructor load information to " $datfil
echo "for year " $in_yr

isql cars << END_SQL > & meetingdim.err

SET ISOLATION TO DIRTY READ;

unload to '$datfil'
select
 0,
 instr_rec.id,
 dwseckey.seckey,
 instr_rec.yr,
 instr_rec.sess,
 instr_rec.obj,
 db_instr2_rec.tot_hrs,
 instr_rec.fte,
 instr_rec.flt,
 to_char(sec_rec.beg_date, '%y-%m-%d') as beg_date,
 to_char(sec_rec.end_date, '%y-%m-%d') as end_date,
 mtg_rec.tot_hrs,
 0.0,
 instr_rec.instr_no,
 dwinstrpay_rec.crspay,
 mtg_rec.im
from
 dwseckey,
 sec_rec,
 secmtg_rec,
 mtg_rec,
 instr_rec,
```

```
 outer db_instr2_rec,
 outer dwinstrpay_rec
where
 dwseckey.yr = $in_yr
 and dwseckey.yr = secmtg_rec.yr
 and dwseckey.sess = secmtg_rec.sess
 and dwseckey.crs_no = secmtg_rec.crs_no
 and dwseckey.sec_no = secmtg_rec.sec_no
 and sec_rec.yr = secmtg_rec.yr
 and sec_rec.sess = secmtg_rec.sess
 and sec_rec.crs_no = secmtg_rec.crs_no
 and sec_rec.sec_no = secmtg_rec.sec_no
 and secmtg_rec.mtg_no = mtg_rec.mtg_no
 and mtg_rec.mtg_no = instr_rec.mtg_no
 and instr_rec.instr_no = db_instr2_rec.instr_no
 and instr_rec.instr_no = dwinstrpay_rec.instr_no;

END_SQL

echo "data file created"
##                                    ##
## End of the extraction sql and execution          ##
############################################################
############################################################
## Now we load the extracted information into mysql     ##
##                                    ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL

 delete from meeting_dim
 where meeting_dim.yr = $in_yr;

 load data LOCAL infile '$datfil'
  REPLACE into table meeting_dim fields terminated by '|';

ADD_MYSQL

echo "done"
##  End of MYSQL data load                    ##
```

paydetail.scr: instructor pay information

```
#!/bin/csh -f | more
#!/bin/sh
#################################################################
## This script is used to extract instructor pay and load it into pay_detail
##
## It then loads the information into the mysql system
#################################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/paydetail.dat

echo "unloading instructor pay information to " $datfil
echo "for year " $in_yr

isql cars << END_SQL > & paydetail.err

SET ISOLATION TO DIRTY READ;

unload to '$datfil'
select
 0,
 vch_rec.jrnl_no,
 gle_rec.doc_id,
 to_char(vch_rec.jrnl_date, '%Y') as yr,
 'SU',
 gltr_rec.fund,
 gltr_rec.subfund,
 gltr_rec.obj,
 gltr_rec.func,
 vch_rec.fscl_yr,
 gltr_rec.amt,
 to_char(vch_rec.jrnl_date, '%y-%m-%d') as jrnl_date
from
 vch_rec,
 gle_rec,
 gltr_rec
where
 vch_rec.jrnl_date >= '01/01/' || $in_yr
 and vch_rec.jrnl_date <= '12/31/' || $in_yr
 and vch_rec.vch_ref = gle_rec.jrnl_ref
```

```
and vch_rec.jrnl_no = gle_rec.jrnl_no
and gle_rec.jrnl_ref = gltr_rec.jrnl_ref
and gle_rec.jrnl_no = gltr_rec.jrnl_no
and gle_rec.gle_no = gltr_rec.ent_no
and gle_rec.ca_amt < 0
and vch_rec.amt_type = 'ACT'
and gltr_rec.stat = 'P'
and (gltr_rec.obj = '52001' or
    gltr_rec.obj = '52101' or
    gltr_rec.obj = '52102' or
    gltr_rec.obj = '52103' or
    gltr_rec.obj = '56001');

END_SQL

echo "data file created"
##                                    ##
## End of the extraction sql and execution          ##
############################################################
############################################################
## Now we load the extracted information into mysql    ##
##                                    ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL

 delete from pay_detail
 where pay_detail.yr = $in_yr;

 load data LOCAL infile '$datfil'
  REPLACE into table pay_detail fields terminated by '|';

ADD_MYSQL

echo "done"
##  End of MYSQL data load                    ##
```

facultytbl.scr: information about the instructors

```
#!/bin/csh -f | more
#!/bin/sh
################################################################
## This script extracts data and loads it into faculty_tbl
##
## It then loads the information into the mysql system
################################################################

set in_yr=$1

set datfil=/work/hardebj/dwdata/facultytbl.dat

echo "unloading instructor load information to " $datfil
echo "for year " $in_yr

isql cars << END_SQL > & facultytbl.err

SET ISOLATION TO DIRTY READ;

unload to '$datfil'
select distinct
 0,
 fac_rec.id,
 id_rec.fullname,
 instr_rec.yr,
 instr_rec.sess,
 fac_rec.obj,
 fac_rec.flt,
 fac_rec.dept,
 fac_rec.ctrct_mos,
 fac_rec.ctrct_amt
from
 dwseckey,
 secmtg_rec,
 mtg_rec,
 instr_rec,
 fac_rec,
 id_rec
where
 dwseckey.yr = $in_yr
 and dwseckey.yr = secmtg_rec.yr
 and dwseckey.sess = secmtg_rec.sess
 and dwseckey.crs_no = secmtg_rec.crs_no
```

```
and dwseckey.sec_no = secmtg_rec.sec_no
and secmtg_rec.mtg_no = mtg_rec.mtg_no
and mtg_rec.mtg_no = instr_rec.mtg_no
and instr_rec.id = fac_rec.id
and fac_rec.id = id_rec.id;

END_SQL

echo "data file created"
##                                    ##
## End of the extraction sql and execution          ##
##############################################################
##############################################################
## Now we load the extracted information into mysql    ##
##                                    ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL

 delete from faculty_tbl
 where faculty_tbl.yr = $in_yr;

 load data LOCAL infile '$datfil'
  REPLACE into table faculty_tbl fields terminated by '|';

ADD_MYSQL

echo "done"
##  End of MYSQL data load                    ##
```

runinstrdw.scr: Runs all scripts and stored procedures in the correct order

```
#!/bin/csh -f | more
#!/bin/sh
#########################################################
## This script is used to all of the  instructor        ##
## pay info scripts                                     ##
##                                                      ##
#########################################################

set in_yr=$1

set in_beg_date=$in_yr"-01-01"
set in_end_date=$in_yr"-12-31"

echo " Running instructor data warehouse scripts."
echo "for year " $in_yr

./acadcaltbl.scr $in_yr;

./coursedim.scr $in_yr;

./meetingdim.scr $in_yr;

./noninstr.scr $in_yr;

./paydetail.scr $in_yr;

./facultytbl.scr $in_yr;

##                                                    ##
## End of the extraction scripts                      ##
########################################################
## Now run the mysql stored procedures                ##
##                                                    ##

/mysql/mysql/bin/mysql johntest << ADD_MYSQL

call upd_mtg_obj($in_yr);

call upd_fac_load($in_yr);

call set_pay_sess($in_beg_date, $in_end_date, $in_yr);

call load_pay_dim($in_yr);
```

```
call calc_load_rate($in_yr);

call load_pay_fact($in_yr);

ADD_MYSQL

echo "Done"
##  End of MYSQL stored procedures              ##
################################################
```

APPENDIX E: STORED PROCEDURES

upd_mtg_obj:

```
UPDATE meeting_dim
SET obj = (select obj from pay_detail
        where pay_detail.yr = meeting_dim.yr
        and pay_detail.sess = meeting_dim.sess
        and pay_detail.id = meeting_dim.id
        and pay_detail.jrnl_date >= meeting_dim.beg_date
        and pay_detail.jrnl_date <= meeting_dim.end_date
        and pay_detail.obj = '52001')
WHERE meeting_dim.yr = in_yr
AND meeting_dim.obj = '';

UPDATE meeting_dim
SET obj = (select obj from pay_detail
        where pay_detail.yr = meeting_dim.yr
        and pay_detail.sess = meeting_dim.sess
        and pay_detail.id = meeting_dim.id
        and pay_detail.jrnl_date >= meeting_dim.beg_date
        and pay_detail.jrnl_date <= meeting_dim.end_date
        and pay_detail.obj = '52101')
WHERE meeting_dim.yr = in_yr
AND meeting_dim.obj = '';

UPDATE meeting_dim
SET obj = (select obj from pay_detail
        where pay_detail.yr = meeting_dim.yr
        and pay_detail.sess = meeting_dim.sess
```

```
        and pay_detail.id = meeting_dim.id
        and pay_detail.jrnl_date >= meeting_dim.beg_date
        and pay_detail.jrnl_date <= meeting_dim.end_date
        and pay_detail.obj = '56001')
WHERE meeting_dim.yr = in_yr
AND meeting_dim.obj = '';
```

upd_fac_load:

```
CREATE PROCEDURE `johntest`.`update_fac_load`(IN in_yr smallint)
BEGIN
 UPDATE meeting_dim
 SET fac_load = alt_load
 WHERE fac_load = 0.0
 AND alt_load > 0.0
 AND yr = in_yr;
END
```

adjust_load:

```
CREATE PROCEDURE `johntest`.`adjust_load`(IN in_yr smallint)
BEGIN
 UPDATE meeting_dim
 SET meeting_dim.fac_load =
 ((meeting_dim.fac_load / cast(meeting_dim.flt as UNSIGNED))
  * (select cast(faculty_tbl.flt as UNSIGNED)
  from faculty_tbl
  where faculty_tbl.yr = meeting_dim.yr
  and faculty_tbl.sess = meeting_dim.sess
  and faculty_tbl.id = meeting_dim.id
  and faculty_tbl.flt <> 'PR'
  and faculty_tbl.flt <> ''
  and faculty_tbl.flt > 0))
 WHERE meeting_dim.fac_load > 0
 AND meeting_dim.flt <> 'PR'
 AND meeting_dim.flt <> ''
 AND meeting_dim.yr = in_yr;
END
```

set_pay_sess:

```
CREATE PROCEDURE `johntest`.`set_pay_sess`(IN beg_yr date, IN end_yr date, IN
in_yr smallint)
BEGIN
 UPDATE pay_detail
 SET pay_detail.sess = 'SP'
 WHERE pay_detail.yr = in_yr
 AND pay_detail.jrnl_date >= beg_yr
 AND pay_detail.jrnl_date < (select acad_cal_tbl.beg_date
                  from acad_cal_tbl
                  where acad_cal_tbl.sess = 'SU'
                  and acad_cal_tbl.yr = in_yr);

 UPDATE pay_detail
 SET pay_detail.sess = 'FA'
 WHERE pay_detail.yr = in_yr
 AND pay_detail.jrnl_date <= end_yr
 AND pay_detail.jrnl_date >= (select acad_cal_tbl.beg_date
                   from acad_cal_tbl
                   where acad_cal_tbl.sess = 'FA'
                   and acad_cal_tbl.yr = in_yr);
END
```

load_pay_dim:

```
CREATE PROCEDURE `johntest`.`load_pay_dim`(IN in_yr smallint)
BEGIN
 DELETE FROM pay_dim
 WHERE yr = in_yr;

 INSERT INTO pay_dim
  SELECT 0, id, yr, sess, obj, sum(amt)
  FROM pay_detail
  WHERE yr = in_yr
  GROUP BY id, yr, sess, obj;
END
```

calc_load_rate:

```
CREATE PROCEDURE `johntest`.`calc_load_rate`(IN load_yr smallint)
BEGIN
 DECLARE in_yr int;

 DELETE FROM load_rate_tbl WHERE  yr = load_yr;

 INSERT INTO load_rate_tbl
 SELECT meeting_dim.id, meeting_dim.yr, meeting_dim.sess, meeting_dim.obj,
SUM(fac_load), 0.0, 0.0
 FROM meeting_dim
 WHERE meeting_dim.yr = load_yr
 GROUP BY meeting_dim.id, yr, sess, meeting_dim.obj;

 UPDATE load_rate_tbl
 SET load_rate_tbl.amt_sum = (select sum(tot_amt)
  from pay_dim
  where pay_dim.yr = load_yr
  and pay_dim.id = load_rate_tbl.id
  and pay_dim.yr = load_rate_tbl.yr
  and pay_dim.sess = load_rate_tbl.sess
  and pay_dim.obj = load_rate_tbl.obj
  group by id, yr, sess, obj)
 WHERE load_rate_tbl.yr = load_yr;

 UPDATE load_rate_tbl
 SET load_rate = amt_sum/load_sum
 WHERE amt_sum > 0
 AND load_sum > 0;
END
```

load_pay_fact:

```
CREATE PROCEDURE `johntest`.`load_pay_fact`(IN in_yr int)
BEGIN
 DELETE FROM pay_fact;

 INSERT INTO pay_fact
 SELECT
  meeting_dim.meetingkey,
  course_dim.crskey,
  pay_dim.paykey,
  meeting_dim.fac_load,
  load_rate_tbl.load_rate,
  meeting_dim.crspay
FROM
  course_dim,
  meeting_dim,
  pay_dim,
  load_rate_tbl
WHERE
  course_dim.yr = in_yr
  AND course_dim.crskey = meeting_dim.crskey
  AND meeting_dim.id = load_rate_tbl.id
  AND meeting_dim.yr = load_rate_tbl.yr
  AND meeting_dim.sess = load_rate_tbl.sess
  AND meeting_dim.obj = load_rate_tbl.obj
  AND meeting_dim.id = pay_dim.id
  AND meeting_dim.yr = pay_dim.yr
  AND meeting_dim.sess = pay_dim.sess
  AND meeting_dim.obj = pay_dim.obj;

UPDATE pay_fact
```

```
  SET tot_amt = tot_load * load_rate
  WHERE tot_load >= 0
  AND load_rate >= 0;
END
```

APPENDIX F: EXAMPLE C++ CODE USED TO CALCULATE INSTRUCTOR LOAD VALUES

Load calculation excerpt

```
//Calculate load for "Advanced and Professional" and "PostSecondary
//Vocational" Co-Op & DIS courses.
qryUpdt->Close();
qryUpdt->SQL->Clear();
qryUpdt->SQL->Add("UPDATE db_facload_rec");
qryUpdt->SQL->Add("SET tothrs2 = ((reg_num * hrs) / 15),");
qryUpdt->SQL->Add("tot_hrs = ((reg_num * hrs) / 15)");
qryUpdt->SQL->Add("WHERE (im = 'CO' OR im = 'D')");
qryUpdt->SQL->Add("AND crsctgry IN ('PSV', 'AP')");
qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
qryUpdt->ExecSQL();

//Calculate load for "Apprenticeship" and "PostSecondary
//Adult Vocational" Co-Op & DIS courses.
qryUpdt->Close();
qryUpdt->SQL->Clear();
qryUpdt->SQL->Add("UPDATE db_facload_rec");
qryUpdt->SQL->Add("SET tothrs2 = ((reg_num * hrs) / 24),");
qryUpdt->SQL->Add("tot_hrs = ((reg_num * hrs) / 24)");
qryUpdt->SQL->Add("WHERE (im = 'CO' OR im = 'D')");
qryUpdt->SQL->Add("AND crsctgry IN ('PSAV', 'APPR')");
qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
qryUpdt->ExecSQL();

//Calculate load for On the Job Training.
qryUpdt->Close();
qryUpdt->SQL->Clear();
qryUpdt->SQL->Add("UPDATE db_facload_rec");
qryUpdt->SQL->Add("SET tothrs2 = (reg_num / 5),");
qryUpdt->SQL->Add("tot_hrs = (reg_num / 5)");
qryUpdt->SQL->Add("WHERE im = 'JT'");
qryUpdt->SQL->Add("AND crsdept NOT IN ('APP','DAS')");
```

```
    qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
    qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
    qryUpdt->ExecSQL();

    //Calculate load for Online.
    qryUpdt->Close();
    qryUpdt->SQL->Clear();
    qryUpdt->SQL->Add("UPDATE db_facload_rec");
    qryUpdt->SQL->Add("SET tothrs2 = ((enr_num * hrs) / 15),");
    qryUpdt->SQL->Add("tot_hrs = ((enr_num * hrs) / 15)");
    qryUpdt->SQL->Add("WHERE im = 'ON'");
    qryUpdt->SQL->Add("AND crsdept = 'APP'");
    qryUpdt->SQL->Add("AND crsctgry IN ('PSV', 'AP')");
    qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
    qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
    qryUpdt->ExecSQL();


    //Calculate the "adjusted" load.

    //Use fields and tables created for KCM.  Specifically,
    //the tot_hrs field on db_instr2_rec.
    if(use_new_catmaint_tables)
    {
      qryUpdt->Close();
      qryUpdt->SQL->Clear();
      qryUpdt->SQL->Add("UPDATE db_facload_rec");
      qryUpdt->SQL->Add("SET adjLoad = ((tothrs2/CAST(flt as int))*CAST(facflt as int))");
      qryUpdt->SQL->Add("WHERE tot_hrs > 0");
      qryUpdt->SQL->Add("AND (flt <> 'PR' and facflt <> 'PR')");
      qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
      qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
      qryUpdt->ExecSQL();
    }
    else  //Use the "traditional" CARS fields (mtg_rec.tot_hrs) for calculating faculty load.
    {
     qryUpdt->Close();
     qryUpdt->SQL->Clear();
     qryUpdt->SQL->Add("UPDATE db_facload_rec");
     qryUpdt->SQL->Add("SET adjLoad = ((tot_hrs/CAST(flt as int))*CAST(facflt as int))");
     qryUpdt->SQL->Add("WHERE tot_hrs > 0");
     qryUpdt->SQL->Add("AND (flt <> 'PR' and facflt <> 'PR')");
     qryUpdt->SQL->Add("AND app_name = '"+app_name+"'");
     qryUpdt->SQL->Add("AND user_login = '"+winuser+"'");
     qryUpdt->ExecSQL();
    }
```