FJET Database Project: Extract, Transform, and Load

Kevin Samms
Kennedy Space Center
Major: Computer Science

Data Mining and Knowledge Discover
Fall Session
Date: 11 20 2015

FJET Database Project: Extract, Transform, and Load

Kevin O. Samms[1]
University of Central Florida, Orlando, FL, 32816

The Data Mining & Knowledge Management team at Kennedy Space Center is providing data management services to the Frangible Joint Empirical Test (FJET) project at Langley Research Center (LARC). FJET is a project under the NASA Engineering and Safety Center (NESC). The purpose of FJET is to conduct an assessment of mild detonating fuse (MDF) frangible joints (FJs) for human spacecraft separation tasks in support of the NASA Commercial Crew Program. The Data Mining & Knowledge Management team has been tasked with creating and managing a database for the efficient storage and retrieval of FJET test data. This paper details the Extract, Transform, and Load (ETL) process as it is related to gathering FJET test data into a Microsoft SQL relational database, and making that data available to the data users. Lessons learned, procedures implemented, and programming code samples are discussed to help detail the learning experienced as the Data Mining & Knowledge Management team adapted to changing requirements and new technology while maintaining flexibility of design in various aspects of the data management project.

Nomenclature

Excel VBA	=	the programming language of Excel.
Flat File	=	a text file in which rows and columns of data are represented by delimited and/or enclosed text.
Front-end	=	an interface that allows the user to connect to the database without having direct manipulative control over database objects.
GUI	=	an interface allowing users to interact with a computer through pictures and clickable buttons.
MS SQL	=	a relational database system developed by Microsoft.
MySQL	=	an open source relational database.
Schema	=	the logical grouping of objects within a database such as tables, views, and stored procedures.
SSMS	=	a software application used for configuring, managing, and administering all components within Microsoft SQL Server.
SSRS	=	server-based report generation software accessed by data users through a web browser.
SQL	=	the query language used to communicate with a database.

I. Introduction

FJET (Frangible Joint Empirical Test) is a project under the NASA Engineering and Safety Center (NESC). Its purpose is to conduct an assessment of mild detonating fuse (MDF) frangible joints (FJs) for human spacecraft separation tasks in support of the NASA Commercial Crew Program. The FJET database project is intended to bring centralized data management, ease of access to the data, and the ability to view all or portions of the data from various perspectives as needed by FJET data users. The Data Mining & Knowledge Management team at Kennedy Space Center (KSC) has been responsible for providing data mining and data management services to various NASA projects, and has been tasked with creating and managing this database. FJET had already begun testing, generating data, and evolving in complexity before the database project had started. The challenges for the Data Mining & Knowledge Management team were to understand FJET data relationships, catch up to the data being generated, and help the FJET team to understand the implications of database storage, all while the requirements for storage and access to the data were mostly undefined. Both the FJET testing and the FJET database project continued to evolve throughout the months over which the project goals were accomplished.

Both data mining and data management begin with the Extract, Transform, and Load (ETL) process in order to retrieve data from its source, transform it in ways that make it amenable to database use, and then load that data into a database. Extraction usually involves downloading data as a Microsoft Excel spreadsheet or as a simple flat file

[1] IT-G Intern, IT-G, KSC, University of Central Florida.

from the data source. The transformation process requires the manipulation of anywhere from a few rows to a few million rows of data. This is where Excel Visual Basic for Applications (Excel VBA), the programming language of Excel, proves advantageous in automating the transformation. Manual transformation using built-in Excel functions alone could prove extremely time-consuming and open to the introduction of anomalies due to human error. After transformation, data are loaded into the database through an interface. In the case of MySQL, an open source database used in the early stages of the database project, the interface is called Workbench; for Microsoft SQL Server (MS SQL), the interface is called SQL Server Management Studio (SSMS). At the end of the ETL process, data are presented to the user either through some form of preconfigured report or through a query tool that allows the user to communicate directly with the database through Structured Query Language (SQL), the native language of the database. Whether report or query tool, both serve as a front-end, or intermediary to the database. Sensible practice is to not give users direct access to the database itself since such access could create a loss of control over the data for those responsible for managing and administering the database. The FJET team originally stated that some of their members would like to query the database directly, however, this was not the option preferred by most of their team nor the option preferred by the Data Mining & Knowledge Management team for interacting with the database.

The FJET data management project began with MySQL community edition; the free and open source relational database management system. This choice allowed for simplicity and flexibility in setting up an initial database by allowing time to discover requirements and potential problems before committing to more expensive and proprietary options. We learned more about the data and the requirements of the FJET team as we performed the ETL process and managed the data over a few months. As the challenges of storing, maintaining, and granting access to the data became more clear, we decided to switch over to Microsoft SQL Server (MS SQL) for the advantage of its integrated SQL Server Reporting Services (SSRS) tool, through which we could provide users with customized and secure access to sensitive test data, and eliminate any need for writing SQL queries directly to the database. At one point while MySQL was being used as the database, it was considered that some users would write SQL queries directly to the database or perhaps we would use a front-end query tool that would make it easier to write SQL queries for those users not familiar with SQL syntax. A number of front-end query tools were considered, including FlySpeed SQL Query, dbForge Query Builder, and Crystal Reports which seemed the most promising of the three. However, security concerns and a preference for an integrated package made MS SQL and its integrated SSRS the appropriate solution. The data was migrated from MySQL to MS SQL using the Microsoft SQL Server Migration Assistant (SSMA) for MySQL, version 5.3.0. The SSMA tool connects to both databases simultaneously, allows the generation of a report to test the conversion before performing it, then the tool converts the MySQL database schema, connects MS SQL to that converted schema, and finally migrates the data from the MySQL database to the MS SQL database. After transferring the data from its initial MySQL environment to its MS SQL environment, and having set up both a development and a production database there, the remaining work was to create procedures for the tracking and managing of data, finding ways to provide fast and efficient access to the data, and generating reports to satisfy the perspectives on the data requested by the data users. FJET data grew in size to many tables and millions of records over the months that this database project evolved.

The next sections of this paper discuss the ETL process in detail (with code examples), define a few types of objects created to help manage the database, discuss the lessons learned about gathering requirements and managing large amounts of data, and concludes with a discussion about the impact of the FJET database project.

II. The Extraction Process

Extraction means taking data, possibly existing in different forms, from its source or sources and combining it into a common form that can then move through the other stages of the ETL process. FJET data are clearly separated by the different categories of data produced by the different instruments in a frangible joint test. These categories of data, or data elements as they will be referred to from this point forward, are data produced by a specific instrument and classified as a data element based on that instrument. Provided that the output of the data producing instrument does not change in terms of the type of output and measuring units, then disparity within a data element is a non-issue; the same extraction procedure can be followed each time: check that the units have not changed between extractions, and the data are already in a form common to other data of that data element. Thus any disparities in the data are confined to a data element. All that is required to put data into a common form is agreement of units within the data element. However, when data comes not from an instrument but from something more subjective and variable, such as a test design, then disparity can be high, but still confined to that data element. Following this idea, changes within a data element lead to the creation of new sub-elements based on the original data element. Basically, where one extraction is different from another but still within the same data element, columns that are to

match up later in a union should be of the same data type (character, integer, etc.) but different sub-elements will simply have different column names and a different number of columns. These variations within a data element are loaded into database tables separate from other sub-elements; different sub-elements go into different tables.

The FJET team provides most test data in the form of an Excel spreadsheet and some data in the form of some document type from which the data must be manually transcribed into an Excel spreadsheet. The latter form of extraction can involve reading handwritten data which at times must be interpreted within the context of the surrounding data due to imprecise handwriting or problems with image clarity. This necessitates that any interpreted data be marked by the data manager, and then sent back to the data producer for verification. Once verified, the data manager can then update the data or mark is as approved. This round-trip time can add days or weeks for a particular test product to move through the ETL process, and become available in the database. Whereas other data placed in the definite form of a spreadsheet can easily move to the transformation stage. This is not guaranteed, however, as data entered into a spreadsheet cell, while well defined, can still be entered in a way that does not match the data type or context of the cell into which it was entered. An example is entering text where a number should be or entering a number into a cell as 40,00 when it should be 4,000. These errors require a round-trip to the data producer in order to find out what was the intended value. Provided that all data are entered correctly and entered into a definite form like a spreadsheet, then at least from the perspective of the data manager, the extraction process is easiest part of the ETL process. The data are downloaded from the staging area, or "outbox" of the data producer to the working folder of the data manager. However, the ease of this part of ETL process does not consider the difficulty experienced by the FJET testing team in producing and preparing the data on their end, before it is placed in an agreed upon location where the data managers can extract the data, beginning the ETL process.

III. The Transformation Process

The transformation process is about making changes to the data that make it suitable for storage within a database. It seems to be a simple idea, just put the data into a database and it becomes accessible from one place. No more files and folders to search through, and we gain the ability to see multiple data elements in various ways. However, the idea is not only to store data but to make it easily retrievable as well. Data must be unique and easily accessible through SQL, the language used to communicate with the database. Uniqueness is not a common sense requirement in that those unfamiliar with operating a database would know that two things in the same set of data cannot have the same name. It may be common in a given occupation to label a diagram with fields that all have the same name or to reuse a device name in multiple places. Data in that form cannot be placed into a database. The database used in this project follows the relational database model of data storage, meant to model real world entities. In this model, each table represents an entity, each table row represents the instance of an entity, and every table column is a different attribute of that entity. A table of persons thus represents people. A name entered into that table is then an instance of a person, and that person can have an address column and a phone number column, both attributes of that person. Uniqueness requires that every row, or instance of an entity be unique. Just as there cannot be two copies of the same exact person, there cannot be two rows, or two of the same entities in the same table of a database. The simple way to mitigate problems where the same dataset, or the same table may receive more than one of a specific entity is to create a unique identifier for each copy of that entity. This allows the two entities to be separated by that identifier. Just as two people with the same name would have different social security numbers. In fact that is a good example of how two people with the same name would be separated in a relational database.

In this project there were multiple tests, each with their own set of data elements. The tests and sub-tests stood out as the most appropriate attribute upon which to base uniqueness within the database. Each entity would be uniquely identified by the test series and test ID from which it came. Some tables in which the data producing device is also named within the data itself additionally have that device name to add uniqueness to the measurement data that is output from a test. Adding unique identifiers like a test series and test ID to identify test data, is what we call adding "intelligence" to the data. It helps to make the data unique, easy to find, and facilitates performing operations on the data through SQL. However, in some cases adding intelligence derived from the dataset itself is still not enough, since for a given test series, test ID, and device name, that device may output multiple rows of the same output data. Furthermore, there may be no way for either the data producer or the data manager to tell whether this will or will not happen, especially considering that there may be many thousands or millions of rows of generated data. Therefore, even with as much uniqueness as we can get from the data itself, we also had to look external to the data and fall back to the ultimate guarantee of uniqueness; the unique identification number generated by the database. In this case the table is given an "id" column that serves to simply give a number to a row, then add some increment to that number for the next row, and similarly give new rows their own unique number. A counting column if you will. Basically every row (entity) gets its own ID (ID card), guaranteeing uniqueness. This is another

layer of intelligence added to the data. The transformation process for FJET thus involved guaranteeing uniqueness by adding intelligence to the data. This prepared the data for upload to the database.

The transformation process is carried out within Microsoft Excel through the use of Excel VBA coding; the programming language of Microsoft Excel. For those extractions where the data is provided in the same format, with the same attributes (columns), and differing only in the amount of data, it is appropriate to write a program in Excel VBA code specific to transforming that particular data element. After extraction, the code is run on this uniform presentation of the data, and the output is data ready for upload to, and use within, the database. Examples of such code with explanations are provided in figures 1 and 2 below.

Figure 1 shows a relatively simple code example. The function is called "PrepPDVdata()." It formats a sheet of data to the correct data types, ensuring that text columns are set to the text data type and numbers columns are set the appropriate number type; decimal with a precision of 8 in this case. The "letter:letter" syntax is the range of columns upon which the operation (the code on the next line below it), is performed. Towards the end of the code the entire sheet is expanded so that the header names and data within each column can be seen clearly.

```
Sub PrepPDVdata()
    '
    ' This code puts the PDV data sheet into the proper format for uploading, and expands all cells to fit
    '
    Columns ("A:A").Select
    Selection.NumberFormat = "0"
    Columns ("B:D").Select
    Selection.NumberFormat = "@"
    Columns ("E:F").Select
    Selection.NumberFormat = "0.00000000"
    Range ("G1").Select
    Cells.Select
    Cells.EntireColumn.AutoFit
    End Sub
```

Figure 1. Example Simple VBA Code.

A slightly more complicated example, shown in figure 2 below, demonstrates more robust code in that it does not assume a static location for the "test_id" column. It searches for it on any given spreadsheet, and performs the operation of repairing any test ID that does not meet the standard ID format. The program ends with a message box to inform of how many rows were affected and where the appropriate stopping point might have been. The code note below the function name explains the purpose of the program.

```
    Sub FixTestID()
    '
    ' This program adds the zero "0" or double zero "00" prefix to test id's that have lost the prefix due to
    ' Excel bug
    ' that removes leading zeroes when opening a csv file where the test id's had leading zeros.
    '
    Dim output As String
    Dim s2 As String
    Dim info As String

    s2 = "test_id"
    info = ""

    'fix test_id format'
    i = 1
    j = 1
    Do While (Cells(1, j).Text <> s2 And j < 20)  'find the right column - j'
       j = j + 1
    Loop
```

```
    'end program if s2 column not found'
    If (j = 20) Then
        output = MsgBox("Error: the " & s2 & " column was not found", vbInformation, "Error")
        Exit Sub
    End If

    'set column types to string'
    Columns(j).NumberFormat = "@"

    i = 2     'start at row 2, i.e., ignore header'
    Do While Not (IsEmpty(Cells(i, j)) And Cells(1, j).Text <> s2) 'change all numbers to zero prefixed
    numbers - move down i, except row 1'
       If Len(Cells(i, j).Text) = 2 Then
          Cells(i, j) = "0" & Cells(i, j).Text

    ElseIf Len(Cells(i, j).Text) = 1 Then
          Cells(i, j) = "00" & Cells(i, j).Text
       Else 'If ((Len(Cells(i, j).Text) > 2) Or (Len(Cells(i, j).Text) <= 0)) Then'
             output = MsgBox("Error: A test id is blank or has more than 2 characters " & Cells(i, j).Text
             & " i: " & i & " j: " & j, vbInformation, "Error")
          Exit Sub
       End If

       'end loop if cell on next row is empty'
       If IsEmpty(Cells(i + 1, j)) Then
          info = info & " " & i
       End If

       If Len(info) > 20 Then
          info = info & vbNewLine
          Exit Do
       End If

       i = i + 1

    Loop

    'Expand worksheet columns to width necessary to show all data and titles'
    Cells.Select
       Cells.EntireColumn.AutoFit
       Cells.EntireColumn.AutoFit

    output = MsgBox("Empty cell(s) encountered at " & vbNewLine & info & vbNewLine & "Be sure all
    rows have been processed", vbInformation, "Information")

End Sub
```

Figure 2. Example VBA Code.

Programs such as those shown above are a necessary part of the transformation process. For most data elements in the FJET database project, there are too many rows of data to be handled manually. Using the built-in Excel functions could be done, but a program to perform a given transformation in seconds is much preferred over minutes and portions of an hour to do the same with a mouse and keyboard. In the interests of brevity, more complicated examples of the Excel VBA code used for transformations are not included in this paper.

IV. The Load Process

The load process is where the data is finally placed into the database. In less automated cases this can involve writing a script which specifies the source file, file attributes, destination table, and what parts of the source file go into what columns. This was the case for the MySQL database that this project started with. An example of such a script is provided in figure 3. The "@" symbol delineates variables into which data are read for each attribute of each row of data. Those values are then entered into the columns and rows of the destination table via the "SET" command. Those variables for which there is no corresponding destination in the SET command are variables that must be read by the process, but should not go into the destination table. Those "read-only" destination attributes could be database controlled, such as an automatically generated ID or a time stamp. The benefit of this Load Data Infile command script is that it transfers the source file to the server. Making the file local to the server allows the data to be read into its table at high speed. The alternative is to use the native import/export tool in MySQL which keeps the file on the local computer, transferring data row by row to the server; unreasonably slow for large data.

```
LOAD DATA LOW_PRIORITY LOCAL INFILE 'C:/Filename.csv'
INTO TABLE `schemaName`.`tableName_clone`
CHARACTER SET utf8
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY ""
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(@readOnlyColumn,
@test_series,
@test_id,
@pdv,
@measure_mm,
@speed_ms,
@readOnlyColumn2)
SET
test_series=nullif(@test_series,''),
test_id=nullif(@test_id,''),
pdv=nullif(@pdv,''),
measure_mm=nullif(@measure_mm,''),
speed_ms=nullif(@speed_ms,'');
```

Figure 3. Example MySQL Upload Script.

Migrating the data over to MS SQL database came with the ability to use an upload wizard in which the process is performed in a more automated fashion, using a Graphical User Interface (GUI). The source file is selected using a Microsoft Explorer dialogue, the appropriate client program for the particular database is selected to perform the upload, and then the wizard allows the selection of destination table and loading options. The option to view the data types, as the wizard has interpreted them, and the option to preview what the table will look like after the upload, are two useful options to help ensure that the upload will go smoothly. A safe procedure when loading data into a "live", or production database is to create a clone of the destination table and its data, then perform the upload on the clone. This allows room for error. If the upload experiences problems or data are unintentionally deleted, then it happened to the clone, not to the production table. It can be of great inconvenience to the data users if the table with their data suddenly disappears or its data gets wiped right when it is being used to accomplish important work. After the data has been uploaded, it should be verified through a "Select" query to show specifically that subset of data separately from other data already in the table. Provided that the upload has gone smoothly, the original table can be deleted and the clone table can be immediately renamed to take its place. Alternatively, the same script/procedure, having worked once without problems, can then be performed on the production table. The clone can be kept temporarily as a backup in the case that something still goes wrong. The first method requires one upload and less time whereas the second method requires two uploads. In this project the second method was considered to be the safest option. Once the data are uploaded and verified, the ETL process is complete. Provided that the appropriate intelligence has been added to the data; the stages of extraction, transformation, and loading result in data that is efficiently stored in the

database. It is centralized, easily accessible through SQL, and available to the data users most likely through front-end query software or web-based reporting software such as Microsoft SQL Reporting Services (SSRS).

V. Tables, Views, Stored Procedures, and Triggers

The ETL process has been covered as the main focus of this paper, however it is important to note that the tables into which data are loaded is a matter to be addressed even before the ETL process is performed. Considerations for the efficient storage of data, tracking changes to the database, and maintaining flexibility are important parts of the database design. This database was built while the requirements for how it would be used were still being developed. Furthermore, those requirements continued to change as the project progressed. Considering such an environment, the most important design consideration implemented for this database was its flexible table design. Each data element was given its own table, and each table had the same primary key design. The primary key for every table consists of at least an ID field, test series, and test ID. Having a common test series and test ID field among the individual tables allows the association of their rows where those fields match. So we can keep camera data, drawing data, and device data in separate tables, but join those tables together into a "view", or virtual table making it possible to see all of the camera, drawing, and device data for a particular test series and test ID. A view is a stored SQL statement also called a virtual table; a table existing only temporarily in memory and not using permanent storage space on the database hard drive. Normally a database would include parent and child tables existing in linked relationships where changes in one table immediately necessitate changes in the other. This project could not follow such a design due to constantly changing requirements. Separating the data into individual tables, and then using views to link them together as needed by the data users, was the most appropriate design because it offered a great deal of flexibility in how we could access and present the data.

Having successfully performed the ETL process, and loaded the data into a well-designed system of tables, the remaining considerations were managing the data and making it available to the data users. Adding a time stamp column is one way to assist in the managing of the data by keeping track of when records were updated. This was accomplished by creating a column and setting it to the data type of "datetime2", allowing NULL values, and setting the default value to the getdate() function. This ensured that every time a record is entered into the database it gets the current date and time in that time stamp column. Allowing a NULL value means the column can have no time stamp, as it would when the table is just created. We also created a table which is a combination of a few other tables with one column holding a value calculated according to an equation specified by the data producers. The idea was to bring in all of the necessary values from the other tables into one table and perform the calculation there. This was a way to run the calculation only once and have it available to be referenced multiple times. The alternative was to join the tables together into a view, and run the calculation at viewing time. Since a view is a stored query that is run whenever the view is accessed, this would mean running the same calculation, multiple times for multiple references; not an efficient use of database resources. However, the downside is having to maintain the table of centralized data and calculations. Whenever one table in the group changes, the central table had to be updated. The way we accomplished this was through a Stored Procedure; a program stored within the database that can perform SQL queries and use variables. The program was written to reload the table and recalculate the values. By combining this with a Trigger to call the Stored Procedure, we automated the process of maintaining the central table. Each of the tables in the group has its own Trigger to detect changes and any one of them can execute the stored procedure. Once implemented, there is no longer a need to manually manage the central table. However, such automated procedures can be forgotten, and the way we avoid this is to write the Trigger and Stored Procedure to log their activity into a table specifically set up for such logging activity. This is a way for these programs to "check-in" and make database managers aware of their existence and activity. Using a Trigger along with a Stored Procedure has security considerations that were addressed through research on the concern, and adjusting procedures accordingly.

VI. Lessons Learned

The most important lessons learned from this project are in regards to requirements and flexibility. The requirements regarding what would be stored in the database, how it was to be used, and who was going to be using it were all in flux as this project evolved. We had many teleconferences with the FJET testing team and there were often gaps in understanding. They knew their data well, and we knew databases well, but bridging the understanding toward us understanding their data, and them understanding what had to be done to it in order to get it into the database, was a major hurdle that took months to eventually overcome. Our discussions with them also inspired conversations within their group that helped to create additional flux in the requirements. An important idea that we can take from this project is that it is necessary to get the data producers to communicate early within their own

groups about what data needs to be accessed in report format, and what that report should look like. The persons using the data on the reporting end should also be included in those discussions throughout the process. Knowing more about what end result is desired would help greatly in determining how to set up the database to achieve the desired result. Another important lesson is that the measurement units and the names of fields need to be as well-defined as possible, as early as possible in the process, and procedures should be established early to ensure that those agreements are met. Variation in the agreed upon terms is inevitable, but procedures agreed upon and then followed by either the data managers or the data producers should catch the resulting anomalies before they become time consuming tasks to go back and fix or reformat data. We learned that the idea of uniqueness and its importance to the data are concepts that need to be fully explained, and stressed to the data producers. This is not a natural concept to those unfamiliar with database management, and can become a repeating hurdle to overcome whenever the data producer makes a change. Of all the concepts and concerns involved in setting up a database, the uniqueness requirement and the need to standardize as early as possible are two areas where much effort needs to be expended before moving forward.

VII. Conclusion

The ETL process as used in this project was relatively uncomplicated compared to those previously performed by the Data Mining & Knowledge Management team at KSC. The major efforts were in managing the data, and writing Excel VBA code for the transformation process. Usually the extraction process involves pulling in data from many dissimilar sources varying in the names of fields, the number of fields used, and in the data types employed to represent the data. The transformation process then has to normalize these disparities; i.e., transform the different forms of data into a common form by which they can be related or combined. The key features that reduced complexity in the ETL process as applied to this project were: data separated into different data elements, each element delimited by the instrument producing those data (thus minimizing the need to normalize); a database design maintaining each data element within its own table but sharing a common primary key structure; and the fact that all data came from one data producer, as opposed to coming from many different producers with possibly different goals and different perspectives.

The FJET database project was a real-world application of database design and management. This project was in flux on both sides. The data management environment evolved from MySQL free edition with a search for front-end query solutions, to a paid MS SQL database solution with integrated reporting services. The FJET project evolved from its early test designs, few test instruments, and thousands of rows of data, to more detailed and complicated tests, greater number of test instruments, and millions of rows of data produced per test.

Prior to this project, FJET test data was stored in files and folders on a server for access by data users. The problem with this setup is that accessing data meant a time consuming search of the various folders to find specific data, and then the entire file had to be accessed, large or small, even if only a portion of the file is what was desired. This simpler form of access is much the same as what one would perform when searching for files on a home computer. In this case however, real-world use of the data would result in the repeated transferring of very large files across networks, the effort to maintain multiple copies of the file, and then having to work with the entire file, not just the portion of the file that a typical data user is interested in. Then there is the fact that without their own reporting tools, the data user will be unable to readily view different portions of different data, at the same time or from a particular perspective. The FJET database project has eliminated those concerns by bringing to the FJET team the abilities of centralized data storage and management, organization of data by data element, and the ability to view all or portions of any data loaded into the database from different perspectives. The data users only need communicate to the data managers what data or combination of data they would like to see, and what filtering options they would like in order to get them to the portion of data that they would like to investigate. The data users can communicate the desired perspective verbally, through a drawing or by spreadsheet. To assist in the accomplishment of FJET database project goals, I had to learn Excel VBA, MySQL and MS SQL syntax, research ways to improve various data management operations, and take an active role in directly managing the database, while keeping track of meetings, procedures, and changes to the data.