Jeffrey Donahue

donahuej@bu.edu

Final Report

MET CS 779

MET CS 682 Assignment Template

Table of Contents

[1. Introduction 2](#_Toc16684243)

[2. The Operational Databases 2](#_Toc16684244)

[3. The Project Data Mart 3](#_Toc16684245)

[1) The Schema 3](#_Toc16684246)

[2) Creating the Data Mart And Indexes 9](#_Toc16684247)

[3) The ETL Process 14](#_Toc16684248)

[4) The Results 14](#_Toc16684249)

[4. The Customer Data Mart 15](#_Toc16684250)

[1) Design, Implementation and ETL 15](#_Toc16684251)

[2) Security Measure 15](#_Toc16684252)

[3) Results 15](#_Toc16684253)

[5. Conclusion 16](#_Toc16684254)

[References 18](#_Toc16684255)

# Introduction

To help myself engage with the topics learned throughout this course, I decided to apply what I have learn to a real problem at my work. As a small company, we do not have a dedicated Database Administrator as a full time role, thus the responsibilities have in the past fallen onto workers who do not have great administration experience. This resulted in a poorly designed schema and limited usability. I decided, with approval from my manager, to take the task of rebuilding from scratch one of our highly inefficient data marts and to create a new data mart. At times throughout this report, I will be redacting client or employee personal information in an attempt to protect their identity.

This report will introduce my company’s three operational databases, along with their limitations which will require a creative solution. I will walk through the essential business questions and aspects needed for the redesign of the Project Data Mart, as well as the design of the brand new Customer Data Mart. The Customer Data Mart will require us to take additional security measures, such as encryption and redaction, to help protect our customers. Lastly, I will provide what I have done to streamline the ETL process, as well as show the performance improvement with the new data marts.

# The Operational Databases

My company has three different sources of data that we use in our day to day operations. Unfortunately, these three sources as not well integrated and it takes a lot of effort for the report author to connect these databases for reporting needs. Each database has its own unique web interface and is stored independently. The first database is our accounting system. This is where we keep our billing information, vendor information, the official records of processed timesheets and paychecks to our employees, and other typical accounting information. The next one is our customer relationship manager (CRM) database, that tracks customer information, potential corporate leads, and official bid and proposal information. These two systems are easy to deal with and we can pull all of our historic data from either whenever we want. However, the third system is rather limited.

The third operational database is our timesheet system, where employees put their daily hours in to track which projects they are working on, for future billing purposes. Our contract with the provider does not allow us to have a direct link to the behind the scenes database, rather we receive a nightly extract in a CSV file, that shows all the hours, by employee, by project and date for the past three years. As soon as a data item is older than 3 years, it falls off of this CSV file and we lose it forever unless we specifically request historic data from the provider. In addition to the two data marts that I am implementing, during the ETL portion of this report, I will create a way to capture history before we lose that record.

But even with just three years of data, each nightly CSV file is around 300,000 records since it is showing such granular data. Currently there are no indexes, which makes querying from this table to be incredibly laborious, but this will change in the new design.

# The Project Data Mart

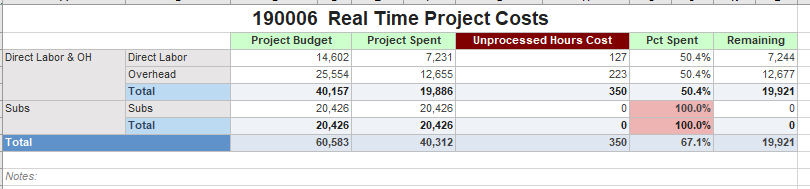
The current version of the Project Data Mart was created to support our project details workbook asset. Project managers can go to our reporting tool, Cognos, find their project, and download a massive 9-tab excel sheet that is rebuilt with live data every time it is downloaded. This report touches data in all three operational databases, making it a very laborious report to run. The current version takes about 6 minutes to run for some bigger projects, and even longer at high traffic times. In addition to this, we have been trying to create a job that emails that report as an attachment to the project manager at critical checkpoints of the project. As a consulting company, we have a lot of projects going at the same time, and we have observed that it is common to have 15-20 projects needing an email per day, which would cause our Cognos system to do 3 or more hours of work per day if the report was to stay in its current state. The ultimate goal here is to get the report to run in under 30 seconds for any given project, which can only be achieved with a new data mart.

The new Project Data Mart will have a constellation schema. We will have to be able to capture employee hours and costs, expense cost, subcontractor costs, revenue and billing information. The next section is going to layout what my proposed schema is. My process is to analyze each business question individually as if I was building a data mart meant for only that question, then combine it all and reduce any unnecessary duplication, such as repeating dimension tables.

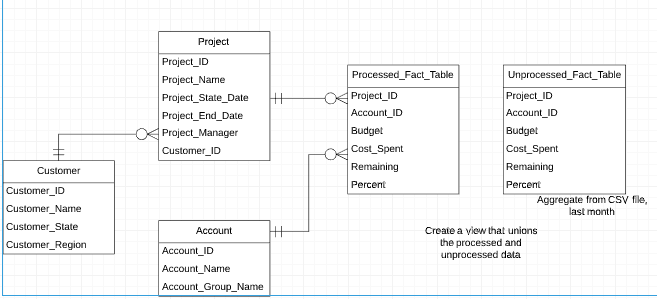
## The Schema

The 9-tab excel worksheet is the best way to gather what our essential business questions are, since the worksheet provides a master project detail view. Any question a project manager could possibly have, should be answered with this worksheet, thus if we can build a data mart that satisfies everything in that excel sheet, then the data mart will be a success. Each tab will serve as a new business question that we will need to address.

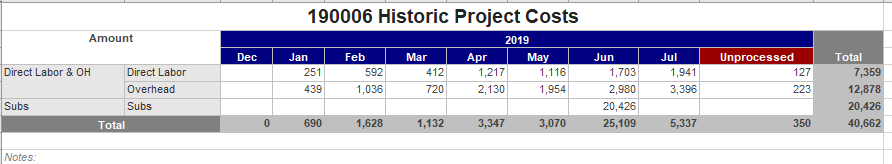
1. Tab 1: The Summary



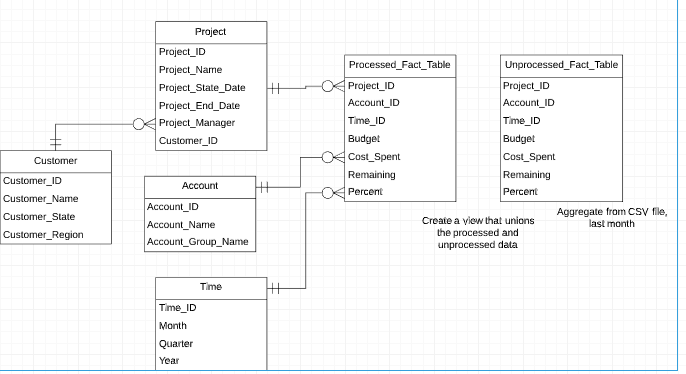
The first tab of the excel sheet give a high level overview of the current state of the project. We are showing the aggregate budget and spend of direct labor and expense (overhead) costs, as well as any subcontractor costs. The last two columns show how much is remaining to the budget, in dollars and percent. This entire tab can be pulled out of our accounting system, except for that red unprocessed hours cost column, which makes this tab tricky. Our accounting system only tracks fully processed information, which is done on a monthly basis. To make this a live report, we need to union our timesheet database onto the processed accounting database information. In my design, I will create to fact tables, one for processed time and the other for unprocessed time. This is the best option here, just in case we change a provider in the future, we can just swap out the source for that one in the ETL process and not have it effect the whole data mart. The only dimensions here are the project dimension and the account information, which determines if the cost/ budget is labor, expense, or subcontractor. The unprocessed table is just going to be a summary table of the CSV file that we get from our timesheet provider that we can union with the processed hours in the future. If this was the only question needed to be answered, the design would look like:



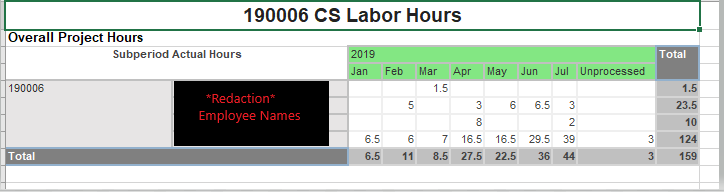
2. Tab 2: Historical Costs

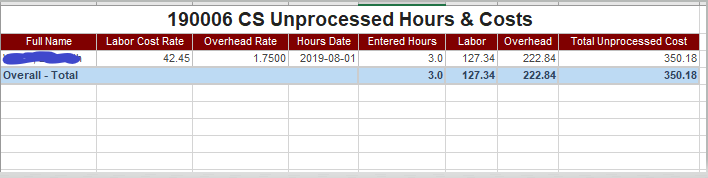


This is similar to the first tab, but now there is a time dimension. We have dropped the budgets from the first tab, but now we track the costs per month, again with a union to keep track of the unprocessed hours. While it is possible to add a time dimension to the first tab’s design and have those fact tables serve as the source for both tabs, it is smarter to separate the fact tables. It is pretty common at my company to just observe the aggregated total project information found on the first tab and adding the time dimension will greatly impact the size and performance of those fact tables, thus it is best to store them separately. The design for this tab is:



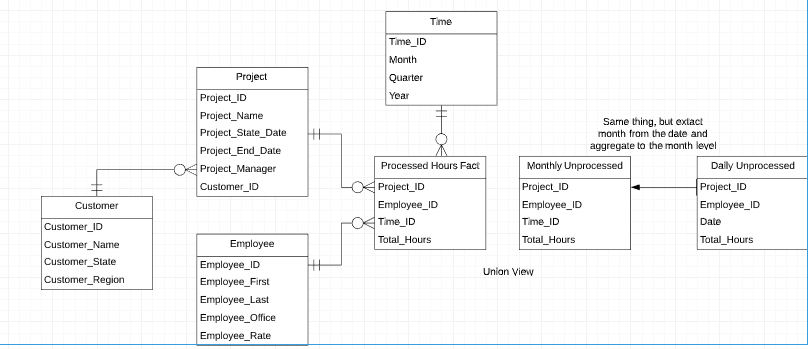
3. Tabs 3 and 4: Employee Hours



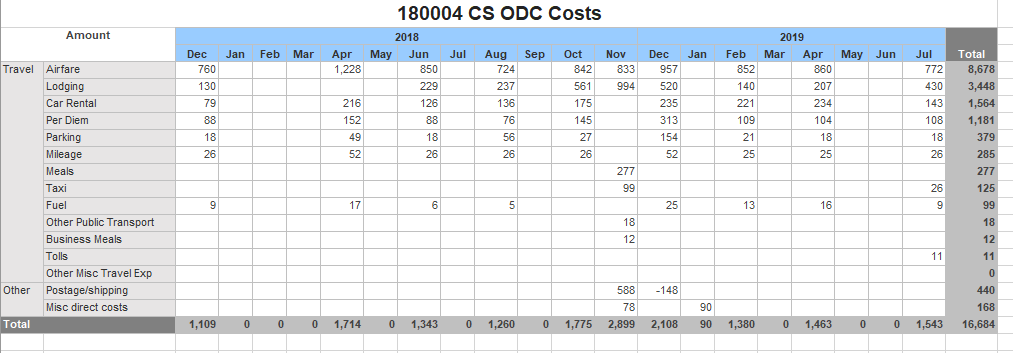


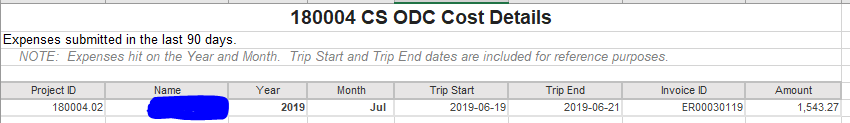
The next two tabs deal with similar information: hours worked by employee. The first of the two shows the hours broken out by employee, by month, along with the unprocessed hours of the current month. The next tab breaks out the unprocessed hours down to the day, and the expect cost to the project, which is just an estimate until it becomes processed.

Again, we are dealing with two fact tables. The first being the processed hours, with a monthly time dimension, as well as the project and employee dimension. The next table is just the CSV file that we get nightly from our provider, filtered to the current month. There is no need for a denormalized schema in the unprocessed hours table since we are just going to load in every night the updated information. The design looks like:

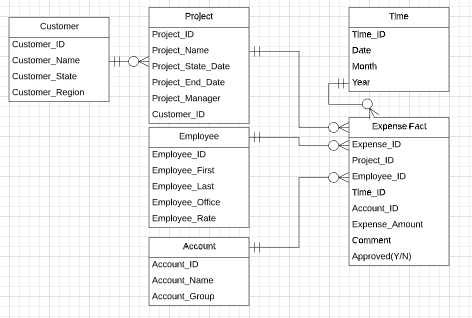


4. Tabs 5 and 6: Summary and Detailed Expenses

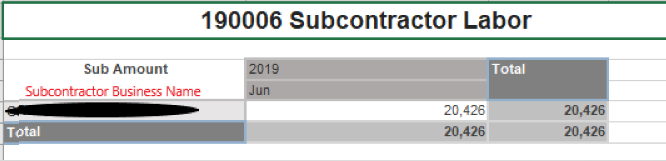




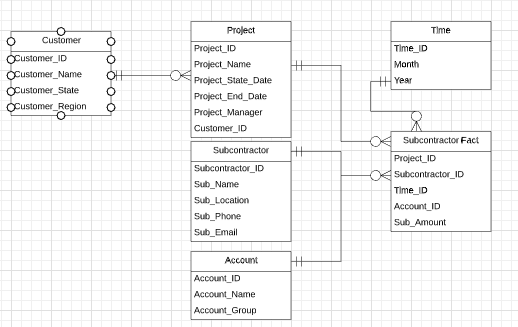
The next two tabs concern the expense costs for each project. The first of the two shows the summary level, then the next one shows the detailed view. This time there is no need for two fact tables, since this is all pulled out of one system, the accounting system, and there aren’t too many expenses. We have about 10-20 expense reports filed per project, thus the detailed version of the fact table doesn’t get very large and we can aggregate the summary table easily. Thus, it is best to not duplicate the storage in this case. The dimensions are the expense category, the employee, the project and the time period. The design is:



5. Tab 7: Subcontractors

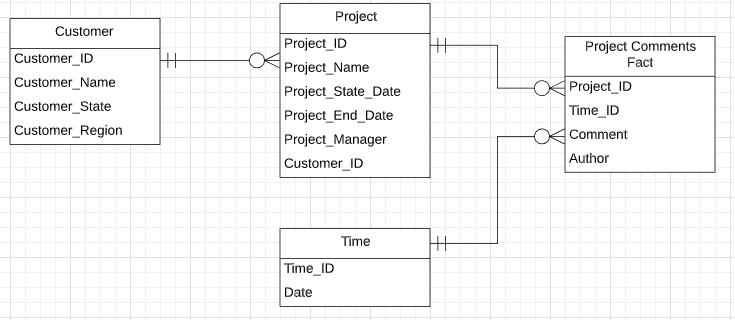


Some projects, we contract out work to subcontractors. This tab is to help project managers keep track of the costs they are incurring with the subcontractors. The key dimensions here are the project, the subcontractor and the time dimension. Again, the design follows:



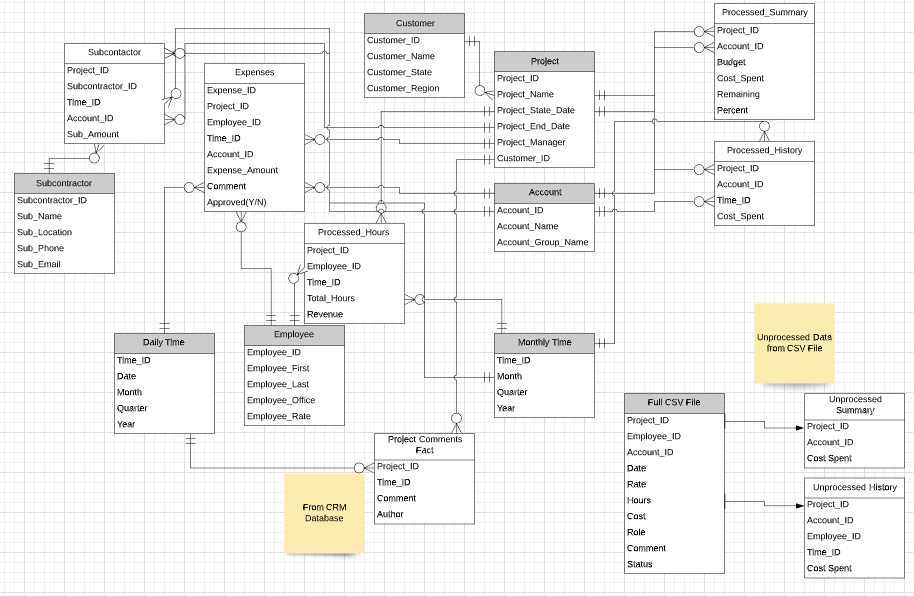
6. Tab 8: Customer Notes

This tab I cannot show do to proprietary information. However, the purpose of this tab is to show key project and customer information to help ensure the project runs smoothly. Included is customer contract information and any comments inserted into our customer relationship management database regarding the project or customer. The dimensions here are just the project and the comment section of the CRM database. The design is as follows:



The Constillation Schema

I’ve declared a lot of dimensions and fact tables above, but some of them are repetitive. We do have two reoccuring time dimensions, one that goes to the monthly level and the other is a daily detailed view. These two should stay separate. The project, account information, customer information show up on different tabs as the same information, thus those dimensions can be combined. The final schema for the Project Data Mart is:



The shaded headers are the dimension tables and the unshaded white headers are the fact tables. The unprocessed hours and cost tables are just aggregated values from the CSV file with auto created Time\_ID from extracting the month and year from the data record. I believe that this schema addresses each and every tab the most efficient way possible, and this will be the best design for my company moving forward.

## Creating the Data Mart And Indexes

Attached in this section is screenshots of the DDL required to create all the tables defined above, as well as indexes that I have decided to implement to help the performance of the data mart and views to easily query each group of data.

The DDL:

Create Table Customer (

Customer\_ID varchar(32) PRIMARY KEY,

Customer\_Name varchar(255),

Customer\_State varchar(32),

Customer\_Region varchar(32)

)

create table Project (

Project\_ID varchar(12) Primary Key,

Project\_Name varchar(255),

Project\_Start\_Date Date,

Project\_End\_Date Date,

Project\_Manager varchar(64),

Customer\_ID varchar(32) references Customer(Customer\_ID)

)

Create Table Employee (

Employee\_ID varchar(12) Primary Key,

Employee\_First varchar(32),

Employee\_Last varchar(32),

Employee\_Office varchar(32),

Employee\_Rate decimal(10,2)

)

Create Table Subcontractor (

Subcontractor\_ID varchar(32) Primary Key,

Sub\_Name varchar(255),

Sub\_Location varchar(255),

Sub\_Phone varchar(16),

Sub\_Email varchar(32)

)

Create Table DailyTime (

DayTime\_ID int Primary Key, --This is just the day count from a given point

DateOfRecord Date,

MonthOfRecord int,

QuarterOfRecord int,

YearOfRecord int

)

Create Table MonthlyTime (

MonthTime\_ID int Primary Key, --This is just the month count from a given point

MonthOfRecord int,

QuarterOfRecord int,

YearOfRecord int

)

Create Table Account (

Account\_ID varchar(32) Primary Key,

Account\_Name varchar(255),

Account\_Group varchar(255)

)

Create Table Processed\_Summary (

Project\_ID varchar(12) references Project(Project\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

Budget decimal(10,2),

Cost\_Spent decimal(10,2),

Remaining decimal(10,2),

PercentOf decimal(3,2),

Primary Key(Project\_ID, Account\_ID)

)

Create Table Processed\_History (

Project\_ID varchar(12) references Project(Project\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

Time\_ID int references MonthlyTime(MonthTime\_ID),

Cost\_Spent decimal(10,2),

Primary key(Project\_ID, Account\_ID, Time\_ID)

)

Create Table Project\_Comment\_Field (

Comment\_ID int Primary Key,

Project\_ID varchar(12) references Project(Project\_ID),

Time\_ID int references DailyTime(DayTime\_ID),

Comment varchar(255),

Author varchar(64)

)

Create Table Processed\_Hours (

Project\_ID varchar(12) references Project(Project\_ID),

Time\_ID int references MonthlyTime(MonthTime\_ID),

Employee\_ID varchar(12) references Employee(Employee\_ID),

Total\_Hours decimal(10,1),

Revenue decimal(10,2),

Primary Key(Project\_ID,Time\_ID,Employee\_ID)

)

Create Table Expenses (

Expense\_ID varchar(12) Primary Key,

Project\_ID varchar(12) references Project(Project\_ID),

Employee\_ID varchar(12) references Employee(Employee\_ID),

Time\_ID int references DailyTime(DayTime\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

Expense\_Amount decimal(10,2),

Comment varchar(255),

CurrStatus varchar(12)

)

Create Table Subcontractor\_Fact (

Project\_ID varchar(12) references Project(Project\_ID),

Subcontractor\_ID varchar(32) references Subcontractor(Subcontractor\_ID),

Time\_ID int references MonthlyTime(MonthTime\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

Sub\_Amount decimal(10,2)

)

--Unprocessed Table

Create Table Unprocessed\_Hours (

Line\_ID int Primary Key,

Project\_ID varchar(12) references Project(Project\_ID),

Employee\_ID varchar(12) references Employee(Employee\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

DateOfRecord Date,

Rate decimal(10,2),

Hours decimal(10,1),

Cost decimal(10,2),

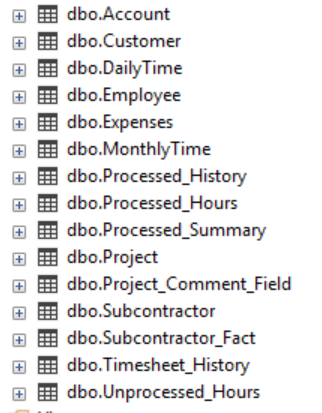
Role varchar(64),

Comment varchar(255),

ProcessedStatus varchar(12)

)

The above DDL models the constellation schema outlined in the previous section. Data types and naming convention is based off the company’s existing naming conventions.



The Indexes:

A clear index needed is any fact table’s project id field. Each of these tables are going to be filtered by that field to create the excel report, thus having this index will improve performance greatly. While it is smart to limit the amount of indexes we have to conserve storage space, I believe the other crucial indexes other than the project id are the Time\_ID in the Processed\_History and the Date in the Unprocessed\_Hours tables since both of these tables are going to be rather large and will need to order the data by these fields. I think anything additional to these field indexes will be too much, but if we observe some performance issues, account\_id and employee\_id fields would be good candidates to consider adding indexes for. The current indexes:

Create index i1 on Processed\_Summary(Project\_ID);

Create index i2 on Processed\_History(Project\_ID);

Create index i3 on Processed\_Hours(Project\_ID);

Create index i4 on Processed\_Summary(Project\_ID);

Create index i5 on Expenses(Project\_ID);

Create index i6 on Subcontractor\_Fact(Project\_ID);

Create index i7 on Project\_Comment\_Field(Project\_ID);

Create index i10 on Unprocessed\_Hours(Project\_ID);

Create index i8 on Processed\_History(Time\_ID);

Create index i9 on Unprocessed\_Hours(DateOfRecord);

The next step is to create the views to setup pulling the data into our visualizer. First we need to setup the unprocessed hours views that mirror the summary and history tables, so that we can union the two tables together easily. That setup is here:

Create View Unprocessed\_Summary AS

Select Project\_ID, Account\_ID, 0 as Budget, Cost, 0 as Remaining, 0 as PercentOf

from Unprocessed\_Hours

--Unprocessed hours have no budgets and do not directly effect the official amount remaining or percentage until they become processed, so we zero out those columns

Create View Unprocessed\_Hours\_View AS

Select Project\_ID, Account\_iD, NULL as Time\_ID, Cost

from Unprocessed\_Hours

where DateOfRecord >= DATEADD(mm, DATEDIFF(mm, 0, GETDATE()) - 1, 0)

--Select all data from the current month since this is the only unprocessed month, the time\_ID is going to be null so it can be easily pulled out in the reporting product

Create View Unprocessed\_Hours\_2 AS

Select Project\_ID,NULL as Time\_ID,Employee\_ID,Hours,Hours\*Rate as Revenue

from Unprocessed\_Hours

The next step is to create the unions to connect the processed hours and the unprocessed hours. Eventually, these views are going to be used in the framework manager of Cognos to model the data on the tool. These views are here:

Create View Summary\_Final AS

Select \* from Processed\_Summary

UNION

Select \* from Unprocessed\_Summary

Create View History\_Final AS

Select \* from Processed\_History

UNION

Select \* from Unprocessed\_Hours\_View

Create View Hours\_Final as

Select \* from Processed\_Hours

UNION

Select \* from Unprocessed\_Hours\_2

The setup of the data mart is now complete. I believe that this data mart will be able to handle all the business questions that I addressed in the first section. Now, the next part of this report is going to address how we pull in the data. With different licensing agreements and access to data, we are going to take an interesting approach to make sure we have updated and right data.

## The ETL Process

To make sure we have the most up to date and correct data, we need to schedule a job that runs periodically. For each system, we want to update the tables at different rates. For example, since we only get the raw timesheet data in a CSV file every night, we will only have to update it once a day. But for the accounting and CRM data, as employees edit and add more data throughout the day, this will have to be updated more often, every 3 hours should suffice.

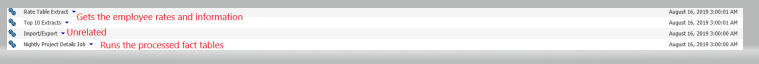
In addition to the timesheet data, we need to have the accounting and CRM data in a location ready to pull into the data mart. This can be accomplished using our reporting studio, Cognos, to pull raw data directly from the system into a CSV on our shared drive every few hours. A benefit of this process is that I can shape the data to look exactly how I want it to, thus essentially being able to skip the transformation stage. I am writing CSV files with the same columns and data types as each of the tables, with only data with timestamps later than the previous pull to avoid wiping and replacing the whole table. To assist transform and load into the data mart, I have used visual studio to create multiple scripts to update or insert into the tables.

For tables with CRM or accounting data, the process is the following:

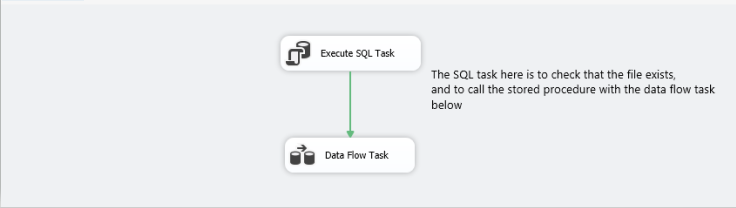
The scheduled process starts every 3 hours.

The script checks if an updated CSV file exists in the shared drive, if not the process is canceled and we are notified.

The CSV file is read in and inserted into the respected table using a stored procedure. The job is complete.



Above is the Cognos jobs for the extract. Below is the Visual Studio script outline.



An example of the SQL code is below (for the Processed\_Hours table). This code is entered into visual studio as part of the SQL Task:

Create procedure LoadProcessedHours

@ProjectID varchar(12),

@Time\_ID int,

@Employee\_ID varchar(12),

@Total\_Hours decimal(10,1),

@Revenue decimal(10,2)

as Begin

insert into Processed\_Hours

values(@ProjectID,@Time\_ID,@Employee\_ID,@Total\_Hours,@Revenue)

End

Exec LoadProcessedHours

Updating the timesheets data is a slightly different process. As I mentioned before, the data is only the last 3 years of data. I wanted to create a timesheet history table to be able to have all that data available to us. To start this, I created another table Timesheet\_History, which mirrors the Unprocessed\_Hours table:

Create Table Timesheet\_History (

Line\_ID int Primary Key,

Project\_ID varchar(12) references Project(Project\_ID),

Employee\_ID varchar(12) references Employee(Employee\_ID),

Account\_ID varchar(32) references Account(Account\_ID),

DateOfRecord Date,

Rate decimal(10,2),

Hours decimal(10,1),

Cost decimal(10,2),

Role varchar(64),

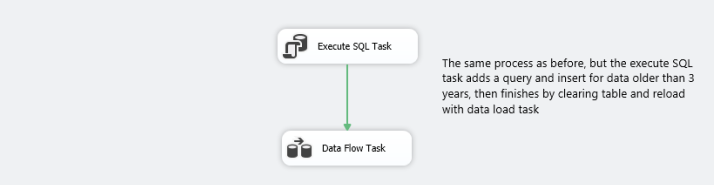
Comment varchar(255),

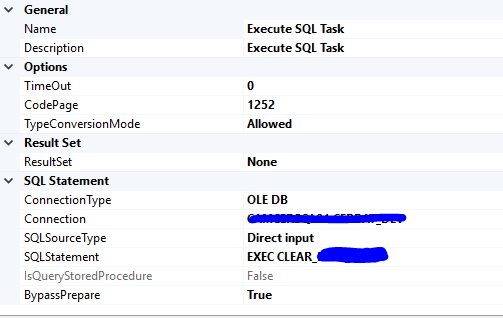
ProcessedStatus varchar(12)

)

The Unprocessed\_Hours table is going to undergo a complete wipe and replace every night, since technically an employee can change an old day of hours, so that will have to be reflected. To catch the falling off data before it is wiped and replaced, I could do this in one of two ways. The first would be to create a trigger upon deletion from the Unprocessed\_Hours table to insert data 3 years or older into the history table. I highly considered doing this, and it would work perfectly, but I decided to implement the second option. The second option adds an additional step to the visual studio script for this update to the table. The whole process is this:

1. Early in the morning, the job is triggered. First, it checks that the updated CSV file is in the shared drive. If not, the process is cancelled and we get a notification.
2. The current Unprocessed\_Hours table is queried, filtering on just data from 3 years or older (the data that has fallen out of the CSV file).
3. That query is inserted into the Timesheet\_History table.
4. The Unprocessed\_Hours table is deleted. Then, the table is repopulated with the new CSV file.
5. The job is completed.





To schedule all these visual studio scripts to run on time every day, I had to use the SQL Server Agent to do so. In all, each individual script takes only a few seconds to run, with the longest being the Timesheet one around 10-15 seconds, but that doesn’t really matter since it runs before anyone is using the system early in the morning.

## The Results

To measure the performance of the new Project Data Mart, I pulled the tables and recently updated data into the Cognos Framework Manager and modeled the data. I repointed the source of the excel project details report and ran it while looking at the administration portal. To test the difference in performance, I selected a medium sized project, and ran it using the old Project Data Mart, then with the new one. The result of the old data mart took a little more than 6 minutes to run.

The Original Data Mart:

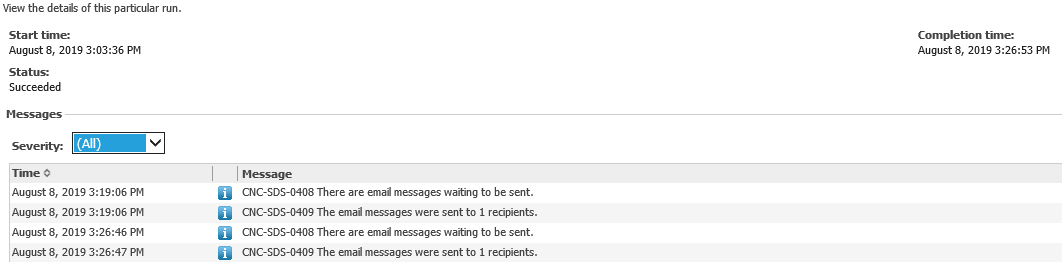


The following step was to run it again with the new source modeled from the new Project Data Mart. This took 33 seconds to run the same project as the first test.

The New Data Mart:



The new source took less than 9% of the time it took for the old source, which is an amazing improvement and it will allow us to perform the additional reporting tool that we wanted to implement from the beginning of this report: emailing copies of the project detail report at important points of the project. I tested running the new report on 30 projects. What happens is that it runs an unique report for each project, searches for the project manager, and sends the report as an excel attachment in an email to the project manager. Like I mentioned before, this was not feasible with the old report because running 30 different versions of this report would take multiple hours alone plus the time to takes to build and send the email. However, with the new report utilizing the new Project Data Mart, it took 23 minutes to finish all 30 projects.



The design and implementation of the Project Data Mart answered our key business needs addressed in this section of the report easily and efficiently. Also, we have a new table to track our whole history of timesheet data, which is hugely important for auditing purposes. With this data mart now operational, the next part of this report will focus on building the Customer Data Mart to address other key needs for my company to succeed.

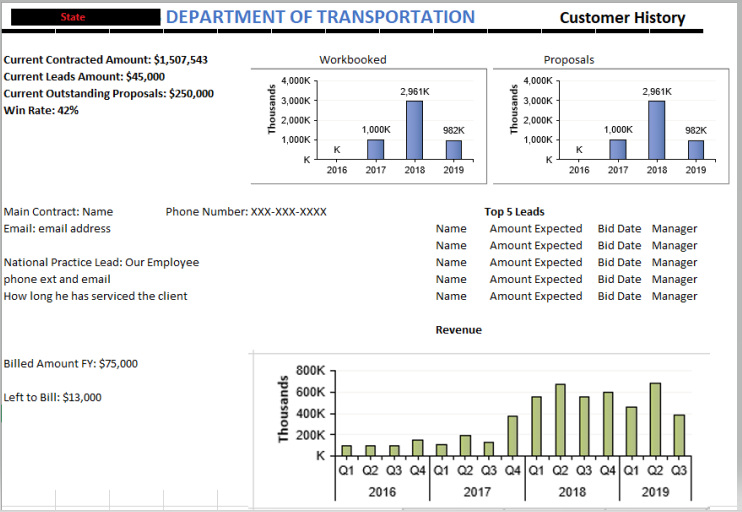
# The Customer Data Mart

While creating a full data warehouse that encompasses our entire enterprise operational data, resources and office politics limit what I can actually implement. However, creating what I call the Customer Data Mart is another essential need for our business, but unlike the Project Data Mart, it has not existed until this point. The Customer Data Mart will allow our sales team to bring a one pager of statistics and information on their meetings with our current and prospective clients, which will give them a sense of who they are talking to, the amount of work we have done with them, what proposals and leads we have, as well as trends for the past few quarters.

I will dive into answering the key business questions, the schema and ETL for this data mart (not as quite detailed as the last section since there are a lot of repeated concepts and ideas). Also, I will explore taking security measures for sensitive customer information, such as encryption and redaction. At the conclusion of this section I will show the results using sample data to again protect our clients’ and our employees’ privacy. A well designed Customer Data Mart will allow our sales team to be confident and prepared going into pitches and meeting.

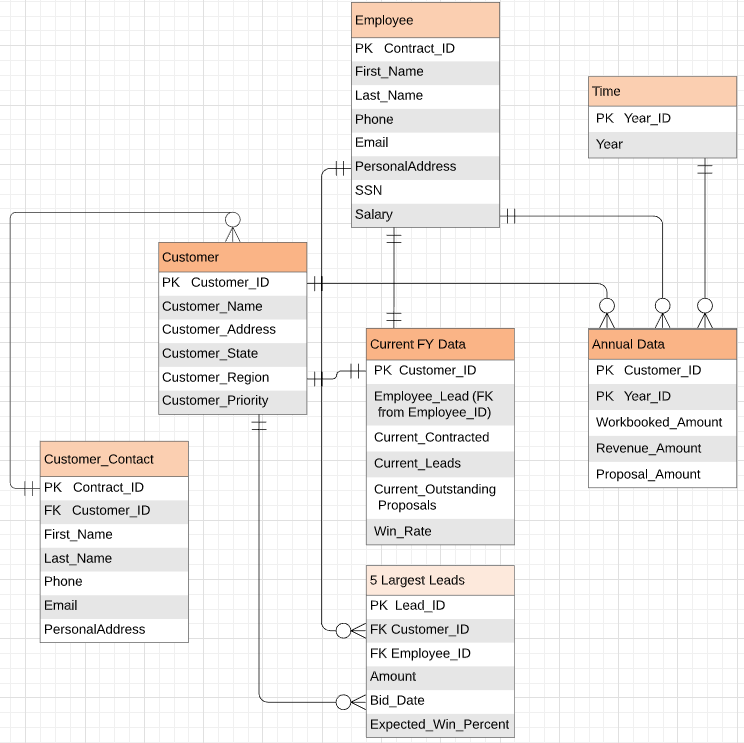
## Design, Implementation and ETL

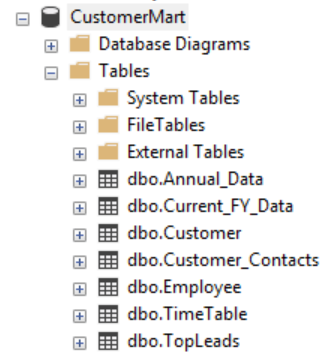
The design of the Customer Data Mart is much simpler than that of the Project Data Mart. The main thing that it has to support is a one pager that our sales team can bring to meetings with clients. A mockup of this report is this:



The main needs are the current contracted, lead, outstanding proposal amounts, as well as the win rate, all on a total aggregated level for the current fiscal year. In addition, we want to be able to show workbooked, proposal and revenue amounts on a yearly basis. Plus, there are important static information that we need to have, such as contact information and lead employee information and the top 5 leads for this particular client.

The design from these key needs is pretty straightforward. The dimensions are customer (with a separate contact table since there may be multiple contacts), employee, and time. We have 3 main facts tables to store everything that we need. A current Fiscal Year fact table stores all the summary data mentioned above, and depends on customer and employee. The annual data is pretty similar, but also has a time dimension to store annual data. The 5 largest leads are stored in a separate table. The schema is as below:





The ETL process is the same as the Project Data Mart, but we do not have to use timesheet data, so there is no need for that extra step. We extract the data from Cognos in the layout of the tables with only newly updated information. Then we insert those CSV files into the respected tables using Visual Studio and SQL Server Agent.

## Security Measure

As you may have noticed in the schema above, this data mart deals with a lot more personal and private information that the last one. Storing Social Security Numbers and customer information, especially federal officials information since we are a government contractor, in plain text is a security risk and stupid to do.

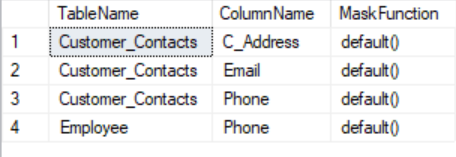
I determined that there are two levels of security that I need. The first is less critical data, but we still need to protect it at a certain level. The columns that fall under this category are customer email addresses, phone numbers, personal address and employee phone and personal address. To protect this data, I create dynamic data masks. If an unauthorized user tries to select one or more of these columns, then the data is replaced with ‘xxx’ instead of actually showing the value. Authorization is given by the administrator to specific users, so this is a safe way to keep some of this data hidden. I tested this process out myself:

I added Sample code into the customer\_contacts table and masked the phone number, email and address fields (as well as the employee phone number, but that is not going to be shown here).

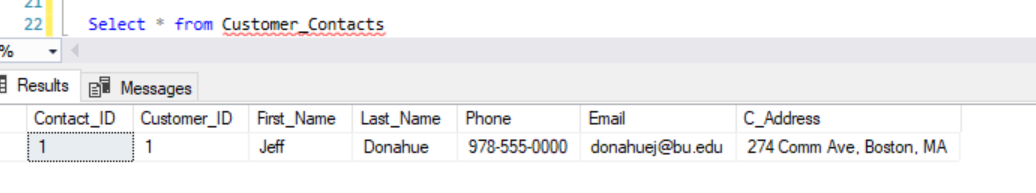
Alter Table Customer\_Contacts

ALTER Column Email ADD MASKED WITH (FUNCTION = 'default()')

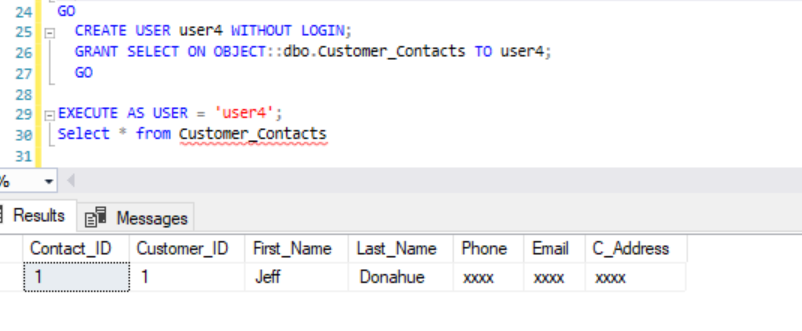
You can select the data of masked columns from the sys table:



Now, as admin, if I select \* from the customer\_contacts table, I still see all the data:

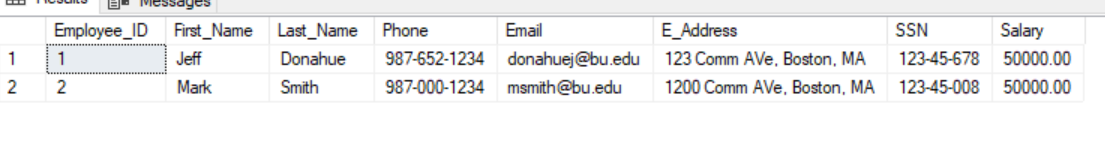


But if I create a local user, without the rights to see the unmasked data, the result is:



For data of this nature, this level of security will suffice. But when it comes to social security numbers and salary information, additional steps need to be taken. That is where the second level of security comes in; I created encrypted columns for the social security number and the salary for our employees.

Pulling from the employee table with an administrator account before encryption has been implemented:



Data with the encrypted SSN:

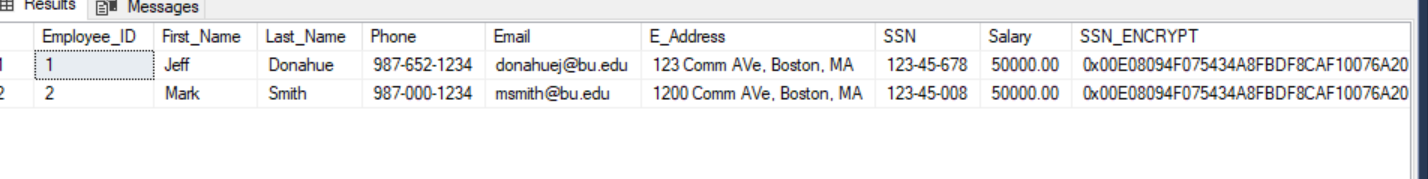
OPEN SYMMETRIC KEY SSN\_Key\_01

DECRYPTION BY CERTIFICATE EmployeeSSN;

UPDATE Employee

SET SSN\_ENCRYPT = EncryptByKey(Key\_GUID('SSN\_Key\_01'), SSN);

GO



After this is encrypted, we drop the SSN column, so that data cannot be pulled without the decrypting key. To decrypt the SSN, we have to use select the data, with the symmetric key open, like so:

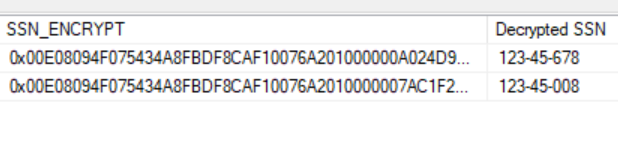
SELECT Employee.\*,

CONVERT(varchar, DecryptByKey(SSN\_ENCRYPT))

AS 'Decrypted SSN'

FROM Employee;

The result of the query is:



This process is repeated for the salary column. These two actions taken for security gives me the peace of mind that my data is protected. Semi-sensitive data is limited to the administators by the data masking tool and more sensitive data requires a decryption password.

Outlined in this section is a good start to building the Customer Data Mart. I believe that the implementation of this data mart will allow our company to gather important information that we haven’t been able to gather before, and very efficiently and securely.

# 5. Conclusion

The implementation of the two data marts is a start to building a fully encompassing data warehouse at my company. I hope the successful implementation of what is outlined above will give me the permission and trust to continue forward with expanding to answer more business questions. The unique setup of our operational databases forced me to consider complicated solutions, but the real world application of administrating a database is not as straightforward as it is in the classroom. Building security features allowed me to explore different SQL tools, and select the best ones for my certain cases.

The process of building these data marts helped me engage with the class material and led to me researching more on data warehousing and security. This final project prepared me to bring my knowledge to the professional world and take a step forward with my database administrator role at my company.

# References

(n.d.). Retrieved from <https://social.msdn.microsoft.com/Forums/sqlserver/en-US/b88c6ca1-9d24-4af3-9de7-1b06c33d0881/table-level-security?forum=sqlsecurity>

Aliceku. (n.d.). Encrypt a Column of Data - SQL Server. Retrieved from <https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/encrypt-a-column-of-data?view=sql-server-2017>

CarlRabeler. (n.d.). CREATE COLUMNSTORE INDEX (Transact-SQL) - SQL Server. Retrieved August 10, 2019, from <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql?view=sql-server-2017>

London training centre. (n.d.). Retrieved August 12, 2019, from <https://www.wiseowl.co.uk/blog/s231/schedule_data_import_in_sql_server_pt1.htm>

MashaMSFT. (n.d.). Import data from Excel to SQL - SQL Server. Retrieved August 12, 2019, from <https://docs.microsoft.com/en-us/sql/relational-databases/import-export/import-data-from-excel-to-sql?view=sql-server-2017>

What is columnar database? - Definition from WhatIs.com. (n.d.). Retrieved August 10, 2019, from <https://searchdatamanagement.techtarget.com/definition/columnar-database>

Sheldon, R. (2017, August 11). Encrypting SQL Server: Dynamic Data Masking. Retrieved from <https://www.red-gate.com/simple-talk/sql/sql-development/encrypting-sql-server-dynamic-data-masking/>