January 2012

Tactical Data Management

Christopher Pryor
Worcester Polytechnic Institute

BNP PARIBAS

Tactical Data Management

A Major Qualifying Project Report
For BNP Paribas New York, NY

Submitted to the Faculty of the
Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

_____

Christopher Pryor

_____

Siqi Wang

Date: December 16th, 2011

Onsite Liaisons:
Philip Coleman, Managing Director
Jordan Prevé, Analyst

Project Advisors:
Professor Jon Abraham, Primary Advisor
Professor Arthur Gerstenfeld, Advisor and Project Head

Abstract

The project focused on delivering a business intelligence system to BNP Paribas that would store data for easy access by the end users, while providing important auditing information for the business side. The solution integrated Microsoft Excel, Tableau, a Python module, and a standalone CSV module that could each communicate with a central database. By normalizing each data item, the data was independent of the originating program and could be successfully imported by any of the appropriate parts of the system. The intent is that this system will help to facilitate collaboration within the organization, reduce the time spent on data intensive tasks, and provide important records for compliance purposes.

Acknowledgements

The team is very grateful for the support and guidance of their primary liaisons at BNP Paribas: Philip Coleman and Jordan Prevé. Additionally, the team would like to thank their advisors; Professors Arthur Gerstenfeld and Jon Abraham, for helping to prepare them for their time in New York and for providing invaluable reviews of the final presentation and report. Lastly, the team would like to thank the entire BNP organization for allowing them to use their facilities during the course of their stay in New York and for providing examples of real-life problems that they hoped the project would address.

In addition to the names above, the team would like to specifically thank the following people for their involvement in the project:

Arnaud Remy
Terry Leaves
Arjun Kohli

Authorship

Abstract	Christopher Pryor
Acknowledgements	Christopher Pryor
Authorship	Christopher Pryor
Table of Contents	Christopher Pryor
Table of Figures	Christopher Pryor
1. Introduction	Siqi Wang
2. Background	Siqi Wang
3. Methodology	Christopher Pryor
3.1 User Discussions	Christopher Pryor
3.2 Database Design	Siqi Wang
3.2.1 Normalization	Siqi Wang
3.2.2 Separation of Metadata and the Data	Siqi Wang
3.2.3 Project Structure	Christopher Pryor
3.3 Excel Add-In	Christopher Pryor
3.4 Python API	Christopher Pryor
3.5 CSV Module	Christopher Pryor
3.6 Project Browser	Christopher Pryor
3.7 Timeline	Christopher Pryor
4. Results	Christopher Pryor
4.1 Database	Siqi Wang
4.2 Excel Add-in	Christopher Pryor
4.3 Python API	Christopher Pryor
4.4 CSV Module	Siqi Wang
4.5 Object Browser	Siqi Wang
5. Conclusion	Christopher Pryor
6. Recommendations	Christopher Pryor
References	Siqi Wang
Appendix	Siqi Wang

Table of Contents

Abstract i
Acknowledgements ii
Authorship iii
Table of Contents iv
Table of Figures vi
1. Introduction 1
2. Background 3
2.1 BNP Paribas 4
2.1.1 BNP Paribas Divisions 4
2.2 Vertica Database Platform 5
2.3 Traditional Row Oriented Database 5
2.4 Tableau 6
2.5 Business Intelligence 7
2.6 Data Normalization 7
2.7 Data Structure 8
2.8 Data Visualization 9
2.9 Current Systems 9
2.9.1 Library of Functions 10
2.9.2 Tactical Solution 10
3. Methodology 11
3.1 User Discussions 11
3.2 Database Design 12
3.2.1 Normalization 12
3.2.2 Separation of Metadata and the Data 12
3.2.3 Project Structure 13
3.3 Excel Add-In 15
3.4 Python API 15
3.5 CSV Module 15
3.6 Project Browser 15
3.7 Timeline 16
4. Results 17
4.1 Database 18
4.1.1 Normalization 18
4.1.2 The Metadata Table 19
4.1.3 Version denotation: Version and Revision 20
4.1.4 Permission 21
4.1.5 Save and Load 22
4.2 Excel Add-in 24

4.3 Python API 27
4.4 CSV Module 28
4.5 Object Browser 30
5. Conclusion 31
5.1 Facilitating Collaboration 31
5.2 Cross-Platform Functionality 32
5.3 Centralized Data Store 32
5.4 Auditing Information 32
6. Recommendations 33
6.1 More Robust Dimensional Compatibility 33
6.2 Optimize Read and Write Algorithms 35
6.3 Graphical Project Visualization 36
6.4 Object Browser 2.0 37
6.5 Permissioning 37
6.6 Full PyWest Integration 38
6.7 Multi-Measure Functionality 38
References 39
Appendix 40
Technical Terms 40
Object-Oriented Programming 40
Scripting 40
Microsoft Excel 40
Comma-Separated Values (CSV) 40
Application programming interface (API) 40
Visual Basic for Applications (VBA) 41
C# 41
Python 41
Structured Query Language (SQL) 41
ActiveX Data Object for .NET (ADO.NET) 41
Open Database Connectivity (ODBC) 42
Java Database Connectivity (JDBC) 42

Table of Figures

Figure 1 Comparison of the functionality of selected systems 3
Figure 2: Visualization of a project 13
Figure 3: Entity relationship diagram of the project structure 14
Figure 4: GANTT chart of different tasks over the course of the MQP 16
Figure 5: Project Structure 17
Figure 6: Example of data normalization in three dimensions 18
Figure 7: Sample metadata for a single revision 19
Figure 8: Saving process flow for sample data 23
Figure 9: Loading process flow for sample data 23
Figure 10: Import UserForm populated with sample information 25
Figure 11: Export UserForm populated with sample information 26
Figure 12: Upload sample CSV file from local drive with preview and default settings . 29
Figure 13: Object Browser screenshot showing sample information 30
Figure 14: Selected functionality compared to the proposed system 31
Figure 15: Visualization of a project's worflow 36

1. Introduction

In project planning, the ability to easily collaborate with your colleagues can be crucial. Although most people will achieve collaboration with shared files or email attachments, these methods are sometimes sloppy and can lead to lost information. More advanced users will sometimes develop their own databases to manage this information, but these are not always implemented in the best way for other users to access. By providing some centralized system with uniform controls, project collaboration, or just remote access, can become simplified and add value to a business.

BNP Paribas ('BNP') does not currently have a system that was purposefully designed for this task implemented throughout the firm. There is a shared drive, a number of ad-hoc spreadsheets, a market parameters tool that has been repurposed by some to store data items, and an assortment of databases that some users have built for themselves. Of these options, email attachments appear to be the primary way to move these files around, which can lead to messy conglomerations when finalizing a project. However, each of the current systems presents their own advantages. The market parameters tool is a very complete system that allows for easy sharing of Excel range objects with some auditing information and permissioning, although this is not the intended purpose of the system and is being used this way due to the demands of the end user. File shares allow users to share any type of data, not just numerical data, provide multidimensional support, and can support permissioning depending on how it is implemented. Email, through the Lotus Notes client, is relatively fast, very familiar to the current user base, and can handle almost any type of data. Additionally, ad-hoc databases can offer some advantages but they are entirely dependent on how they are created, plus they are not necessarily a "renewable resource"; they must be re-created for each new project.

While these systems can satisfy most needs of the end user, there are some tasks that they cannot handle as well as is desired. The market parameters system is the most similar to the proposed solution. It is a storage system of Excel range objects and is thus unable to store n-dimensional data for n>2. File shares and email both suffer from an inability to keep track of changes to the data or the dates in which the data has been edited. Finally, while ad-hoc databases are very flexible and useful, they aren't very user friendly, and can require a lot of work to develop – plus there is no centralized control structure, which may result in much duplication of effort from project to project.

Before the proof of concept was developed, BNP also stressed the need for a system that could store sufficient amounts of audit data, as well as provide project tracking and compatibility with established systems. The current methods of collaboration offered almost no audit information short of file created dates and information about when the emails were sent. Even then, the email system at BNP offers an inbox size of 1 GB per user, so emails with attachments will need to be deleted in order to receive new mail when the limit is reached, potentially deleting critical files. By storing critical metadata in its own location, it can be preserved for review at a later time, even after the specific revision is no longer the most current production data.

	Market Parameters	File Sharing	Lotus Notes	Cfg. Mgr.	BYO Database
N-Dimensional					?
Versioning					?
Permissioning					?
Excel Interface					?
Dates					?
Central Storage					?
Collaboration					?

Figure 1 Comparison of the functionality of selected systems

2. Background

The preparation for the Tactical Data Management MQP began in August 2011, while the project was conducted at BNP Paribas in New York from October 24th to December 16th. The background section covers the information necessary to understand the project. The team primarily worked with an analytics database management system called Vertica, which in the future will provide data storage and business intelligence for the different divisions of BNP Paribas.

2.1 BNP Paribas

BNP Paribas is one of the world's largest financial institution and a leading global banking group. With a full range of banking, investing, asset management and other financial and risk-management products and services, BNP Paribas serves individual consumers, small and middle market businesses and large corporations all over the world. BNP Paribas's four domestic markets are France, Italy, Belgium and Luxembourg. It also has significant retail operations in the United States, Poland, Turkey, Ukraine and North Africa as well as large scale investment banking operation in New York, London, Hong Kong and Singapore.

2.1.1 BNP Paribas Divisions

Within the company, there are three divisions: the Corporate and Investment Banking, Retail Banking, and Investment Solutions. Over the years BNP Paribas has become an important player in international markets. BNP Paribas is listed on the Paris Stock Exchange. The shares are also traded on SEAQ International in London, in Milan on the MTA International and on the Frankfurt Stock Exchange. This project was completed for Mr. Philip Coleman and Mr. Jordan Prevé of Fixed Income IT. Fixed Income IT is a division under the Corporate and Investment Bank division of BNP Paribas.

2.2 Vertica Database Platform

Vertica is an analytic database designed to manage large, fast-growing volumes of data and to provide fast query performance when utilized for data warehouses and other query-intensive applications. Its design features include column-oriented storage organization, which increases the performance of sequential record access; out-of-place updates and hybrid storage organization, which increase the performance of queries, insertions, and loads; compression, which reduces storage cost and insert/load bandwidth; and shared nothing architecture, which reduces system contention for shared resources and allows gradual degradation of performance.

Vertica also has a scale-out architecture, which will allow users to increase storage capacity and increase performance by adding compute and storage recourses without taking Vertica offline. The clusters grow incrementally, making the operation very simple. Also, any node in a Vertica cluster is capable of initiating loads or queries, and can evenly distribute the workload to other nodes when it makes sense to do so.

2.3 Traditional Row Oriented Database

In a typical relational database management system, data values are collected and managed as individual rows and events containing related rows. This reflects the history wherein most data begins life in transactional applications which generally create or modify one or a few records at a time for performance reasons. For example, Microsoft Access is based on row-oriented database system and has been used for many years. Conversely, business intelligence and analytic applications, generated reports, and ad hoc queries often call upon the database to analyze selected attributes of vast numbers of rows or records, needing only those columns or aggregates of those columns to support the user's needs.

Because of their row-based functions, a row-oriented database must read the entire record or "row" in order to access the needed attributes or column data. As a result, analytic and business intelligence queries most often end up reading significantly more data than should be necessary. This creates very large input/output (I/O) burdens. In addition, the row-oriented relational database management system, having been designed for transactional activities, is most often built for optimum retrieval and joining of small data sets rather than large ones, further burdening the I/O subsystems that support the analytic store.

2.4 Tableau

Tableau Desktop is a data visualization software based on technology that enables users to simply visual data. Tableau works with the human natural ability to process data visually, and it also provides very fast processing, up to millions of rows of data in seconds. The intuitive drag-and-drop interface means every change can be visualized immediately as a user performs it. Furthermore, Tableau is able to combine multiple views into a dashboard, which allows insertion of web pages and documents directly from the sources. Tableau's direct connect feature will allow users to connect directly to databases, and to the files and spreadsheets in those databases. With direct connection the user can see up-to-minute data without needing to manually refresh.

Tableau can read data either directly from a database or from an Excel file. In both scenarios, Tableau is able to set up live connection to the data file and present users the latest changes in the data if there is any editing or updates.

2.5 Business Intelligence

Business Intelligence (BI) refers to computer-based techniques used in identifying, extracting, and analyzing business data. BI provides historical, current and predictive views of business operations. BI aims to benefit business decision making by providing supports in data warehouse, reports searching, data analysis, data mining, and data recovering. In fact, BI generally facilitates the business decision making process, powers the bottom line, and helps to achieve a fully coordinated organization. Functions that considered BI technologies in this project are analytical data processing, data reporting and complex analytics processing.

In the project, BI applications use data gathered from a data warehouse. In this case, BI is a set of methodologies, processes, architectures, and technologies that transform raw data into meaningful and useful information used to enable more effective strategic, tactical, and operational insights and decision-making in the business.

2.6 Data Normalization

In the design of a database management system, the process of organizing data to minimize redundancy is called normalization. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. The objectives of normalization, according to Edgar F. Codd, the inventor of the relational model, are to free the collection of relations from undesirable insertion, update and deletion dependencies; to reduce the need for restructuring the collection of relations as new types of data are introduced, and thus increase the life span of application programs; to make the relational model more informative to users; and to make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

Having data saved in a universal format will greatly increase the efficiency and processing speed of the database. Usually, normalization involves dividing large tables into less redundant ones so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database. Normalization also involves managing data into a standard form that is required by intended usage. In general, any actions to reduce data inefficiency or to improve data consistency can be considered as data normalization.

2.7 Data Structure

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications. For example, an array stores a number of elements of the same type in a specific order while a record is a value that contains other values, while a linked list will store the defined relationship for certain records. In this project, a structure that could fulfill the purpose of supporting data transferring via different platforms is required.

In many programming projects, a suitable data structure will be the priority factor to consider. Especially for a relatively large system, the quality of data structure will determine its implementation difficulty and the quality of final product directly. Also, the algorithm for programming can be easily generated once the data structure has been finalized. Nowadays, the in depth understanding of the relations between data structure and algorithm has driven the invention of many new designs and programming languages.

2.8 Data Visualization

Visualization is the process of presenting data in a useful way to interested parties. The main goal of data visualization is to communicate information clearly and effectively through graphical means. Visualization is not only to make the data aesthetically attractive, but also to enhance the functionality. In order to achieve the balance between design and function, certain tasks are required. These tasks include data acquisition, data analysis, data mining in order to find patterns or hidden information, data transforms, and enabling the user to interact with the data. A proper visualization project uses qualitative data analysis and qualitative research to study and to summarize data, with the intent to extract useful information and develop conclusions.

Data visualization requires certain tools and certain technologies to achieve, and the method for achieving effective data visualization has been concluded to use different shapes, image processing, computer animation and user interfaces to express and to model the data to provide visible explanation. Data visualization usually involves third party software development, which in our case is the Tableau Software.

2.9 Current Systems

The system, in its current form, will only support one measure of data. However, the ability to store more than one measure would greatly improve the usability of the system, particularly when comparing similar datasets. For example, if one were to look at a risk object for the EUR:USD rate and the same object for the GBP:USD they could be put into different measures of the same object instead of entirely different projects. It would help the user to save both objects, assuming they have the same dimensions, in the same object to allow for easier comparison and better organization within the database.

2.9.1 Library of Functions

BNP Paribas library of pricing and risk management functions for various interest rate derivatives and bonds contains over 1000 functions and has over 1,600 users. It provides market data and analytics for many BNP Paribas systems. Users are able to call functions from Excel spreadsheets and from C++ and Visual Basic programs. Calling functions from spreadsheets allows users to accomplish tasks such as combine real books with scratch trades, combine official market data with their own data, and write spreadsheets for pricing, hedging, P&L and risk management. This library of functions are used globally by traders, middle office and market risk.

2.9.2 Tactical Solution

BNP Paribas currently executes a tactical solution within an ad-hoc environment of spreadsheets and other as-needed components. There is an auxiliary explorer provided to allow users to view all data in the database directly. Furthermore, the explorer will provide detailed audit information of the data such as version, time-created, author and machine. However, the current system does not offer the visibility, automation, and governance that are required by the business. Malfunction of one step within the solution could take days to determine the fault point in that records are not kept for review purposes. The tests must be run multiple times and, even then, different results could be generated. Furthermore, every solution comes from the collaboration of various departments and constant interaction with the overarching business, making periodical review a crucial part for the daily business operation.

3. Methodology

The project ultimately resulted in the delivery of a proof of concept for the database and the integration with existing systems. The solution that was developed took into account the needs of the business as well as functionality that would be valuable to the end user. After confirming the needs of the business and its end users, the different software projects began and were completed to showcase the functionality of the system to give the BNP the best information about how the system would function if it were to be used within the bank.

3.1 User Discussions

In order for the project to be well received by the bank, it was imperative for it to function in a way that satisfies the needs of the end users. Different classifications of users were taken into account throughout the project as they are materially different, although not necessarily disjoint. The end-user is more often concerned with the ability to quickly access and save data, and perform these operations as simply as possible. Whereas someone in a more research oriented role would like the ability to test different versions of the same data without necessarily releasing it to the general employees to use in their daily work. Lastly, there are the members of compliance who would want to be able to bring up important identifying information about any piece of data within the system.

3.2 Database Design

The database is the core structure of the solution and is intended to provide accessibility from every environment and software that has integrated in the solution.

3.2.1 Normalization

The data that BNP Paribas are dealing with contain almost every aspect of the market. They are often huge datasets with rows and columns indicators and multiple sheets aggregated together, multiple sheets usually being the indicator of date and time. In order to provide a clear picture of the dataset, it is necessary to perform normalization to all dataflow in the database to make sure that the database is able to store 3-dimentional data or even n-dimensional data in the future.

Combining the indicators and dimension variables together, the following rule has been set to store an n-dimensional object: store each single measure with its unique row, column and dimension indicators. In another word, store every data entry with its position parameters. For 4-dimensional data or n-dimensional data in the future, the user can just add more column as dimension indicators to store the data. The way that the data stored is a simple design, but with strong functionality and easy adaptability.

3.2.2 Separation of Metadata and the Data

Most current database systems work in a way that the database embeds identifying information of a data set to the dataset itself, integrating two parts linked with each other in the database. This is not an ideal process, because it generates extra workload for the database management system. A metadata table that stores all the identifying information of a set of data, such as name, author, version and revision, could solve this problem.

By using the index function inside SQL, indices can be created for certain columns of data. By doing so, the data can be found quickly and efficiently. The database would therefore save one piece of data into two different parts: the data part, which consist the data/values/keys and the metadata part, which contains all the information associated with the specific data. An ideal data metadata table would be formed by name, version, revision, author, date, permission and etc.

3.2.3 Project Structure

In the design, a 'project' is an object which contains a series of data. Each project can contain a series of versions and revisions. By default, for a project to exist it must have at least one version, this will be referred to as version 1 and contain the production level data for this project. By default, the associated systems will attempt to pull down this version when no specific paramenter is entered. Then the revisions represent edits to the object in time, with the highest revision number representing the most recent data object within a version.

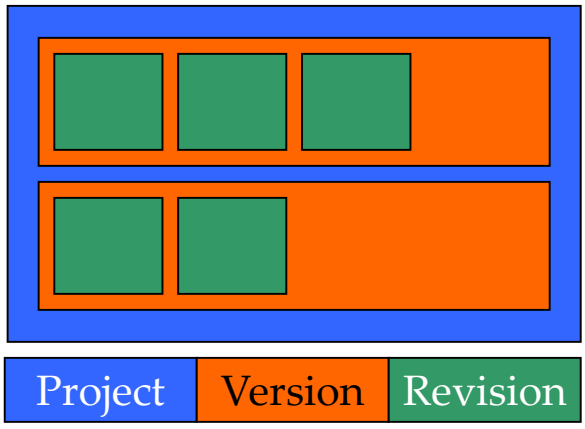

Figure 2: Visualization of a project

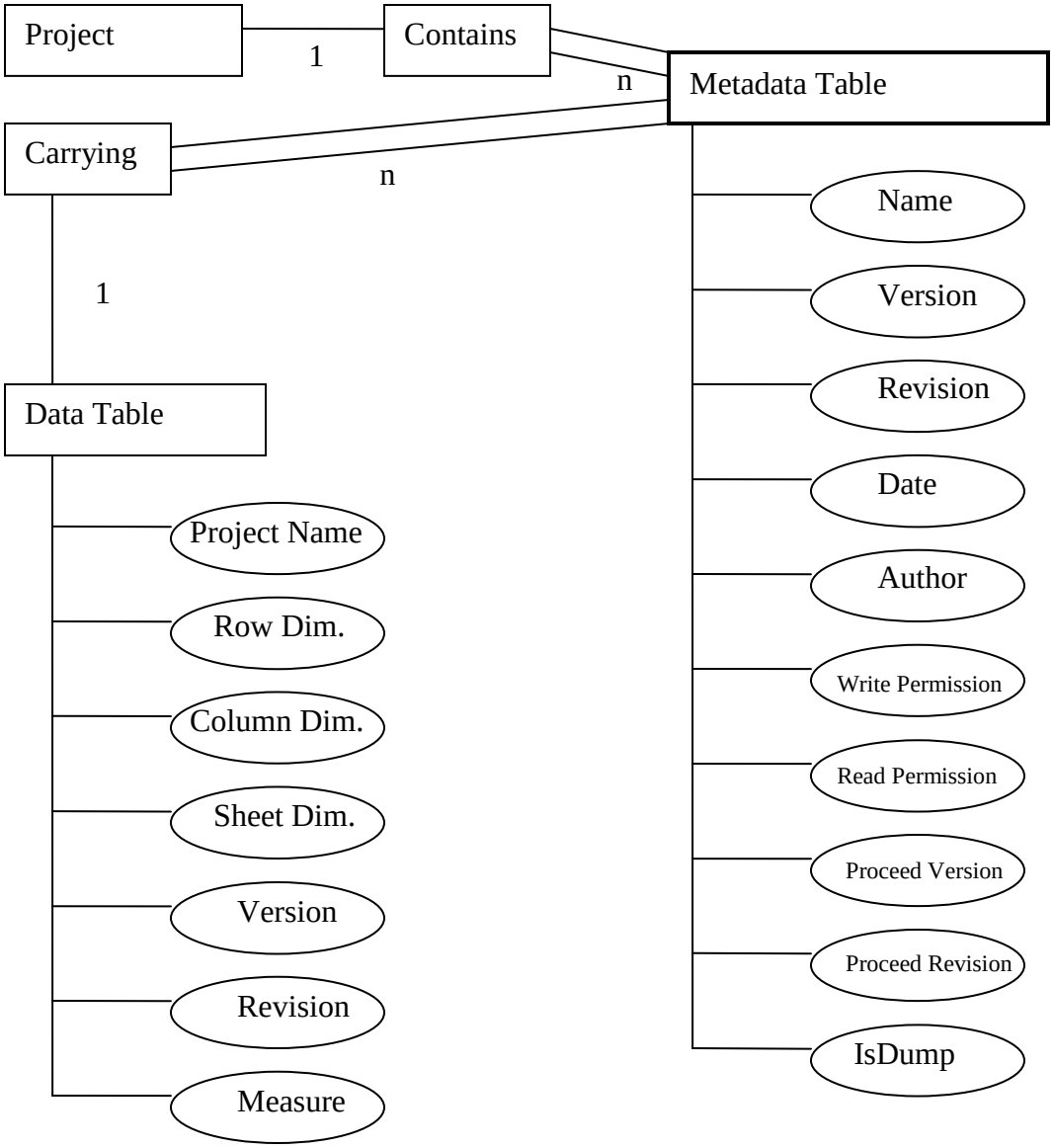

Figure 3: Entity relationship diagram of the project structure

3.3 Excel Add-In

As most employees within the bank use Microsoft Excel for data analysis, the delivery of an add-in for this platform seemed natural. If the system required users to learn how to use a new program, then it would require training and detract from the overall value to the business. Using Visual Basic for Applications, it is possible to write and query the database in a way that is familiar to the current Excel user base.

3.4 Python API

Similarly, there is a sizable user base within the BNP Paribas organization that heavily relies on the use of Python for day-to-day calculations. The goal of the Python API was to allow these users to "drop in" functions to read from, or write to, the database without needing to extensively rewrite their code. Compatibility with the current version of Python used within the bank was also stressed.

3.5 CSV Module

The goal of the CSV module is to provide a stand-alone executable that users can utilize alongside existing programs which output CSV files. Due to the common nature of CSV outputs, a tool to specifically handle these was seen as desirable, particularly because a solution to communicate with the database would not be necessary for each CSV generator.

3.6 Project Browser

Going back to the different functionality required by the different users, the Project Browser will allow users to view a nest tree to navigate through the entire database. The goal is to facilitate ease-of-use because it would not be reasonable to assume that a user would know the identifying information for each data object.

3.7 Timeline

The first two weeks of the project were spent getting set up with the necessary resources and continuing the research from the preparation period in A term. Additionally, the first week was also when most of the database structure was decided upon after time was spent discussing the needs of different users. From the third week to the sixth week, the focus shifted to prototyping the different modules and Excel add-in. During this time the structure of the database underwent some changes as the group became more familiar with the limitations of the different platforms. When one change was needed in a system, say the Excel add-in, it may have necessitated a change another system, such as a Python script. The first workable instance of the Excel add-in, which was ultimately refined over time, was completed during the fourth week, the Python API in the fifth week, the CSV module in the sixth week, and the Project Browser in the seventh week. The sixth week also marked the start of preparing for the final presentation and report. A concrete plan was developed to run through the key points in the system and showcase the functionality to senior management. The seventh week continued with practicing and refining the presentation. The eight week saw the on-site portion of the project come to a close with the delivery of the final presentation and related materials in case this platform was chosen to enter production

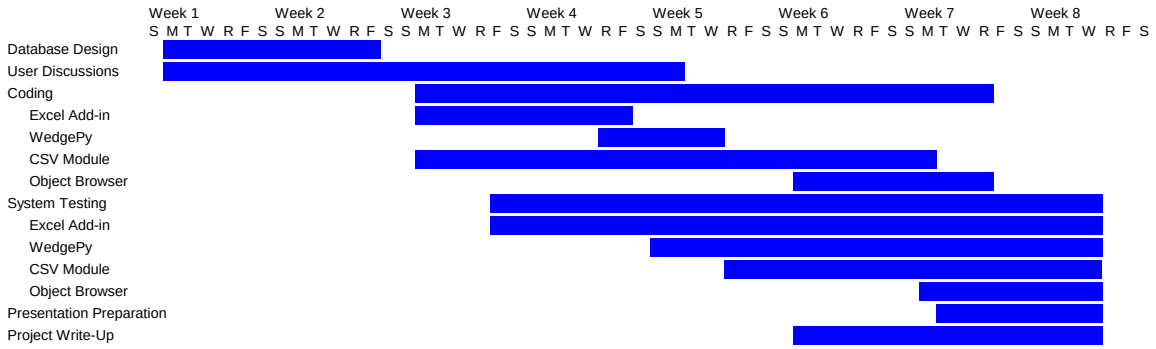

Figure 4: GANTT chart of different tasks over the course of the MQP

4. Results

The project set out to complete a proof-of-concept that would showcase the value of a centralized storage for user-generated data that also contained auditing information for compliance or other related purposes. The result was just that, the proof-of-concept consisted of: a database, an Excel add-in, a Python API, a CSV module, and the Project Browser. These systems were able to communicate with each other properly, allowing users to choose whichever was best for them and avoid needing specific systems. Since the facilitation of collaborative efforts was seen as the most important goal of this project, the ability to choose whichever tool fit one's goals, without concern for "Will this work?", was seen as a relatively important step towards ease-of-use. Below is a chart of what the proposed solution looks like. The arrows represent the flow of data so that we see the Tableau can only read data from the database while the Excell Add-in is capable of both reading from and writing to the database.

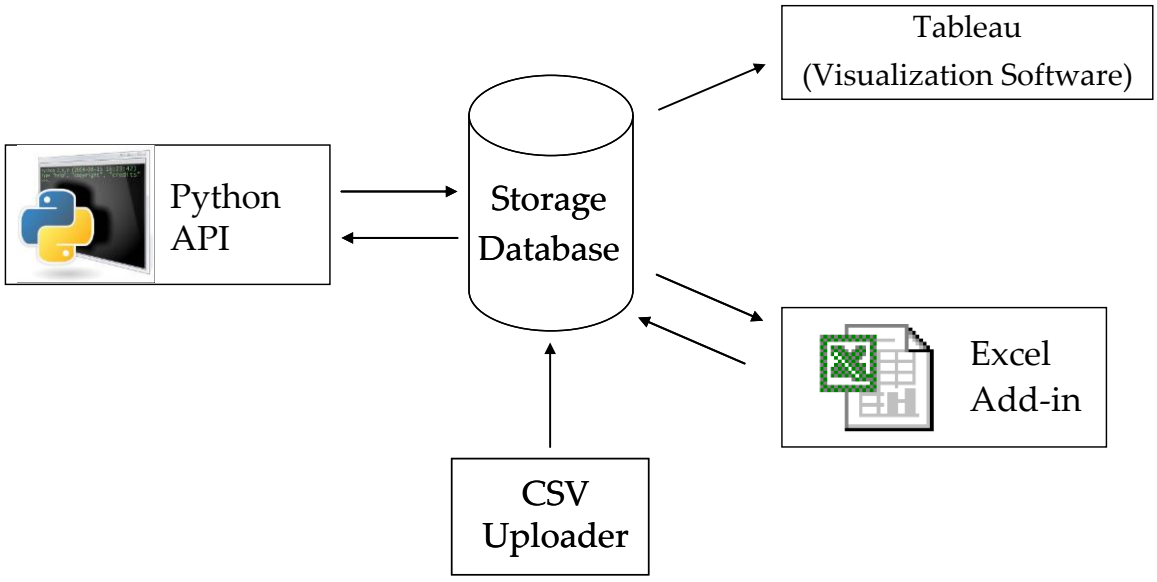

Figure 5: Project Structure

4.1 Database

4.1.1 Normalization

The data will be stored in a way which every measure will be saved with its row, column and dimension indicator. For a given data set, every key value will be saved and there won't be any duplicate value in the data table because the dimension indicator for every key value is unique.

An example to store 3-dimentional data is provided below

X-axis	Y-axis	Z-axis	Measure
1	1	1	91
1	1	2	84
1	1	3	100
1	2	1	90

Figure 6: Example of data normalization in three dimensions

For 4-dimensional data or n-dimensional data in the future, the user can just add more columns as dimension indicators in order to store the data. Also, if any new identifying information is required to be saved in the data table, such as author, adding a new column will solve the problem as well.

4.1.2 The Metadata Table

Metadata Table is a table that will store all the background information of a set of data, such as name, author, version and revision.

ProjectName	Version	Revision	Time_Created	Author	WritePermission	ReadPermission	PreceedingV	PreceedingR	IsDump
Salary	2	3	12/4/2011 13:41	wangs			2	2	FALSE

Figure 7: Sample metadata for a single revision

Compared to the traditional design of the database and information table, the architecture shown above has several new features. Firstly, the metadata table still uses the column oriented structure. With the Vertica database management system, the time for parsing through a column and find the target key is brief. When extracting information about a particular table in the database, the required time for checking through all levels of information starting from ProjectName to Version then to Revision will be shorter compared to the conventional method. This will benefit users when loading either the data or the database browser.

In addition, the traditional database system usually has its data table and metadata table connected to each other (if there is a metadata table). The term 'connected' here means selecting the data table will automatically call the corresponding data item in the metadata table. In this scenario, both the speed of the database and the storage of the database will be constrained by this action, thus affecting efficiency and usage of the system. In the database structure described above, the two parts of the data are not really connected to each other. The metadata table is designed solely for viewing and browsing purpose. The data table is designed solely for saving and loading data. This design can therefore increase the processing speed of the database and improve the overall performance of the system to better compete in a market that values information received as quickly as possible.

Thirdly, this design emphasizes the concept of a 'project', which represents the cluster of data that are related to the target usage, including all versions and revisions. Traditional databases usually store different versions and revisions of the data as single entity in the database binding with its background information. When a user calls for a particular piece of data, the system will read the information that the user has input and transmit it to the data process level and present the user with the requested dataset. This design will save the background information into the metadata table and transform the data table into whatever view is needed. In the data storage level, each single cell of information is saved under the column Measure with its Row Dimension, Column Dimension, Sheet Dimension, Version, and Revision. Each table is named by its project name and every version and revision will be saved in to this table using the criteria above. When user needs to pull information out of the database, the system will require him to input the project name of the dataset, and Version and Revision of the data. With the provided information the system can easily search through the data table level to match the project name first, and then select everything in that table with given Version and Revision. The whole loading process can be completed with a series of SQL commands.

4.1.3 Version denotation: Version and Revision

A given set of data (e.g., 7-Dec-2011/ USD Bond Market) might be edited multiple times before final versions are put into the database, and during the editing time the author may drop several versions of the data table into the database. Therefore, using version and revision to denote a specific edition of the dataset is a good solution. When a user finishes an Excel/CSV file, they may upload the table(s) to the database with corresponding version number and revision number to distinguish between different versions/users. Users can choose to manually input version and revision number or use default settings, and default settings will be offered for both auto-new-revision and auto-new-version.

Utilizing version and revision can be a useful way to save and load data with two more columns added to the data table in the database: Version, Revision. By querying SQL Create Table/Insert/Select command it will be very easy for users to read and upload data using SQL conditional restriction WHERE. However, sometimes mistakes may occur, and when they do they might affect the entire dataset given that people who are editing the data after an error has been made might use the wrong version. So it is necessary to have a denotation of what previous version and previous revision are of the current dataset in order to trace back where the mistakes occurs and who made the mistake. Putting the previous version and previous revision information into the data table will be extra workload for the data table itself thus decreasing the efficiency of the system, so using metadata table to record the current version, current revision, previous version and previous revision would be the solution. Like the settings for version and revision, the previous version and previous revision can be either manually input or set to default. When default setting is selected it will automatically record the previous version of the highest version number currently exists with the highest revision number under it.

4.1.4 Permission

The NYSE index is crucial to the research team while the foreign exchange rates are the foundation for foreign exchange swaps groups. Different groups need different information, and authentication and permission levels are needed to prevent leakage of information within the company or even worse, outside the company.

Permission usually has at least two levels of access: read, and write. Unfortunately, neither Tableau nor Excel has protected view settings that could be used for this purpose. Consequently, permission functional structure can be only implemented within the database structure. Writing the permission information into the data table is both inefficient and unnecessary, thus metadata table would be the solution, again. By adding two columns for write permission and read

permission into the metadata table, the permission setting structure is done. Users can simply input information about read permission and write permission in the textbox when saving the data. If nothing has been input, the default setting will allow only the author of the original dataset to have read and write permission. The administrator of the database will have authority to change the permission information for a particular dataset when such action is needed.

4.1.5 Save and Load

The fundamental operations that a database system needs to handle are save/upload data, and load/extract data. With the version and revision included and also the requirement to perform the permission check, both the save and the load need to perform a series of checks before functioning. The way that the system handles the permission setting is to prevent people without write permission from updating the current existing data, and people without read permission from downloading data from database. Therefore, the core information required for input is the ProjectName, for both saving and loading. The permission check also needs to be executed as soon as possible. For loading, the permission check needs to be executed at the very beginning, while for saving the system needs to know whether the dataset that the user is trying to save is a new one or not. After the initial checking the system can just get information from data table in a load case, or write data into both data table and information table in a save case. Besides, the user interface will have some default settings for version and revision in case people don't know what version and revision they are looking for/trying to save. Other information that is required by the metadata table, such as date & time and author, will be generated by automatic system command. Complete flow chart of save and load is provided below:

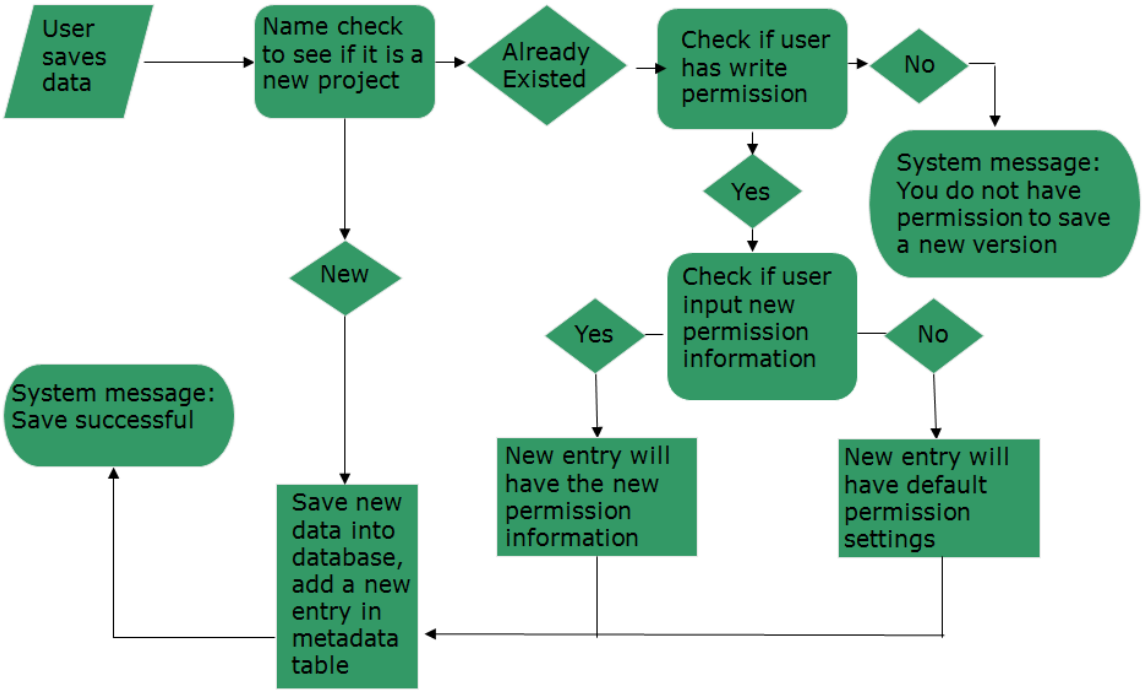

Figure 8: Saving process flow for sample data

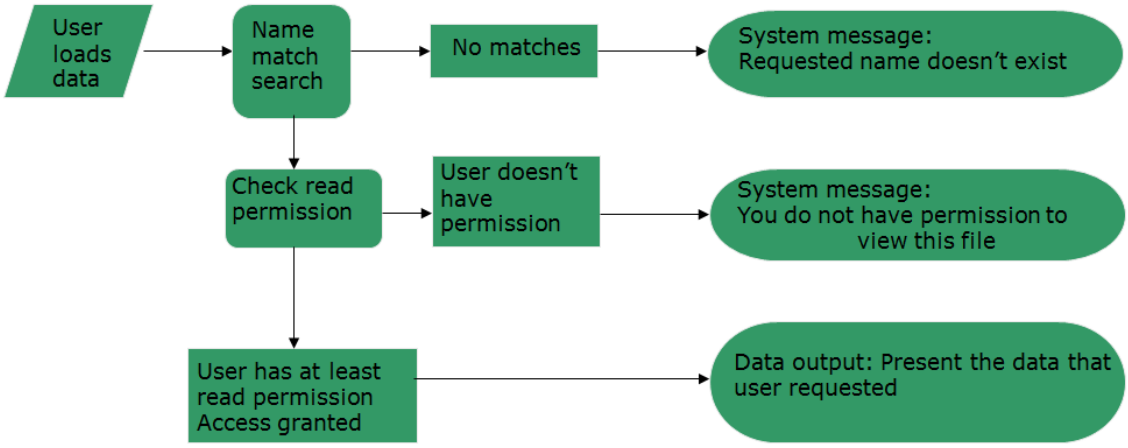

Figure 9: Loading process flow for sample data

4.2 Excel Add-in

Arguably the tool which would see the largest user base within the bank, the Excel add-in was designed to be as intuitive as possible while still giving the user full control over their data. It allows the user to act on any object, dense or sparse, keyed or not, by either importing it into Excel or exporting it to the database. The add-in was written in Visual Basic for Applications and relies on ADO objects and an ODBC driver to connect to the Vertica database.

Permissions and project existence were built into both the export and import functionality of the Excel add-in. Within the database, the permissions are broken down by author, read and write permission. The Excel module automatically grabs the current logged on Windows user and uses their username to try to login to the database. If they are not an authorized user, the add-in will return an error and they will be unable to continue the operation. The existence of the project is checked first by parsing the Metadata table for any instance where the project name exists where the Version and Revision are determined by whether data is being imported into Excel or exported to the database. For importing, an error will be return if there is an error with the selected version and revision, and if Version 1 Revision 0 does not exist while exporting, a new project will be created to handle the data. Should the user be authorized their name is then checked against the author of the object they are attempting to access. If there is a match, the program continues to either import or export data. If not, the program will check the write permission and then, if the import function is being used, check against those with read permission.

The import function is able to intelligently choose the most recent production object, within a project, when the only input is the project's name. The module queries the database to discover the most recent revision of Version 1, the production data, and populates the current worksheet with the data. For keyed data, the axes will also be brought into the current Worksheet. Because the import function loads a specific version and revision, and because a version and revision cannot be changed, the values loaded into cells cannot be updated. The function must be run again with different parameters if the user wishes to update the data since the last time it was imported into the spreadsheet.

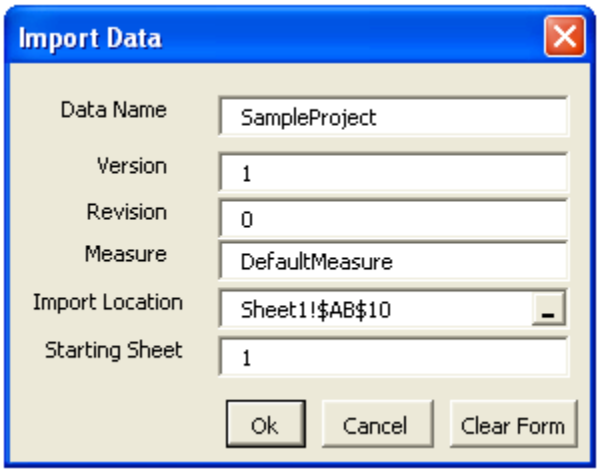

Figure 10: Import UserForm populated with sample information

The export function offers a few more controls over how data is written into the database. However, it only requires that the user input the project name, version, and the range of cells in which the data resides. After the necessary permission and existence queries have been performed, the exporting function intelligently parses the selected range, and additional pages if the data is presented in three dimensions, and performs SQL commands to insert rows into the project's data table. If axes or additional sheets are selected, the module is able to interpret these and assign the appropriate key to each value in the data range. The Version, Revision, preceding Version, and preceding Revision are handled intelligently when not specifically inputted by the user. There are defaults in place that will allow the program to run properly, although checks to make sure that the project progression is being properly constructed are not in place. This should not matter for the most part, and the end users should refer to the Object browser to see the existing structure when uploading new versions.

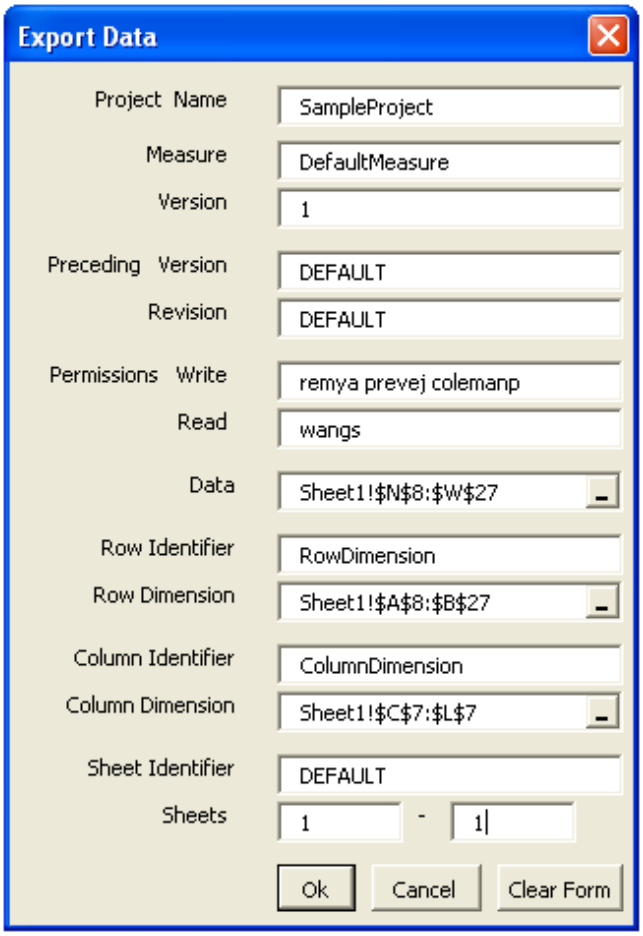

Figure 11: Export UserForm populated with sample information

4.3 Python API

The Python API, although it may see the most use from a volume standpoint, will likely have a much smaller user base than the Excel module. It can be directly embedded into existing Python scripts, projects, etc. and provide functionality to call an object in the database or to automatically write into the database after completing an operation.

In a similar way to the Excel add-in, the Python API also checks for project existence and handles object permissions in the same way that Excel does. The logged on user is checked against the different permission strings and the project name is searched for within the master table depending on whether the user is attempting to import or export data.

The import module relies on the IsDump metadata to determine if the desired object will be returned as a dictionary object or as a nested list. The module then selects the measure and dimensional columns from the data table for the selected version and revision. With the dictionary, a nested dictionary is generated so that for a dictionary "d", you can call d[x][y][z], within Python, to grab some unique element of the data object, where x, y, and z correspond to the different dimensions of the data.

ImportData(projectname, measure, version, revision)

The export functionality of the Python API relies on taking in either a dictionary, or a list, and sending it off to the database in a normalized format. As with all other modules, a series of defaults and permission checks are implemented to make the use of the module very easy.

ExportData(projectname, version, write, read, preceedingv, preceedingr, rowname, colname, shtname, measure, data)

4.4 CSV Module

The CSV Module, known as CSV Uploader, will provide user an easy way to upload data in .CSV format directly into the database.

A comma-separated values (CSV) file stores tabular data (numbers and text) in plain-text form. As a result, such a file is easily human-readable (e.g., in a text editor). Not only is CSV a simple file format that is widely supported by business and scientific applications, but CSV files are among the most popular file format in BNP Paribas. Among its most common uses is to move tabular data between programs that naturally operate on a more efficient or complete proprietary format. For example: a CSV file can be saved from an Excel Spreadsheet, object text of a tactical solution, or a word document.

In that CSV format is built for 2 dimensional, downloading a 3 dimensional data from the database and convert it into CSV format is not necessary. Therefore the CSV Module is designed to only support saving data into the database.

The core function of the CSV Uploader is to parse a CSV file into a C# dataset object, and then insert the data in the dataset into the database. The user interface of the CSV Uploader is provided below:

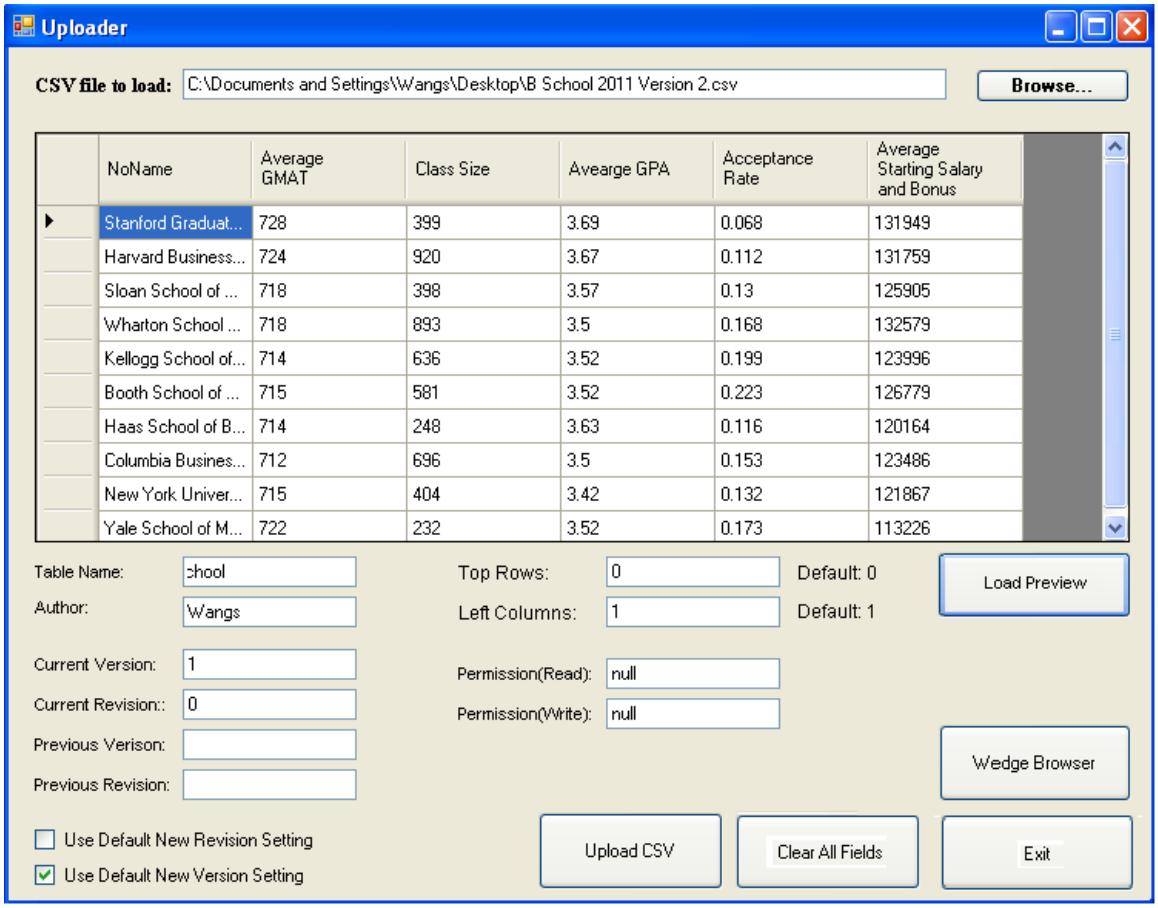

Figure 12: Upload sample CSV file from local drive with preview and default settings

As with the Excel Add-in, the user needs to input the project name before uploading. The author will be automatically generated from their Windows user name. There are several unique properties for the CSV Uploader: a user can always use the preview function to take a look at the dataset he has selected. The CSV Uploader is coded in a way that a user can upload the CSV file directly into the database without even open it. Users can select automatic version or automatic revision according to their own requirements. If a specific version is needed, users can always use the Object Browser to look at what is already in the database.

4.5 Object Browser

The Object Browser is a browser with functions as a database visualizer. Unlike normal database visualization tools though, the Object Browser will not allow users to view the data table inside the database. The main purpose for developing this browser is to let users acknowledge what is already in the database and what is not.

A view of a typical selection of the Object Browser is provided below:

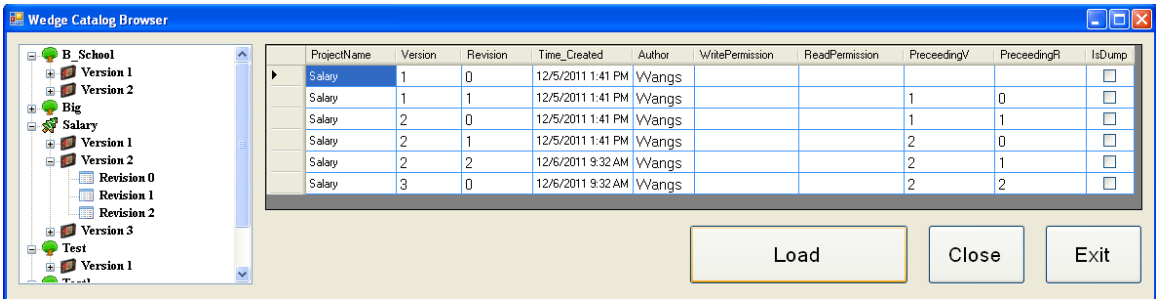

Figure 13: Object Browser screenshot showing sample information

The browser would be able to embed into any platforms supported by our applications and by using the browser the user should have a clear view of the data table and its version and revision information in a tree structure. If a user needs more information about a specific data table, he will be able to pull out the detailed information corresponding to the item that he selected. Once he confirms the selection and press at the button, everything in the metadata table that is related to the selected node will be presented to the user. If a user needs more information about a specific version of a data table, he can still pull out all the revisions this version has. By examining at preceding versions and preceding revisions we can easily have a relation map among different versions. Also, by looking at author and date & time we can always find out who made changes in the past and when those changes occurred.

5. Conclusion

The project's goal was to develop and test a system for BNP Paribas that would facilitate collaboration within the bank and add value in reducing time spent on current tasks. During the seven weeks at BNP Paribas, different modules were constructed and tested to ensure the feasibility of what was trying to be accomplished. A central database and normalized data allowed for all modules to communicate without a loss of information or data. Without this database structure, the other aspects of the project would not have been possible.

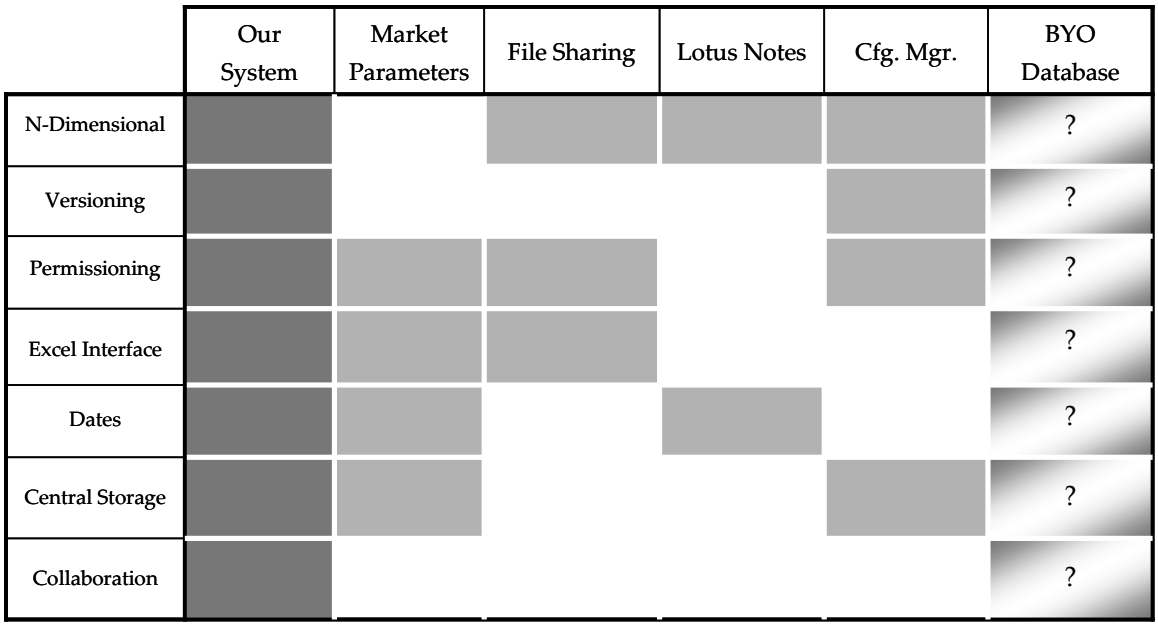

Figure 14: Selected functionality compared to the proposed system

5.1 Facilitating Collaboration

By allowing each module to communicate with the others, the collaborative aspect of the project became extremely easy to implement. Where the market parameters system currently supports something similar, there are a number of restrictions to what it can store and how the data can be accessed. By allowing the user to choose the most appropriate solution for them, be it the Excel add-in, Python API, or CSV module, it allows the user to use what they know and avoid

unnecessary training. As users become more familiar with this system, it is hoped that they will begin to use it to share raw data instead of sending it in emails or through a shared drive.

5.2 Cross-Platform Functionality

With the chosen normalization scheme, all modules were able to properly read from and write to the database without loss of information between the different platforms. Although the Python API may import information as a dictionary from the database, the associated dimensional values for each element of the measure remain the same.

5.3 Centralized Data Store

The movement away from storing data on the end user's machine allows for significantly more control over the data. If multiple users were collaborating on a single project, the data would be stored in a central location and be readily accessible even if one user was absent from work, or otherwise unavailable to send off the data. The data is also not lost by some user error, as could be the case where one forgets where they have saved an important file and cannot locate it. Provided the system is used to keep track of it, the data should never be lost. Of course, there is a downside to a centralized data store in that if the storage system fails all data could be lost.

5.4 Auditing Information

A key difference between the proposed platform and the existing market parameters system is that latter does not have as comprehensive compliance functionality as the proposed platform. Within the proposed system, all data objects exist in perpetuity and each revision of a single object is saved for review purposes. If someone was to use a specific data object in the system, and that object was revised before another user wished to run the same test, they would be able to get the same results with this system. However, if the revision history was not saved no tests would be perfectly replicable, and this could potentially let the bank incur unnecessary losses, and do nothing to identify what data was being used, so to prevent the error from occurring in the future.

6. Recommendations

The following recommendations should be treated as though a production version of this solution is being considered. It addresses some of the shortcomings of the proof-of-concept and highlights how some of the proposed functionalities would add value to the business. Although this proof-of-concept can showcase some of the benefits of this type of system, they can not be fully realized until a number of these concerns are addressed either to the specific modules or to the data structure.

6.1 More Robust Dimensional Compatibility

While the solution presented is fully capable of storing three dimensional data, there are restrictions for what raw data can be interpreted as such. Within the Excel module, the third dimension is only possible through the use of different worksheets and not by expressing the data in an example similar to the one to the right.

	1	**1**	**2**	**2**
	1	**2**	**1**	**2**
1	0.3	0.8	0.1	0.2
2	0.1	0.0	0.7	0.5
3	0.1	0.7	0.5	0.6
4	0.5	0.7	0.7	0.1
5	0.7	0.2	0.5	0.9
6	0.4	0.7	0.1	0.1

Consider that the bold numbers represent three distinct dimensions and the un-bolded numbers represent elements of a measure. Each unique combination of keys exists only once within this data set. However, the current Excel add-in is unable to assign proper dimension to this type of data as it looks only for assign column, row and sheet dimensions. It was a design constraint that was not considered to be material to the proof of concept because most data would be in the row, column, sheet orientation, even then most would not use multiple sheets.

By providing the support to assign an array, like the figure to the right which shows four dimensional data, there is the potential for the platform to see more widespread use. With the current architecture, it is not possible to create a table to properly communicate the instance of four dimensions. If anything, two dimensions might be considered if there was some concatenation of either the opposite row or column dimensions. This feature would allow the system to be even more seamless in that any, intuitively; dimensionally organized data could be stored within the database while still preserving all information.

	A	**B**	**A**	**B**	
1	0.8	0.1	0.1	0.5	**3**
2	0.8	0.6	0.8	0.7	**3**
1	0.4	0.4	0.2	0.6	**4**
2	0.5	0.7	0.4	0.8	**4**
	C	**C**	**D**	**D**	

The Python API can also benefit from more robust dimensional support. Currently, it can only read data in three dimensions. These dimensions do not necessarily need to have more than one element, but without the third dimension the module will be unable to read the data properly. For example, two dimensional information is stored in three dimensions with the third being a single, default value. This would avoid requiring the user to "cleanse" the inputs for the API and further integrate it into the banks existing Python system.

6.2 Optimize Read and Write Algorithms

At the end of the seven weeks, the group was able to write functional code for reading and writing to the database. Unfortunately, the algorithms developed would not be fast enough to put into production. This applies to all modules except for the Python API's Import function, which is able to import a data object almost instantly, as opposed to the Excel add-in, CSV module, and API's export function. That fact that at least one of the modules performs at the desired speed implies that the code for the other modules is not necessarily optimized, particularly the Excel import function, and the problem does not lie within the database itself. The for loops used in the different modules appear to be the culprits, and are likely spending the most time performing an INSERT command for each element of a measure.

Without the ability to let users read and write in real time, or close to real time, they will likely revert to the systems they embraced prior to the introduction of this system. One goal, if this would enter production, is that the targeted user base would embrace the new system so that the development is not a waste. If the system that is put into production cannot actually improve the users' experience, perhaps by slow performance, then, what was the point of implementing it in the first place?

6.3 Graphical Project Visualization

The ability to create graphical representations of the individual projects, instead of the text-based display of the current Project Browser, would likely make the system easier to understand. By displaying the concurrent versions and the revisions, a user could see how the project has developed over time. Then the user could bring up the project within Tableau or Excel to compare the versions or revisions while having some understanding of how the different data objects relate to each other.

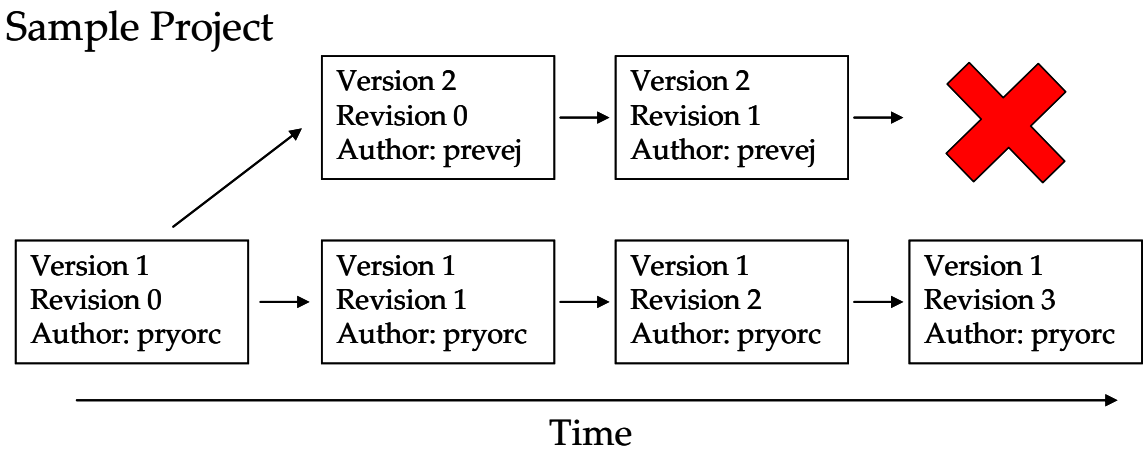

Figure 15: Visualization of a project's worflow

6.4 Object Browser 2.0

The Project Browser, in its current form, is only capable of displaying text-based information about the different projects, and the internal versions and revisions. If the user wishes to load a piece of data into Excel, or Tableau, with the browser open they must enter the identifying information to pull the correct object. However, if the Project Browser was capable of populating the identifying information for the Excel add-in, or the Tableau interface, it would also make the system easier to use. This functionality would also help to avoid entering the identifying information for a different object.

Additionally, using the object browser as an interface for Tableau could help with the relevant permission handling that was not addressed by this proof-of-concept. With the current design, there was no established way to grant unique read permissions throughout a single project. By that, one user with permission to view only version 1, or 2, of a single project could view all projects while using Tableau.

6.5 Permissioning

While the current system contains a way to handle read/write permissions, it is not perfect. There are ways around them for the savvy user, and it would likely be rejected by any compliance department for production purposes. Integration with the existing Windows logons used by BNP, perhaps even integration with the new SmartPass system being rolled out, would likely alleviate these concerns and not add to the complexity of the system.

As mentioned before, the Tableau permissioning leaves some things to be desired. It only works for specific users who have been created in the database, something that has caused segmentation faults and thus not been included in the proof-of-concept. Currently, the database administrative account can look at each element in the database. But, if the create-user issue is resolved, there would still be an issue protecting specific versions.

6.6 Full PyWest Integration

The Python module also allows for additional compatibility with an internal project related to the current Westminster system. An object-oriented Python module being developed to provide direct access to the Westminster library from the Python command line. Instead of requiring Westminster objects to be inputted into Excel only, this allows for any Westminster object to be directly loaded into Python where batch files, etc. can be run on the data as it normally would be.

Additionally, array support for the Python API would be useful to directly handle the PyWest objects. The Python API can only handle lists and dictionaries in its current state, and it would simplify the process even more if no conversion to a dictionary or list was required. For most use cases the existing system works fine, but for a production version more compatibility is generally better.

6.7 Multi-Measure Functionality

The system, in its current form, will only support one measure of data. However, the ability to store more than one measure would greatly improve the usability of the system, particularly when comparing similar datasets. For example, if one were to look at a risk object for the EUR:USD rate and the same object for the GBP:USD they could be put into different measures of the same project instead of seperate projects. It would help the user to save both objects, assuming they have the same dimensions, in the same object to allow for easier comparison and better organization within the database.

References

Codd, E.F. "Further Normalization of the Data Base Relational Model"

A Fast CSV Reader, Sebastien Lorion,

http://www.codeproject.com/KB/database/CsvReader.aspx

Importing CSV Data and saving it in database, Mukund Pujari,

http://www.codeproject.com/KB/database/FinalCSVReader.aspx

A simple Database Viewer, Uri Lavi,

http://www.codeproject.com/KB/database/DBViewer.aspx

Appendix

Technical Terms

Object-Oriented Programming

Object-oriented programming is a programming paradigm that using an ‘object’ to design and programs and applications. ‘Object’ is a concept of data fields and methods together with their interactions in data structures. Using Object-oriented programming can greatly increase the flexibility, extendibility, and re-usability of programs.

Scripting

Scripting language is a programming language that allows control of one or more applications. It serves the purpose to shorten the required time for traditional edit-compile-link-run flow. Usually, scripts are created by end-users in a different language than the core code of application.

Microsoft Excel

Microsoft Excel is a spreadsheet application that features calculation, graphing tools, pivot table and VBA macros. Its friendly user-interface, well-rounded built in functions and vivid graphical charts made Excel one of the most popular software in the field.

Comma-Separated Values (CSV)

CSV file stores numerical and text data in plain-text form, and data in CSV file is separated by comma, thus making CSV file easily human-readable. Because CSV is a simple file format, it is widely supported by business applications.

Application programming interface (API)

API is the source code that allows different software to communicate with each other. Due to the increasing complexity of software in recent years, the API is more and more designed based on the specification of intended use of software. The specifications contain routines, data structures, object classes and variables.

Visual Basic for Applications (VBA)

VBA is an implementation of Microsoft's event-driven programming language Visual Basic and its associated integrated development environment. It is built to extend the functionality of Windows applications especially Microsoft Office. VBA is considered a kind of Basic Script which supports visualization for executive applications. VBA is heavily used in banks and other financial institutions in that it `enables building user defined functions and accessing Windows API.

C#

C# (C sharp) is an Object-oriented programming language developed by Microsoft within its .NET initiatives. It is intended to be simple, modern and general-purpose. By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure

Python

Python is an Object-oriented programming, interpreted language with strong emphasizes on code readability. Python is famous for its large and comprehensive standard library. Its grammar is very efficient and clear, because Python uses indentation of block delimiters, unlike any other popular programming languages.

Structured Query Language (SQL)

SQL is a programming language designed for managing data in relational database management systems. SQL became a standard of the International Organization for Standard in 1987. However, SQL code portability between different major relational database management systems still exist due to lack of full compliance with the standard.

ActiveX Data Object for .NET (ADO.NET)

ADO.NET is a set of computer software components that programmers can use to access data and data services. It is a part of the base class library that is included with the Microsoft .NET Framework. ADO.NET uses OLE DB technology and programming languages associated with .NET Framework for its development, and it allows any programming languages to establish connections and to access relational database systems. ADO.NET also supports access to non-relational database systems such as XML and Excel.

Open Database Connectivity (ODBC)

ODBC is a standard C programming language interface for accessing database management systems. Most API of ODBC uses SQL to complete majority of missions. An application can use ODBC to query data from any database management systems directly, in that ODBC is independent of specific programming language, database system, and operating system.

Java Database Connectivity (JDBC)

JDBC is an API for the Java programming language that defines how user may connect and access to a database. Like ODBC, JDBC is oriented for relational database management systems and it provides methods for querying, searching and updating data in the database.