ABSTRACT

A Call Center is the focal point of customer service for most companies today. Using a variety of technologies, call centers connect the customer and the organization, in real-time, to provide customer service. Call centers take on different forms. A Call Center can provide pre-sales, sales and sales support or a variety of other types of service. In the context of non-profit organizations like public universities call centers are used to raise donations from the community and alumni. This is more true in the current situation where many universities have been burdened by budget cuts.

The Call Center management software has been designed to provide support for the Call Center located at the Office of Institutional Advancement at Texas A&M – Corpus Christi. The system consists of three different modules, which complement each other. The first module is a Web-based module, which assists the callers in tracking their calls, response based prompting etc. The second is a standalone module that will be used for management purposes like managing the callers, generating reports, generating personalized thank you letters for the donors etc. The last and the third module is a standalone telecommunications application, which eliminates the need for individual telephone units for the callers.

The software was developed using Microsoft .Net technology. The interfaces and logic Web module was designed and developed using ASP .NET with C# with MS SQL server as the backend. The Call Center management and the telecommunications module are standalone applications and are developed using C#.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND BACKGROUND

The Department of Institutional Advancement at Texas A&M University – Corpus Christi provides programs and services for fund raising and friend raising for the University. The department is responsible for timely and appropriate stewardship of all donations received by Texas A&M University-Corpus Christi. The department is also responsible for the collection and maintenance of the alumni records database. The department requests and accepts donations from alumni, parents, and friends of the university. The Call Center at the department handles an important task of locating future donors for the university. The donor information for different categories is obtained from various sources. The primary source of data for the alumni is the Student Information System (SIS) database. Parent donor data is obtained during the new student orientation events. The donors are divided into various segments depending on the giving preferences. For example the alumni are divided into different segments based on the college they have graduated from.

1.1 Call Center Operations

The Call Center operates both during the spring and fall semesters every year and hires 8-10 student workers every year to talk to the donors and to keep track of the donations. The Call Center is managed by a development officer who is responsible for gathering data and to assign the calling tasks to the callers. The development officer is also responsible for generating reports on donations raised on a weekly and monthly basis and to keep track of the giving history of the donors. The giving history is what helps the department identify the future donors for the University.

The center operates five days a week from Monday to Friday from 6 PM to 9 PM and on Sundays from 2 PM to 6 PM. The Call Center currently uses a database system designed using Microsoft Access to keep track of the donors. During a typical calling day each of the callers is given a list of people to call asking for donations for the University. The caller then uses the system to get the details of the donor before making a call. The callers are provided with the complete details of the donor including the name, phone number, college the donor graduated from (in case of alumni), the names and other details of the donor's children (parent donors). After the call is completed the caller will update the result of the call in the database and will proceed to the next call.

1.2 Need for New Software

The software that was being used prior to developing this software was adapted from proprietary Call Center management software called SmartCall. The software was designed to be used for a few number of donors, but as the number of donors has increased so is the complexity of managing and tracking the donations. The increase in the number of donors and the lack of proper software to manage the Call Center has hampered the productivity of the call center. With the newly developed software which provides the callers and manager with a lot of new features the Call Center would be able to generate more donations and will be able to do a better job in identifying future donors. The old Access based software system had a number of drawbacks, which were addressed in the new software. The following are some of the drawbacks that have been identified and have been corrected in the new software.

1. **Difficult to use**: The old system is not very user friendly. Every time a call is made the caller must manually retrieve the donor’s record from the database. The caller has to perform several operations to retrieve a donor’s record, which was both time consuming and involves some learning on part of the user. Because of this most of the time was spent searching for records and updating information in the database rather than spending time talking to the donors.

   The new system has the ability to automatically retrieve the donor information for the caller. The donor information is just a button click away for the caller. The new system is easy to use and needs no training before a caller can start his regular work.
2. **Report Generation**: The current system has no built-in report generation feature which would enable the Call Center manager to track the activity and progress of the call center. Reports would enable the manager identify callers who have not been performing and also would help them plan the operations to maximize the donations. The new system has built-in report generation capabilities. This helps the Call Center manager to track the progress more easily and to make maximum use of the limited resources available.
3. **Call Backs**: The current system keeps track of the results of each of the calls made but the caller has to manually retrieve the call if he wishes to call that particular donor again. For example if a call has been made to a donor and the donor wants the caller to call back at a later time, the caller enters the appropriate date and time to call the donor again. This list of callbacks are stored in the database but they have to be retrieved manually everyday. This results in a loss of productivity and, also, is very error prone.

The new system automatically retrieves any call backs. The system searches the database for any callbacks during the time of a call and retrieves the donor information. The system presents the user with all the required information like when the donor was last called, who the caller was, and the response the donor gave to the caller.

4. **Follow Up**: The Call Center manager sends thank you letters every week to all the donors. This operation is currently being done manually. The manager runs a query every Monday to retrieve the names and addresses of the people who have donated the previous week and then uses mail merge to create thank you letters to each of the donors. This process is too time consuming and is error prone.

   The new system has the capability to retrieve all the donations made between any specified time periods and print thank you letters addressed to each donor. This feature works like mail merge software and makes the letters more personal in a short time and with out a lot of effort.

5. **Management**: Each of the callers is assigned to call a particular segment of donors. The current system has no built-in option to manage this automatically. The manager currently assigns the segments to each caller on paper and the caller retrieves the records for that particular segment to make the call. This is time consuming both for the manager and the user since the manager has to check the availability of each of the segments before making the calling assignments. The users also have to sift through numerous records to retrieve the donors in the segment assigned to him.

   The new system provides features that help the Call Center manager to easily manage the Call Center. The Call Center manager can assign the callers to different segments and the system will retrieve only the donors that belong to the particular segment.

6. **Prompts**: An ideal Call Center software would prompt the user with appropriate information to present to the user depending on the users response. This would help the callers give correct and information when asking for donations. This has been one of the major drawbacks of the current system. Since the donors belong to different segments a College of Science and Technology donor would like to hear more about the current developments being done in that college. The current system lacks this feature and the users are provided with typed notes, which are difficult to handle when talking on the phone.

   The new software provides response based prompts to the callers so that they can present the donors with more accurate and update information about the university. The Call Center manager can customize the responses so that the information presented to the donors is up to date.

7. **Caller Performance**: The current system does not have a feature to help the manager track caller performance. This currently is being done manually. The manager retrieves the total donations raised by each caller and the information is updated on a white board located in the call center. This is done to show the amounts raised by each caller to instill a sense of competition among the callers.

   The new system updates the donations raised by each caller dynamically and presents it to the callers when they first log in. This would instill a sense of competition among the callers and would increase the Call Center performance.

2. CALL CENTER MANAGEMENT SOFTWARE

The Call Center management software is designed to completely automate all tasks from making a call to sending out thank you letters to the donors. The software is designed to have three separate modules to handle different tasks. The three modules complement each other to completely automate the tasks of the call center. Two of these three modules are data driven and use data from a single SQL 2000 database. The third module is a stand-alone application, which can be used to make the calls using a telephone access unit connected to the serial port of the computer.

The first module is Web-based software, which can be used to assist the callers in making the calls to the donors, updating any relevant information about the donors. This module has a built-in capability to prompt the users with suggestions during a call. The prompts can be customized by the Call Center manager to provide the callers with some useful tips and information about the University that needs to be conveyed to the donors.

The second module (management module) is a stand-alone application, which can be used by the administrator for purposes like adding, modifying and deleting Call Center users. Apart from user administration this module is capable of importing data in to the database from Microsoft Excel files that are in a predetermined format. The manager can use this module to generate thank you letters for donors who have pledged to support the university.

The third and the final module (telephony module) interacts with the telephone access unit and can be used by the callers. Telephone access units eliminate the need of a separate telephone unit for each caller; instead the callers can use the telephone access units, which are more handy and cheaper than an actual telephone unit. The telephone access units are plugged into the serial port of a computer and the software will interact with this unit to dial the number requested by the user. Figure 2.1 shows the interaction between the three modules.

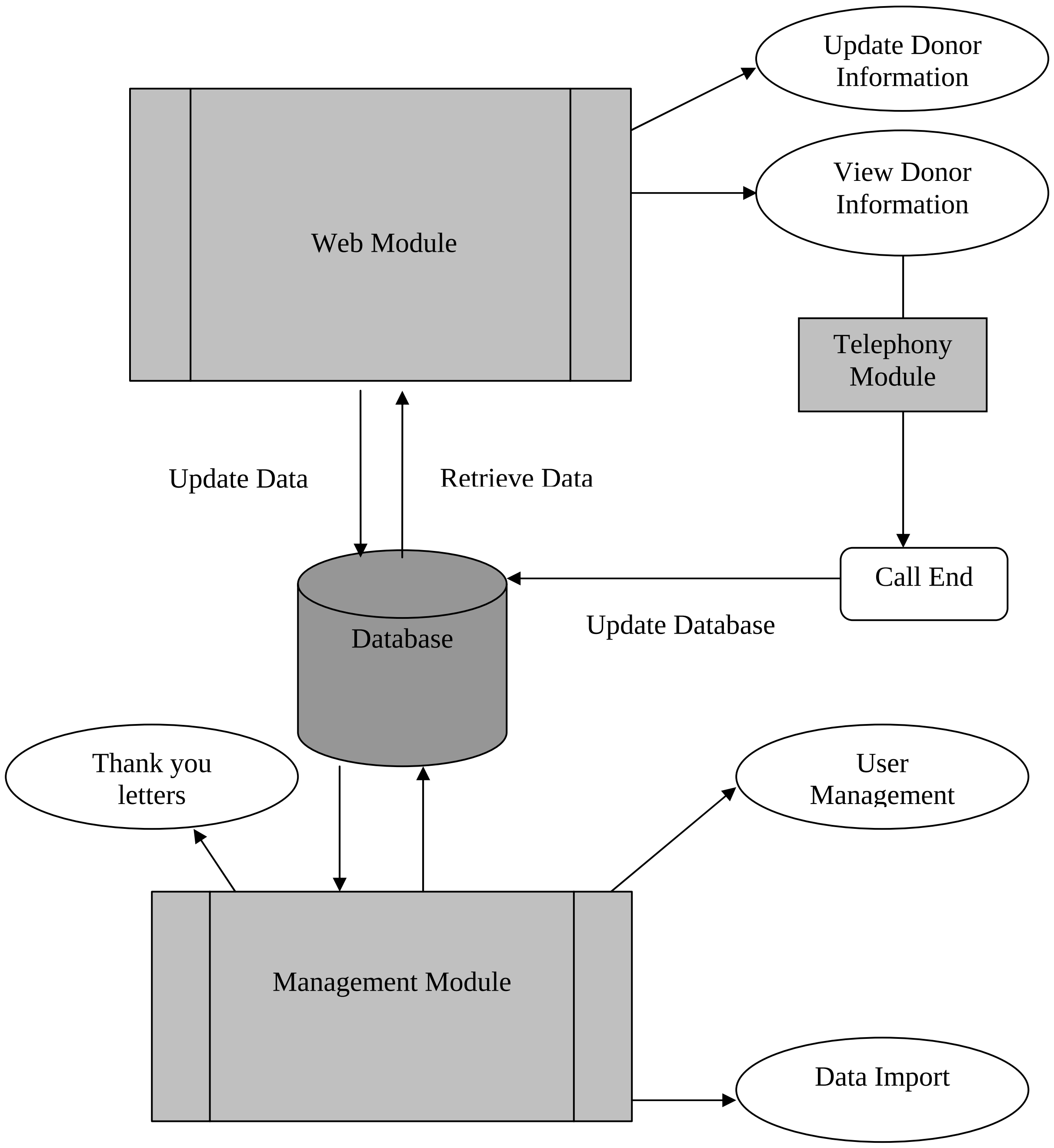
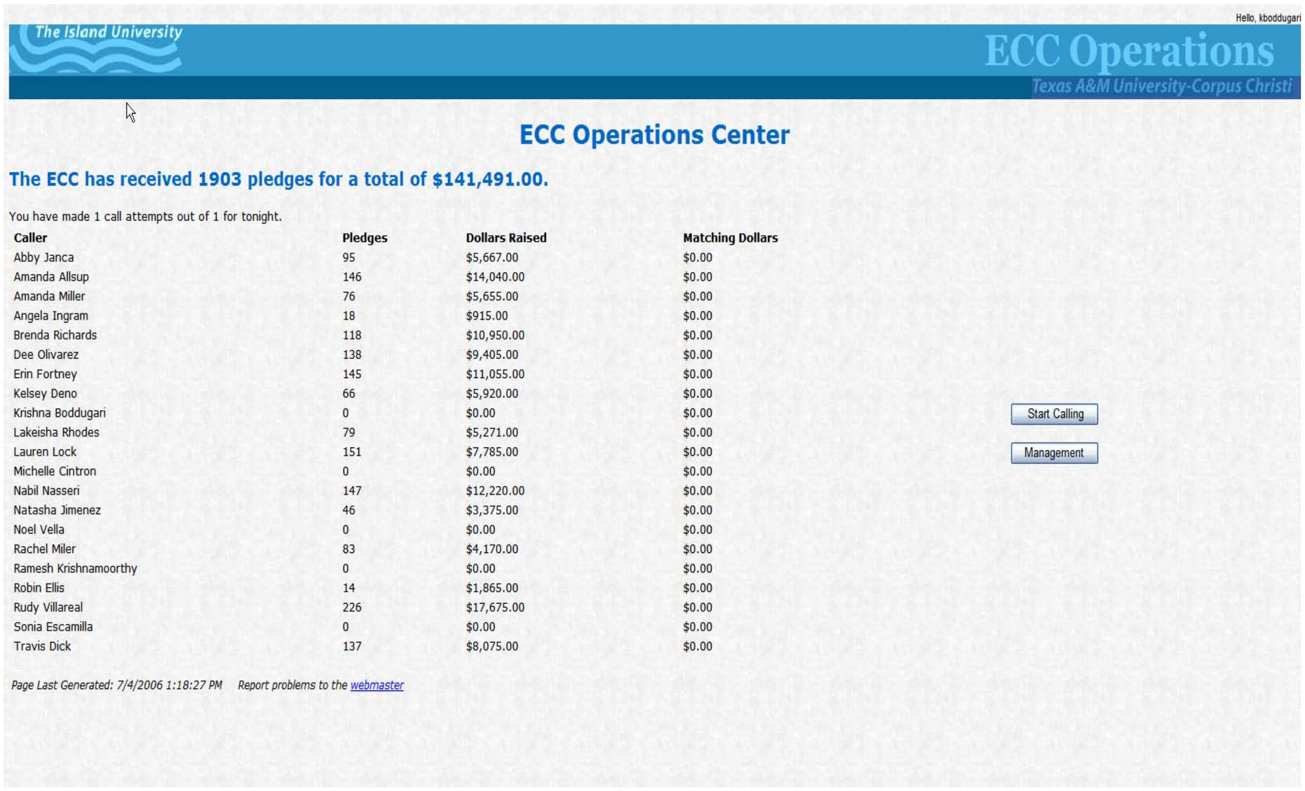


Figure 2.1 Three Software Modules

2.1 Web Module

The Web module is primarily used by the callers when calling the donors. After the caller has logged into the system and is ready to call the system retrieves the donor information. Figure 2.2 shows the screenshot of the firstpage the user sees after logging in the system. The user calls the donor using this information and will also be able to view the complete details of the donor including the details of the donor's spouse, children and the donors giving history with out navigating away from the current page. The caller can also update any of the information related to the donor on the same page. The page also provides useful tips and information to the caller that needs to be presented to the donor during the call. This information can be dynamic and can change with the donor's response to the caller.



ECC Operations Center

The ECC has received 1903 pledges for a total of $141,491.00.

You have made 1 call attempts out of 1 for tonight.

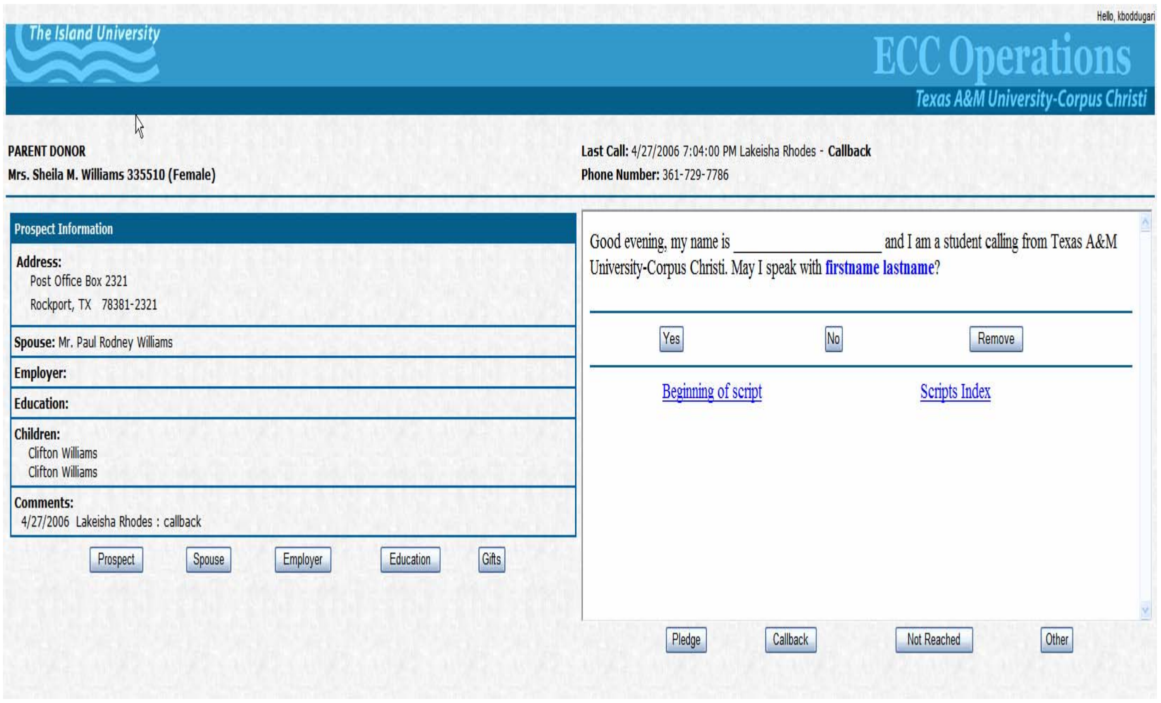
Caller	Pledges	Dollars Raised	Matching Dollars
Abby Janca	95	$5,667.00	$0.00
Amanda Allsup	146	$14,040.00	$0.00
Amanda Miller	76	$5,655.00	$0.00
Angela Ingram	18	$915.00	$0.00
Brenda Richards	118	$10,950.00	$0.00
Dee Olivarez	138	$9,405.00	$0.00
Erin Fortney	145	$11,055.00	$0.00
Kelsey Deno	66	$5,920.00	$0.00
Krishna Boddugari	0	$0.00	$0.00
Lakeisha Rhodes	79	$5,271.00	$0.00
Lauren Lock	151	$7,785.00	$0.00
Michelle Cintron	0	$0.00	$0.00
Nabil Nasseri	147	$12,220.00	$0.00
Natasha Jimenez	46	$3,375.00	$0.00
Noel Vella	0	$0.00	$0.00
Rachel Miler	83	$4,170.00	$0.00
Ramesh Krishnamoorthy	0	$0.00	$0.00
Robin Ellis	14	$1,865.00	$0.00
Rudy Villareal	226	$17,675.00	$0.00
Sonia Escamilla	0	$0.00	$0.00
Travis Dick	137	$8,075.00	$0.00

Page Last Generated: 7/4/2006 1:18:27 PM Report problems to the webmaster

Figure 2.2 Screenshot of the Operations Center

When the caller first retrieves the donor information the system selects a donor from two different sets of data. If the caller has any callbacks to be made during that particular time the system retrieves that information; if not the donors information is obtained randomly from the pool of donors. For example if a caller X calls donor Y on Monday and if the donor is busy during that time and asks the caller to call back during particular time on Tuesday the donor updates this information in the database and this goes into the callback pool. When the caller is ready to call on Tuesday the system searches for the callbacks to be made during that particular time and presents the caller with the information. The caller gets all the information like the previous date and the time of the call so that the caller can tell the donor what they are calling them about and why they are calling him at that particular time. Figure 2.3 shows a screenshot of the first page when the caller starts calling. The user then proceeds with the call and if a pledge is made the user will enter the information about the pledge in a separate form, screenshot of which is shown in Figure 2.5.

The Web module, in addition, provides some of the management capabilities provided in the management module. Incorporating this functionality into both the modules gives the user the flexibility to either use the Web based application or the standalone application. The Web module does not provide the complete set of functionalities that the stand-alone management module provides. Some of the functions such as data imports and generating thank you letters are not implemented in the web module. Figure 2.4 shows a screenshot of user update function of the web module.



PARENT DONOR
Mrs. Sheila M. Williams 335510 (Female)

Last Call: 4/27/2006 7:04:00 PM Lakeisha Rhodes - Callback
Phone Number: 361-729-7786

Prospect Information

Address:
Post Office Box 2321
Rockport, TX 78381-2321

Spouse: Mr. Paul Rodney Williams

Employer:

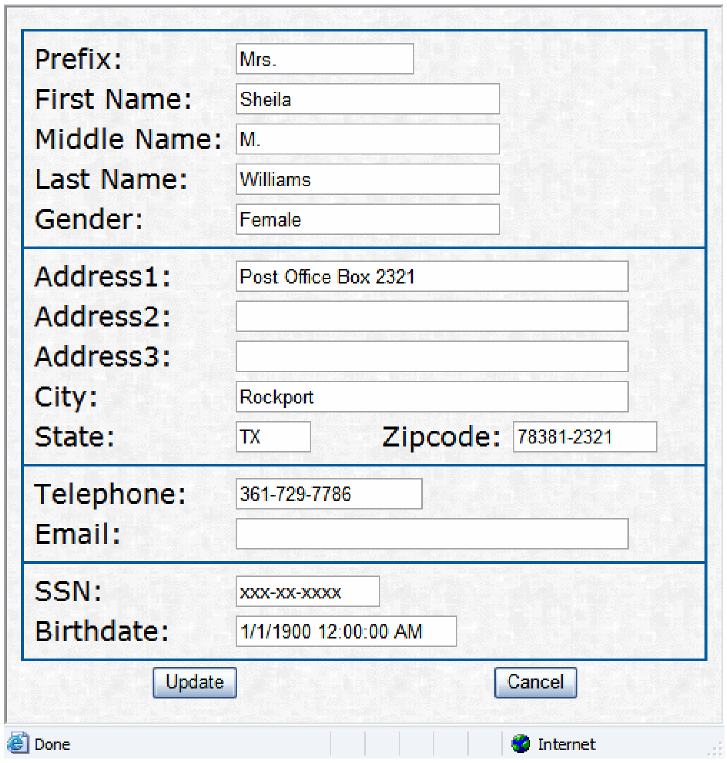
Education:

Children:
Clifton Williams
Clifton Williams

Comments:
4/27/2006 Lakeisha Rhodes : callback

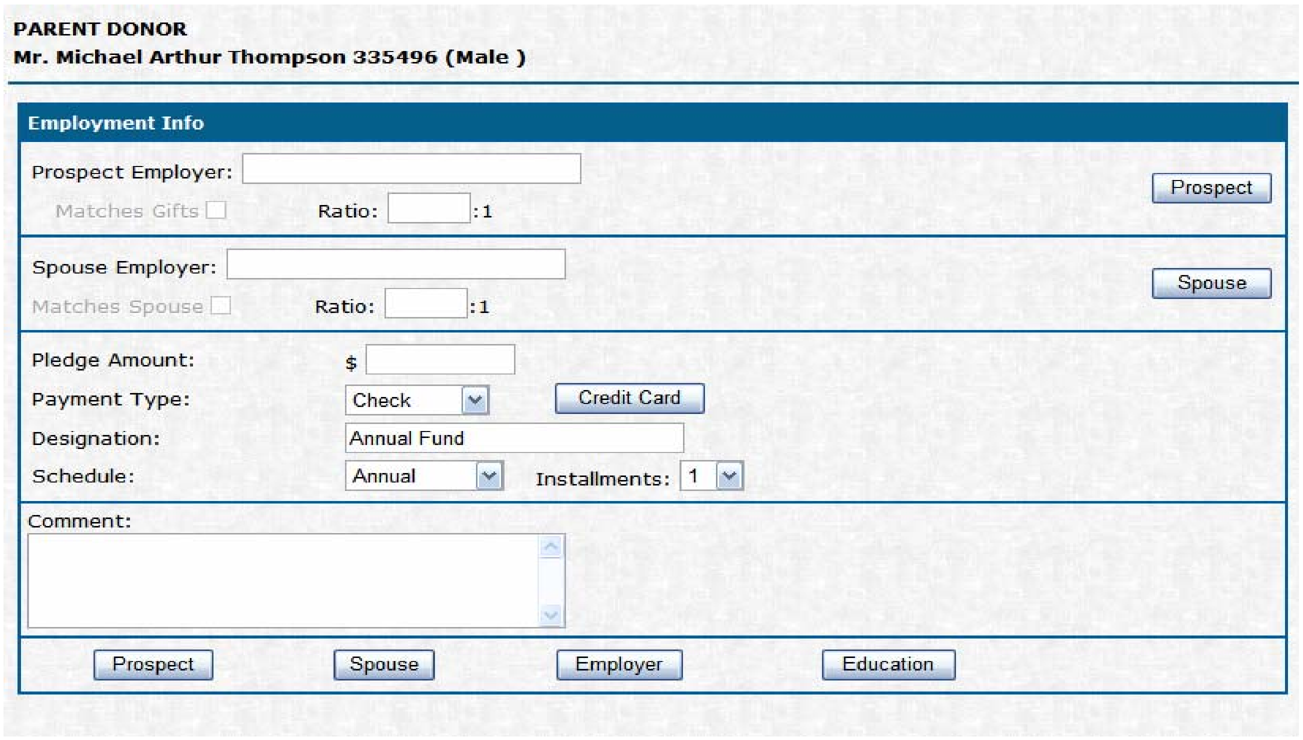
Good evening, my name is ______________ and I am a student calling from Texas A&M University-Corpus Christi. May I speak with firstname lastname?

Figure 2.3 Screenshot of the Web Module



Prefix:	Mrs.
First Name:	Sheila
Middle Name:	M.
Last Name:	Williams
Gender:	Female
Address1:	Post Office Box 2321
Address2:	
Address3:	
City:	Rockport
State:	TX
Zipcode:	78381-2321
Telephone:	361-729-7786
Email:	
SSN:	xxx-xx-xxxx
Birthdate:	1/1/1900 12:00:00 AM

Figure 2.4 Screenshot of User Update screen


PARENT DONOR
Mr. Michael Arthur Thompson 335496 (Male)

Employment Info

Prospect Employer:
Matches Gifts ☐ Ratio: :1

Spouse Employer:
Matches Spouse ☐ Ratio: :1

Pledge Amount:	$
Payment Type:	Check
Designation:	Annual Fund
Schedule:	Annual
Installments:	1

Comment:

Figure 2.5 Screenshot of Pledge Page

2.2 Management Module

The management module is designed as a stand-alone application and is used primarily for managing the Call Center operations like managing users, generating reports etc. Figure 2.6 shows a screenshot of the management module. The application is data driven and uses the same database used by the Web module to retrieve and update the data.

Another major function of the management module is to import data. The data is obtained every semester from the SIS database as a list of comma separated values. This list will be parsed and imported into the database. For the import operation to be successful the data must be in a predetermined format, provided along with the application. A sample excel template file is provided in Appendix A.

The module also provides the manager the ability to assign the users to the various calling segments. This functionality enables the manager to plan and divide the calls depending on the number of donors in each segment. The manager can view the details of the donations received and print the thank you letters to be mailed to the donors (Figure 2.7). The letters are generated from a Microsoft word template, which is provided with the application. The thank you letter template is provided in Appendix B. The content of the thank you letter can be modified to include the details the manager would like to include at any time. The module can also be used to lookup and update the donor information. Figure 2.8 shows the donor lookup and update screenshot.

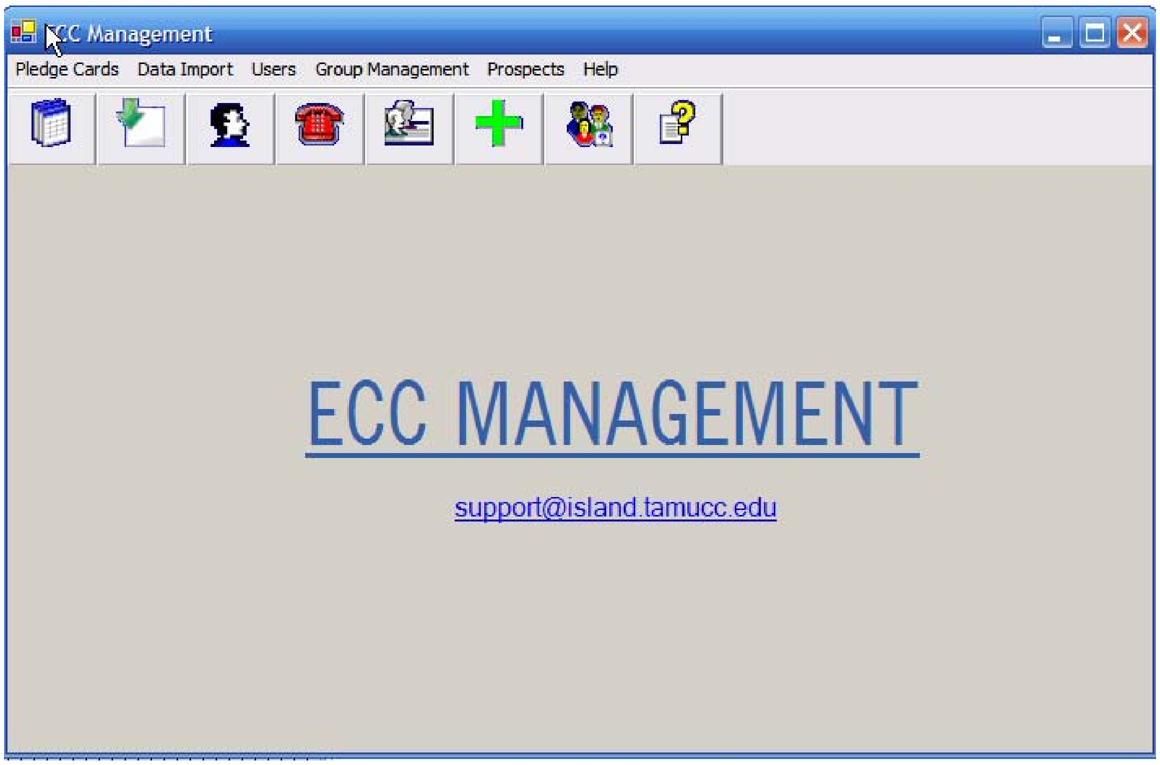
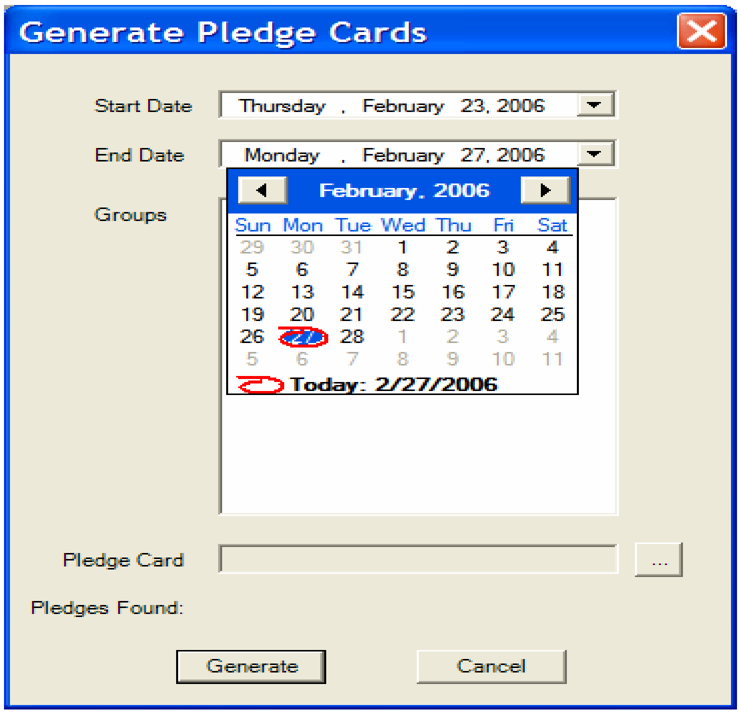


Figure 2.6 Basic Functionality of the Management Module



Generate Pledge Cards

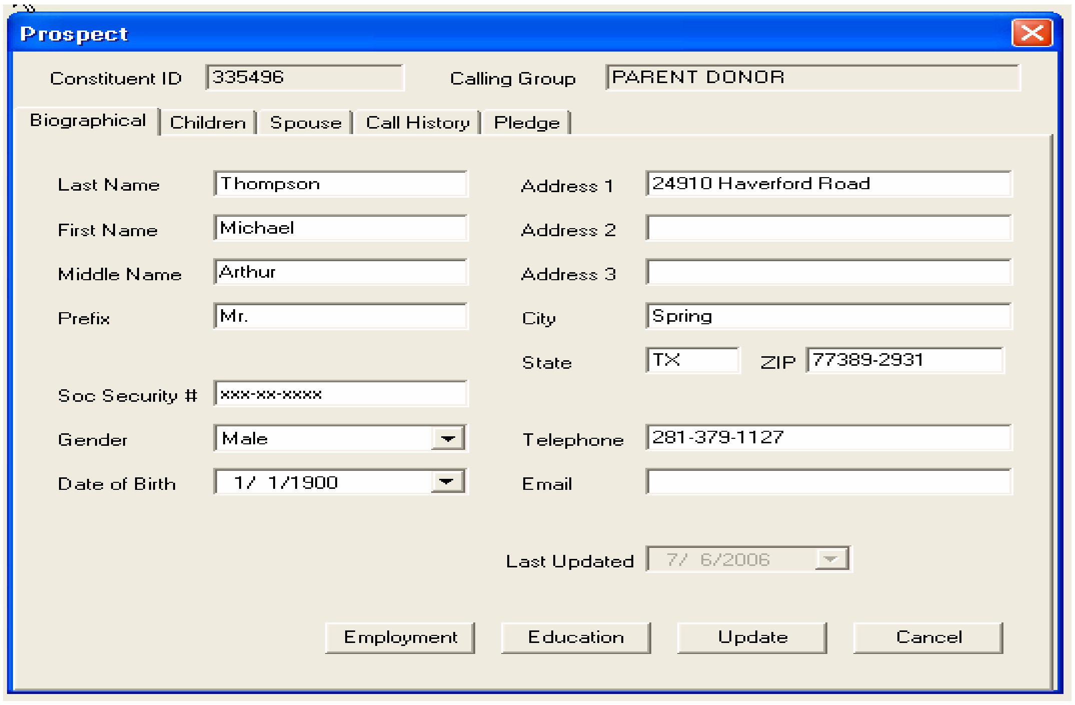
Start Date: Thursday , February 23, 2006

End Date: Monday , February 27, 2006

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	1	2	3	4
5	6	7	8	9	10	11

Today: 2/27/2006

Figure 2.7 Print Thank You Letters



Prospect

Constituent ID: 335496 | Calling Group: PARENT DONOR

Last Name: Thompson
First Name: Michael
Middle Name: Arthur
Prefix: Mr.
Soc Security #: xxx-xx-xxxx
Gender: Male
Date of Birth: 1/ 1/1900

Address 1: 24910 Haverford Road
Address 2:
Address 3:
City: Spring
State: TX ZIP: 77389-2931
Telephone: 281-379-1127
Email:
Last Updated: 7/ 6/2006

Figure 2.8 Donor Lookup and Update

2.3 Telephony Module

The telephony module is designed to interact with the telephone access unit (Figure 2.9) that can used to make calls. Use of a telephone access unit makes calling more efficient and productive giving the user the power of personal computer and telephone unit integration. Using the telephone access unit computers can make and answer telephone calls, and information about those calls can be returned to the computer. While a modem only transmits data from computer to computer over telephone lines, the unit allows communication between called and calling parties.

Figure 2.9 Telephone Access Unit [Teltone 2004]

Figure 2.10 shows a screen shot of the telephony application. The application has a simple interface with a text box to enter the phone number to dial and a group of buttons to enter the phone number. The application interacts with the telephone access unit connected to the serial port of the computer and dials the number entered by the user. The serial port on the computer needs to be enabled at the BIOS level for this application to work correctly. Once the user has finished pressing the ‘Hangup’ button on the application ends the call.

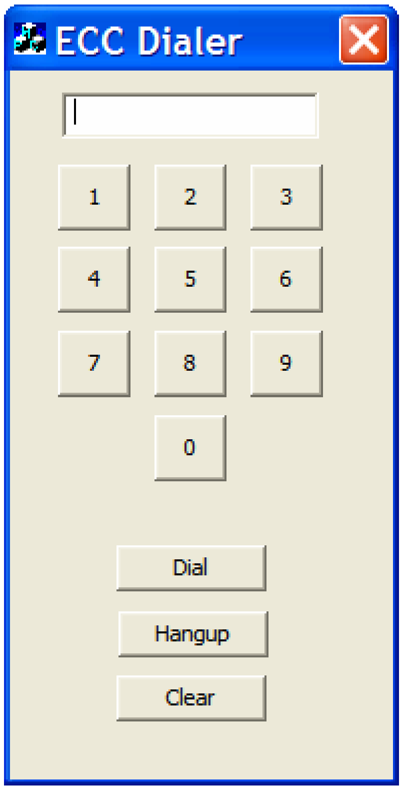


Figure 2.10 Telephony Application

3. SYSTEM DESIGN

The application consists of three different modules, which complement each other. The Web module and the management modules are designed as n-tier applications and the telephony module is a stand-alone application. The Web and the Management modules are developed using the Microsoft .NET technology. The user interfaces are designed and developed using Microsoft Visual Studio 2003. The database for the application is designed using MS SQL 2000 standard edition. The client application uses ADO .NET to communicate with the database. The telephony application is developed using the Microsoft Telephony API (TAPI) with Visual C++ as the programming language.

The Web module is designed to run using the ASP .NET framework version 1.1. The module is currently installed on a server running Inter Information Server (IIS). The management module and the telephony modules are packaged and can be a distributed as installable packages. The management module can be installed and run from any windows based computers that have the .NET framework installed.

3.1 System Requirements

The Call Center management system consists of a web module and two stand alone modules. The web module needs to be installed on a server running Internet Information Services 5.0 or higher. The stand alone modules are distributed as installable programs which can be run on any system running Windows 98 or higher and must have the .NET frame work installed.

3.1.1 Database Server Requirements

The machine chosen for running the database must meet the following minimum system requirements:

- Operating System: Windows 2000 server or higher.
- Memory: 64 Megabytes of RAM (128MB) recommended).
- Hard Disk: 95–270 MB of available hard disk space for the server; 250 MB for a typical installation.
- Drive: CD-ROM drive.
- Display: VGA or higher resolution monitor.

3.1.2 Web Server Requirements

The system for installing and running the Web Module must meet the following minimum requirements:

- Operating System: Windows 2000 server or higher.
- Memory: 64 Megabytes of RAM (128MB) recommended).
- Software: IIS 6.0 with .NET framework installed.
- Drive: CD-ROM drive.
- Display: VGA or higher resolution monitor.

3.1.3 Client System Requirements

The client systems for running the web module and the stand alone applications must meet the following minimum requirements:

- Operating System: Windows 98 or higher.
- Memory: 32 Megabytes of RAM (64 MB) recommended).
- Software: Any Web browser (Internet Explorer 4.0 or higher, Netscape, Mozilla). For stand alone applications the system must have Microsoft .NET Framework installed.
- Display: VGA or higher resolution monitor.
- Other: For the Telephony application the system must have a serial port and must be enabled in the BIOS.

3.2 Database Design

The database for the application is designed and implemented using Microsoft SQL server 2000 Enterprise Edition. Web and the management applications use the database to store and retrieve donor data. Both the applications use the same database. Data retrieval and updates are implemented as stored procedures with in the SQL server. The applications connect to the database using ADO .NET technology.

3.2.1 Microsoft SQL Server 2000

Microsoft SQL Server is a relational database management system (RDBMS) produced by Microsoft. Its primary query language is Transact-SQL, an implementation of the ANSI/ISO standard Structured Query Language (SQL) used by both Microsoft and Sybase. SQL Server is commonly used by businesses for small- to medium-sized databases, but the past five years have seen greater adoption of the product for larger enterprise databases.

The SQL Server 2000 database engine includes integrated XML support. It also has the scalability, availability, and security features required to operate as the data storage component of the largest Web sites. The SQL Server 2000 programming model is integrated with the Windows DNA architecture for developing Web applications, and SQL Server 2000 supports features such as English Query and the Microsoft Search Service to incorporate user-friendly queries and powerful search capabilities in Web applications.

The same database engine can be used across platforms ranging from laptop computers running Microsoft Windows 98 through large, multiprocessor servers running Microsoft Windows 2000 Data Center Edition. SQL Server 2000 Enterprise Edition supports features such as federated servers, indexed views, and large memory support that allow it to scale to the performance levels required by the largest Web sites.

The SQL Server 2000 relational database engine supports the features required to support demanding data processing environments. The database engine protects data integrity while minimizing the overhead of managing thousands of users concurrently modifying the database. SQL Server 2000 distributed queries allow to reference data from multiple sources as if it were a part of a SQL Server 2000 database, while at the same time, the distributed transaction support protects the integrity of any updates of the distributed data. Replication allows to maintain multiple copies of data, while ensuring that the separate copies remain synchronized. [Petkovic 2000].

3.2.2 Database Design

Database for the application is designed using MS SQL 2000 standard edition. The Web module and the management module use the same database to store and retrieve the data. Figure 3.1 shows the ER diagram for the database design. The applications access the data using the stored procedures. A stored procedure is a group of SQL statements that form a logical unit and perform a particular task [Henderson 2003]. Stored procedures are used to encapsulate a set of operations or queries to execute on a database server. Stored procedures can be compiled and executed with different parameters and results, and they may have any combination of input, output, and input/output parameters.

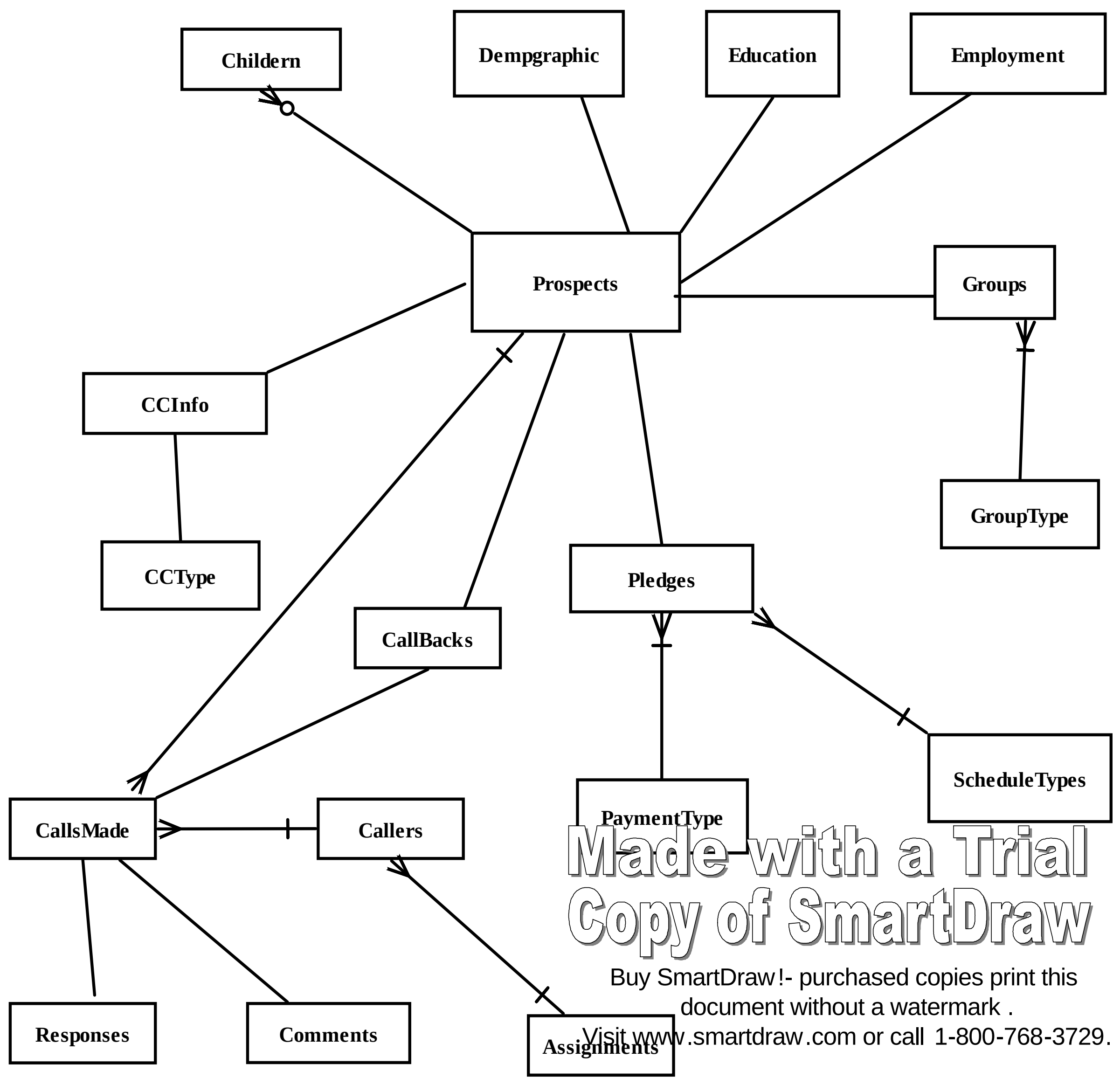


Figure 3.1 Entity Relationship Diagram

3.3 Application Design

The application has been designed using Microsoft .NET technology. The Web module was designed using ASP .NET with C# and the management module is a

Windows forms application designed using C#. Both the applications are designed as n-tier applications where the business logic is separated from user interface design.

3.3.1 Microsoft .NET Framework

The .NET Framework is a new computing platform that simplifies application development in the highly distributed environment of the Internet. The .Net Framework enables developers to develop applications for various devices and platforms like windows application web applications windows services and web services [Jones 2002].

The .NET Framework has two components the .NET Framework class library and, the common language runtime (CLR). The .NET Framework class library facilitates types that are common to all .NET languages [Deitel 2003]. The common language runtime consists of class loader that load the Intermediate Language (IL) code of a program into the runtime, which compiles the IL code into native code, and executes and manages the code to enforce security and type safety, and provide thread support.

The .NET Framework Architecture has languages at the top such as VB .NET C#, VJ#, VC++ .NET; developers can develop applications such as Windows Forms, Web Form, Windows Services and XML Web Services. Bottom two layers consist of .NET Framework class library and Common Language Runtime.

3.3.2 User Interface Design

User interface for the application was designed using Microsoft Visual Studio 2003. Cascading Style Sheets (CSS) are used to give the Web module a consistent look.

This makes it easier to change the layout of the Web interface. The access to the SQL 2000 database is provided by Microsoft® ActiveX® Data Objects (ADO). ADO enables the client applications to access and manipulate data from a variety of sources through an OLE DB provider.

Visual Studio .NET 2003 is the comprehensive tool for rapidly building Microsoft .NET–connected applications for Microsoft Windows and the Web. Visual Studio .NET 2003 includes integrated support for the Microsoft .NET Compact Framework [Onion 2005]. Visual Studio 2003 supports integrated emulation, which enables for integrated development and debugging without requiring any additional devices.

3.3.3 Database Connectivity

ADO.NET provides consistent access to data sources such as Microsoft SQL Server or MSDE 2000, as well as data sources exposed through OLE DB, SQL Client, ODBC and XML [Thomsen 2002]. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. ADO.NET leverages the power of XML to provide disconnected access to data. ADO.NET was designed hand-in-hand with the XML classes in the .NET Framework [MSDN 2004].

The ADO.NET Dataset object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML [MSDN 2004]. “ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET Dataset object in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources" [MSDN 2004]. "The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. When compiling code that uses the System. Data namespace, reference both System.Data.dll and System.Xml.dll" [MSDN 2004].

3.3.4 Telephony Application

The telephony application is designed using Microsoft's telephony API. The telephony application is designed using Visual Studio 2003 with Visual C++ as the programming language. Telephony API, or Telephony Application Programming Interface (TAPI) was developed jointly by Microsoft and Intel, with input from a number of telephony companies, and originally released in 1994. TAPI enables Windows applications to share telephony devices with each other and provides a common means of handling different media (voice, data, fax, video, etc.) on a wide range of hardware platforms.

TAPI is available on all Windows platforms from Windows 3.1 to Windows XP. Different versions of TAPI are available on different versions of Windows. Windows 95 was the first version of Windows to integrate TAPI directly into the operating system.

TAPI interacts with the hardware using a driver called Telephony Service Provider (TSP). Any TAPI application can interact directly with modems but when any other hardware like a telephone access unit or voice processing cards have to be used TSP driver provided by the manufacturer has to be used to access and control the hardware.

4. EVALUATION AND RESULTS

Software testing is the process used to help identify the correctness, completeness, security and quality of developed computer software. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation [Pressman 2000]. A set of testing schemes were used to test the functionality and usability of the application. Testing was done both during the project development (unit testing) and after the application has been completely developed. To minmize the number of defects unit testing was done during the application development. Each part was tested individually to test the correct functionality. Once the application has been developed completely all the individual parts of the application were integrated and integration tests were performed.

4.1 Application Testing

The functionality of the application has been tested using use cases and test cases. The use cases were provided mostly by the Call Center manager. A use case is a technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal. Use cases typically avoid technical jargon, preferring instead the language of the end user or domain expert [Patton 2001].

Several use cases were developed to test the functionality and usability of all the three modules. Each use covered specific functionality of the application. Separate use cases were written for each of the three modules. After the application has been completely developed several users from the Call Center were given the use cases to test the application. Errors found during this testing were fixed before moving to the next stage.

4.1.1 Functional Testing

Data for testing the application was provided by the Call Center manager. The donor data from the previous calling season was provided as excel sheets. The data from Microsoft Excel sheets was imported into the application using a perl script that was written just to import this data.

After the data has been imported the application was tested during the last calling cycle during Spring 2006. During this phase the application was used as a production system and seemed to perform up the user expectations. Any defects found during this cycle were fixed and a next round of testing was conducted to make sure that all the defects found during the first phase were corrected.

Testing was performed by the callers at the call center by using test cases. The callers performed the actions specified in the test case and verified the results. Each module had different test cases to test the functionality of that particular module. The test cases were written as a series of actions for the end user to perform with verification points which were used to indicate the functionality of the application. Table 4.1 shows the results of the functional testing.

Table 4.1 Functional Testing Results

Test Phase	Module Tested	Number of Test Cases	Number of Defects
Phase 1	Web	26	4
	Management	12	0
	Telephony	3	0
Phase 2	Web	26	0
	Management	12	0
	Telephony	3	0

4.1.2 Usability Testing

Student workers and the Call Center manager were involved in usability testing. Their feedback was collected and any suggested changes if possible were implemented.

Usability testing was done to identify how users actually interact with the application. During this testing the users were asked to work with the application while the Call Center manager took notes on how well the users were able to use the application and, the ease of navigation. The application was tested for the ease of navigation through the forms, appearance, consistency so that the data is displayed as expected.

All the pages in the Web module and the forms in the stand alone application were checked for the uniformity in the appearance. The Web module in addition was tested for any broken links and also if the links are pointing to the correct pages. Table 4.2 shows the results of usability testing.

Table 4.2 Usability Testing Results

Usability Metric	Score (Out of 5)
Ease of Use	5
Look and Feel	4
Colors and Fonts	4
Ease of Navigation	5
Features	5

5. FUTURE WORK

The Call Center application is designed for raising donations for a university. This application suffices the need for any Call Center in a university scenario. With some minor modifications and additions the application can be designed to be customizable for any non-profit organization.

The application currently uses a separate module for data imports and to print thank you letters. This is done because importing data and generating word documents using a browser is a difficult and time consuming task. This task can be incorporated into the Web application by designing Active X controls that can be used to import data and generate thank you letters. This would eliminate the need for a separate application just for the data imports and thank you letter generation.

The current application lets the Call Center manager customize the prompts that are shown to the caller during a call. Any such changes can be done by editing the HTML pages which requires a WYSIWYG HTML editor for easy editing. New functionality can be added to the application so that the Call Center manager can modify the prompts on the web. The new functionality can mimic any of the WIKI software that lets the user modify web pages from any browser software.

The telephony module currently has limited functionality. Currently the user can enter and dial the required number or end the call when the call is finished. The caller still needs to use the controls on the telephone access unit to change the volume or to mute. New functionality can be added to the telephony application has all the controls needed by the caller.

6. CONCLUSION

The application can be used in the Call Center to automate all the tasks from calling the donors to report generation. The callers use the application to make the calls and also to keep track of who they called, when they called, and the outcome of the call. Dynamic prompting capability of the application helps the callers when talking to the donors. The callers need not search for their notes as to what to say while talking to the donors. All the information relating to donor like the donor’s demographic information, employment, education etc. can be obtained just by the click of a button. All these features help the caller to be more efficient while calling and also will help them in presenting more precise information to the donors.

The management module is useful to the manager to generate reports, manage users etc. The module provides the manager with information to make informed decisions as the number callers to assign to a group during the calling season. The manager can generate reports to compare the Call Center performance across a period of time. The manager can spend more time planning rather than managing the callers in the call center.

The telephony module makes calling easy. This cuts the cost of purchasing costly telephone equipment for each caller.

BIBLIOGRAPHY AND REFERENCES

[Deitel 2003] Deitel, H.M; Deitel, P.J; Listfield, J.A; Nieto, T.R; Yaeger, C.H; Zlatkina, M. C# for Experienced Programmers, Prentice Hall, 2003.

[Henderson 2003] Henderson, K. The Guru's Guide to Transact-SQL, Addison-Wesley Press, 2003.

[Jones 2002] Jones, R.A. Mastering ASP .NET with C#. Sybex Publishers 2002.

[MSDN 2004] ADO Data Access.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbmscVisualBasicDocumentationMap.asp (Visited Mar 19, 2006).

[Onion 2005] Onion, F. Essential ASP .NET with examples in C#, Addison-Wesley Press, 2005.

[Patton 2001] Patton, R. Software Testing, Sams Publishing 2001.

[Petkovic 2000] Petkovic, D. SQL Server 2000: A Beginners Guide, Osborne Publishing 2000.

[Pressman 2000] Roger S. Pressman. Software Engineering: A Practitioner's Approach, McGraw Hill, 2000.

[Teltone 2004] Telephone Access Unit.
http://www.teltone.com/downloads/flyer_t311.pdf (Visited Mar 21, 2006).

[Thomsen 2002] Thomsen, C. Database Programming with C#, Apress Publishing, 2002.

APPENDIX A – Data Import File Template

@@dateofcall

@@salutation
@@address
@@city

@@dear

I enjoyed speaking with you regarding the exciting changes at the Island University. I especially want to thank you for joining the Texas A&M-Corpus Christi Parents Council. Your support will provide additional resources that enhance student career development and learning, mentoring and counseling, and provide for the expansion of library resources and technology updates.

The Parents Council is one of the programs instrumental in bridging the gap between a good and excellent University. Your commitment to the University demonstrates that you believe in a quality education.

On behalf of the University and President Killebrew, thank you for joining the Parents Council. We really appreciate your gift. Please visit our website at
http://kanga.tamucc.edu/development/parent/parent.htm.
Have a great year!

Sincerely,

Celeste Jennings

Celeste Jennings, Development Officer, Annual Funds

Date of Pledge	@@pledgedate	Pledge Amount	@@pledgeamt
Matching Gift Amount	@@matchamt	Total Amount	@@totalamt
Installment Amount	@@installamt	Payment Number	1
Payment Method	@@paymethod	Schedule	@@schedule
Matching Gift Co.	@@matchco	Matching Gift Ratio	@@matchratio

Enclosed is my check for ______________________
I understand my gift will be designated to @@designation
Unless I have indicated otherwise ______________________

Membership Categories
@@Level5
@@Level4
@@Level3
@@Level2
@@Level1

@@salutation
@@address
@@city

@@identification

APPENDIX B – Thank You Letter Template

The data in Microsoft Excel should be in the following format. The following fields must be in the in the same order and must be the first line of the Excel sheet

1) cosnId
2) ssn
3) Prefix
4)Firstname
5) lastname
6) gender
7) address1
8) address2
9) address3
10) city
11) state
12) zipcode
13) telephone
14) Email
15) birthdate
16) spousessn
17) spouseprefix
18) spousemiddlename
19) spouselastname
20) spousegender
21) spouseEmail
22) yr1
23) Yr2
24) school1
25) school2
26) campus1
27) campus2
28) degree1
29) degree2
30) major1_1
31) major2_1
32) spyear1
33) spyear2
34) spschool1
35) spschool2
36) spmajor1
37) spmajor2
38) employername
39) jobtitle
40) employeraddress1
41) employeraddress2
42) employeraddress3
43) employercity
44) employerstate
45) employerzip
46) matchingcompany
47) matchingratio
48) spemployer
49) spousejobtitle
50) spousematchingcompany
51) matchingratio2
52) giftlstamt
53) giftlstdate
54) giftlstreference
55) childfirstname
56) childlastname
57) childssn

APPENDIX C – Database SQL File

Table Creation Scripts

```
CREATE TABLE [dbo].[Assignments] (
	[ID] [int] IDENTITY (1, 1) NOT NULL ,
	[CallerID] [smallint] NOT NULL ,
	[GroupID] [tinyint] NOT NULL ,
	[StartTime] [smalldatetime] NOT NULL ,
	[StopTime] [smalldatetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[CallBacks] (
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[StartTimeblock] [smalldatetime] NOT NULL ,
	[EndTimeblock] [smalldatetime] NULL ,
	[LastCallID] [int] NOT NULL ,
	[IsPersonal] [bit] NOT NULL ,
	[InUse] [bit] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Callers] (
	[CallerID] [smallint] IDENTITY (1, 1) NOT NULL ,
	[UserNTLM] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[FirstName] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[LastName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[IsManager] [bit] NOT NULL ,
	[IsActive] [bit] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[CallingPool] (
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[LastCallID] [int] NOT NULL ,
	[InUse] [bit] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[CallsMade] (
	[CallID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS 
NOT NULL ,
	[CallerID] [smallint] NOT NULL ,
	[CallStart] [smalldatetime] NOT NULL ,
	[CallEnd] [smalldatetime] NOT NULL ,
	[Result] [tinyint] NOT NULL 
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Children] (
	[ChildID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS 
NOT NULL ,
	[FirstName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL 
,
	[LastName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL 
,
	[SSN] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NULL 
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Comments] (
	[ID] [int] IDENTITY (1, 1) NOT NULL ,
	[CallID] [int] NOT NULL ,
	[Comment] [varchar] (240) COLLATE SQL_Latin1_General_CP1_CI_AS NOT 
NULL 
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[CreditCardInfo] (
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS 
NOT NULL ,
	[CardType] [tinyint] NOT NULL ,
	[CardNumber] [char] (16) COLLATE SQL_Latin1_General_CP1_CI_AS NOT 
NULL ,
	[ExpirationDate] [smalldatetime] NOT NULL ,
	[NameOnCard] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS 
NOT NULL 
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[CreditCardTypes] (
	[ID] [tinyint] IDENTITY (1, 1) NOT NULL ,
	[Name] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NOT 
NULL 
) ON [PRIMARY]
```

```
GO

CREATE TABLE [dbo].[Demographic] (
	[DemographicID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[SSN] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Prefix] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[FirstName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
	[MiddleName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
	[LastName] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
	[Gender] [char] (6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address1] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address2] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address3] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[City] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[State] [char] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Zipcode] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Telephone] [char] (16) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Email] [varchar] (60) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[BirthDate] [smalldatetime] NULL ,
	[IsSpouse] [bit] NOT NULL ,
	[Updated] [smalldatetime] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Education] (
	[ID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[Year] [int] NULL ,
	[SchoolName] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
	[CollegeName] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
	[Degree] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Major] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IsSpouse] [bit] NOT NULL ,
	[Updated] [smalldatetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Employment] (
	[EmploymentID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[Employer] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
	[JobTitle] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address1] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address2] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Address3] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[City] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[State] [char] (3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Zipcode] [char] (11) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Telephone] [char] (16) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Matches] [bit] NULL ,
	[Ratio] [float] NULL ,
	[IsSpouse] [bit] NOT NULL ,
	[Updated] [smalldatetime] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Gifts] (
	[GiftID] [int] IDENTITY (1, 1) NOT NULL ,
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[GiftDate] [smalldatetime] NULL ,
	[Amount] [money] NULL ,
	[Reference] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[GroupTypes] (
	[GroupTypeID] [int] IDENTITY (1, 1) NOT NULL ,
	[GroupType] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Groups] (
	[GroupID] [tinyint] IDENTITY (1, 1) NOT NULL ,
	[GroupName] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[DefaultDesignation] [varchar] (50) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
	[ScriptURL] [varchar] (80) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
	[GroupTypeID] [int] NOT NULL
```

```
) ON [PRIMARY]
GO


CREATE TABLE [dbo].[LastCall] (
	[ConsID] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
	[CallID] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[MatchingGift] (
	[ID] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[ParentID] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Name] [varchar] (80) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Alias] [varchar] (80) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Parent] [varchar] (80) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Fund] [varchar] (80) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Minimum] [money] NULL ,
	[Maximum] [money] NULL ,
	[Ratio] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[MatchesSpouse] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL ,
	[MatchesRetired] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[PaymentTypes] (
	[ID] [tinyint] IDENTITY (1, 1) NOT NULL ,
	[Name] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Pledges] (
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[Amount] [money] NOT NULL ,
	[Schedule] [tinyint] NOT NULL ,
	[Installments] [tinyint] NOT NULL ,
	[PayMethod] [tinyint] NOT NULL ,
	[DueDate] [smalldatetime] NOT NULL ,
	[Designation] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[IsJointGift] [bit] NOT NULL
```

```
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Prospects] (
	[ConstituentID] [varchar] (11) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL ,
	[GroupID] [tinyint] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Responses] (
	[ResponseID] [tinyint] IDENTITY (1, 1) NOT NULL ,
	[Name] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL ,
	[IsComplete] [bit] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[ScheduleTypes] (
	[ID] [tinyint] IDENTITY (1, 1) NOT NULL ,
	[Name] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL
) ON [PRIMARY]
GO
```

Stored Procedures

```
CREATE PROCEDURE AddAssignment (@CallerID smallint,
                                @GroupID tinyint,
                                @StartTime smalldatetime,
                                @StopTime smalldatetime) AS
INSERT INTO Assignments
(CallerID, GroupID, StartTime, StopTime)
VALUES
(@CallerID, @GroupID, @StartTime, @StopTIme)


CREATE PROCEDURE AddCall (@ConstituentID varchar (11),
                          @CallerID smallint,
                          @CallStart smalldatetime,
                          @CallEnd smalldatetime,
                          @Result tinyint) AS
INSERT INTO CallsMade
(ConstituentID, CallerID, CallStart, CallEnd, Result)
VALUES
(@ConstituentID, @CallerID, @CallStart, @CallEnd, @Result)
CREATE PROCEDURE AddCallBack (@ConstituentID varchar (11),
                              @StartTimeblock smalldatetime,
                              @EndTimeblock smalldatetime,
                              @LastCallID int,
                              @IsPersonal bit) AS
INSERT INTO CallBacks
(ConstituentID, StartTimeblock, EndTimeblock, LastCallID, IsPersonal, InUse)
VALUES
(@ConstituentID, @StartTimeblock, @EndTimeblock, @LastCallID, @IsPersonal, 0)


CREATE PROCEDURE AddCaller (@UserNTLM varchar (20),
                            @FirstName varchar (20),
                            @LastName varchar (25),
                            @IsActive bit,
                            @IsManager bit) AS
INSERT INTO Callers
(UserNTLM, FirstName, LastName, IsActive, IsManager)
VALUES
(@UserNTLM, @FirstName, @LastName, @IsActive, @IsManager)


CREATE PROCEDURE AddChild (@ConstituentID varchar(11),
                           @FirstName varchar(25),
                           @LastName varchar(25),
                                    @SSN char(11))
AS
INSERT INTO Children
(ConstituentID, FirstName, LastName, SSN)
VALUES
(@ConstituentID, @FirstName, @LastName, @SSN)

CREATE PROCEDURE AddComment (@CallID int,
                                          @Comment varchar (240)) AS
INSERT INTO Comments
(CallID, Comment)
VALUES
(@CallID, @Comment)


CREATE PROCEDURE AddCreditCardInfo (@ConstituentID varchar (11),
                                                   @CardType tinyint,
                                              @CardNumber char (16),
                                              @ExpirationDate smalldatetime,
                                              @NameOnCard varchar (50)) AS
INSERT INTO CreditCardInfo
(ConstituentID, CardType, CardNumber, ExpirationDate, NameOnCard)
VALUES
(@ConstituentID, @CardType, @CardNumber, @ExpirationDate, @NameOnCard)


CREATE PROCEDURE AddDegree (@ID int OUT,
                                    @ConstituentID varchar (11),
                                       @Year int,
                                       @Degree varchar (10),
                                       @Major varchar (40),
                                       @SchoolName varchar (40),
                                       @CollegeName varchar (40),
                                       @IsSpouse bit) AS
INSERT INTO Education
(ConstituentID, Year, SchoolName, CollegeName, Degree, Major, IsSpouse, Updated)
VALUES
(@ConstituentID, @Year, @SchoolName, @CollegeName, @Degree, @Major,
@IsSpouse, GetDate())
SET @ID = @@IDENTITY


CREATE PROCEDURE AddDemographic (@DemographicID int OUT,
                                          @ConstituentID varchar(11),
                                              @SSN char (11),
                                              @Prefix varchar(15),
                                              @FirstName varchar(25),
                                              @MiddleName varchar(25),
                                              @LastName varchar(25),
                                              @Gender char(6),
                                              @Address1 varchar(40),
                                              @Address2 varchar(40),
                                              @Address3 varchar(40),
                                              @City varchar(40),
                                              @State char(3),
                                              @Zipcode char(11),
                                              @Telephone char(16),
                                              @Email varchar(60),
                                              @Birthdate smalldatetime=NULL,
                                              @IsSpouse bit) AS
INSERT INTO Demographic
(ConstituentID, SSN, Prefix, FirstName, MiddleName, LastName, Gender, Address1,
Address2, Address3, City, State, Zipcode, Telephone, Email, Birthdate, IsSpouse,
Updated)
VALUES
(@ConstituentID, @SSN, @Prefix, @FirstName, @MiddleName, @LastName,
@Gender, @Address1, @Address2, @Address3, @City, @State, @Zipcode,
@Telephone, @Email, @Birthdate, @IsSpouse, GetDate())
SET @DemographicID = @@IDENTITY


CREATE PROCEDURE AddEmployment (@EmploymentID int OUT,
                                    @ConstituentID varchar(11),
                                       @Employer varchar (40),
                                       @JobTitle varchar(40),
                                       @Address1 varchar(40),
                                       @Address2 varchar(40),
                                       @Address3 varchar(40),
                                       @City varchar(40),
                                       @State char(3),
                                       @Zipcode char(11),
                                       @Telephone char(16),
                                       @IsSpouse bit) AS
INSERT INTO Employment
(ConstituentID, Employer, JobTitle,  Address1, Address2, Address3, City, State,
Zipcode, Telephone, IsSpouse, Updated)
VALUES
(@ConstituentID, @Employer, @JobTitle, @Address1, @Address2, @Address3, @City,
@State, @Zipcode, @Telephone, @IsSpouse, GetDate())
SET @EmploymentID = @@IDENTITY
```

```
CREATE PROCEDURE AddGift (@GiftID int OUT,
                          @ConstituentID varchar (11),
                            @GiftDate smalldatetime,
                            @Amount money)
AS
INSERT INTO Gifts
(ConstituentID, GiftDate, Amount)
VALUES
(@ConstituentID, @GiftDate, @Amount)
SET @GiftID = @@IDENTITY


CREATE PROCEDURE AddGroup (@GroupName varchar(30),
                            @DefaultDesignation varchar(50),
                            @ScriptURL varchar(80))
AS
INSERT INTO Groups
(GroupName, DefaultDesignation, ScriptURL)
VALUES
(@GroupName, @DefaultDesignation, @ScriptURL)


CREATE PROCEDURE AddPledge (@ConstituentID varchar (11),
                             @Amount money,
                             @Schedule tinyint,
                             @Installments tinyint,
                             @PayMethod tinyint,
                             @DueDate smalldatetime,
                             @Designation varchar (50),
                             @IsJointGift bit) AS
INSERT INTO Pledges
(ConstituentID, Amount, Schedule, Installments, PayMethod, DueDate, Designation,
IsJointGift)
VALUES
(@ConstituentID, @Amount, @Schedule, @Installments, @PayMethod, @DueDate,
@Designation, @IsJointGift)


CREATE PROCEDURE AddProspect(@ConstituentID varchar(11), @GroupID tinyint)
AS
INSERT INTO Prospects
(ConstituentID, GroupID)
VALUES
(@ConstituentID, @GroupID)
INSERT INTO CallingPool
(ConstituentID, LastCallID, InUse)
```

```
VALUES
(@ConstituentID, 0, 0)


CREATE PROCEDURE AddScript (@ScriptName varchar(40)) AS

DECLARE @ScriptID smallint, @ScreenID int

INSERT INTO Scripts
(ScriptName, ScreenZero)
VALUES
(@ScriptName, 0)

SET @ScriptID = @@IDENTITY

INSERT INTO Screens
(ScriptID)
VALUES
(@ScriptID)

SET @ScreenID = @@IDENTITY

UPDATE Scripts
SET ScreenZero = @ScreenID WHERE ScriptID = @ScriptID


CREATE PROCEDURE DeleteCall (@CallID int) AS
DECLARE
@Result tinyint, @Complete bit, @ConsID varchar(11), @LastCallID int
BEGIN TRANSACTION WipeCall
SELECT @ConsID = ConstituentID FROM CallsMade WHERE CallID = @CallID
IF @@ERROR <> 0
        GOTO ERRORHANDLER
SELECT @Result = Result FROM CallsMade WHERE CallID = @CallID
IF @@ERROR <> 0
        GOTO ERRORHANDLER
SELECT @Complete = IsComplete FROM Responses WHERE ResponseID = @Result
IF @@ERROR <> 0
        GOTO ERRORHANDLER
DELETE CallBacks WHERE ConstituentID = @ConsID
IF @@ERROR <> 0
        GOTO ERRORHANDLER
DELETE CallingPool WHERE ConstituentID = @ConsID
IF @@ERROR <> 0
        GOTO ERRORHANDLER
DELETE Comments WHERE CallID = @CallID
```

```
IF @@ERROR <> 0
        GOTO ERRORHANDLER
DELETE CallsMade WHERE CallID = @CallID
IF @@ERROR <> 0
        GOTO ERRORHANDLER
IF @Complete = 1
        BEGIN
                SELECT @LastCallID = CallID FROM CallsMade WHERE
ConstituentID = @ConsID
                IF @@ERROR <> 0
                        GOTO ERRORHANDLER
                IF @LastCallID = NULL
                        SET @LastCallID = 0
                INSERT INTO CallingPool
                (ConstituentID, LastCallID, InUse)
                VALUES
                (@ConsID, @LastCallID, 0)
                IF @@ERROR <> 0
                        GOTO ERRORHANDLER
                IF @Result = 1
                        BEGIN
                                DELETE Pledges WHERE ConstituentID = @ConsID
                                IF @@ERROR <> 0
                                        GOTO ERRORHANDLER
                        END
        END
COMMIT TRANSACTION WipeCalls
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION WipeCalls
RETURN @@ERROR


CREATE PROCEDURE DeleteCaller (@CallerID smallint) AS
BEGIN TRANSACTION CallerDelete
DELETE Assignments WHERE CallerID = @CallerID
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE CallsMade
SET CallerID = 0
WHERE CallerID = @CallerID
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE Callers
WHERE CallerID = @CallerID
IF @@ERROR <> 0
```

```
  GOTO ErrorHandler
COMMIT TRANSACTION CallerDelete
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION CallerDelete
RETURN @@ERROR

CREATE PROCEDURE DeleteCallingAssignments (@CallerID smallint) AS
DELETE Assignments WHERE CallerID = @CallerID

CREATE PROCEDURE DeleteFromCallBacks (@ConstituentID varchar (11)) AS
DELETE CallBacks WHERE ConstituentID = @ConstituentID

CREATE PROCEDURE DeleteFromCallingPool (@ConstituentID varchar (11)) AS
DELETE CallingPool WHERE ConstituentID = @ConstituentID


CREATE PROCEDURE DeleteGroup (@GroupID tinyint) AS
BEGIN TRANSACTION GroupDelete
DELETE FROM CallBacks WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM CallingPool WHERE ConstituentID IN (SELECT ConstituentID
FROM Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Children WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM CreditCardInfo WHERE ConstituentID IN (SELECT ConstituentID
FROM Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Demographic WHERE ConstituentID IN (SELECT ConstituentID
FROM Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Education WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Employment WHERE ConstituentID IN (SELECT ConstituentID
FROM Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
```

```
  GOTO ErrorHandler
DELETE FROM Gifts WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM LastCall WHERE ConsID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Pledges WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Comments WHERE CallID IN (SELECT CallID FROM CallsMade
WHERE ConstituentID IN (SELECT ConstituentID FROM Prospects WHERE GroupID
= @GroupID))
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM CallsMade WHERE ConstituentID IN (SELECT ConstituentID FROM
Prospects WHERE GroupID = @GroupID)
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Assignments WHERE GroupID = @GroupID
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Prospects WHERE GroupID = @GroupID
IF @@ERROR <> 0
  GOTO ErrorHandler
DELETE FROM Groups WHERE GroupID = @GroupID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION GroupDelete
RETURN 0
ErrorHandler:
  ROLLBACK TRANSACTION GroupDelete
  RETURN @@ERROR


CREATE PROCEDURE GetAssignmentsByCallerID (@CallerID smallint)  AS
SELECT FirstName, LastName, GroupID, GroupName, StartTime, StopTime
FROM vw_CallingAssignments
WHERE CallerID = @CallerID


CREATE PROCEDURE GetCall (@CallID int) AS
SELECT ConstituentID, UserNTLM, CallStart, CallEnd, ResultName
```

```
FROM vw_CallDetail
WHERE CallID = @CallID

CREATE PROCEDURE GetCallFromCallingPool (@CallerID smallint,
                                         @GroupID tinyint,
                                         @CallStart smalldatetime,
                                         @CallCheck smalldatetime)
AS
DECLARE @ConsID varchar (11)
BEGIN TRANSACTION GetCall
SET @ConsID = (SELECT TOP 1 ConstituentID FROM vw_CallingPoolDetail
WHERE InUse = 0 AND GroupID = @GroupID AND  (LastCallID = 0 OR CallEnd <
@CallCheck)
ORDER BY CallEnd ASC)
IF @@ERROR <> 0
GOTO ErrorHandler
UPDATE CallingPool
SET InUse = 1
WHERE ConstituentID = @ConsID
IF @@ERROR <> 0
GOTO ErrorHandler
SELECT @ConsID
IF @@ERROR <> 0
GOTO ErrorHandler
COMMIT TRANSACTION GetCall
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION GetCall
RETURN @@ERROR


CREATE PROCEDURE GetCallStatsByGroupType(@GroupTypeID tinyint)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int,
@WrongNumber int, @Spanish int, @MailInfo int,
@hangup int,  @Solicitation int, @Callback int, @Nocalls int,
@NoContact int, @Divorced int, @Deceased int, @AlreadyGave int
SELECT @Total = Count(ConstituentID) FROM vw_CallDetail
WHERE GroupTypeID = @GroupTypeID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupTypeID = @GroupTypeID
SELECT @NotReached = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=2 AND GroupTypeID = @GroupTypeID
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=3 AND GroupTypeID = @GroupTypeID
SELECT @Spanish= Count(ConstituentID) FROM vw_CallDetail
```

```
WHERE Result=4 AND GroupTypeID = @GroupTypeID
SELECT @WrongNumber = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=5 AND GroupTypeID = @GroupTypeID
SELECT @Deceased= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=6 AND GroupTypeID = @GroupTypeID
SELECT @Divorced = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=7 AND GroupTypeID = @GroupTypeID
SELECT @MailInfo= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=8 AND GroupTypeID = @GroupTypeID
SELECT @NoCalls = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=9 AND GroupTypeID = @GroupTypeID
SELECT @NoContact = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=10 AND GroupTypeID = @GroupTypeID
SELECT @Solicitation= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=11 AND GroupTypeID = @GroupTypeID
SELECT @CallBack= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=12 AND GroupTypeID = @GroupTypeID
SELECT @HangUp = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=13 AND GroupTypeID = @GroupTypeID
SELECT @AlreadyGave = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=14 AND GroupTypeID = @GroupTypeID

CREATE PROCEDURE GetCallerByID (@CallerID smallint) AS
SELECT CallerID, UserNTLM, FirstName, LastName, IsManager, IsActive
FROM Callers
WHERE CallerID = @CallerID


CREATE PROCEDURE GetCallerDetailedCallStats(@CallerID smallint,
                                            @StartDate smalldatetime,
                                            @EndDate smalldatetime)  AS

SELECT ConstituentID, GroupName, CallStart, CallEnd, ResultName FROM
vw_CallDetail WHERE CallerID = @CallerID AND CallStart BETWEEN @StartDate
AND @EndDate
ORDER BY CallStart ASC


CREATE PROCEDURE GetCallerList AS
SELECT CallerID, UserNTLM, FirstName, LastName, IsActive, IsManager
FROM Callers

CREATE PROCEDURE GetCallerNightlyCallStats(@UserNTLM varchar(20),
                                           @StartDate smalldatetime,
                                           @EndDate smalldatetime)  AS
DECLARE
```

```
@Total int, @Pledge int, @NotReached int, @Hangup int, @Refusal int, @Solicitation
int, @Callback int, @Other int, @CallerID smallint
SET @CallerID = (SELECT CallerID FROM Callers WHERE UserNTLM =
@UserNTLM)
CREATE TABLE #TempTable
(
        CallID int,
        Result tinyint
)
INSERT INTO #TempTable
SELECT CallID, Result FROM CallsMade WHERE CallerID = @CallerID AND
CallStart BETWEEN @StartDate AND @EndDate
SELECT @Total = Count(*) FROM #TempTable
SELECT @Pledge = Count(*) FROM #TempTable WHERE Result=1
SELECT @NotReached = Count(*) FROM #TempTable WHERE Result=2
SELECT @Refusal = Count(*) FROM  #TempTable
WHERE Result=3
SELECT @Solicitation= Count(*) FROM  #TempTable
WHERE (Result=9 OR Result=10 OR Result=11)
SELECT @Callback= Count(*) FROM  #TempTable
WHERE Result=12
SELECT @Hangup= Count(*) FROM #TempTable
WHERE Result=13
SELECT @Other= Count(*) FROM #TempTable
WHERE (Result=4 OR Result=5 OR Result=6 OR Result=7 OR Result=8)
DROP TABLE #TempTable
SELECT  @Total AS Total, @Pledge AS Pledge, @NotReached AS NotReached,
@Refusal AS Refusal, @Solicitation AS Solicitation,
  @Callback AS Callback, @Other AS Other, @Hangup AS Hangup


CREATE PROCEDURE GetCallerPledgeStats (@CallerID smallint)  AS
SELECT Count(CallerID) AS PledgeCount, SUM(Amount) AS PledgeSum
FROM vw_PledgesByCaller
WHERE CallerID = @CallerID
ORDER BY PledgeCount


CREATE PROCEDURE GetCallerPledgeSummary (@CallerID smallint) AS
BEGIN TRANSACTION GetPledgesByCaller
CREATE TABLE #TempData
(
  ConstituentID varchar(11),
  Amount smallmoney,
  Matches bit,
  Ratio float
)
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData
SELECT ConstituentID, NULL, NULL, NULL
FROM CallsMade
WHERE CallsMade.CallerID = @CallerID AND Result = 1
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData
SET Amount = (SELECT Amount FROM Pledges
WHERE Pledges.ConstituentID = #TempData.ConstituentID)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData
SET Matches = (SELECT TOP 1 Matches FROM Employment
WHERE Employment.ConstituentID = #TempData.ConstituentID AND IsSpouse = 0
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData
SET Ratio = (SELECT TOP 1 Ratio FROM Employment
WHERE Employment.ConstituentID = #TempData.ConstituentID AND IsSpouse = 0
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
SELECT * FROM #TempData
IF @@ERROR <> 0
  GOTO ErrorHandler
DROP TABLE #TempData
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION GetPledgesByCaller
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION GetPledgesByCaller
RETURN @@ERROR

CREATE PROCEDURE GetCallingAssignments AS
SELECT ID, CallerID, FirstName, LastName, GroupName, StartTime, StopTime,
GroupName
FROM vw_CallingAssignments
WHERE IsActive = 1
ORDER BY CallerID ASC
```

```
CREATE PROCEDURE GetChildren (@ConstituentID varchar (11)) AS
SELECT ChildID, FirstName, LastName, SSN
FROM Children
WHERE ConstituentID = @ConstituentID


CREATE PROCEDURE GetComments (@ConstituentID varchar (11)) AS
SELECT CallStart, FirstName, LastName, Comment
FROM vw_Comments
WHERE ConstituentID = @ConstituentID
ORDER BY CallStart DESC


CREATE PROCEDURE GetCreditCardDetail (@ConstituentID varchar (11)) AS
SELECT CardNumber, ExpirationDate, NameOnCard, Name
FROM vw_CreditCardDetail
WHERE ConstituentID = @ConstituentID

CREATE PROCEDURE GetCurrentCallFromCallingPool (@CallerID smallint) AS
SELECT TOP 1 ConstituentID
FROM CallingPool
WHERE InUse = 1

CREATE PROCEDURE GetDeceasedList (@StartDate smalldatetime,
                                   @EndDate smalldatetime) AS
SELECT ConstituentID, CallStart, CallerFirstName, CallerLastName, GroupName
FROM vw_CallDetail
WHERE CallStart >= @StartDate AND CallStart <= @EndDate AND Result=6
ORDER BY ConstituentID


CREATE PROCEDURE GetDegreeDetail (@DegreeID int) AS
SELECT Year, SchoolName, CollegeName, Degree, Major
FROM Education
WHERE ID = @DegreeID


CREATE PROCEDURE GetDemographicChangeList (@StartDate smalldatetime,
                                           @EndDate smalldatetime) AS
BEGIN TRANSACTION ListDemoChanges
CREATE TABLE #TempData1
(
  ConstituentID varchar(11),
  Updated smalldatetime
)
IF @@ERROR <> 0
```

```
  GOTO ErrorHandler
INSERT INTO #TempData1
SELECT ConstituentID, Updated
FROM Demographic
WHERE Updated >= @StartDate AND Updated <= @EndDate
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData1
SELECT ConstituentID, Updated
FROM Education
WHERE Updated >= @StartDate AND Updated <= @EndDate
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData1
SELECT ConstituentID, Updated
FROM Employment
WHERE Updated >= @StartDate AND Updated <= @EndDate
IF @@ERROR <> 0
  GOTO ErrorHandler
CREATE TABLE #TempData2
(
  ConstituentID varchar(11),
  FirstName varchar(25),
  LastName varchar(25),
  GroupName varchar (30),
  Updated smalldatetime
)
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData2
SELECT ConstituentID, NULL, NULL, NULL, MAX(Updated)
FROM #TempData1
GROUP BY ConstituentID
IF @@ERROR <> 0
  GOTO ErrorHandler
DROP TABLE #TempData1
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET FirstName = (SELECT TOP 1 FirstName FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET LastName = (SELECT TOP 1 LastName FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET GroupName = (SELECT GroupName FROM vw_ProspectGroupDetail
WHERE vw_ProspectGroupDetail.ConstituentID = #TempData2.ConstituentID)
IF @@ERROR <> 0
  GOTO ErrorHandler
SELECT * FROM #TempData2
IF @@ERROR <> 0
  GOTO ErrorHandler
DROP TABLE #TempData2
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION ListDemoChanges
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION ListDemoChanges
RETURN @@ERROR


CREATE PROCEDURE GetEmployerName (@ConstituentID varchar (11)) AS
SELECT TOP 1 Employer
FROM Employment
WHERE ConstituentID = @ConstituentID AND IsSpouse = 0
ORDER BY Updated DESC


CREATE PROCEDURE GetEmploymentChanges(@ConstituentID varchar(11),
@IsSpouse bit) AS
SELECT TOP 2 * FROM Employment
WHERE ConstituentID = @ConstituentID AND IsSpouse = @IsSpouse
ORDER BY Updated DESC


CREATE PROCEDURE GetGiftHistory (@ConstituentID varchar (11)) AS
SELECT GiftDate, Amount, Reference
FROM Gifts
WHERE ConstituentID = @ConstituentID
ORDER BY GiftDate DESC


CREATE PROCEDURE GetGroupByID (@GroupID tinyint) AS
SELECT GroupName, DefaultDesignation, ScriptURL
FROM Groups
```

```
WHERE GroupID = @GroupID


CREATE PROCEDURE GetGroupByName (@GroupName varchar(30)) AS
SELECT GroupID, DefaultDesignation, ScriptURL
FROM Groups
WHERE GroupName = @GroupName


CREATE PROCEDURE GetGroupCallStats(@GroupID tinyint)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int, @Solicitation int, @Callback
int, @Other int
SELECT @Total = Count(ConstituentID) FROM vw_CallDetail
WHERE GroupID = @GroupID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupID = @GroupID
SELECT @NotReached = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=2 AND GroupID = @GroupID
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE (Result=3 OR Result=13) AND GroupID = @GroupID
SELECT @Solicitation= Count(ConstituentID) FROM vw_CallDetail
WHERE (Result=9 OR Result=10 OR Result=11) AND GroupID = @GroupID
SELECT @Callback= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=12 AND GroupID = @GroupID
SELECT @Other= Count(ConstituentID) FROM vw_CallDetail
WHERE (Result=4 OR Result=5 OR Result=6 OR Result=7 OR Result=8) AND
GroupID = @GroupID
SELECT  @Total AS Total, @Pledge AS Pledge, @NotReached AS NotReached,
@Refusal AS Refusal, @Solicitation AS Solicitation,
  @Callback AS Callback, @Other AS Other


CREATE PROCEDURE GetGroupCallbackTotals (@GroupID tinyint)  AS
SELECT GroupID, Count (*)  AS NumProspects
FROM vw_CallBackPoolDetail
WHERE GroupID = @GroupID
GROUP BY GroupID


CREATE PROCEDURE GetGroupPoolTotals (@GroupID tinyint)  AS
SELECT GroupID, Count (*)  AS NumProspects
FROM vw_CallingPoolDetail
WHERE GroupID = @GroupID
GROUP BY GroupID
```

```
CREATE PROCEDURE GetGroupProspectTotals (@GroupID tinyint)  AS
SELECT GroupID, Count (*)  AS NumProspects
FROM Prospects
WHERE GroupID = @GroupID
GROUP BY GroupID


CREATE PROCEDURE GetGroupResponses(@GroupID tinyint, @EndDate
smalldatetime)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int, @SpanishOnly int,
@WrongNumber int, @Deceased int,
@Divorced int, @MailInfo int, @NoCalls int, @NoContact int, @NoSolicit int,
@Callback int, @NotAttempted int
CREATE TABLE #Table1
(
	ConstituentID varchar(11),
	Result tinyint,
	CallStart smalldatetime
)
INSERT INTO #Table1
SELECT v1.ConstituentID, v1.Result, v1.CallStart FROM vw_CallDetail v1
WHERE v1.GroupID = @GroupID AND v1.CallStart = (SELECT MAX(v2.CallStart)
FROM vw_CallDetail v2
WHERE v2.ConstituentID = v1.ConstituentID)
DELETE #Table1 WHERE CallStart > @EndDate
SELECT @Total = Count(ConstituentID) FROM Prospects
WHERE GroupID = @GroupID
SELECT @Pledge = Count(ConstituentID) FROM #Table1
WHERE Result=1
SELECT @NotReached = Count(ConstituentID) FROM #Table1
WHERE Result=2
SELECT @Refusal = Count(ConstituentID) FROM #Table1
WHERE (Result=3 OR Result=13)
SELECT @SpanishOnly = Count(ConstituentID) FROM #Table1
WHERE Result=4
SELECT @WrongNumber = Count(ConstituentID) FROM #Table1
WHERE Result=5
SELECT @Deceased = Count(ConstituentID) FROM #Table1
WHERE Result=6
SELECT @Divorced = Count(ConstituentID) FROM #Table1
WHERE Result=7
SELECT @MailInfo = Count(ConstituentID) FROM #Table1
WHERE Result=8
SELECT @NoCalls= Count(ConstituentID) FROM #Table1
```

```
WHERE Result=9
SELECT @NoContact = Count(ConstituentID) FROM #Table1
WHERE Result=10
SELECT @NoSolicit = Count(ConstituentID) FROM #Table1
WHERE Result=11
SELECT @Callback= Count(ConstituentID) FROM #Table1
WHERE Result = 12
SELECT @NotAttempted = @Total -  Count(ConstituentID) FROM #Table1
SELECT  @Total AS Total, @Pledge AS Pledge, @NotReached AS NotReached,
@Refusal AS Refusal,
        @SpanishOnly AS SpanishOnly, @WrongNumber AS WrongNumber,
@Deceased AS Deceased,
        @Divorced AS Divorced, @MailInfo AS MailInfo, @NoCalls AS NoCalls,
@NoContact AS NoContact,
        @NoSolicit AS NoSolicit, @CallBack AS CallBack, @NotAttempted AS
NotAttempted


CREATE PROCEDURE GetGroupResponses2(@GroupID tinyint)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int, @SpanishOnly int,
@WrongNumber int, @Deceased int,
@Divorced int, @MailInfo int, @NoCalls int, @NoContact int, @NoSolicit int,
@Callback int, @NotAttempted int
SELECT @Total = Count(ConstituentID) FROM Prospects
WHERE GroupID = @GroupID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupID = @GroupID
SELECT @NotReached = Count(ConstituentID) FROM Prospects
WHERE ConstituentID IN (SELECT ConstituentID FROM vw_CallDetail WHERE
Result=2 AND GroupID=@GroupID)
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE (Result=3 OR Result=13) AND GroupID = @GroupID
SELECT @SpanishOnly = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=4 AND GroupID = @GroupID
SELECT @WrongNumber = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=5 AND GroupID = @GroupID
SELECT @Deceased = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=6 AND GroupID = @GroupID
SELECT @Divorced = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=7 AND GroupID = @GroupID
SELECT @MailInfo = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=8 AND GroupID = @GroupID
SELECT @NoCalls= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=9  AND GroupID = @GroupID
SELECT @NoContact = Count(ConstituentID) FROM vw_CallDetail
```

```
WHERE Result=10 AND GroupID = @GroupID
SELECT @NoSolicit = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=11 AND GroupID = @GroupID
SELECT @Callback= Count(ConstituentID) FROM CallBacks
WHERE ConstituentID IN (SELECT ConstituentID FROM Prospects WHERE GroupID
= @GroupID)
SELECT @NotAttempted = Count(ConstituentID) FROM Prospects
WHERE GroupID=@GroupID AND ConstituentID NOT IN
        (SELECT ConstituentID FROM vw_CallDetail WHERE GroupID=@GroupID)
SELECT  @Total AS Total, @Pledge AS Pledge, @NotReached AS NotReached,
@Refusal AS Refusal,
        @SpanishOnly AS SpanishOnly, @WrongNumber AS WrongNumber,
@Deceased AS Deceased,
        @Divorced AS Divorced, @MailInfo AS MailInfo, @NoCalls AS NoCalls,
@NoContact AS NoContact,
        @NoSolicit AS NoSolicit, @CallBack AS CallBack, @NotAttempted AS
NotAttempted

CREATE PROCEDURE GetGroupTypeCallStats(@GroupTypeID int)  AS

DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int,
@WrongNumber int, @Spanish int, @MailInfo int,
@hangup int,  @Solicitation int, @Callback int,
@Nocalls int, @NoContact int, @Divorced int,
@Deceased int, @AlreadyGave int

SELECT @Total = Count(ConstituentID) FROM vw_CallDetail
WHERE GroupTypeID = @GroupTypeID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupTypeID = @GroupTypeID
SELECT @NotReached = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=2 AND GroupTypeID = @GroupTypeID
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=3 AND GroupTypeID = @GroupTypeID
SELECT @Spanish= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=4 AND GroupTypeID = @GroupTypeID
SELECT @WrongNumber = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=5 AND GroupTypeID = @GroupTypeID
SELECT @Deceased= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=6 AND GroupTypeID = @GroupTypeID
SELECT @Divorced = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=7 AND GroupTypeID = @GroupTypeID
SELECT @MailInfo= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=8 AND GroupTypeID = @GroupTypeID
SELECT @NoCalls = Count(ConstituentID) FROM vw_CallDetail
```

```
WHERE Result=9 AND GroupTypeID = @GroupTypeID
SELECT @NoContact = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=10 AND GroupTypeID = @GroupTypeID
SELECT @Solicitation= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=11 AND GroupTypeID = @GroupTypeID
SELECT @CallBack= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=12 AND GroupTypeID = @GroupTypeID
SELECT @HangUp = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=13 AND GroupTypeID = @GroupTypeID
SELECT @AlreadyGave = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=14 AND GroupTypeID = @GroupTypeID

SELECT  @Total AS Total, @Pledge AS Pledge,
@NotReached AS NotReached, @Refusal AS Refusal,
@Solicitation AS Solicitation,
 @Callback AS Callback,
@NoContact AS NoContact,
@NoCalls AS NoCalls,
@AlreadyGave AS AlreadyGave,
@Divorced AS Divorced,
@Deceased AS Deceased,
@hangup AS Hangup,
@WrongNumber as WrongNumber,
@Spanish AS Spanish,
@MailInfo AS MailInfo

CREATE PROCEDURE GetGroupTypeCallbackTotals (@GroupTypeID tinyint)  AS
SELECT Count (*)  AS NumProspects
FROM vw_CallBackPoolDetail
WHERE GroupID in (SELECT GroupID from Groups where GroupTypeID =
@GroupTypeID)


CREATE PROCEDURE GetGroupTypePoolTotals (@GroupTypeID tinyint)  AS
SELECT Count (*)  AS NumProspects
FROM vw_CallingPoolDetail
WHERE GroupID in (SELECT GroupID from Groups WHERE GroupTypeID =
@GroupTypeID)


CREATE PROCEDURE GetGroupTypeProspectTotals (@GroupTypeID tinyint)  AS
SELECT Count (*)  AS NumProspects
FROM Prospects
WHERE GroupID In (SELECT GroupID from Groups where GroupTypeID =
@GroupTypeID)
```

```
CREATE PROCEDURE GetGroupTypeRefusalCount (@GroupTypeID tinyint)
 AS
BEGIN
Select count(*) as RefusalCount from CallsMade C,Prospects P
where P.ConstituentID = C.ConstituentID and P.GroupID in( SELECT GroupID from
Groups where GroupTypeID = @GroupTypeID) and C.Result in (3,9,11,13)
END


CREATE PROCEDURE GetLastCallFromCallbacks (@ConstituentID varchar (11)) AS
SELECT LastCallID FROM Callbacks
WHERE ConstituentID = @ConstituentID AND InUse = 1


CREATE PROCEDURE GetLastCallFromCallingPool (@ConstituentID varchar (11))
AS
SELECT LastCallID FROM CallingPool
WHERE ConstituentID = @ConstituentID AND InUse = 1

CREATE PROCEDURE GetLastCallMade (@ConstituentID varchar (11)) AS
SELECT TOP 1 CallID FROM CallsMade
WHERE ConstituentID = @ConstituentID
ORDER BY CallEnd DESC


CREATE PROCEDURE GetMailInfoByGroup (@GroupName varchar(30), @StartDate
smalldatetime, @EndDate smalldatetime) AS
SELECT ConstituentID, MailInfoDate AS MailDate, CallerFirstName, CallerLastName,
GroupName
FROM vw_MailInfoReport
WHERE GroupName = @GroupName AND @StartDate <= MailInfoDate AND
@EndDate >= MailInfoDate


CREATE PROCEDURE GetMatchingRatio (@CompanyID varchar (15)) AS
SELECT Ratio
FROM MatchingGift
WHERE ID = @CompanyID


CREATE PROCEDURE GetNightlyCallStats(@StartDate smalldatetime,
                                        @EndDate smalldatetime)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Hangup int, @Refusal int, @Solicitation
int, @Callback int, @Other int
```

```
CREATE TABLE #TempTable
(
        CallID int,
        Result tinyint
)
INSERT INTO #TempTable
SELECT CallID, Result FROM CallsMade WHERE  CallStart BETWEEN @StartDate
AND @EndDate
SELECT @Total = Count(CallID) FROM #TempTable
SELECT @Pledge = Count(CallID) FROM #TempTable WHERE Result=1
SELECT @NotReached = Count(CallID) FROM #TempTable WHERE Result=2
SELECT @Refusal = Count(CallID) FROM  #TempTable
WHERE Result=3
SELECT @Solicitation= Count(CallID) FROM  #TempTable
WHERE (Result=9 OR Result=10 OR Result=11)
SELECT @Callback= Count(CallID) FROM  #TempTable
WHERE Result=12
SELECT @Hangup= Count(CallID) FROM #TempTable
WHERE Result=13
SELECT @Other= Count(CallID) FROM #TempTable
WHERE (Result=4 OR Result=5 OR Result=6 OR Result=7 OR Result=8)
DROP TABLE #TempTable
SELECT  @Total AS Total, @Pledge AS Pledge, @NotReached AS NotReached,
@Refusal AS Refusal, @Solicitation AS Solicitation,
  @Callback AS Callback, @Other AS Other, @Hangup AS Hangup


CREATE PROCEDURE GetNightlyPledgeStats(@StartDate smalldatetime,
                                        @EndDate smalldatetime)  AS
SELECT Count(ConstituentID) AS NumPledges, SUM(Amount) AS AmountPledged
FROM vw_PledgeStats
WHERE @StartDate <= CallStart AND @EndDate >= CallStart


CREATE PROCEDURE GetNonpersonalCallbackFromPool (@CallerID smallint,
                                                  @CallStart smalldatetime,
                                                  @CallCheck smalldatetime)
AS
DECLARE
@ConsID varchar (11)
BEGIN TRANSACTION GetCall
SET @ConsID = (SELECT TOP 1 ConstituentID FROM vw_CallBackPoolDetail
WHERE InUse = 0 AND IsPersonal = 0 AND StartTimeblock <= @CallStart AND
EndTimeblock >= @CallStart AND (LastCallID = 0 OR CallEnd < @CallCheck)
ORDER BY CallEnd ASC)
UPDATE CallBacks SET InUse = 1
```

```
WHERE ConstituentID = @ConsID
SELECT @ConsID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION GetCall
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION GetCall
RETURN @@ERROR


CREATE PROCEDURE GetPersonalCallbackFromPool (@CallerID smallint,
                                              @CallStart smalldatetime,
                                              @CallCheck smalldatetime)
AS
DECLARE
@ConsID varchar (11)
BEGIN TRANSACTION GetCall
SET @ConsID = (SELECT TOP 1 ConstituentID FROM vw_CallBackPoolDetail
WHERE InUse = 0 AND IsPersonal = 1 AND CallerID = @CallerID AND
StartTimeblock <= @CallStart AND EndTimeblock >= @CallStart AND (LastCallID =
0 OR CallEnd < @CallCheck)
ORDER BY CallEnd ASC)
UPDATE CallBacks SET InUse = 1
WHERE ConstituentID = @ConsID
SELECT @ConsID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION GetCall
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION GetCall
RETURN @@ERROR


CREATE PROCEDURE GetPledgeCallInfo (@ConstituentID varchar (11)) AS
SELECT GroupName, ConstituentID, FirstName, LastName, CallStart
FROM vw_PledgeCalls
WHERE ConstituentID = @ConstituentID


CREATE PROCEDURE GetPledgeDetail (@ConstituentID varchar(11)) AS
SELECT Designation, Amount, DueDate, PaymentName, ScheduleName, Installments
FROM vw_PledgeDetail
WHERE ConstituentID = @ConstituentID
```

```
CREATE PROCEDURE GetPledgeList (@StartDate smalldatetime,
                                          @EndDate smalldatetime) AS
BEGIN TRANSACTION ListPledges
CREATE TABLE #TempData
(
  ConstituentID varchar(11),
  FirstName varchar(25),
  LastName varchar (25),
  GroupName varchar (30),
  CallerFirstName varchar (20),
  CallerLastName varchar (25),
  CallStart smalldatetime
)
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData
SELECT ConstituentID, NULL, NULL, GroupName, FirstName, LastName, CallStart
FROM vw_PledgeCalls
WHERE CallStart >= @StartDate AND CallStart <= @EndDate
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData
SET FirstName = (SELECT TOP 1 FirstName FROM Demographic
WHERE Demographic.ConstituentID = #TempData.ConstituentID
AND Demographic.IsSpouse = 0 ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData
SET LastName = (SELECT TOP 1 LastName FROM Demographic
WHERE Demographic.ConstituentID = #TempData.ConstituentID
AND Demographic.IsSpouse = 0 ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
SELECT * FROM #TempData
IF @@ERROR <> 0
  GOTO ErrorHandler
DROP TABLE #TempData
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION ListPledges
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION ListPledges
RETURN @@ERROR
```

```
CREATE PROCEDURE GetPledgeStatsByGroup (@GroupID tinyint)  AS
SELECT GroupName, COUNT(GroupID) AS NumPledges, SUM(Amount) AS
AmountPledged
FROM vw_PledgeGroups
WHERE GroupID = @GroupID
GROUP BY GroupName


CREATE PROCEDURE GetPledgeStatsByGroupType (@GroupTypeID tinyint)  AS
SELECT GroupType, COUNT(GroupTypeID) AS NumPledges, SUM(Amount) AS
AmountPledged
FROM vw_PledgeGroupTypes
WHERE GroupTypeID = @GroupTypeID
GROUP BY GroupType

CREATE PROCEDURE GetPledgeSummary AS
SELECT Count(ConstituentID), SUM(Amount)
FROM Pledges

CREATE PROCEDURE GetPledgesByGroup (@GroupName varchar(30), @StartDate
smalldatetime, @EndDate smalldatetime) AS
SELECT ConstituentID, PledgeDate, Amount, DueDate, PaymentName, ScheduleName,
Installments, Designation, Caller, GroupName
FROM vw_PledgeReport
WHERE GroupName = @GroupName AND @StartDate <= PledgeDate AND
@EndDate >= PledgeDate


CREATE PROCEDURE GetProspectCallHistory (@ConstituentID varchar(11)) AS
SELECT CallID, CallStart, ResultName, CallerFirstName, CallerLastName
FROM vw_CallDetail
WHERE ConstituentID = @ConstituentID
ORDER BY CallStart


CREATE PROCEDURE GetProspectDemographic (@ConstituentID varchar (11)) AS
SELECT TOP 1 SSN, Prefix, FirstName, MiddleName, LastName, Gender, Address1,
Address2, Address3, City, State, Zipcode, Telephone, Email, Birthdate, Updated
FROM Demographic
WHERE ConstituentID = @ConstituentID AND IsSpouse = 0
ORDER BY Updated DESC


CREATE PROCEDURE GetProspectEducation (@ConstituentID varchar (11)) AS
SELECT ID, Year, SchoolName, CollegeName, Degree, Major
FROM Education
```

```
WHERE ConstituentID = @ConstituentID AND IsSpouse = 0
ORDER BY Year DESC

CREATE PROCEDURE GetProspectEmployment (@ConstituentID varchar (11)) AS
SELECT TOP 1 Employer, JobTitle, Address1, Address2, Address3, City, State,
Zipcode, Telephone, Matches, Ratio
FROM Employment
WHERE ConstituentID = @ConstituentID AND IsSpouse = 0
ORDER BY Updated DESC

CREATE PROCEDURE GetProspectGroupDetail (@ConstituentID varchar (11))
AS
SELECT GroupID, GroupName, ScriptURL, DefaultDesignation
FROM vw_ProspectGroupDetail
WHERE ConstituentID = @ConstituentID


CREATE PROCEDURE GetRefusalCount (@GroupID tinyint)
 AS
BEGIN
Select count(*) as RefusalCount from CallsMade C,Prospects P
where P.ConstituentID = C.ConstituentID and P.GroupID = @GroupID and C.Result in
(3,9,11,13)
END

CREATE PROCEDURE GetSpouseDemographic (@ConstituentID varchar (11)) AS
SELECT TOP 1 SSN, Prefix, FirstName, MiddleName, LastName, Gender, Address1,
Address2, Address3, City, State, ZipCode, Telephone, Email, Birthdate, Updated
FROM Demographic
WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
ORDER BY Updated DESC

CREATE PROCEDURE GetSpouseEducation (@ConstituentID varchar (11)) AS
SELECT ID, Year, SchoolName, CollegeName, Degree, Major
FROM Education
WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
ORDER BY Year DESC

CREATE PROCEDURE GetSpouseEmployerName (@ConstituentID varchar (11)) AS
SELECT TOP 1 Employer
FROM Employment
WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
ORDER BY Updated DESC

CREATE PROCEDURE GetSpouseEmployment (@ConstituentID varchar (11)) AS
```

```
SELECT TOP 1 Employer, JobTitle, Address1, Address2, Address3, City, State,
Zipcode, Telephone, Matches, Ratio
FROM Employment
WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
ORDER BY Updated DESC

CREATE PROCEDURE GetSpouseName (@ConstituentID varchar (11)) AS
SELECT TOP 1 Prefix, FirstName, MiddleName, LastName
FROM Demographic
WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
ORDER BY Updated DESC


CREATE PROCEDURE [dbo].[GetTodaysCallBacks]
(@GroupId int)
AS SELECT GroupID,Count (*)  AS NumProspects
FROM vw_CallBackPoolDetail
WHERE GroupID = @GroupId AND StartTimeBlock > GETDATE() AND
EndTimeBlock < (DATEADD(Hour,20,GetDate()))
group by GroupId

CREATE PROCEDURE GetValidAssignmentsByCallerID (@CallerID smallint,
                                                @CallingTime smalldatetime)
AS
SELECT GroupID
FROM Assignments
WHERE CallerID = @CallerID AND StartTime <= @CallingTime AND StopTime >=
@CallingTime
ORDER BY GroupID

CREATE PROCEDURE GetWrongNumberList (@StartDate smalldatetime,
                                     @EndDate smalldatetime) AS
SELECT ConstituentID, CallStart, CallerFirstName, CallerLastName, GroupName
FROM vw_CallDetail
WHERE CallStart >= @StartDate AND CallStart <= @EndDate AND Result=5
ORDER BY ConstituentID


CREATE PROCEDURE ListCreditCardTypes AS
SELECT ID, Name
FROM CreditCardTypes
ORDER BY ID


CREATE PROCEDURE ListGroupTypes AS
SELECT GroupTypeID, GroupType
```

```
FROM GroupTypes
ORDER BY GroupTypeID

CREATE PROCEDURE ListGroups AS
SELECT GroupID, GroupName, DefaultDesignation, ScriptURL
FROM Groups
ORDER BY GroupName

CREATE PROCEDURE ListPaymentTypes AS
SELECT ID, Name
FROM PaymentTypes
ORDER BY ID

CREATE PROCEDURE ListResponseTypes AS
SELECT ResponseID, Name, IsComplete
FROM Responses
ORDER BY ResponseID

CREATE PROCEDURE LoadEducation (@ConstituentID varchar (11),
                                @Year int,
                                @SchoolName varchar (40),
                                @CollegeName varchar (40),
                                @Degree varchar (10),
                                @Major varchar (40),
                                @IsSpouse bit) AS
BEGIN TRANSACTION EducationAdd
INSERT INTO Education
(ConstituentID, Year, SchoolName, CollegeName, Degree, Major, IsSpouse, Updated)
VALUES
(@ConstituentID, @Year, @SchoolName, @CollegeName, @Degree, @Major,
@IsSpouse, '1/1/1900')
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION EducationAdd
RETURN @@IDENTITY
ERRORHANDLER:
ROLLBACK TRANSACTION EducationAdd
RETURN @@ERROR

CREATE PROCEDURE LoadEmployment (@ConstituentID varchar (11),
                                 @Employer varchar (40),
                                 @JobTitle varchar (40),
                                 @Address1 varchar (40),
                                 @Address2 varchar (40),
                                 @Address3 varchar (40),
                                 @City varchar (40),
                                                    @State char (3),
                                                    @Zipcode varchar (11),
                                                    @Telephone char (16),
                                                    @Matches bit,
                                                    @Ratio float,
                                                    @IsSpouse bit) AS
BEGIN TRANSACTION EmploymentAdd
INSERT INTO Employment
(ConstituentID, Employer, JobTitle, Address1, Address2, Address3, City, State, Zipcode,
Telephone, Matches, Ratio, IsSpouse, Updated)
VALUES
(@ConstituentID, @Employer, @JobTitle, @Address1, @Address2, @Address3, @City,
@State, @Zipcode, @Telephone, @Matches, @Ratio, @IsSpouse, '1/1/1900')
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION EmploymentAdd
RETURN @@IDENTITY
ERRORHANDLER:
ROLLBACK TRANSACTION EmploymentAdd
RETURN @@ERROR

CREATE PROCEDURE LoadGifts (@ConstituentID varchar (11),
                                          @GiftDate smalldatetime,
                                          @Amount money,
                                          @Reference varchar (25)) AS
BEGIN TRANSACTION GiftsAdd
INSERT INTO Gifts
(ConstituentID, GiftDate, Amount, Reference)
VALUES
(@ConstituentID, @GiftDate, @Amount, @Reference)
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION GiftsAdd
RETURN @@IDENTITY
ERRORHANDLER:
ROLLBACK TRANSACTION GiftsAdd
RETURN @@ERROR

CREATE PROCEDURE LoadProspectDemographic (@ConstituentID varchar (11),
                                                            @SSN char (11),
                                                            @Prefix varchar (15),
                                                            @FirstName varchar (25),
                                                            @MiddleName varchar (25),
                                                            @LastName varchar (25),
                                                            @Gender char (6),
                                                            @Address1 varchar (40),
                                                            @Address2 varchar (40),
                                                            @Address3 varchar (40),
                                                            @City varchar (40),
                                                            @State char (3),
                                                            @Zipcode char (11),
                                                            @Telephone char (16),
                                                            @Email varchar (60),
                                                            @Birthdate smalldatetime,
                                                            @IsSpouse bit) AS
BEGIN TRANSACTION DemographicAdd
INSERT INTO Demographic
(ConstituentID, SSN, Prefix, FirstName, MiddleName,  LastName, Gender, Address1,
Address2, Address3, City, State, Zipcode, Telephone, Email, Birthdate, IsSpouse,
Updated)
VALUES
(@ConstituentID, @SSN, @Prefix, @FirstName, @MiddleName,  @LastName,
@Gender, @Address1, @Address2, @Address3, @City, @State, @Zipcode,
@Telephone, @Email, @Birthdate, @IsSpouse, '1/1/1900')
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION DemographicAdd
RETURN @@IDENTITY
ERRORHANDLER:
ROLLBACK TRANSACTION DemographicAdd
RETURN @@ERROR

CREATE PROCEDURE LoadProspects (@ConstituentID varchar (11),
                                @GroupID tinyint) AS
BEGIN TRANSACTION ProspectsAdd
INSERT INTO Prospects
(ConstituentID, GroupID)
VALUES
(@ConstituentID, @GroupID)
IF @@ERROR <> 0
  GOTO ERRORHANDLER
INSERT INTO CallingPool
(ConstituentID, LastCallID, InUse)
VALUES
(@ConstituentID, 0, 0)
COMMIT TRANSACTION ProspectsAdd
RETURN @@IDENTITY
ERRORHANDLER:
ROLLBACK TRANSACTION ProspectsAdd
RETURN @@ERROR

CREATE PROCEDURE NightlyCleanup AS
```

```
BEGIN TRANSACTION Cleanup
UPDATE CallingPool
SET InUse = 0
IF @@ERROR <> 0 GOTO ErrorHandler
UPDATE CallBacks
SET InUse = 0
IF @@ERROR <> 0 GOTO ErrorHandler
INSERT INTO CallingPool
SELECT ConstituentID, LastCallID, InUse
FROM CallBacks
WHERE EndTimeBlock < GETDATE()
IF @@ERROR <>  0 GOTO ErrorHandler
DELETE FROM CallBacks
WHERE EndTimeBlock < GETDATE()
IF @@ERROR <> 0 GOTO ErrorHandler
COMMIT TRANSACTION Cleanup
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION Cleanup
RETURN @@ERROR

CREATE PROCEDURE RecreateCallbackEntry (@ConstituentID varchar (11),
                                        @LastCallID int)
AS
DECLARE
@StartTimeBlock smalldatetime,
@EndTimeBlock smalldatetime,
@IsPersonal bit
BEGIN TRANSACTION RecreateCall
SELECT @StartTimeBlock = StartTimeBlock FROM CallBacks WHERE ConstituentID
= @ConstituentID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
SELECT @EndTimeBlock = EndTimeBlock FROM CallBacks WHERE ConstituentID
= @ConstituentID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
SELECT @IsPersonal = IsPersonal From CallBacks WHERE ConstituentID =
@ConstituentID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
DELETE CallBacks WHERE ConstituentID = @ConstituentID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
INSERT INTO CallBacks
(ConstituentID, StartTimeBlock, EndTimeBlock, LastCallID, IsPersonal, InUse)
```

```
VALUES
(@ConstituentID, @StartTimeBlock, @EndTimeBlock, @LastCallID, @IsPersonal, 0)
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION RecreateCall
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION RecreateCall
RETURN @@ERROR


CREATE PROCEDURE RecreateCallingPoolEntry (@ConstituentID varchar (11),
                                                          @LastCallID int)
AS
BEGIN TRANSACTION RecreateCall
DELETE CallingPool WHERE ConstituentID = @ConstituentID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
INSERT INTO CallingPool
(ConstituentID, LastCallID, InUse)
VALUES
(@ConstituentID, @LastCallID, 0)
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION RecreateCall
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION RecreateCall
RETURN @@ERROR

CREATE PROCEDURE SPGetGroupTypeCallStats(@GroupTypeID tinyint)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int,
@WrongNumber int, @Spanish int, @MailInfo int,
@hangup int,  @Solicitation int, @Callback int, @Nocalls int,
@NoContact int, @Divorced int, @Deceased int, @AlreadyGave int
SELECT @Total = Count(ConstituentID) FROM vw_CallDetail
WHERE GroupTypeID = @GroupTypeID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupTypeID = @GroupTypeID
SELECT @NotReached = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=2 AND GroupTypeID = @GroupTypeID
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=3 AND GroupTypeID = @GroupTypeID
SELECT @Spanish= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=4 AND GroupTypeID = @GroupTypeID
```

```
SELECT @WrongNumber = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=5 AND GroupTypeID = @GroupTypeID
SELECT @Deceased= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=6 AND GroupTypeID = @GroupTypeID
SELECT @Divorced = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=7 AND GroupTypeID = @GroupTypeID
SELECT @MailInfo= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=8 AND GroupTypeID = @GroupTypeID
SELECT @NoCalls = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=9 AND GroupTypeID = @GroupTypeID
SELECT @NoContact = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=10 AND GroupTypeID = @GroupTypeID
SELECT @Solicitation= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=11 AND GroupTypeID = @GroupTypeID
SELECT @CallBack= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=12 AND GroupTypeID = @GroupTypeID
SELECT @HangUp = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=13 AND GroupTypeID = @GroupTypeID
SELECT @AlreadyGave = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=14 AND GroupTypeID = @GroupTypeID

CREATE PROCEDURE SP_GetGroupTypeCallStats(@GroupTypeID tinyint)  AS
DECLARE
@Total int, @Pledge int, @NotReached int, @Refusal int,
@WrongNumber int, @Spanish int, @MailInfo int,
@hangup int,  @Solicitation int, @Callback int, @Nocalls int,
@NoContact int, @Divorced int, @Deceased int, @AlreadyGave int
SELECT @Total = Count(ConstituentID) FROM vw_CallDetail
WHERE GroupTypeID = @GroupTypeID
SELECT @Pledge = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=1 AND GroupTypeID = @GroupTypeID
SELECT @NotReached = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=2 AND GroupTypeID = @GroupTypeID
SELECT @Refusal = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=3 AND GroupTypeID = @GroupTypeID
SELECT @Spanish= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=4 AND GroupTypeID = @GroupTypeID
SELECT @WrongNumber = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=5 AND GroupTypeID = @GroupTypeID
SELECT @Deceased= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=6 AND GroupTypeID = @GroupTypeID
SELECT @Divorced = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=7 AND GroupTypeID = @GroupTypeID
SELECT @MailInfo= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=8 AND GroupTypeID = @GroupTypeID
SELECT @NoCalls = Count(ConstituentID) FROM vw_CallDetail
```

```
WHERE Result=9 AND GroupTypeID = @GroupTypeID
SELECT @NoContact = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=10 AND GroupTypeID = @GroupTypeID
SELECT @Solicitation= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=11 AND GroupTypeID = @GroupTypeID
SELECT @CallBack= Count(ConstituentID) FROM vw_CallDetail
WHERE Result=12 AND GroupTypeID = @GroupTypeID
SELECT @HangUp = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=13 AND GroupTypeID = @GroupTypeID
SELECT @AlreadyGave = Count(ConstituentID) FROM vw_CallDetail
WHERE Result=14 AND GroupTypeID = @GroupTypeID

CREATE PROCEDURE SearchProspectMatchCompanies (@SearchText varchar (40))
AS
SELECT ID, Name, Ratio
FROM MatchingGift
WHERE Name LIKE @SearchText
ORDER BY Name

CREATE PROCEDURE SearchProspectMatching (@Employer varchar (40)) AS
SELECT ID, Ratio, MatchesSpouse FROM MatchingGift
WHERE
Name LIKE @Employer OR Alias LIKE @Employer OR Parent LIKE @Employer

CREATE PROCEDURE SearchProspectsByName (@FirstName varchar(25),
                              @LastName varchar(25))
AS
BEGIN TRANSACTION SearchProspects
CREATE TABLE #TempData1
(
  ConstituentID varchar(11)
)
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData1
SELECT DISTINCT ConstituentID
FROM Demographic
WHERE IsSpouse = 0 AND (FirstName LIKE @FirstName AND LastName LIKE
@LastName)
IF @@ERROR <> 0
  GOTO ErrorHandler
CREATE TABLE #TempData2
(
  ConstituentID varchar(11),
  FirstName varchar(25),
  LastName varchar(25),
  MiddleName varchar(25),
  GroupName varchar(30),
  Telephone char(16)
)
IF @@ERROR <> 0
  GOTO ErrorHandler
INSERT INTO #TempData2
SELECT ConstituentID, NULL, NULL, NULL, NULL, NULL
FROM #TempData1
IF @@ERROR <> 0
  GOTO ErrorHandler
DROP TABLE #TempData1
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET FirstName = (SELECT TOP 1 FirstName FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET MiddleName = (SELECT TOP 1 MiddleName FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET LastName = (SELECT TOP 1 LastName FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET GroupName = (SELECT GroupName FROM vw_ProspectGroupDetail
WHERE vw_ProspectGroupDetail.ConstituentID = #TempData2.ConstituentID)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE #TempData2
SET Telephone = (SELECT TOP 1 Telephone FROM Demographic
WHERE Demographic.ConstituentID = #TempData2.ConstituentID
ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
SELECT * FROM #TempData2
ORDER BY LastName, FirstName
IF @@ERROR <> 0
```

```
  GOTO ErrorHandler
DROP TABLE #TempData2
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION SearchProspects
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION SearchProspects
RETURN @@ERROR


CREATE PROCEDURE SearchSpouseMatchCompanies (@SearchText varchar (40))
AS
SELECT ID, Name, Ratio
FROM MatchingGift
WHERE Name LIKE @SearchText AND MatchesSpouse = 'Y'
ORDER BY Name


CREATE PROCEDURE SetCallerActiveState (@CallerID smallint, @State bit) AS
BEGIN TRANSACTION SetState
UPDATE Callers
SET IsActive = @State WHERE CallerID = @CallerID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION SetState
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION SetState
RETURN @@ERROR

CREATE PROCEDURE SetCallerManagerState (@CallerID smallint, @State bit) AS
BEGIN TRANSACTION SetState
UPDATE Callers
SET IsManager = @State WHERE CallerID = @CallerID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION SetState
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION SetState
RETURN @@ERROR

CREATE PROCEDURE UpdateCaller (@CallerID smallint,
                               @UserNTLM varchar (20),
                               @FirstName varchar (20),
                                        @LastName varchar (25),
                                        @IsActive bit,
                                        @IsManager bit) AS
BEGIN TRANSACTION CallerUpdate
UPDATE Callers
SET UserNTLM = @UserNTLM, FirstName = @FirstName, LastName = @LastName,
IsActive = @IsActive, IsManager = @IsManager
WHERE CallerID = @CallerID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION CallerUpdate
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION CallerUpdate
RETURN @@ERROR

CREATE PROCEDURE UpdateChild (@ChildID int,
                                        @FirstName varchar (25),
                                        @LastName varchar (25),
                                        @SSN char(11)) AS
UPDATE Children
SET  FirstName = @FirstName, LastName = @LastName, SSN = @SSN
WHERE ChildID = @ChildID

CREATE PROCEDURE UpdateDegree (@DegreeID int,
                                          @Year int,
                                          @Degree varchar (10),
                                          @Major varchar (40),
                                          @SchoolName varchar (40),
                                          @CollegeName varchar (40)) AS
BEGIN TRANSACTION DegreeUpdate
UPDATE Education
SET Year = @Year, Degree = @Degree, Major = @Major, SchoolName =
@SchoolName, CollegeName = @CollegeName, Updated = GetDate()
WHERE ID = @DegreeID
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION DegreeUpdate
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION DegreeUpdate
RETURN @@ERROR


CREATE PROCEDURE UpdateDemographic (@ConstituentID varchar(11),
                                                  @SSN char (11),
                                                  @Prefix varchar(15),
                                                  @FirstName varchar(25),
                                                  @MiddleName varchar(25),
                                                  @LastName varchar(25),
                                                  @Gender char(6),
                                                  @Address1 varchar(40),
                                                  @Address2 varchar(40),
                                                  @Address3 varchar(40),
                                                  @City varchar(40),
                                                  @State char(2),
                                                  @Zipcode char(11),
                                                  @Telephone char(16),
                                                  @Email varchar(60),
                                                  @Birthdate smalldatetime=NULL,
                                                  @IsSpouse bit) AS
BEGIN TRANSACTION DemographicUpdate
INSERT INTO Demographic
(ConstituentID, SSN, Prefix, FirstName, MiddleName, LastName, Gender, Address1,
Address2, Address3, City, State, Zipcode, Telephone, Email, Birthdate, IsSpouse,
Updated)
VALUES
(@ConstituentID, @SSN, @Prefix, @FirstName, @MiddleName, @LastName,
@Gender, @Address1, @Address2, @Address3, @City, @State, @Zipcode,
@Telephone, @Email, @Birthdate, @IsSpouse, GetDate())
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION DemographicUpdate
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION DemographicUpdate
RETURN @@ERROR


CREATE PROCEDURE UpdateEmployment (@ConstituentID varchar(11),
                                              @Employer varchar (40),
                                              @JobTitle varchar(40),
                                              @Address1 varchar(40),
                                              @Address2 varchar(40),
                                              @Address3 varchar(40),
                                              @City varchar(40),
                                              @State char(3),
                                              @Zipcode char(11),
                                              @Telephone char(16),
                                              @IsSpouse bit) AS
BEGIN TRANSACTION EmploymentUpdate
INSERT INTO Employment
```

```
(ConstituentID, Employer, JobTitle,  Address1, Address2, Address3, City, State,
Zipcode, Telephone, IsSpouse, Updated)
VALUES
(@ConstituentID, @Employer, @JobTitle, @Address1, @Address2, @Address3, @City,
@State, @Zipcode, @Telephone, @IsSpouse, GetDate())
IF @@ERROR <> 0
  GOTO ERRORHANDLER
COMMIT TRANSACTION EmploymentUpdate
RETURN 0
ERRORHANDLER:
ROLLBACK TRANSACTION EmploymentUpdate
RETURN @@ERROR

CREATE PROCEDURE UpdateGroup (@GroupID tinyint,
                              @GroupName varchar(30),
                              @DefaultDesignation varchar(50),
                              @ScriptURL varchar(80))
AS
UPDATE Groups
SET GroupName = @GroupName, DefaultDesignation = @DefaultDesignation,
ScriptURL = @ScriptURL
WHERE GroupID = @GroupID


CREATE PROCEDURE UpdateProspectMatchingInfo (@ConstituentID varchar (11),
                                             @Ratio float)
AS
DECLARE
@EmploymentID int
BEGIN TRANSACTION UpdateRatio
SET @EmploymentID = (SELECT TOP 1 EmploymentID FROM Employment
  WHERE ConstituentID = @ConstituentID AND IsSpouse = 0
  ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE Employment
SET Matches = 1, Ratio = @Ratio
WHERE EmploymentID = @EmploymentID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION UpdateRatio
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION UpdateRatio
RETURN @@ERROR
```

```
CREATE PROCEDURE UpdateSpouseMatchingInfo (@ConstituentID varchar (11),
                                                        @Ratio float)
AS
DECLARE
@EmploymentID int
BEGIN TRANSACTION UpdateRatio
SET @EmploymentID = (SELECT TOP 1 EmploymentID FROM Employment
  WHERE ConstituentID = @ConstituentID AND IsSpouse = 1
  ORDER BY Updated DESC)
IF @@ERROR <> 0
  GOTO ErrorHandler
UPDATE Employment
SET Matches = 1, Ratio = @Ratio
WHERE EmploymentID = @EmploymentID
IF @@ERROR <> 0
  GOTO ErrorHandler
COMMIT TRANSACTION UpdateRatio
RETURN 0
ErrorHandler:
ROLLBACK TRANSACTION UpdateRatio
RETURN @@ERROR


CREATE PROCEDURE UploadDegree (@ID int OUT,
                                      @ConstituentID varchar (11),
                                         @Year int,
                                         @Degree varchar (10),
                                         @Major varchar (40),
                                         @SchoolName varchar (40),
                                         @CollegeName varchar (40),
                                         @IsSpouse bit) AS
INSERT INTO Education
(ConstituentID, Year, SchoolName, CollegeName, Degree, Major, IsSpouse, Updated)
VALUES
(@ConstituentID, @Year, @SchoolName, @CollegeName, @Degree, @Major,
@IsSpouse, '1/1/1900')
SET @ID = @@IDENTITY

CREATE PROCEDURE UploadDemographic (@DemographicID int OUT,
                                           @ConstituentID varchar(11),
                                               @SSN char (11),
                                               @Prefix varchar(15),
                                               @FirstName varchar(25),
                                               @MiddleName varchar(25),
                                               @LastName varchar(25),
                                               @Gender char(6),
                                               @Address1 varchar(40),
                                               @Address2 varchar(40),
                                               @Address3 varchar(40),
                                               @City varchar(40),
                                               @State char(3),
                                               @Zipcode char(11),
                                               @Telephone char(16),
                                               @Email varchar(60),
                                               @Birthdate smalldatetime=NULL,
                                               @IsSpouse bit) AS
INSERT INTO Demographic
(ConstituentID, SSN, Prefix, FirstName, MiddleName, LastName, Gender, Address1,
Address2, Address3, City, State, Zipcode, Telephone, Email, Birthdate, IsSpouse,
Updated)
VALUES
(@ConstituentID, @SSN, @Prefix, @FirstName, @MiddleName, @LastName,
@Gender, @Address1, @Address2, @Address3, @City, @State, @Zipcode,
@Telephone, @Email, @Birthdate, @IsSpouse, '1/1/1900')
SET @DemographicID = @@IDENTITY

CREATE PROCEDURE UploadEmployment (@EmploymentID int OUT,
                                    @ConstituentID varchar(11),
                                               @Employer varchar (40),
                                               @JobTitle varchar(40),
                                               @Address1 varchar(40),
                                               @Address2 varchar(40),
                                               @Address3 varchar(40),
                                               @City varchar(40),
                                              @State char(3),
                                              @Zipcode char(11),
                                              @Telephone char(16),
                                              @IsSpouse bit) AS
INSERT INTO Employment
(ConstituentID, Employer, JobTitle,  Address1, Address2, Address3, City, State,
Zipcode, Telephone, IsSpouse, Updated)
VALUES
(@ConstituentID, @Employer, @JobTitle, @Address1, @Address2, @Address3, @City,
@State, @Zipcode, @Telephone, @IsSpouse, '1/1/1900')
SET @EmploymentID = @@IDENTITY
```