Gopi Krishna Katakam

Zander Pegues

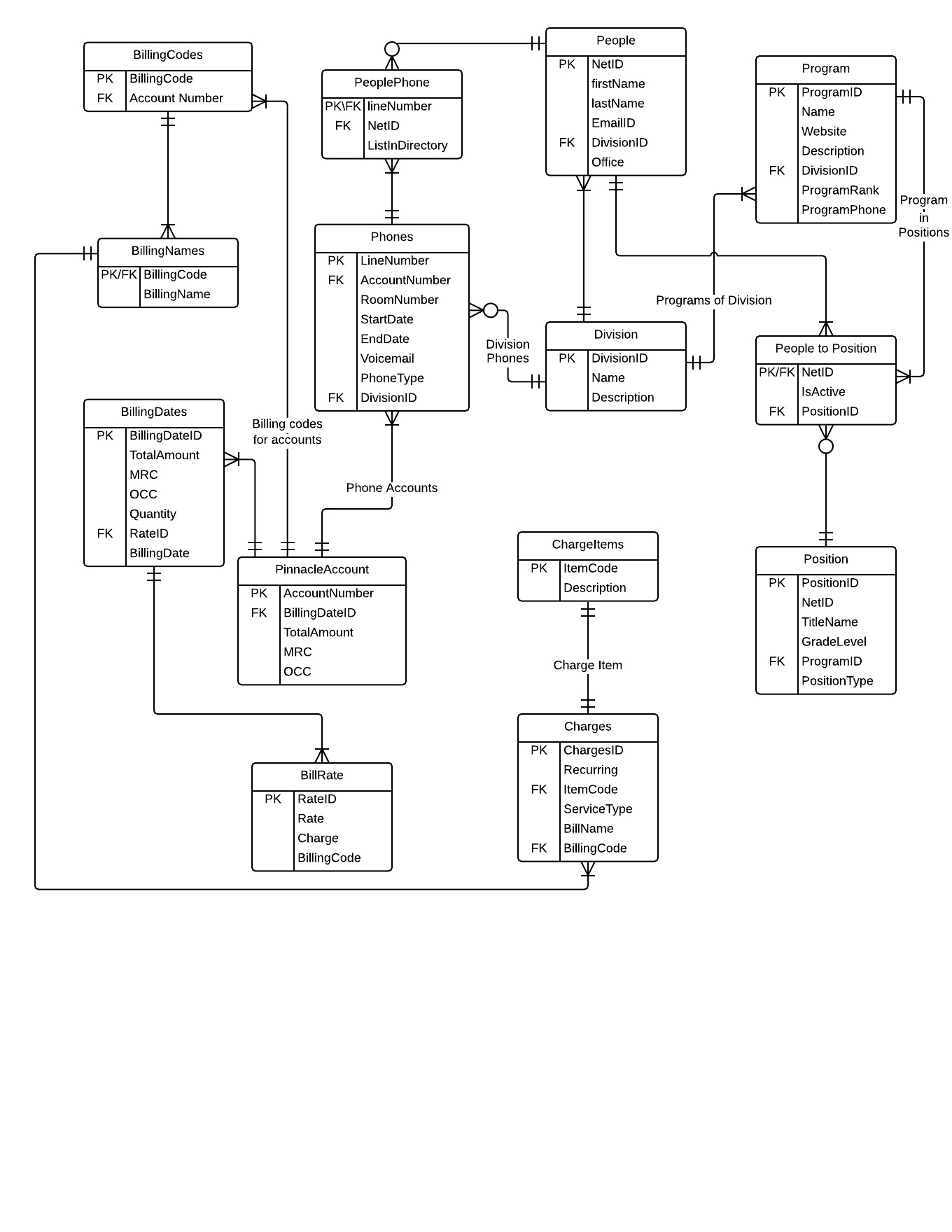
Technical Document

1. **Problem Description**: The goal of our project is to replace the legacy system that is used to capture reconciliation data for tracking the phone systems at the Center for Development and Disability. Additionally, the project should improve the functionality of the phone database by increasing the scope of the data being captured. Furthermore, the database can be integrated into the existing directory database in order to streamline directory updating services. Lastly, if possible, we would like to create a script that will pull data straight from Telecomm and import it directly into the new database (removing the current dependencies on running macros in excel to filter the data).
2. **Analysis**
   1. Existing System: Currently users run reports from Pinnacle (Telecom’s database system) filter the Pinnacle data through an excel macro and then import the data into the Access database. The new process will bypass the excel macros and rely on querying to filter the relevant data.
   2. Users & Abilities: Our users are basic users (non-power users) with rudimentary technical skills. There are currently only two users of the database, but it is possible that up to 5 additional users may need to access the system. The two main users are “worker bees”, not supervisors. However, three database analysts, one system analyst and one supervisor also have access to the system to troubleshoot technical issues. The analysts and supervisor all have advanced technical skills. Potential stakeholders include Supervisors, Division Coordinators, Inventory Control administrators and all staff members who use the directory.
   3. New System: We are migrating from a Microsoft Access database to a SQL Server database. This migration will make the data more secure and it will increase performance. Furthermore, we are combining the directory database with the phone DB to give users more data to query from and create a single source to contain both databases.
3. **Assumptions**

We considered the following assumptions when creating our database.

* Consider NetID as 10 characters long (typically it is 9 characters, but it might increase by one more character in future).
* A NetID refers to each indivual (one to one relationship between Netid and people).
* The Office column is not mandatory to store in the people table.
* The ProgramRank and ProgramPhone columns in table Program can be null.
* Program and division must have its name and description to be stored
* Need to retrieve a type of phone and also whether it has voicemail or not.
* ChargesID is a unique identifier for charges table.
* BillingDateID is a unique identifier for BillingDates table
* ListInDirectory is not mandatory for table PeoplePhone.
* ProgramID must be 4 characters in length.
* DivisionID is considered same as Acronym of the Division and it must be 5 characters in length.
* The PeoplePhone and People to Position tables are used as bridge tables between many to many relationships.
* ChargesID is an alias for Authorization Code (a field in the Pinnacle report)

1. **Solution Description**: The goal of the project is to create a SQL Server database with the data from the existing Access database. Furthermore, we will need to secure the database so that only authorized users can modify the data. If possible, we would like to link the new database to the existing directory database, which also uses SQL Server. Finally, depending on our access to Pinnacle, we would also like to create a script that will pull the data straight from Pinnacle into SQL Server. To ensure that our solution can be implemented we need access to the current Microsoft Access database. Additionally, we need to collaborate with the database analyst to sync our new database with the current directory database. Furthermore, we need to contact Pinnacle to find out about streamlining the data retrieval process.
   1. Process: To begin we met with users to learn the basic requirements of the current Access system. In this meeting the users provided us with a copy of the Pinnacle report from which their data is entered. We then increased the scope of the data being captured by including all of the data from the pinnacle report (where previously they only captured a few fields from the report). Furthermore, we were able to include the current directory database into our new phone database to have a single source for storing the data. Next, we built the ERD to depict the tables, columns and the relationships between the tables. Finally, we built database in SQL Server and set user permissions and encrypted the data.
   2. Technologies Used: We used Anderson’s VMs to collaborate on the creation of a single SQL Server database. We also used Visual studio to test vulnerabilities and attack group A’s database.
   3. Requirements of Solution: Upon meeting with our users they requested that the database be able to capture all of their current data as well as provide an avenue to create reports based on the data. Furthermore, the users requested that we merge the database with the current directory database in order to better streamline data entry and retrieval through the single source point for data. Moreover, we were able to document the exact tables and columns that were captured in the Pinnacle report so that we could replicate them in a SQL Server database.
2. **ERD**:



1. **Table Descriptions:**

Level 0 tables:

BillRate (RateID, Rate, Charge, BillingCode)

RateID, int, not null

Rate, real, not null

Charge, real, not null

BillingCode, varchar(64), not null

ChargeItems (ItemCode, Description)

ItemCode, varchar(64), not null

Description, varchar(64), not null

Division (DivisionID, Name, Description)

DivisionID, char(5), not null

Name, varchar(64), not null

Description, varchar(128), not null

Level 1 tables:

BillingDates (BillingDate, TotalAmount, MRC, OCC, Quantity, *RateID*),

BillingDateID, int, not null

TotalAmount, real, not null

MRC, real, not null

OCC, real, not null

Quantity, int, not null

RateID, int, not null

BillingDate, date, not null

*RI: BillingDates.RateID must exist in BillRate.RateID*

People (NetID, FirstName, LastName, EmailID, *Division*, Office)

NetID, char(10), not null

FirstName, varchar(64), not null

LastName, varchar(64), not null

EmailID, varchar(64), not null

DivisionID, char(5), not null

Office, varchar(64), not null

*RI: People.DivisionID must exist in Division.DivisionID*

Program (ProgramID, Name, Website, Description, *DivisionID*, ProgramRank, ProgramPhone)

ProgramID, char(4), not null

Name, varchar(64), not null

Website, varchar(1024), not null

Description, varchar(128), not null

DivisionID, char(5), not null

ProgramRank, char(4), null

ProgramPhone, varchar(12), null

*RI: Program.DivisionID must exist in Division.DivisionID*

Level 2 tables:

PeopleToPosition (NetID, IsActive, *PositionID*)

NetID, char(10), not null

IsActive, bit, not null

PositionID, smallint, not null

*RI: PeopleToPosition.PositionID must exist in Position.PositionID*

PinnacleAccount (AccountNumber, *BillingDateID*, TotalAmount, MRC, OCC)

AccountNumber, varchar(64), not null

BillingDateID, date, not null

TotalAmount, real, not null

MRC, real, not null

OCC, real, not null

*RI: PinnacleAccount.BillingDateID must exist in BillingDates.BillingDateID*

Position (PositionID, PersonID, TitleName, GradeLevel, *ProgramID*, PositionType)

PositionID, smallint, not null

PersonID, char(10), not null

TitleName, varchar(32), not null

GradeLevel, varchar(5), not null

ProgramID, char(4), not null

PosisitionType, varchar(32), not null

*RI: Position.ProgramID must exist in Program.ProgramID*

Level 3 tables:

Billing Codes (BillingCode, *AccountNumber*),

BillingCode, varchar(64), not null

AccountNumber, vachar(64), not null

*RI: BillingCodes.AccountNumber must exist in PinnacleAccount.AccountNumber*

Phones (LineNumber, *AccountNumber*, RoomNumber, StartDate, EndDate, Voicemail, PhoneType, *DivisionID*)

LineNumber, varchar(12), not null

AccountNumber, varchar(64), not null

RoomNumber, varchar(6), not null

StartDate, date, not null

EndDate, date, not null

Voicemail, bit, not null

PhoneType, varchar(24), not null

DivisionID, char(5), not null

*RI: Phone.AccountNumber must exist in PinnacleAccount.AccountNumber*

*RI: Phone.DivisionID must exist in Division.DivisionID*

Level 4 tables:

Billing Names (*BillingCode*, BillingName),

BillingCode, varchar(64), not null

BillingName,varchar(64), not null

RI: BillingNames.BillingCode must exist in BillingCodes.BillingCode

PeoplePhone (*LineNumber*, *NetID*, ListInDirectory)

LineNumber, varchar(12), not null

NetID, char(10), not null

Listindirectory, bit, null

*RI: PeoplePhone.NetID must exist in People.NetID*

*RI: PeoplePhone.LineNumber must exist in Phones.LineNumber*

Level 5 tables

Charges (ChargesID, Recurring, *ItemCode*, ServiceType, BillName, BillCode)

ChargesID, varchar(64), not null

Recurring, bit, not null

ItemCode, varchar(64), not null

ServiceType, varchar(64), not null

BillName, varchar(64), not null

BillCode, varchar(64), not null

*RI: Charges.ItemCode must exist in ChargeItems.ItemCode*

*RI: Charges.BillingCode must exist in BillingCodes.BillingCode*

1. **Object Descriptions**

Views:

**vwCharges:**

* Returns a list of Charge information of the lines along with their respective BillingName and Billing Rates.
* **Tables Included:** Charges, ChargeItems
* **View Columns:** ChargesID, ServiceType, BillName, BillCode, ItemCode,Description

**vwDivisionPhones:**

* Returns a list of all phone line numbers associated to each division along with division name and division description.
* **Tables included**: division and phone tables
* **View Columns:** divisionID, name, description, linenumber

**vwPhoneList:**

* Returns a list of all phones associated to each person along with their division and office information
* **Tables included:** phone, division, people
* **View Columns:** firstName, lastName, lineNumber, divisionID, office

**vwpositions:**

* Returns a list of positions associated to each person along with their details like first name, last name, email id, title name, position and program
* **Tables included:** people, peopleToPosition, position
* **View Columns:** firstName, lastName, EmailID, titleName, positionID, programID

**vwprograms:**

* Returns a list of all programs associated to each division
* **Tables included:** program, division
* **View Columns:** programName, programDescription, programPhone, DivisionName

**vwTitle:**

* Returns a list of people along with their title.
* **Tables included:** people, peopleToPosition, position
* **View Columns:** firstName, lastName, titleName

Stored Procedures:

**getBillingRates:**

Returns a list of Rates based on BIllingDate

**Parameters:**

@BillingDate date (required)

**Returns:** Data Set

* BillingDate
* TotalAmount
* Rate
* Charge

**getBillingReport:**

Returns a list of all BillingNames, BillingCodes alomg with Billingdates based on AccountNumber

**Parameters:**

@AccountNumber VARCHAR(64)

**Returns:** Data Set

* AccountNumber
* BillingDate
* BillingCode
* BillingName

**getLineCount**

Returns a count of phonelines based on voicemail functionality (yes or no)

**Parameters:**

@voiceMail BIT

**Returns:** LineCount

**getPhoneListReport:**

Returns a list of phones based on AccountNumber

**Parameters:**

@AccountNumber VARCHAR(64)

**Returns:** lastName, firstName, lineNumber, voiceMail, TotalAmount

**getProgramEachDivision:**

Returns a list of programs based on divisionName

**Parameters:**

@divisionName VARCHAR(32)

**Returns:** programName, programDescription, programphone

**getCharges:**

Returns a list of items along with ServiceType based on Bill Name.

**Parameters:**

@BillName VARCHAR(64)

**Returns:** ItemCode, Description, ServiceType

**getActivePeopleLineNumber:**

Returns a list of lineNumbers of people based on their position(active or not-active).

**Parameters:**

@input BIT

**Returns:** NetID, lineNumber

**getPositionsOfEachProgram:**

Returns a List of positions in each program based on divisionID.

**Parameters:**

@division CHAR(5)

**Returns:** positionID, NetID, titleName, programID

**Insert Stored Procedures:**

**insBillingCodes:**

Insert new BIllingCode.

**Parameters:**

* + - @BillingCode VARCHAR(64),
    - @AccountNumber VARCHAR(64)

**Returns:** nothing

**insBillingDates:**

Insert BillingDate.

**Parameters:**

* + - @BillingDate date,
    - @TotalAmount float(3),
    - @MRC float(3),
    - @OCC float(3),
    - @Quantity int,
    - @RateID int

**Returns:** nothing

**insBillingNames:**

Insert BillingNames into BillingName Table.

**Parameters:**

* + - @BillingCode VARCHAR(64),
    - @BillingName VARCHAR(64)

**Returns:** nothing

**insBillRate:**

Insert New BillRate into BillRate Table

**Parameters:**

* + - @RateID int,
    - @Rate float(3),
    - @Charge float(3),
    - @BillingCode varchar(64)

**Returns:** Nothing

**insChargeItems:**

Insert new Item in ChargeItems Table.

**Parameters:**

* + - @ItemCode varchar(64),
    - @Description varchar(64)

**Returns:** Nothing

**insCharges:**

Insert new Charge in Charges Table.

**Parameters:**

* + - @ChargesID varchar(64),
    - @Recurring bit,
    - @ItemCode varchar(64),
    - @ServiceType varchar(64),
    - @BillName varchar(64),
    - @BillingCode varchar(64)

**Returns:** Nothing

**insdivision:**

Insert new division.

**Parameters:**

* + - @divisionID CHAR(5),
    - @name VARCHAR(32),
    - @description VARCHAR(128)

**Returns:** Nothing

**insPeople:**

Insert people to people table

**Parameters:**

* + - @NetID CHAR(10),
    - @firstName VARCHAR(64),
    - @lastName VARCHAR(64),
    - @EmailID VARCHAR(64),
    - @divisionID CHAR(5),
    - @office VARCHAR(64) OPTIONAL

**Returns:** nothing

**inspeopleToPhone:**

Insert NetID, linenumber and listIndirectory data into table.

**Paramters:**

* + - @lineNumber VARCHAR(12),
    - @NetID CHAR(10),
    - @listInDirectory BIT Optional
    - **Returns:** Nothing

**inspeopleToPosition:**

Insert record to match people,position into peopleToPosition Table.

**Paramters:**

* + - @NetID CHAR(10),
    - @isActive BIT,
    - @positionID SMALLINT

**Returns:** Nothing

**insphone:**

Insert new phoneline into phone table.

**Parameters:**

* + - @lineNumber VARCHAR(12),
    - @accountNumber VARCHAR(64),
    - @roomNumber VARCHAR(6),
    - @startDate DATE,
    - @endDate DATE,
    - @voiceMail BIT,
    - @phoneType VARCHAR(24),
    - @divisionID CHAR(5)

**Returns:** Nothing

**inspinnacleAccount:**

Insert new Pinnacle Account into PinnacleAccount Table.

**Parameters:**

* + - @AccountNumber varchar(64),
    - @BillingDateI int,
    - @TotalAmoun float(3),
    - @MRC float(3),
    - @OCC float(3)

**Returns:** Nothing

**insposition:**

Insert new position in position table.

**Parameters:**

* + - @NetID CHAR(10),
    - @titleName VARCHAR(32),
    - @gradeLevel VARCHAR(5),
    - @programID CHAR(4),
    - @positionType VARCHAR(32)

**Returns:** Nothing

**Insprogram:**

Insert new Program into program table.

**Parameters:**

* + - @programID CHAR(4),
    - @name VARCHAR(64),
    - @website VARCHAR(1024),
    - @description VARCHAR(128),
    - @divisionID CHAR(5),
    - @programRank CHAR(4) OPTIONAL
    - @programphone varchar(12) OPTIONAL

**Returns:** Nothing

**Update Stored Procedures:**

**updBillRate:**

Update Table BillRate based on RateID Column Value.

**Parameters:**

* + - @RateID int,
    - @Rate float(3),
    - @Charge float(3)

**Return:** Nothing

**updBillingName:**

Update BillingName Table based on BillingCode

**Parameters:**

* + - @billingName VARCHAR(64),
    - @billingCode VARCHAR(64)

**Returns:** Nothing

**updChargeItemDescription:**

Update Description of ChargeItem in ChargeItems based on ItemCode

**Parameters:**

* + - @description varchar(64),
    - @itemCode VARCHAR(64)

**Returns:** Nothing

**updDirectory:**

Update listInDirectory in table peoplePhone based on lineNumber

**Parameters:**

* + - @ListInDirectory BIT,
    - @lineNumber VARCHAR(12)

**Returns:** Nothing

**updDivisionDescription:**

Update Division Description based on divisionID in division Table.

**Parameters:**

* + - @description varchar(128),
    - @divisionID CHAR(5)

**Returns:** Nothing

**updEndDateOfLine:**

Update endDate of the lineNumber basedon lineNumber in phone table.

**Parameters:**

* + - @description varchar(128),
    - @divisionID CHAR(5)

**Returns**: Nothing

**updLastName:**

Update lastName in table people based on NetiD.

**Parameters:**

* + - @description varchar(128),
    - @divisionID CHAR(5)

**Returns:** Nothing

**updPositionTitle:**

Update title of the position in position table based on last name of the person.

**Paramters:**

* + - @title varchar(32),
    - @lastName VARCHAR(64)

**Returns:** Nothing

**updPositionType:**

Update Postiontype in position table based on positionID.

**Parameters:**

* + - @positionType VARCHAR(32),
    - @positionID smallint

**Returns**: Nothing

**updProgramName:**

Update name in program table based on programID

**Parameters:**

* + - @name varchar(62),
    - @programID CHAR(4)

**Returns:** Nothing

**updquanity:**

Update Quantity in BillingDates table basedon BillingDate

**Parameters:**

* + - @billingDate date,
    - @quantity int

**Returns:** Nothing

**updServicetype:**

Update ServiceType in Charges table based on ChargesID

**Parameters:**

* + - @serviceType varchar(64),
    - @ChargesID VARCHAR(64)

**Returns:** Nothing

**updTotalAmmount:**

Update TotalAmount in PinnacleAccount based on AccountNumber.

**Parameters:**

* + - @totalAmount real,
    - @accountNumber VARCHAR(64)

**Returns:** Nothing

**Delete Stored Procedures:**

**delAccount:**

Delete record from table PinnacleAccount based on AccountNumber

**Parameters:** @accounNumber VARCHAR(64)

**Returns:** Nothing

**delBillingCodes:**

Delete record from BillingCodes table based on BillingCode

**Parameters:** @BillingCode VARCHAR(64)

**Returns:** nothing

**delBillingDate:**

Delete record from BillingDates table based on BillingDateID

**Parameters:** @BillingDateID INT

**Returns:** Nothing

**delBillingName:**

Delete record from BillingNames table based on BillingCode

**Parameters:** @BillingCode VARCHAR(64)

**Returns:** Nothing

**delBillRate:**

Delete record from BillRate table based on RateID

**Parameters:** @rateID int

**Returns:** Nothing

**delChargeItems:**

Delete record from ChargeItems table based on itemCode

**Paramaetrs:** @Itemcode VARCHAR(64)

**Returns:** nothing

**delCharges:**

Delete record from Charges table based on ChargesID

**Parameters:** @ChargesID VARCHAR(64)

**Returns:** Nothing

**deldivision:**

Delete record from division table based on both name and description

**Parameters:**

* @name VARCHAR(32),
* @description VARCHAR(128)

**Returns:** nothing

**delpeople:**

Delete record from people table based on NetID

**Parameters:** @NetID CHAR(10)

**Returns:** Nothing

**delpeoplePhone:**

Delete record from peopelPhone table based on lineNumber

**Parameters:** @lineNumber VARCHAR(12)

**Returns:** Nothing

**delphone:**

Delete record from phone table based on lineNumber

**Parameters:** @linenumber VARCHAR(12)

**Returns:** Nothing

**delposition:**

Delete record from position table based on netID

**Parameters**: @NetID CHAR(10)

**Returns:** Nothing

**delprogram:**

Delete record from program table based on both name and description

**Parameters:**

* @name VARCHAR(64),
* @description VARCHAR(128)

**Returns:** Nothing

And also we have written more stored procedures to create our database by executing just one SP and Drop our Database by using one SP.

**CreateCDDPhoneDB**

**Below mentioned two Stored Procedures must be executed in same sequence to create CDDPhone DB**

* CreateTables;
* CreateRelationships;

Where **CreateTables** Stored Procedure consists of query for creation of all each table and **CreateRelationships** consists of all relationships which must be maintained between tables.

**DropCDDPhoneDB:**

**Below mentioned three Stored Procedures must be executed in same sequence to Drop CDDPhone DB**

* DropRelationships;
* DeleteData;
* DropTables;

Where **DropRelationships** Stored Procedure removes all existing relationships between tables, **DeleteData** Stored Procedure delete all data in each table and finally **DropTables** Stored procedure drops all tables.

**List of Views and Stored procedures:**

Views**:**

1. vwpositions
2. vwprograms
3. vwTitle
4. PhoneListView
5. ChargesView
6. DivisionPhonesView

Stored Procedures**:**

Below are the Stored Procedures used for creating the database.

* CreateTables
* CreateRelationships
* DeleteData
* DropRelationships
* DropTables
* CreateCDDPhoneDB
* DropCDDPhoneDB

Below is a list of Stored Procedures that are used on the tables in our database.

1. programsEachDivision
2. billingRates
3. getBillingRates
4. getProgramsEachDivision
5. updPositionTitle
6. updProgramName
7. updDivisionDescription
8. PhoneListReport
9. LineCount
10. BillingReport
11. updChargeItemDescription
12. updServicetype
13. updTotalAmount
14. updDirectory
15. updEndDateOfLine
16. updBillingName
17. updquantity
18. updPositionType
19. updLastName
20. insPeople
21. inspeoplePhone
22. insprogram
23. insdivision
24. inspeopleToPosition
25. insposition
26. insphone
27. insBillingCodes
28. insBillingNames
29. insBillingDates
30. inspinnacleAccount
31. insBillrate
32. insChargeItems
33. UpdateBill
34. insCharges
35. SelectCharges
36. delpeople
37. SelectPeople
38. delpeoplePhone
39. SelectProgram
40. delprogram
41. deldivision
42. delposition
43. delphone
44. delBillingCodes
45. delBillingName
46. delBillingDate
47. delAccount
48. delBillRate
49. delChargeItems
50. delCharges

**Indexes:**

Below is a list of Indexes that are used to relate the tables in our database. One Index is created on every foreign keys in our tables and Default index on each primary key of table.

**BillingCodes:**

PK\_BillingCodes (clustered)

Non-Clustered Index on AccountNumber

**BillingDates:**

PK\_BillingDates (Clustered)

Non Clustered Index on RateID

**BillingNames:**

Same column for both PK and FK , clustered index on BillingCode

**BillRate:**

Clustered Index on RateID (PK\_BillRate).

**ChargeItems:**

Clustered Index on ItemCode (PK\_ChargeItems)

**Charges:**

Clusterd Index on ChargesID (PK\_Charges)

Non-Clustered Index on two of the columns (ItemCode, BillingCode)

**Division:**

Clustered Index on divisionID (PK\_division)

**People:**

Clustered Index on NetID (PK\_people)

Non-Clustered Index on divisionID

**PeoplePhone:**

Clustered Index on lineNumber (PK\_peoplePhone)

Non-Clustered Index on NetID

**PeopleToPosition:**

Clustered Index on NetID (PK\_PTP)

Non-Clustered Index on positionID

**Phone:**

Clusterd Index on lineNumber (PK\_phone)

Non-Clustered Index on divisionID

**PinnacleAccount:**

Clustered Index on AccountNumber (PK\_PinnacleAccount)

Non-Clustered Index on BillingDateID

**Position:**

Clustered Index on positionID

Non-Clustered Index on programID

**Program:**

Clustered Index on programID

Non-Clustered Index on divisionID

1. **Access to System**:

Two users need generic user access to the system (Cindy Feliz & Yvonne Dominguez). Additionally, the DBA (Vicki Scott), Data Manager (Gayle Osburn), System Analyst (Ken Potter) and IT Director (Dan Wenz) need admin access to the system.

1. **Security Mechanisms**:

**Considerations**:

To be able to access data from a database, a user must pass through two stages of authentication, one at the SQL Server level and the other at the database level. These two stages are implemented using Logins names and User accounts respectively. A valid login is required to connect to SQL Server and a valid user account is required to access a database.

**Login**: A valid login name is required to connect to an SQL Server instance. A login could be:

\* A Windows NT/2000 login that has been granted access to SQL Server

\* An SQL Server login that is maintained within SQL Server

**User**: A valid user account within a database is required to access that database. User accounts are specific to a database. All permissions and ownership of objects in the database are controlled by the user account. SQL Server logins are associated with these user accounts. A login can have associated users in different databases, but only one user per database.

**Implementation:**

In our database we would like to create users with various permission on objects upon request from organization.

We can restrict each user access for retrieve, update, and delete upon request by head of the department by creating a schema for individual, in this case once user creates that user will be assigned to particular schema where only one user (newly created) exists.

**Encryption:** by using Column level encryption we can secure our critical data from users who are not supposed to get that data, we performed column level encryption on AccountNumber column in PinnacleAccount table in our database and we would like to do more once we check with organization.

As per the result from attacking group we thought to make some changes in the user account like give access to view all stored procedures and views instead of restricting them and also restrict all table alter permission, if head of department requests to provide alter permission for set tables then we can do it for them.

Finally we would like create a backup with password protected and also same will be suggest to them for securing their database also would like to suggest create audit which can be used only in particular editions of SQL Server like (Enterprise, developer) we can’t perform audit on standard edition of SQL Server.

**Users**:

The users would like the database to be secure so that only the two users, the IT Director and DBAs can modify the data. Restrict Billing tables (BillingNames, BillingCodes, Charges, ChargeItems, PinnacleAccount, BillRate, BillingDates) permissions to only include the users described above (in the access to system section).

1. **DBA Approaches**:

Make sure that the users have restricted access or full depending on user requirements. We can also encrypt the data if needed. We can also schedule automatic backups that we can use to recover the database from. We can password protect the backups to keep it secure. We can audit the data in the database to ensure proper data is being inserted.

1. **Future Implementations**:
2. Create the user interface (website).
3. Create script to automatically pull Pinnacle data into SQL Server.
4. **Additional Information/Appendices**:

1. DB maintenance information to maintain the database on the DBMS selected: Once the database has begun the operation phase the DBA will need to maintain the data and log file management system, as well as the backups and backup schedule. Furthermore, they should be on the lookout for index fragmentation and DB corruption. Additionally, they may be asked to generate statistics on the database usage or on performance of queries.

2. Description of Data Access Application: describe how the user would access the system; if necessary provide a description of the user interface and the calls to DB objects that may be utilized: Currently a user interface has not been created. However, one should be created in the future so that users can effectively manage the data. Depending on what type of interface is created the user will need to be taught how to use the interface to generate reports on the data. Furthermore, they will need to be taught how to enter, retrieve and update data in the database through use of the interface.