Safe Haven Cat Rescue

Icon

Description automatically generated

Michael Johnson

IST659 M400

March 27, 2021

Contents:

[Part I: Project Summary 2](#_Toc67756167)

[Stakeholders: 2](#_Toc67756168)

[Business Rules: 3](#_Toc67756169)

[Data Questions: 3](#_Toc67756170)

[Data Glossary: 4](#_Toc67756171)

[Conceptual Model: 5](#_Toc67756172)

[Logical Model: 6](#_Toc67756173)

[Part II: Physical Database Design 6](#_Toc67756174)

[Access Forms and Reports: 30](#_Toc67756175)

[Reflection: 32](#_Toc67756176)

[Project Summary: 32](#_Toc67756177)

# Part I: Project Summary

* Safe Haven Cat Rescue is a non-profit organization developed to ensure the safety of the local cat population. The rescue manages a variety of staff members that consist of volunteers, contracted veterinarians, and various cat maintenance workers who provide care and maintenance to the cats living in the shelter.
* Currently, the Safe Haven Cat Rescue data consists of multiple Excel spreadsheets containing pet information, personnel information, donation information, and supply tracking to help with ordering food and medicine. After a person adopts one or multiple cats, the rescue moves them to an archived spreadsheet.
* Building a relational database would help the rescue accurately manage the supply of food and medicine based on the needs of each cat. In addition, the database would help the rescue reach out to potential people who want to adopt a cat, volunteer in the shelter, or people who are willing to donate needed items and money.

# Stakeholders:

* The stakeholders for this project include the entire Safe Haven staff which includes veterinarians, cat bodyguards, shelter maintenance technicians, and food technicians. The primary responsibilities for the stakeholders include maintaining the cat shelter, giving the cats a safe location to live, and ordering supplies. Additional stakeholders include the people who adopt the cats and the individuals donating to shelters.
* Regarding the stakeholders, a database would allow the owners to manage the staff schedule more effectively and identify shortages in food or medicine while encouraging more adoptions from people in the local area. Additionally, the database opens the rescue up to additional marketing techniques like email lists and targeted social media campaigns based on information provided by people who donate and adopt the cats.

# Business Rules:

* A person may adopt zero or many cats.
* A person may donate money to a shelter location.
* A person may donate an item to the item list.
* A person may volunteer or work at a shelter location.
* A person may have zero or more job types.
* The shelter can house zero or many cats.
* The shelter can have one or many items of various quantities.
* The shelter can have one or many jobs.
* A cat has the default statuses of not vaccinated, not adoption ready, and not adopted once rescued.
* An item can have one or more as the quantity.
* An item is dropped from the database when the quantity is 0.
* A donation date is entered immediately upon receipt.
* A donation of $200.00 is made for every cat adopted by the person who adopts the cat.
* A donation could consist of zero or many items.
* A purchased item for the shelter comes out of the donation funds as a negative donation with the addition of an item.
* Payment for the veterinarian service comes out of the donation table as a negative donation.

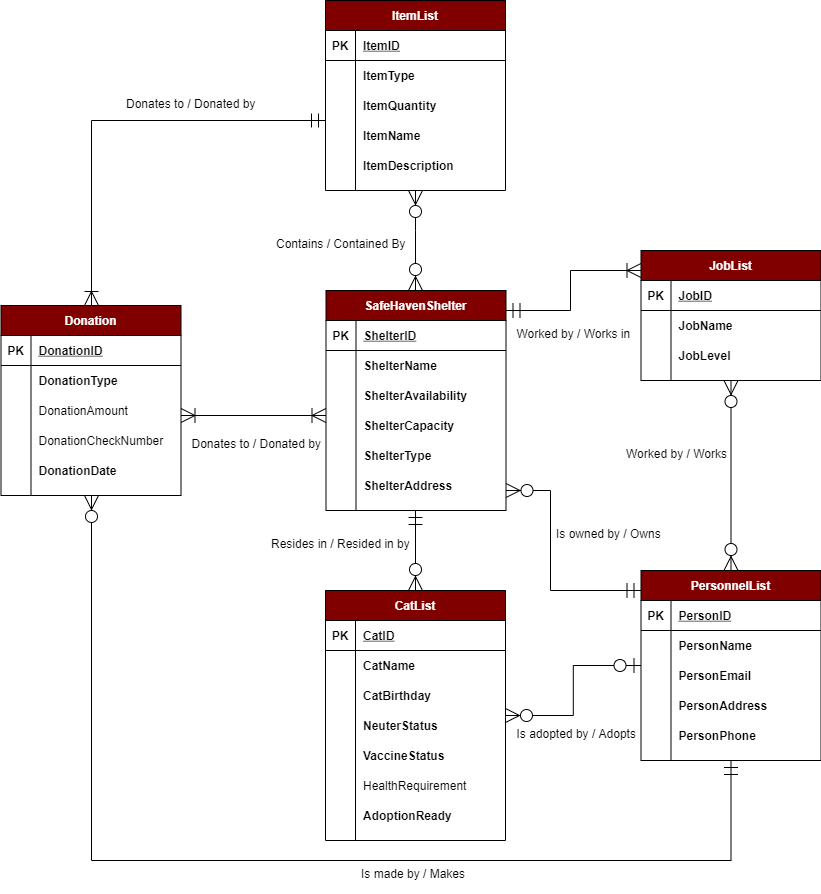
# Data Questions:

* Are more cash and check donations coming in than money going out per week?
* Where do the highest number of donations come from per month?
* How many days left where every cat can have two meals per day?
* How many cats are not ready for adoption and why?
* What city would be the best for marketing based on the number of adoptions?

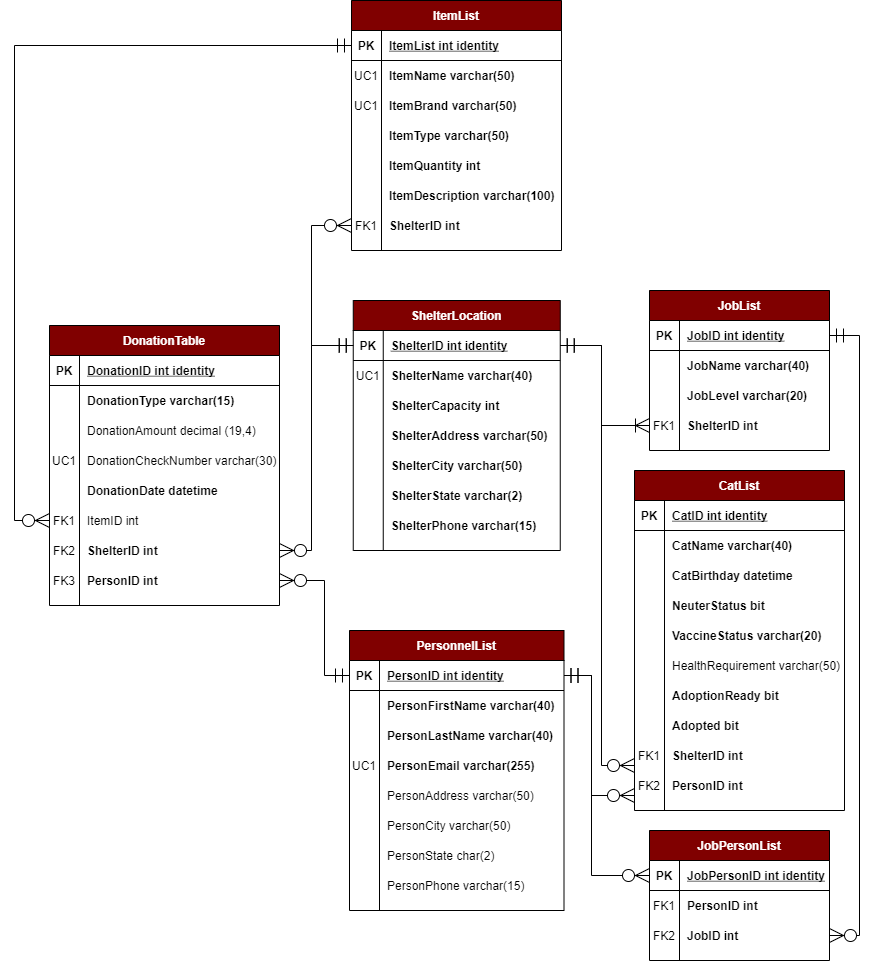
# Data Glossary:

* A ***cat*** is available for adoption or was adopted.
  + ***VaccineStatus*** indicates whether a cat has received all vaccines or not.
  + ***NeuterStatus*** indicates whether a cat is neutered or not.
  + ***HealthRequirement*** indicates whether the cat has special care instructions.
  + ***AdoptionReady*** indicates whether the cat is available to adopt.
  + ***Adopted*** indicates if the cat was adopted.
* ***Donations*** can be either item, cash, purchase, payment, or check.
  + ***DonationAmount*** is the amount of money from cash or check.
  + ***DonationCheckNumber*** is the number of a check donation.
* ***ItemType*** consists of food, medicine, toys, litter, and litter boxes.
  + ***ItemName*** is the name of the item.
  + ***ItemBrand*** is the company that created the item.
  + ***ItemQuantity*** is the amount of an item in a shelter and could be 0.
  + ***ItemDescription*** contains information related to the item.
* ***JobList*** consists of all the jobs required by the shelters.
  + ***JobName*** consists of Owner, Manager, Shelter Maintenance Technician, Secretary, Bodyguard, or Food Technician.
  + ***JobLevel*** consists of volunteer or full-time.
* ***Personnel*** consist of anyone interested in adoption, has adopted a cat, volunteers at the rescue, or donates to the shelter.
* ***Shelter*** is a location a cat resides in.
* ***ShelterLocation*** is the primary location for the Safe Haven rescue with a capacity of five cats.
  + ***ShelterName*** is Safe Haven Rescue.
  + ***ShelterCapacity*** indicates the number of cats allowed at the shelter.
  + ***ShelterAddress*** location of the shelter.

# Conceptual Model:



# Logical Model:



# Part II: Physical Database Design

/\*

Drop database objects.

\*/

DROP PROCEDURE IF EXISTS dbo.spAddJob

DROP PROCEDURE IF EXISTS dbo.spAddPersonnel

DROP PROCEDURE IF EXISTS dbo.spAddShelterLocation

DROP PROCEDURE IF EXISTS dbo.spAddCat

DROP PROCEDURE IF EXISTS dbo.spAddItem

DROP PROCEDURE IF EXISTS dbo.spUsedItem

DROP PROCEDURE IF EXISTS dbo.spDeleteItem

DROP PROCEDURE IF EXISTS dbo.spAddQuantity

DROP PROCEDURE IF EXISTS dbo.spMakeDonation

DROP FUNCTION IF EXISTS dbo.fGetJobIDFromJobName

DROP FUNCTION IF EXISTS dbo.fGetShelterIDFromShelterName

DROP FUNCTION IF EXISTS dbo.fCheckShelterAvailability

DROP FUNCTION IF EXISTS dbo.fGetItemIDFromItemName

DROP FUNCTION IF EXISTS dbo.fGetPersonID

DROP FUNCTION IF EXISTS dbo.fCheckShelterFill

DROP VIEW IF EXISTS dbo.ItemStatus

DROP VIEW IF EXISTS dbo.WeeklyDonation

DROP VIEW IF EXISTS dbo.DonationCity

DROP VIEW IF EXISTS dbo.CheckAdoptionStatus

DROP VIEW IF EXISTS dbo.AdoptionCity

GO

/\*

Drop database tables.

\*/

DROP TABLE IF EXISTS dbo.JobPersonTable

DROP TABLE IF EXISTS dbo.DonationTable

DROP TABLE IF EXISTS dbo.ItemList

DROP TABLE IF EXISTS dbo.CatList

DROP TABLE IF EXISTS dbo.PersonnelList

DROP TABLE IF EXISTS dbo.JobList

DROP TABLE IF EXISTS dbo.ShelterLocation

/\*

Create all database tables.

\*/

-- Creates the PersonnelList Table

CREATE TABLE dbo.PersonnelList (

PersonID int identity NOT NULL PRIMARY KEY,

PersonFirstName varchar(40) NOT NULL,

PersonLastName varchar(40) NOT NULL,

PersonEmail varchar(255) NOT NULL UNIQUE,

PersonAddress varchar(50),

PersonCity varchar(50),

PersonState char(2),

PersonPhone varchar(15),

)

GO

-- Creates the ShelterLocation Table

CREATE TABLE dbo.ShelterLocation (

ShelterID int identity NOT NULL PRIMARY KEY,

ShelterName varchar(40) NOT NULL UNIQUE,

ShelterCapacity int NOT NULL DEFAULT 5,

ShelterAddress varchar(50),

ShelterCity varchar(50),

ShelterState char(2),

ShelterPhone varchar(15),

)

GO

-- Creates the JobList Table

CREATE TABLE dbo.JobList (

JobID int identity NOT NULL PRIMARY KEY,

JobName varchar(40) NOT NULL,

JobLevel varchar(20) NOT NULL,

ShelterID int,

CONSTRAINT FK1\_JobList FOREIGN KEY (ShelterID) REFERENCES ShelterLocation(ShelterID),

)

GO

-- Creates the CatList Table

CREATE TABLE dbo.CatList (

CatID int identity NOT NULL PRIMARY KEY,

CatName varchar(40),

CatBirthday datetime NOT NULL,

NeuterStatus bit NOT NULL,

VaccineStatus bit NOT NULL DEFAULT 'FALSE',

HealthRequirement varchar(100) DEFAULT 'NONE',

AdoptionReady bit NOT NULL DEFAULT 'FALSE',

Adopted bit NOT NULL DEFAULT 'FALSE',

PersonID int,

ShelterID int,

CONSTRAINT FK1\_CatList FOREIGN KEY (PersonID) REFERENCES PersonnelList(PersonID),

CONSTRAINT FK2\_CatList FOREIGN KEY (ShelterID) REFERENCES ShelterLocation(ShelterID)

)

GO

-- Creates the ItemList Table

CREATE TABLE dbo.ItemList (

ItemID int identity NOT NULL PRIMARY KEY,

ItemName varchar(50) NOT NULL,

ItemBrand varchar(50) NOT NULL,

ItemType varchar(50) NOT NULL,

ItemDescription varchar(100),

ItemQuantity int NOT NULL DEFAULT 1,

ShelterID int NOT NULL

CONSTRAINT FK1\_ItemList FOREIGN KEY (ShelterID) REFERENCES ShelterLocation(ShelterID)

)

GO

-- Creates the Donation Table

CREATE TABLE dbo.DonationTable (

DonationID int identity NOT NULL PRIMARY KEY,

DonationType varchar(15) NOT NULL,

DonationAmount decimal(19,4),

DonationCheckNumber varchar(30),

DonationDate datetime NOT NULL DEFAULT GETDATE(),

ItemID int,

ShelterID int,

PersonID int NOT NULL,

CONSTRAINT FK1\_DonationTable FOREIGN KEY (ItemID) REFERENCES ItemList(ItemID),

CONSTRAINT FK2\_DonationTable FOREIGN KEY (ShelterID) REFERENCES ShelterLocation(ShelterID),

CONSTRAINT FK3\_DonationTable FOREIGN KEY (PersonID) REFERENCES PersonnelList(PersonID),

)

GO

CREATE TABLE dbo.JobPersonTable (

JobPersonID int identity NOT NULL PRIMARY KEY,

PersonID int NOT NULL,

JobID int NOT NULL,

CONSTRAINT FK1\_JobPersonTable FOREIGN KEY (PersonID) REFERENCES PersonnelList(PersonID),

CONSTRAINT FK2\_JobPersonTable FOREIGN KEY (JobID) REFERENCES JobList(JobID)

)

GO

/\*

Create all functions.

\*/

-- Gets a JobID when the JobName is known.

CREATE OR ALTER FUNCTION dbo.fGetJobIDFromJobName (@JobName varchar(40))

RETURNS int AS

BEGIN

-- Declare variable to hold the result

DECLARE @ReturnValue int

-- Select the JobID from the JobName.

SELECT @ReturnValue = JobID FROM JobList WHERE JobName = @JobName

-- Return the JobID.

RETURN @ReturnValue

END;

GO

-- Gets an ItemID when the ItemName is known.

CREATE OR ALTER FUNCTION dbo.fGetItemIDFromItemName (@ItemName varchar(50))

RETURNS int AS

BEGIN

-- Declare variable to hold the result

DECLARE @ReturnValue int

-- Select the ItemID from the ItemName.

SELECT @ReturnValue = ItemID FROM ItemList WHERE ItemName = @ItemName

-- Return the ItemID.

RETURN @ReturnValue

END;

GO

-- Gets a ShelterID when the ShelterName is known.

CREATE OR ALTER FUNCTION dbo.fGetShelterIDFromShelterName (@ShelterName varchar(40))

RETURNS int AS

BEGIN

-- Declare variable to hold the result.

DECLARE @ReturnValue int

-- Select the ShelterID from the ShelterName.

SELECT @ReturnValue = ShelterID FROM ShelterLocation WHERE ShelterName = @ShelterName

-- Return the ShelterID.

RETURN @ReturnValue

END;

GO

-- Gets a PersonID with a first and last name.

CREATE OR ALTER FUNCTION dbo.fGetPersonID (@PersonEmail varchar(255))

RETURNS int AS

BEGIN

-- Declare variable to hold the result.

DECLARE @ReturnValue int

-- Select the PersonID from the PersonEmail.

SELECT @ReturnValue = PersonID FROM PersonnelList WHERE PersonEmail = @PersonEmail

-- Return the PersonID.

RETURN @ReturnValue

END;

GO

-- Check shelter capacity and availability before adding a cat.

CREATE OR ALTER FUNCTION dbo.fCheckShelterAvailability (@ShelterName varchar(255))

RETURNS int AS

BEGIN

-- Declare variable to hold the result.

DECLARE @ReturnValue int

-- Finds the number of cats in the shelter by ShelterLocation and subtracts that from the ShelterLocation capacity.

SELECT @ReturnValue = (SELECT SUM(ShelterCapacity) FROM ShelterLocation WHERE ShelterName = @ShelterName) - (SELECT COUNT(ShelterID) FROM CatList WHERE ShelterID = dbo.fGetShelterIDFromShelterName(@ShelterName))

-- Return the ShelterID.

RETURN @ReturnValue

END;

GO

-- Check current shelter fill.

CREATE OR ALTER FUNCTION dbo.fCheckShelterFill (@ShelterName varchar(255))

RETURNS int AS

BEGIN

-- Declare variable to hold the result.

DECLARE @ReturnValue int

-- Finds the number of cats in the shelter by ShelterLocation and subtracts that from the ShelterLocation capacity.

SELECT @ReturnValue = (SELECT COUNT(ShelterID) FROM CatList WHERE ShelterID = dbo.fGetShelterIDFromShelterName(@ShelterName))

-- Return the ShelterID.

RETURN @ReturnValue

END;

GO

/\*

ShelterLocation.

\*/

-- Stored Procedure to add a new shelter location to the ShelterLocation table.

CREATE OR ALTER PROCEDURE dbo.spAddShelterLocation (

@ShelterName varchar(40),

@ShelterCapacity int = 1,

@ShelterAddress varchar(50),

@ShelterCity varchar(50),

@ShelterState char(2),

@ShelterPhone varchar(15)

)

AS

BEGIN

-- Add to PersonnelList

INSERT INTO ShelterLocation(

ShelterName, ShelterCapacity,ShelterAddress, ShelterCity, ShelterState, ShelterPhone

)

-- Using the given variables and the function GetPersonID to obtain the PersonID from PersonEmail

VALUES (

@ShelterName, @ShelterCapacity, @ShelterAddress, @ShelterCity, @ShelterState, @ShelterPhone

)

RETURN @@identity

END;

GO

-- Adding new shelter locations using the spAddShelter Stored Procedure and the fGetPersonID function

EXEC dbo.spAddShelterLocation

@ShelterName = 'Johnson Estate',

@ShelterCapacity = 5,

@ShelterAddress = '123 Address Way',

@ShelterCity = 'Fredericksburg',

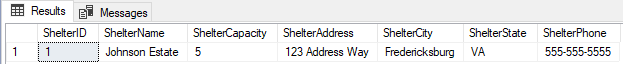
@ShelterState = 'VA',

@ShelterPhone = '555-555-5555'

GO

SELECT \* FROM dbo.ShelterLocation

GO



/\*

JobList.

\*/

-- Stored Procedure to add a job to the JobList.

-- Required variables: JobName, JobLevel.

CREATE OR ALTER PROCEDURE dbo.spAddJob (

@JobName varchar(40),

@JobLevel varchar(20),

@ShelterName varchar(40)

)

AS

BEGIN

-- Add to JobList.

INSERT INTO JobList (

JobName,

JobLevel,

ShelterID

)

-- Using the given variables JobName and JobLevel.

VALUES (

@JobName,

@JobLevel,

dbo.fGetShelterIDFromShelterName(@ShelterName)

)

RETURN @@identity

END;

GO

-- Adding new jobs to the JobList using the Stored Procedure spAddJob.

EXEC dbo.spAddJob

@JobName = 'Veterinarian',

@JobLevel = 'Full-Time',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Shelter Maintenance Technician',

@JobLevel = 'Volunteer',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Bodyguard',

@JobLevel = 'Volunteer',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Food Technician',

@JobLevel = 'Volunteer',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Secretary',

@JobLevel = 'Volunteer',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Owner',

@JobLevel = 'Full-Time',

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddJob

@JobName = 'Manager',

@JobLevel = 'Full-Time',

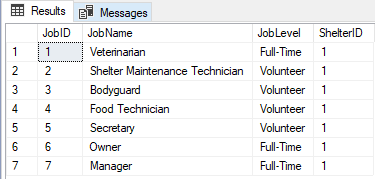
@ShelterName = 'Johnson Estate'

GO

-- Business Rule: The shelter can have one or many jobs.

SELECT \* FROM dbo.JobList

GO



/\*

PersonnelList.

\*/

-- Stored Procedure to add an item to the PersonnelList table.

-- Required variables: @PersonFirstName, @PersonLastName, @PersonEmail.

-- Optional variables: @PersonAddress, @PersonCity, @PersonState, @PersonPhone, @JobName.

CREATE OR ALTER PROCEDURE dbo.spAddPersonnel (

@PersonFirstName varchar(40),

@PersonLastName varchar(40),

@PersonEmail varchar(255),

@PersonAddress varchar(100) = NULL,

@PersonCity varchar(50) = NULL,

@PersonState char(2) = NULL,

@PersonPhone varchar(15) = NULL,

@JobName varchar(40) = NULL

)

AS

BEGIN

-- Check if the person being added will have a job.

IF @JobName = NULL

BEGIN

-- If no job, add to PersonnelList without a job.

INSERT INTO PersonnelList (

PersonFirstName, PersonLastName, PersonEmail, PersonAddress, PersonCity, PersonState, PersonPhone

)

-- Using the given variables.

VALUES (

@PersonFirstName, @PersonLastName, @PersonEmail, @PersonAddress, @PersonCity, @PersonState, @PersonPhone

)

RETURN @@identity

END

ELSE

-- If person will have a job, add person to PersonnelList and to the JobPersonTable bridge table with the PersonID and JobID.

BEGIN

DECLARE @GetJobID AS int

DECLARE @GetPersonID AS int

-- Get the JobID from the @JobName with the fGetJobIDFromJobName Function.

SET @GetJobID = dbo.fGetJobIDFromJobName(@JobName)

-- Add to the PersonnelList.

INSERT INTO PersonnelList (

PersonFirstName, PersonLastName, PersonEmail, PersonAddress, PersonCity, PersonState, PersonPhone

)

-- Using the given variables.

VALUES (

@PersonFirstName, @PersonLastName, @PersonEmail, @PersonAddress, @PersonCity, @PersonState, @PersonPhone

)

-- Get the PersonID after creating the person.

SET @GetPersonID = @@IDENTITY

-- Add to the JobPersonTable both the PersonID and the JobID from the @GetJobID and @GetPersonID variables.

INSERT INTO JobPersonTable (PersonID, JobID)

VALUES (@GetPersonID, @GetJobID)

RETURN @@IDENTITY

END

END;

GO

-- Adding a new person to the PersonnelList with a job from the JobList. Adding a JobName is optional.

EXEC dbo.spAddPersonnel

@PersonFirstName = 'Michael',

@PersonLastName = 'Johnson',

@PersonEmail = 'MJohns39@syr.edu',

@PersonAddress = '123 Address Way',

@PersonCity = 'Fredericksburg',

@PersonState = 'VA',

@PersonPhone = '555-555-5555',

@JobName = 'Manager'

EXEC dbo.spAddPersonnel

@PersonFirstName = 'Savannah',

@PersonLastName = 'Johnson',

@PersonEmail = 'SJohnson@anemail.vet',

@PersonAddress = '123 Address Way',

@PersonCity = 'Fredericksburg',

@PersonState = 'VA',

@PersonPhone = '555-555-5588',

@JobName = 'Veterinarian'

EXEC dbo.spAddPersonnel

@PersonFirstName = 'Pete',

@PersonLastName = 'Catski',

@PersonEmail = 'PCat@notacat.feline',

@PersonAddress = '462 Nyan Drive',

@PersonCity = 'Middleburg',

@PersonState = 'VA',

@PersonPhone = '555-555-5577',

@JobName = 'Shelter Maintenance Technician'

EXEC dbo.spAddPersonnel

@PersonFirstName = 'Crookshanks',

@PersonLastName = 'Micecatcher',

@PersonEmail = 'MiceRCrooks@hogwarts.edu',

@PersonAddress = '1961 Portrait Place',

@PersonCity = 'Springfield',

@PersonState = 'VA',

@PersonPhone = '555-555-5544',

@JobName = 'Food Technician'

EXEC dbo.spAddPersonnel

@PersonFirstName = 'Socks',

@PersonLastName = 'Clinton',

@PersonEmail = 'Socks\_Press@whitehouse.gov',

@PersonAddress = '1 White House',

@PersonCity = 'Washington',

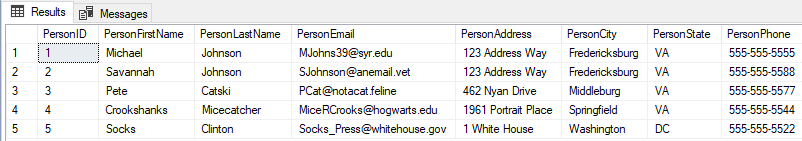
@PersonState = 'DC',

@PersonPhone = '555-555-5522',

@JobName = 'Secretary'

GO

SELECT \* FROM dbo.PersonnelList

GO

-- Business Rule: A person may have multiple jobs.

-- Adding an Owner for the Safe Haven Cat Shelter with the person's email and the job name Owner.

INSERT INTO dbo.JobPersonTable(PersonID, JobID) VALUES (dbo.fGetPersonID('SJohnson@anemail.vet'),dbo.fGetJobIDFromJobName('Owner'))

GO

-- Showing each person who has a job at the shelter.

SELECT

-- Combine first name and last name from the personnel list.

PersonnelList.PersonFirstName + ' ' + PersonnelList.PersonLastName AS [First and Last Name],

JobList.JobName

FROM PersonnelList

-- Join the JobPerson bridge table on the PersonnelList table based on PersonID.

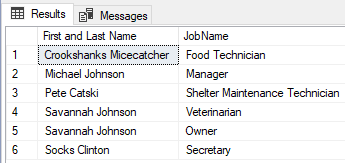
JOIN dbo.JobPersonTable ON PersonnelList.PersonID = JobPersonTable.PersonID

-- Join the JobList with the JobPersonTable bridge based on JobID.

JOIN dbo.JobList ON JobList.JobID = JobPersonTable.JobID

ORDER BY PersonnelList.PersonFirstName

GO



/\*

CatList.

\*/

-- Stored Procedure to add a new cat with a specified shelter location.

CREATE OR ALTER PROCEDURE dbo.spAddCat (

@CatName varchar(40),

@CatBirthday datetime,

@NeuterStatus bit,

-- Business Rules: A cat has the default statuses of not vaccinated, not adoption ready, and not adopted once rescued.

@VaccineStatus bit = 0,

@HealthRequirement varchar(50) = 'None',

@AdoptionReady bit = 0,

@Adopted bit = 0,

@ShelterName varchar(40),

@PersonID int = NULL

)

AS

BEGIN

-- Checks if the shelter has any availability using the fCheckShelterAvailability Function.

-- Business Rule: The shelter can hold zero to five cats.

IF(dbo.fCheckShelterAvailability(@ShelterName) > 0)

BEGIN

-- Add to CatList

INSERT INTO CatList (

CatName,

CatBirthday,

NeuterStatus,

VaccineStatus,

HealthRequirement,

AdoptionReady,

Adopted,

ShelterID,

PersonID

)

-- Using the given variables.

VALUES (

@CatName,

@CatBirthday,

@NeuterStatus,

@VaccineStatus,

@HealthRequirement,

@AdoptionReady,

@Adopted,

dbo.fGetShelterIDFromShelterName(@ShelterName),

@PersonID

)

END

ELSE

-- End the Stored Procedure if the shelter is at max capacity.

BEGIN

PRINT('No Availability in Shelter')

RETURN

END

END

GO

-- Adding a new cat with default values for @VaccineStatus, @HealthRequirement, @AdoptionReady, @Adopted, @PersonID

EXEC dbo.spAddCat

@CatName = 'Athena',

@CatBirthday = '11/2/2012',

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

EXEC spAddCat

@CatName = 'Aries',

@CatBirthday = '2/3/2015',

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

EXEC spAddCat

@CatName = 'Gerald',

@CatBirthday = '6/12/2019',

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

-- Adding a new cat without default values

EXEC spAddCat

@CatName = 'Frank',

@CatBirthday = '2/10/2021',

@VaccineStatus = 'TRUE',

@HealthRequirement = 'Specialized Food',

@AdoptionReady = 'TRUE',

@Adopted = 'FALSE',

@PersonID = NULL,

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

EXEC spAddCat

@CatName = 'Ruby',

@CatBirthday = '7/17/2020',

@VaccineStatus = 'TRUE',

@HealthRequirement = 'None',

@AdoptionReady = 'TRUE',

@Adopted = 'FALSE',

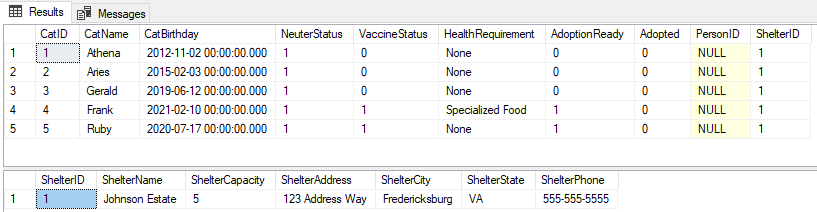
@PersonID = NULL,

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

SELECT \* FROM CatList

SELECT \* FROM ShelterLocation



-- Trying to add a cat to a full shelter

EXEC spAddCat

@CatName = 'Beatrice',

@CatBirthday = '12/31/2020',

@VaccineStatus = 'FALSE',

@HealthRequirement = 'NONE',

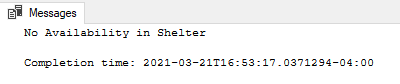
@AdoptionReady = 'FALSE',

@Adopted = 'FALSE',

@PersonID = NULL,

@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'



-- Finding a new home for Frank (the cat) creates some space in the shelter, thanks Pete!

UPDATE dbo.CatList

SET PersonID = dbo.fGetPersonID('PCat@notacat.feline'), Adopted = 'TRUE', ShelterID = NULL

WHERE CatName = 'Frank'

-- Re-adding Beatrice to Johnson Estate now that Frank was adopted.

EXEC spAddCat

@CatName = 'Beatrice',

@CatBirthday = '12/31/2020',

@VaccineStatus = 'FALSE',

@HealthRequirement = 'NONE',

@AdoptionReady = 'FALSE',

@Adopted = 'FALSE',

@PersonID = NULL,

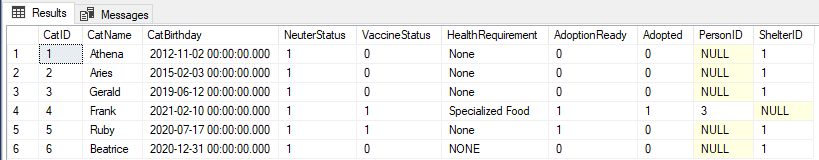
@NeuterStatus = 'TRUE',

@ShelterName = 'Johnson Estate'

GO

SELECT \* FROM dbo.CatList

GO



/\*

Items.

\*/

-- Stored Procedure to add an item to the ItemList table.

CREATE OR ALTER PROCEDURE dbo.spAddItem (

@ItemName varchar(50),

@ItemBrand varchar(50),

@ItemType varchar(50),

@ItemDescription varchar(100),

@ItemQuantity int,

@ShelterName varchar(40)

)

AS

BEGIN

-- Add to ItemList.

INSERT INTO ItemList (

ItemName, ItemBrand, ItemType, ItemDescription, ItemQuantity, ShelterID

)

-- Using the given variables and the function fGetShelterIDFromShelterName to obtain the ShelterID from the @ShelterName.

VALUES (

@ItemName, @ItemBrand, @ItemType, @ItemDescription, @ItemQuantity, dbo.fGetShelterIDFromShelterName(@ShelterName)

)

RETURN @@identity

END;

GO

-- INSERT Items into the ItemList with the following values: ItemName, ItemBrand, ItemType, ItemDescription, ItemQuantity.

INSERT dbo.ItemList(ItemName, ItemBrand, ItemType, ItemDescription, ItemQuantity, ShelterID) VALUES

('True Nature Turkey and Chicken Entree', 'Purina Pro Plan', 'Food', 'Chunks in Gravy', 16, dbo.fGetShelterIDFromShelterName('Johnson Estate')),

('Kitten Classic Pate', 'Purina Fancy Feast', 'Food', 'Pate', 24, dbo.fGetShelterIDFromShelterName('Johnson Estate'))

GO

-- Using the spAddItem Stored Procedure to add items into the ItemList

EXEC dbo.spAddItem

@ItemName = 'Hydrolyzed Protein',

@ItemBrand = 'Royal Canin',

@ItemType = 'Speacilized Food',

@ItemDescription = 'Dry',

@ItemQuantity = 5,

@ShelterName = 'Johnson Estate'

GO

EXEC dbo.spAddItem

@ItemName = 'Scented Clumping Clay Cat Litter',

@ItemBrand = 'Fresh Step',

@ItemType = 'Litter',

@ItemDescription = 'Multi-Cat',

@ItemQuantity = 3,

@ShelterName = 'Johnson Estate'

GO

EXEC dbo.spAddItem

@ItemName = 'Wilderness Chicken Recipe',

@ItemBrand = 'Blue Buffalo',

@ItemType = 'Food',

@ItemDescription = 'Dry',

@ItemQuantity = 2,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Multi-Cat Clumping Litter',

@ItemBrand = 'Arm and Hammer',

@ItemType = 'Litter',

@ItemDescription = 'Multi-Cat',

@ItemQuantity = 12,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'High Sided Cat Litter Box',

@ItemBrand = 'Frisco',

@ItemType = 'Litter Box',

@ItemDescription = 'Extra Large',

@ItemQuantity = 4,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Cyclosporine Oral Solution',

@ItemBrand = 'Atopica',

@ItemType = 'Medicine',

@ItemDescription = 'Feline Dermatitis',

@ItemQuantity = 2,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Flee and Tick Spot Treatment',

@ItemBrand = 'Frontline Plus',

@ItemType = 'Medicine',

@ItemDescription = 'Over 1.5 Pounds',

@ItemQuantity = 3,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Nutrish Natural Chicken and Brown Rice',

@ItemBrand = 'Rachel Ray',

@ItemType = 'Food',

@ItemDescription = 'Dry',

@ItemQuantity = 6,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Flea Spot Treatment For Cats',

@ItemBrand = 'Advantage II',

@ItemType = 'Medicine',

@ItemDescription = 'Over 9 pounds',

@ItemQuantity = 7,

@ShelterName = 'Johnson Estate'

EXEC dbo.spAddItem

@ItemName = 'Tidy Cats Unscented Non-Clumping Cat Litter',

@ItemBrand = 'Purina',

@ItemType = 'Litter',

@ItemDescription = 'Multi-Cat',

@ItemQuantity = 4,

@ShelterName = 'Johnson Estate'

SELECT \* FROM ItemList

GO



-- Changing the ItemType for 'Specialized Food' due to error in spelling where the ItemType has the wrong spelling.

UPDATE dbo.ItemList

SET ItemType = 'Specialized Food' -- Correct Spelling.

WHERE ItemType = 'Speacilized Food' -- Incorrect Spelling.

GO

-- Manually deleting a medicine after the last one has been used.

DELETE FROM dbo.ItemList

WHERE ItemName = 'True Nature Turkey and Chicken Entree'

GO

SELECT \* FROM ItemList

GO



-- Stored Procedure to update item quantities in the ItemList table after using an item.

-- Required variables: @ItemName, @ItemBrand, @NumberOfItemUsed

CREATE OR ALTER PROCEDURE dbo.spUsedItem(

@ItemName varchar(50),

@ItemBrand varchar(50),

@NumberOfItemUsed int

)

AS

BEGIN

-- Find the current item quantity

DECLARE @CurrentQuantity AS int

SET @CurrentQuantity = (

SELECT ItemQuantity AS CurrentQuantity FROM ItemList

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

)

-- Check if used item will drop the item to 0 quantity.

IF(@CurrentQuantity - @NumberOfItemUsed > 0)

BEGIN

-- Update the item with the correct quantity

UPDATE ItemList

SET ItemQuantity = ItemQuantity - @NumberOfItemUsed

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

RETURN @@identity

END

ELSE

BEGIN

-- Delete the item if the quantity falls below 0

DELETE FROM dbo.ItemList

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

END

END

GO

-- Updating an item quantity amount when an item is used using the spUsedItem Stored Procedure.

EXEC spUsedItem

@ItemName = 'Scented Clumping Clay Cat Litter',

@ItemBrand = 'Fresh Step',

@NumberOfItemUsed = 2

GO

SELECT \* FROM ItemList

GO



-- Deleting an item in the ItemList when the item quantity reaches 0 using the spUsedItem Stored Procedure.

-- Business Rules: An item can have one or more as the quantity. An item is dropped from the database when the quantity is 0.

EXEC spUsedItem

@ItemName = 'Scented Clumping Clay Cat Litter',

@ItemBrand = 'Fresh Step',

@NumberOfItemUsed = 1

GO

SELECT \* FROM ItemList

GO



-- Stored Procedure to delete an item from the ItemList table.

-- Required variables: ItemName, ItemBrand

CREATE OR ALTER PROCEDURE dbo.spDeleteItem (

@ItemName varchar(50),

@ItemBrand varchar(50)

)

AS

BEGIN

DELETE ItemList

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

RETURN @@identity

END;

GO

-- Deleting an item from the database using the spDeleteItem Stored Procedure.

EXEC dbo.spDeleteItem

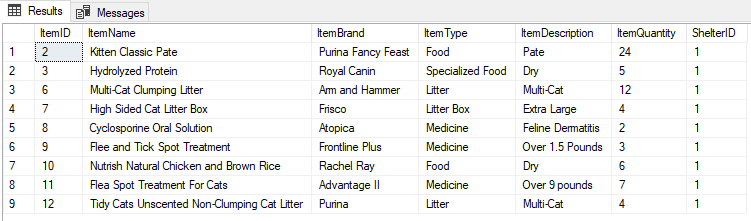
@ItemName = 'Wilderness Chicken Recipe',

@ItemBrand = 'Blue Buffalo'

GO

SELECT \* FROM dbo.ItemList

GO



/\*

Donation.

\*/

-- Stored Procedure to add item quantity to an item in the ItemList.

-- Required variables: @ItemName, @ItemBrand, @NumberOfItemsAdded

CREATE OR ALTER PROCEDURE dbo.spAddQuantity(

@ItemName varchar(50),

@ItemBrand varchar(50),

@NumberOfItemsAdded int

)

AS

BEGIN

DECLARE @CurrentQuantity AS int

SET @CurrentQuantity = (

SELECT ItemQuantity AS CurrentQuantity FROM ItemList

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

)

UPDATE ItemList

SET ItemQuantity = ItemQuantity + @NumberOfItemsAdded

WHERE ItemName = @ItemName AND ItemBrand = @ItemBrand

RETURN @@identity

END

GO

-- Stored Procedure to make a donation.

-- Required variables: @DonationType, @ShelterName, @PersonEmail

-- Optional variables: @DonationAmount, @DonationCheckNumber, @ItemName, @ItemBrand, @ItemType, @ItemQuantity, @ItemDescription

CREATE OR ALTER PROCEDURE dbo.spMakeDonation (

@DonationType varchar(15),

@DonationAmount decimal(19,4) = 0,

@DonationCheckNumber varchar(30) = NULL,

@ShelterName varchar(40),

@PersonEmail varchar(255),

@ItemName varchar(50) = NULL,

@ItemBrand varchar (50) = NULL,

@ItemType varchar(50) = NULL,

@ItemQuantity int = NULL,

@ItemDescription varchar(100) = NULL

)

AS

BEGIN

-- Check the donation type.

IF @DonationType = 'Item'

BEGIN

-- Look for the item in the ItemList.

IF NOT EXISTS (SELECT TOP 1 ItemName FROM ItemList WHERE ItemName = @ItemName)

BEGIN

DECLARE @NewItemID AS int

-- Add new item to the ItemList.

EXEC dbo.spAddItem

@ItemName = @ItemName,

@ItemBrand = @ItemBrand,

@ItemType = @ItemType,

@ItemQuantity = @ItemQuantity,

@ItemDescription = @ItemDescription,

@ShelterName = @ShelterName

SET @NewItemID = @@IDENTITY

--Checks if the item was purchased.

IF @DonationAmount = NULL

BEGIN

-- Add donation to donation table with ItemID, ShelterID from function, and PersonID from function.

INSERT INTO dbo.DonationTable(DonationType, ShelterID, PersonID, ItemID)

VALUES (@DonationType, dbo.fGetShelterIDFromShelterName(@ShelterName), dbo.fGetPersonID(@PersonEmail), @NewItemID)

RETURN @@IDENTITY

END

ELSE

BEGIN

-- Add donation to donation table with ItemID, ShelterID from function, and PersonID from function.

INSERT INTO dbo.DonationTable(DonationType, ShelterID, DonationAmount, PersonID, ItemID)

VALUES (@DonationType, dbo.fGetShelterIDFromShelterName(@ShelterName), @DonationAmount, dbo.fGetPersonID(@PersonEmail), @NewItemID)

RETURN @@IDENTITY

END

END

ELSE

BEGIN

-- Update the item quantity if it does exist in the data base using the spAddQuantity Stored Procedure.

EXEC dbo.spAddQuantity

@ItemName = @ItemName,

@ItemBrand = @ItemBrand,

@NumberOfItemsAdded = @ItemQuantity

-- Add donation to donation table with ItemID, ShelterID from function, and PersonID from function.

IF @DonationAmount = NULL

BEGIN

INSERT INTO dbo.DonationTable(DonationType, ShelterID, PersonID, ItemID)

VALUES (@DonationType, dbo.fGetShelterIDFromShelterName(@ShelterName), dbo.fGetPersonID(@PersonEmail), dbo.fGetItemIDFromItemName(@ItemName))

RETURN @@identity

END

ELSE

BEGIN

-- Add donation to donation table with ItemID, ShelterID from function, and PersonID from function.

INSERT INTO dbo.DonationTable(DonationType, ShelterID, DonationAmount, PersonID, ItemID)

VALUES (@DonationType, dbo.fGetShelterIDFromShelterName(@ShelterName), @DonationAmount, dbo.fGetPersonID(@PersonEmail), dbo.fGetItemIDFromItemName(@ItemName))

RETURN @@IDENTITY

END

END

END

ELSE

BEGIN

-- If donation is a check or cash, add donation to donation table with ShelterID from function and PersonID from function.

INSERT INTO dbo.DonationTable(DonationType, DonationAmount, DonationCheckNumber, ShelterID, PersonID)

VALUES (@DonationType, @DonationAmount, @DonationCheckNumber, dbo.fGetShelterIDFromShelterName(@ShelterName), dbo.fGetPersonID(@PersonEmail))

RETURN @@identity

END

END

GO

-- New Item donation.

EXEC spMakeDonation

@DonationType = 'Item',

@ShelterName = 'Johnson Estate',

@PersonEmail = 'MJohns39@syr.edu',

@ItemName = 'Disposable Cat Litter Box',

@ItemBrand = 'Natures Miracle',

@ItemType = 'Litter Box',

@ItemDescription = 'Jumbo',

@ItemQuantity = 2

GO

-- Donation made for item that already exists.

EXEC spMakeDonation

@DonationType = 'Item',

@ShelterName = 'Johnson Estate',

@PersonEmail = 'MJohns39@syr.edu',

@ItemName = 'Hydrolyzed Protein',

@ItemBrand = 'Royal Canin',

@ItemType = 'Specialized Food',

@ItemDescription = 'Dry',

@ItemQuantity = 5

GO

-- Check donation.

EXEC spMakeDonation

@DonationType = 'Check',

@ShelterName = 'Johnson Estate',

@DonationAmount = 200.00,

@DonationCheckNumber = '1234567899876541001',

@PersonEmail = 'SJohnson@anemail.vet'

EXEC spMakeDonation

@DonationType = 'Check',

@ShelterName = 'Johnson Estate',

@DonationAmount = 200.00,

@DonationCheckNumber = '1274567899876541001',

@PersonEmail = 'MJohns39@syr.edu'

EXEC spMakeDonation

@DonationType = 'Check',

@ShelterName = 'Johnson Estate',

@DonationAmount = 80.00,

@DonationCheckNumber = '1274567899876541001',

@PersonEmail = 'Socks\_Press@whitehouse.gov'

GO

EXEC spMakeDonation

@DonationType = 'Cash',

@ShelterName = 'Johnson Estate',

@DonationAmount = 200.00,

@PersonEmail = 'SJohnson@anemail.vet'

GO

-- Payment for the contracted veterinarian.

-- Business Rules: Payment for the veterinarian service comes out of the donation table as a negative donation.

EXEC spMakeDonation

@DonationType = 'Payment',

@ShelterName = 'Johnson Estate',

@DonationAmount = -200.00,

@PersonEmail = 'SJohnson@anemail.vet'

GO

-- Business Rules: A purchased item for the shelter also comes out of the donation funds as a negative donation with the addition of an item.

EXEC spMakeDonation

@DonationType = 'Item',

@ShelterName = 'Johnson Estate',

@PersonEmail = 'MJohns39@syr.edu',

@DonationAmount = -30.00,

@ItemName = 'Tidy Cats Unscented Non-Clumping Cat Litter',

@ItemBrand = 'Purina',

@ItemType = 'Litter',

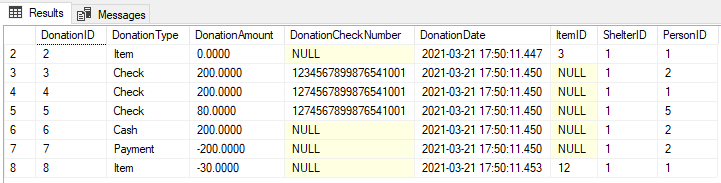
@ItemDescription = 'Multi-Cat',

@ItemQuantity = 6

GO

SELECT \* FROM DonationTable

GO



/\*

Views.

\*/

-- Data Question: Are more cash and check donations coming in than money going out per week?

CREATE OR ALTER VIEW dbo.WeeklyDonation AS

SELECT

-- Takes the sum of all donations as a new column named WeeklyDonationMargin.

SUM(DonationAmount) AS WeeklyDonationMargin

FROM DonationTable

-- Limits the donations based on only those made in the past week.

WHERE DonationDate > DATEDIFF(DAY, DonationDate, -7)

GO

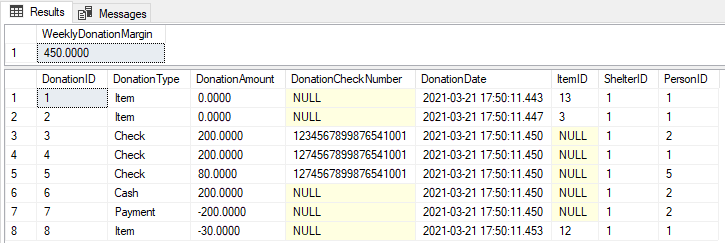
-- Run the view and compare it with the DonationTable. Looks like the shelter is +$450 for the week, awesome!

SELECT \* FROM dbo.WeeklyDonation

GO

SELECT \* FROM dbo.DonationTable

GO



-- Data Question: Where do the highest number of donations come from per month?

CREATE OR ALTER VIEW dbo.DonationCity AS

SELECT

-- Selects the city and state of a person who made a donation as a new City column.

PersonnelList.PersonCity + ', ' + PersonnelList.PersonState AS City,

-- Counts the number of donations made by city as a new column called DonationByCity.

COUNT(PersonnelList.PersonCity) AS DonationByCity

FROM PersonnelList

-- Joins the DonationTable and the PersonnelList taking only the people who have made donations.

JOIN DonationTable ON PersonnelList.PersonID = DonationTable.PersonID

-- Limits the donations to only those made within the past month.

WHERE DonationTable.DonationDate > DATEDIFF(Month, DonationDate, -1)

-- Groups the donation by City, State.

GROUP BY PersonCity, PersonState

GO

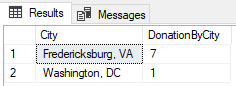
-- Runs the view, looks like Fredericksburg has the most donations. Understandable considering the shelter is in that city.

-- Washington DC with one donation could be an untapped market!

SELECT \* FROM dbo.DonationCity

ORDER BY DonationByCity DESC

GO



-- Data Question: How many days left where every cat in the shelter can have two meals per day?

CREATE OR ALTER VIEW dbo.ItemStatus AS

SELECT

-- Take the sum of all food quantity and divides that by two meals per day

-- and divides that by the number of cats in the shelter found with the fCheckShelterFill function.

((SUM(ItemQuantity) / 2) / dbo.fCheckShelterFill('Johnson Estate')) AS [FoodRemaining (Days)]

FROM ItemList

-- Limits the ItemType to only food.

WHERE ItemType = 'Food'

GO

-- Runs the view, looks like the shelter can go another 3 whole days with current food stocks.

-- Good thing the shelter is $+450 for the week!

SELECT \* FROM dbo.ItemStatus

GO



-- Data Question: Which cats are not ready for adoption and why?

CREATE OR ALTER VIEW dbo.CheckAdoptionStatus AS

SELECT

-- Selects the relevent columns of CatName, NeuterStatus, and VaccineStatus.

CatList.CatName,

CatList.NeuterStatus,

CatList.VaccineStatus

FROM CatList

-- Limits the selection to only those who have not been adopted, those who are not AdoptionReady,

-- and those who still need either a vaccine or neuter.

WHERE CatList.Adopted = 0 AND CatList.AdoptionReady <> 1 AND (CatList.NeuterStatus <> 1 OR CatList.VaccineStatus <> 1)

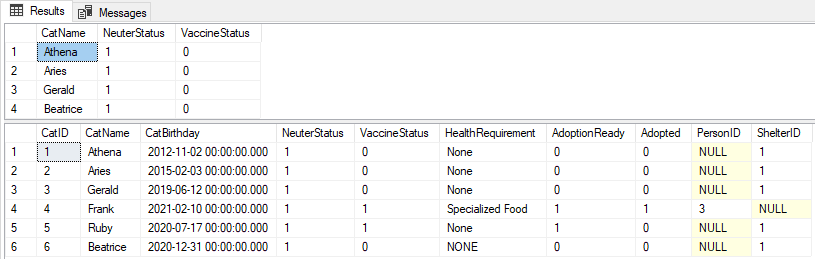
GO

-- Runs the view to check which cats still need some work to get adopted. Looks like Ruby is ready to go though!

SELECT \* FROM dbo.CheckAdoptionStatus

SELECT \* FROM CatList

GO



-- Data Question: What city would be the best for marketing based on the number of adoptions?

CREATE OR ALTER VIEW dbo.AdoptionCity AS

SELECT

-- Selects the city and state of a person who adopted a cat as a new City column.

PersonnelList.PersonCity + ', ' + PersonnelList.PersonState AS City,

-- Counts the number of adoptions by city.

COUNT(PersonnelList.PersonCity) AS AdoptionByCity

FROM PersonnelList

-- Joins the CatList and the PersonnelList taking only the people who have adopted a cat.

JOIN CatList ON PersonnelList.PersonID = CatList.PersonID

-- Groups the selection by city and state.

GROUP BY PersonCity, PersonState

GO

-- Runs the view to find out where the cats are getting adopted from.

-- Looks like the shelter could do some more marketing from within Fredericksburg, the only adoption is from another city!

SELECT \* FROM dbo.AdoptionCity

ORDER BY AdoptionByCity DESC

GO

-- The AdoptionCity view is confirmed using a view that joins the Name, City and State, and CatName with anyone who has adopted a cat.

CREATE OR ALTER VIEW dbo.CatParent AS

SELECT

PersonnelList.PersonFirstName + ' ' + PersonnelList.PersonLastName AS [Name],

PersonnelList.PersonCity + ', ' + PersonnelList.PersonState AS City,

CatList.CatName

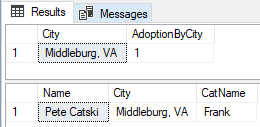
FROM PersonnelList

JOIN CatList ON PersonnelList.PersonID = CatList.PersonID

GO

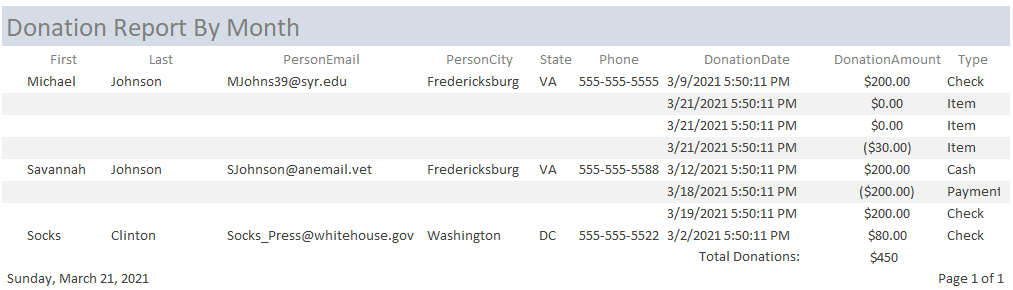
SELECT \* FROM dbo.CatParent

GO

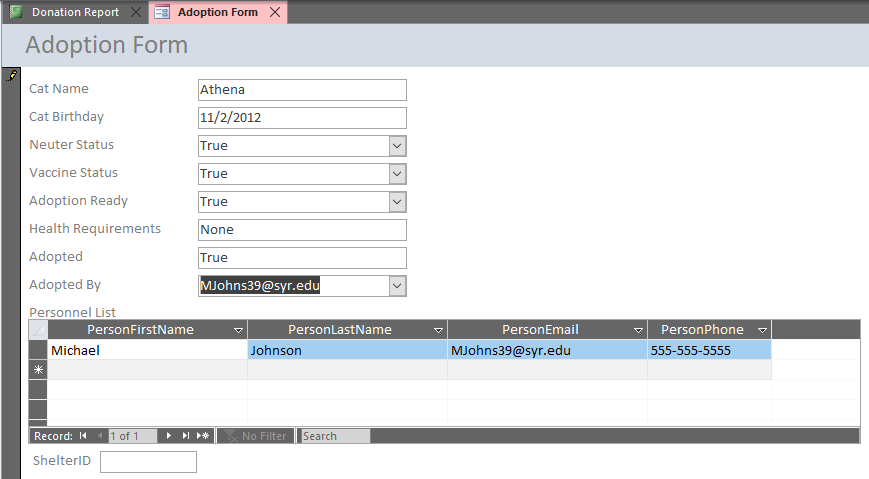


# Access Forms and Reports:

-- Report to view the number of donations made by month with a grand total at the bottom for the entire month:



-- Form to handle adoptions of each cat. Once a person is added to the data base, they can adopt a cat through this form or through SQL Server:



-- The new adoption made through Access is confirmed with the view that joins the Name, City and State, and CatName with anyone who has adopted a cat.

CREATE OR ALTER VIEW dbo.CatParent AS

SELECT

PersonnelList.PersonFirstName + ' ' + PersonnelList.PersonLastName AS [Name],

PersonnelList.PersonCity + ', ' + PersonnelList.PersonState AS City,

CatList.CatName

FROM PersonnelList

JOIN CatList ON PersonnelList.PersonID = CatList.PersonID

GO

SELECT \* FROM dbo.CatParent

GO



# Reflection:

* The assumptions I made prior to this project centered on the coding sections being the most difficult part. After starting, I quickly realized the coding aspect is probably the easiest part when looking at the entirety of the Relational Data Model. The conceptual and logical models assisted greatly with the planning portion and lead me toward the right path for completing this project. However, I still found some difficulties specifically in determining the functions and stored procedures that could help with entering and managing data within the database. In the end, I split almost everything into stored procedures with functions built in to help with getting information to and from the database. This greatly increased my overall time spent on the project but helped me come away feeling confident in using SQL.
* Next time this type of project arises, I plan to spend more time thinking about how the database fits into the organizational structure. I started off with the mindset that I wanted to build a database that had multiple shelters, foster homes, and could deal with a variety of issues but quickly realized I needed to switch to building a database suited for one specific shelter before the tables spiraled out of control. Although I feel like the freedom to do this from scratch was great, I do think working directly with an organization that already has defined rules in place would help temper myself from bloating the database.
* This process highlighted every single part of the Relational Data Model and showed me how important it was not to skip steps. I initially wanted to go straight to the coding part thinking it would be easier to plan out the logical and conceptual models, but I realized there is a good reason for the structure. Building out the models ahead of time allowed me to imagine how the data would sit in the database and that translated well when I actually sat down to specify the data types for each of my tables.

# Project Summary:

* Putting the final database together took some work trying to combine the business rules together with the database objects. The biggest help in managing the business rules came down to using stored procedures to help ensure I met all the requirements. A great example of this came from the stored procedure built to update the quantity of an item after a shelter worked used it. If the quantity of the item fell to zero, a delete statement built into the procedure kicked the item out of the database as per the business rules.
* Answering the data questions was a difficult task as this was where I realized my database was missing quite a few tables if I wanted to run the shelter like it was the size of Amazon or Google (with just as many locations). Taking a fresh look at the data questions with a scaled down database allowed me to accurately identify how I could answer them through views, functions, and carefully worded SELECT statements. Although the amount of data was a limiting factor in seeing the actual power of the database, I had fun imagining how this fake cat shelter could benefit from some simple marketing strategies built from the results of each question.
* Access was the easy choice for implementing this database as I could picture the people working within the cat rescue enjoying the simplicity of the design. I commented in the initial summary that the shelter used Microsoft Excel to manage their data so that hopefully meant the shelter already had a full license for Microsoft Access. Additionally, the flexibility of being able to code with Visual Basic Application in Microsoft Access increases the ease of creating forms that have data validation built into the data entry boxes.