the Center for the prevention of child abuse

PARENT EMPOWERMENT PROGRAM (PEP)

Database Documentation 2017

Carson Badame, Marcos Barbieri, Jimmy Crowley, Jesse Opitz, John Randis, Rachel Ulicni

Executive Summary

This document's objective and purpose is to provide a design for and implementation of a database for The Center for the Prevention of Child Abuse of Duchess County, as well as essential information regarding the architecture of the database. The CPCA requires a database to be able to store, insert, update, delete, and otherwise manipulate data, as well as view reports of data, for their business records and purposes. The database will be key in taking attendance for classes, building and viewing reports on participants, and storing class survey data.

All of the CPCA's data which was previously recorded on plain paper and stored in filing cabinets will now be accounted for in the database, and can quickly and easily be found, rather than manually storing and searching through physical pieces of paper for information. The database will be integrated with the client application, which will provide a clear front-end that will allow the users at the CPCA to easily and effectively retrieve data from the database. The implementation of the database will coincide with the use of physical paper reporting, until the system is proven effective and the users obtain a level of comfortability with the application utilizing the database, ultimately eliminating the need to use paper data storage and recovery.

The design encompasses all aspects of the CPCA's data retrieval, including—but not limited to—details about participants, classes, employees, reports, surveys, referrals and families. This database will only be accessible from the CPCA's secure network at its headquarters due to the high level of sensitive and confidential information that it contains.

Entity Relationship Diagram

An Entity Relationship Diagram, also known as an ER diagram or ERD, is a data modeling technique that graphically illustrates an information system's entities and the relationship between those entities. The elements of an ERD are:

- **Entities**
- **Attributes**
- **Relationships (Cardinality)**

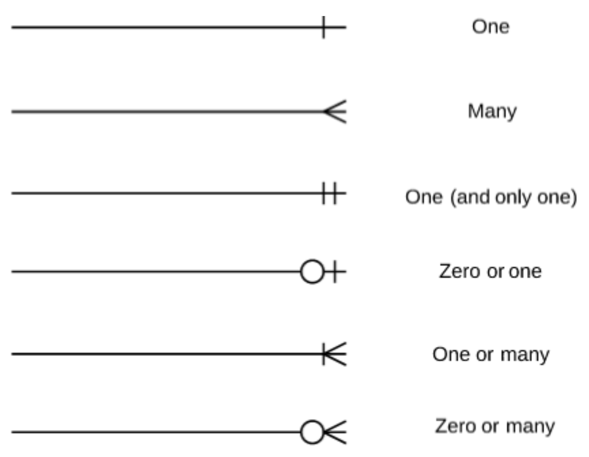
An **entity** is a real-world item or concept that exists on its own, typically a noun—such as a person, object, concept, or event—that can have data stored about it. It is represented by the tables in the database. Each row of the table is an instance of that entity. For example, a customer, student, car, or product.

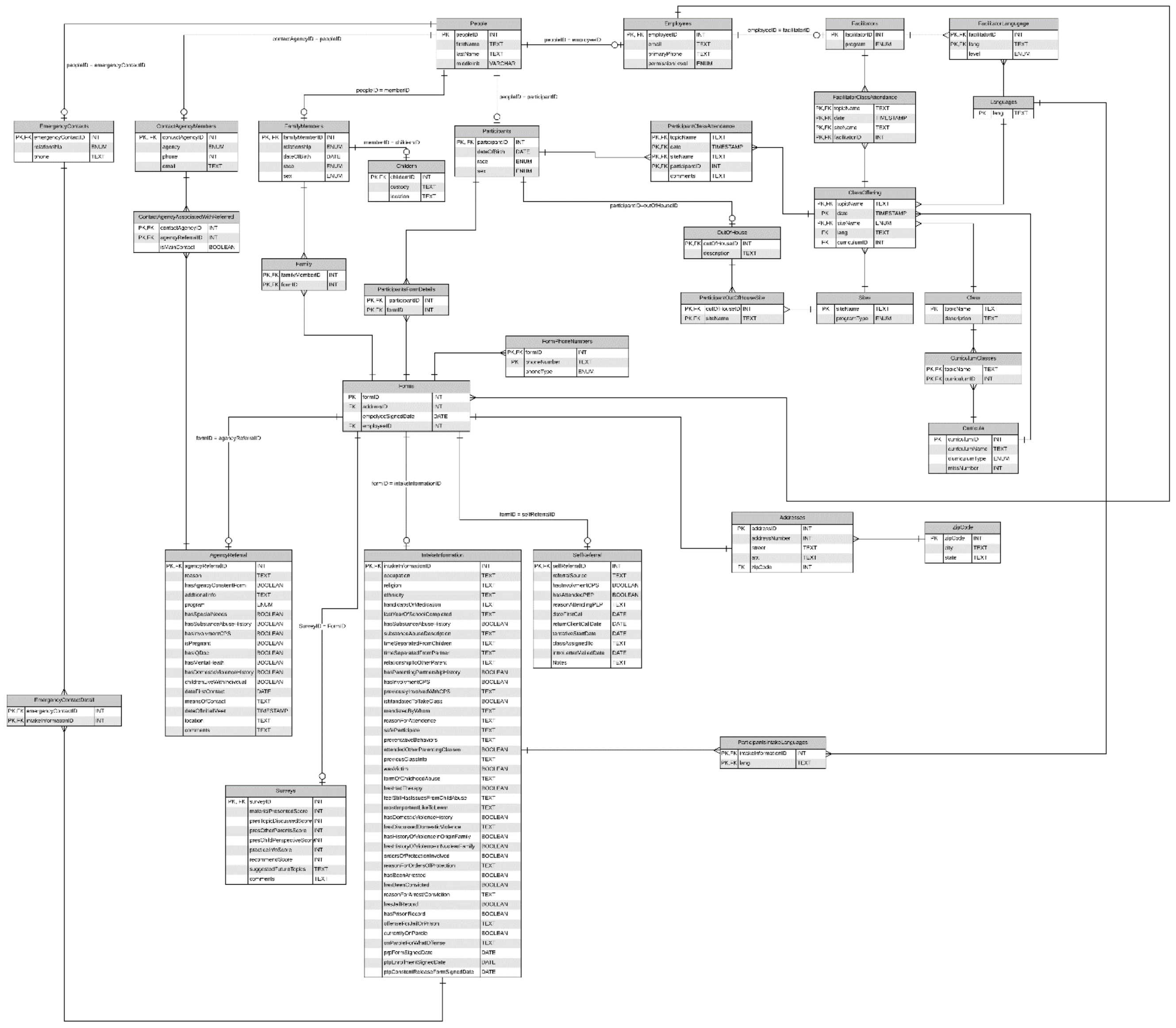
An **attribute** of an entity is a particular property that describes the entity. For example, attributes for a student entity may be: student_id, first_name, last_name, email, phone_number.

A **relationship** is the association that describes the interaction between entities, like a verb. For example, if a student registers for a course, the two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way. Relationships are typically shown as connecting lines.

Cardinality defines relationships in terms of numbers; it is the number of instances that one entity can, or must, be associated with each instance of another entity. This can be **one-to-one, one-to-many,** or **many-to-many** relationships.

- **One-to-One:** Both tables can have only one record on either side of the relationship.
  - For example, one student associated with one mailing address.
- **One-to-Many:** The first table contains only one record that relates to none, one, or many records in the second, related table.
  - For example, one student registers for multiple courses, but all those courses have a single line back to that one student.
- **Many-to-Many:** Each record in one table can relate to any number of records (or no records) in the other table.
  - Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students.




People
PK peopleID INT
firstName TEXT
lastName TEXT
middleInit VARCHAR
peopleID = employeeID
Employees
PK, FK employeeID INT
email TEXT
primaryPhone TEXT
permissionLevel ENUM
employeeID = facilitatorID
Facilitators
PK facilitatorID INT
program ENUM
FacilitatorLanguage
PK,FK facilitatorID INT
PK,FK lang TEXT
level ENUM
FacilitatorClassAttendance
PK,FK topicName TEXT
PK,FK date TIMESTAMP
PK,FK siteName TEXT
PK,FK facilitatorID INT
Languages
PK lang TEXT
peopleID = emergencyContactID
contactAgencyID = peopleID
peopleID = memberID
peopleID = participantID
EmergencyContacts
PK,FK emergencyContactID INT
relationship ENUM
phone TEXT
ContactAgencyMembers
PK, FK contactAgencyID INT
agency ENUM
phone INT
email TEXT
FamilyMembers
PK, FK familyMemberID INT
relationship ENUM
dateOfBirth DATE
race ENUM
sex ENUM
memberID = childrenID
Children
PK,FK childrenID INT
custody TEXT
location TEXT
Participants
PK, FK participantID INT
dateOfBirth DATE
race ENUM
sex ENUM
ParticipantClassAttendance
PK,FK topicName TEXT
PK,FK date TIMESTAMP
PK,FK siteName TEXT
PK,FK participantID INT
comments TEXT
ClassOffering
PK,FK topicName TEXT
PK date TIMESTAMP
PK,FK siteName ENUM
FK lang TEXT
FK curriculumID INT
participantID=outOfHouseID
OutOfHouse
PK,FK outOfHouseID INT
description TEXT
ContactAgencyAssociatedWithReferred
PK,FK contactAgencyID INT
PK,FK agencyReferralID INT
isMainContact BOOLEAN
Family
PK,FK familyMemberID INT
PK,FK formID INT
ParticipantsFormDetails
PK,FK participantID INT
PK,FK formID INT
ParticipantOutOfHouseSite
PK,FK outOfHouseID INT
PK,FK siteName TEXT
Sites
PK siteName TEXT
programType ENUM
Class
PK topicName TEXT
description TEXT
CurriculumClasses
PK,FK topicName TEXT
PK,FK curriculumID INT
Curricula
PK curriculumID INT
curriculumName TEXT
curriculumType ENUM
missNumber INT
FormPhoneNumbers
PK,FK formID INT
PK phoneNumber TEXT
phoneType ENUM
Forms
PK formID INT
FK addressID INT
employeeSignedDate DATE
FK employeeID INT
formID = agencyReferralID
formID = intakeInformationID
formID = selfReferralID
SurveyID = FormID
Addresses
PK addressID INT
addressNumber INT
street TEXT
apt TEXT
FK zipCode INT
ZipCode
PK zipCode INT
city TEXT
state TEXT
AgencyReferral
PK,FK agencyReferralID INT
reason TEXT
hasAgencyConsentForm BOOLEAN
additionalInfo TEXT
program ENUM
hasSpecialNeeds BOOLEAN
hasSubstanceAbuseHistory BOOLEAN
hasInvolvementCPS BOOLEAN
isPregnant BOOLEAN
hasIQDoc BOOLEAN
hasMentalHealth BOOLEAN
hasDomesticViolenceHistory BOOLEAN
childrenLiveWithIndividual BOOLEAN
dateFirstContact DATE
meansOfContact TEXT
dateOfInitialMeet TIMESTAMP
location TEXT
comments TEXT
IntakeInformation
PK,FK intakeInformationID INT
occupation TEXT
religion TEXT
ethnicity TEXT
handicapsOrMedication TEXT
lastYearOfSchoolCompleted TEXT
hasSubstanceAbuseHistory BOOLEAN
substanceAbuseDescription TEXT
timeSeparatedFromChildren TEXT
timeSeparatedFromPartner TEXT
relationshipToOtherParent TEXT
hasParentingPartnershipHistory BOOLEAN
hasInvolvementCPS BOOLEAN
previouslyInvolvedWithCPS TEXT
isMandatedToTakeClass BOOLEAN
mandatedByWhom TEXT
reasonForAttendence TEXT
safeParticipate TEXT
preventativeBehaviors TEXT
attendedOtherParentingClasses BOOLEAN
previousClassInfo TEXT
wasVictim BOOLEAN
formOfChildhoodAbuse TEXT
hasHadTherapy BOOLEAN
feelStillHasIssuesFromChildAbuse TEXT
mostImportantLikeToLearn TEXT
hasDomesticViolenceHistory BOOLEAN
hasDiscussedDomesticViolence TEXT
hasHistoryOfViolenceInOriginFamily BOOLEAN
hasHistoryOfViolenceInNuclearFamily BOOLEAN
ordersOfProtectionInvolved BOOLEAN
reasonForOrdersOfProtection TEXT
hasBeenArrested BOOLEAN
hasBeenConvicted BOOLEAN
reasonForArrestOrConviction TEXT
hasJailRecord BOOLEAN
hasPrisonRecord BOOLEAN
offenseForJailOrPrison TEXT
currentlyOnParole BOOLEAN
onParoleForWhatOffense TEXT
prpFormSignedDate DATE
ptpEnrollmentSignedDate DATE
ptpConstentReleaseFormSignedDate DATE
SelfReferral
PK,FK selfReferralID INT
referralSource TEXT
hasInvolvementCPS BOOLEAN
hasAttendedPEP BOOLEAN
reasonAttendingPEP TEXT
dateFirstCall DATE
returnClientCallDate DATE
tentativeStartDate DATE
classAssignedTo TEXT
introLetterMailedDate DATE
Notes TEXT
EmergencyContactDetail
PK,FK emergencyContactID INT
PK,FK intakeInformationID INT
ParticipantsIntakeLanguages
PK,FK intakeInformationID INT
PK,FK lang TEXT
Surveys
PK, FK surveyID INT
materialPresentedScore INT
presTopicDiscussedScore INT
presOtherParentsScore INT
presChildPerspectiveScore INT
practicalInfoScore INT
recommendScore INT
suggestedFutureTopics TEXT
comments TEXT

Primary and Foreign Keys

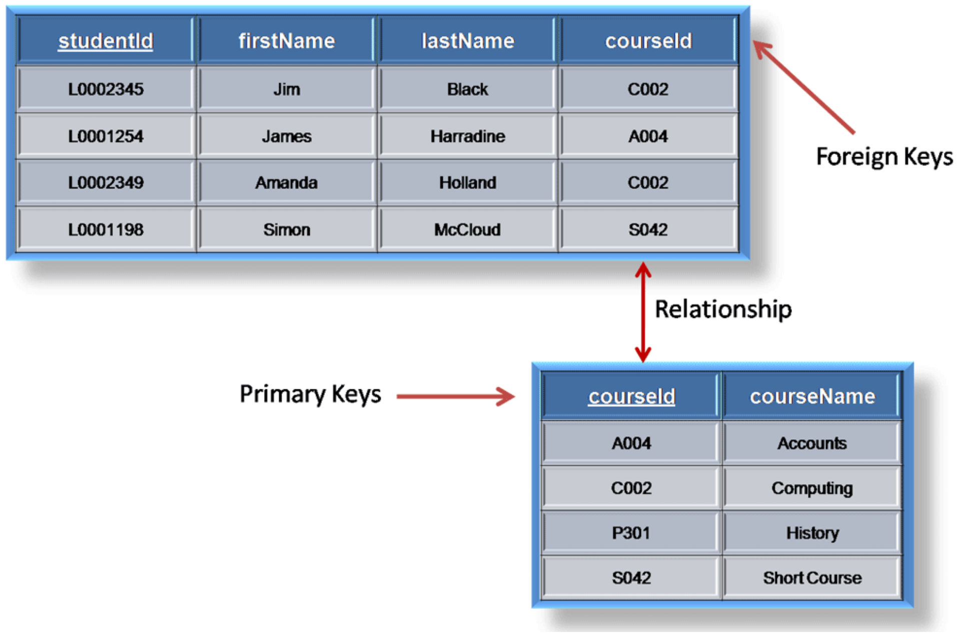
Primary keys and foreign keys are extremely important concepts and are critical to an efficient relational database. Without them, relational databases would not work. A **primary key** (PK) is a column, or columns, in a table that uniquely identifies the rows in that table. In order for a table to qualify as a relational table, it must have a primary key. The primary key's main features are:

- It must contain a unique value for each row of data.
- It cannot contain null values.

For example, the primary key in a *Students* table may be *student_id*, since that is a unique number which identifies a specific student.

A **foreign key** (FK) is a column, or columns, in a table that refers to the primary key in another table. Unlike primary keys, duplicate and null values are allowed in foreign keys. Foreign keys allow for *referential integrity*, meaning that if a foreign key contains a value, this value refers to an existing record in the related table.

In the example below, the *Courses* table has a primary key of *courseId*, which is a foreign key in the *Students* table. The cardinality of their relationship is *many-to-many*, because a student can have many courses, and a course can have many students.

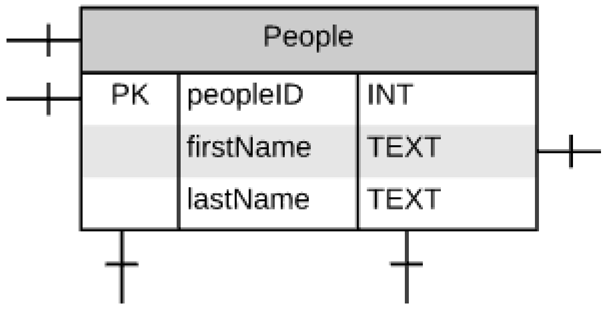


studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

Tables

People Table



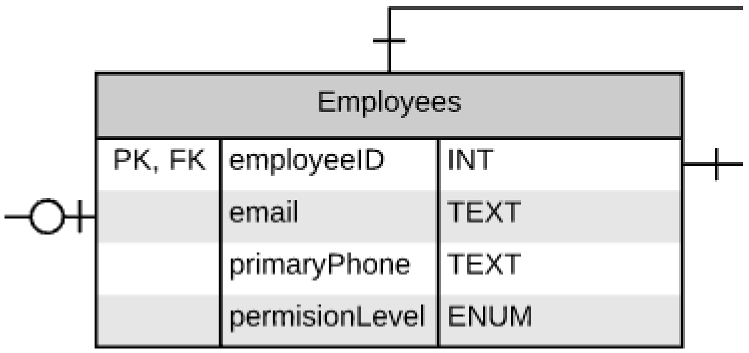
The People Table stores general data regarding any person. This table has the following attributes:

- peopleID (PK)
- firstName
- lastName

The People Table has 5 relationships and they are:

Table	Relationship	Other Table
People	One-to-Zero or One	EmergencyContacts
People	One-to-Zero or One	ContactAgencyMembers
People	One-to-Zero or One	FamilyMembers
People	One-to-Zero or One	Participants
People	One-to-Zero or One	Employees

Employees Table



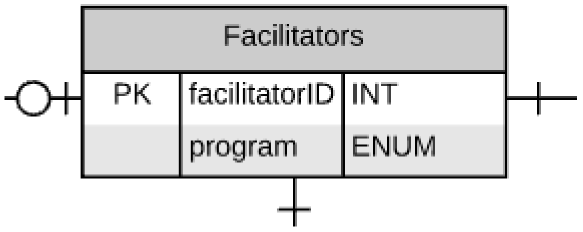
The Employees Table stores an employee's general data. The table has the following attributes:

- employeeID (PK/FK)
- email
- primaryPhone
- permissionLevel

The Employees Table has 3 relationships and they are:

Table	Relationship	Other Table
Employees	Zero or One-to-One	People
Employees	One-to-Zero or One	Facilitators
Employees	One-to-Many	Forms

Facilitators Table



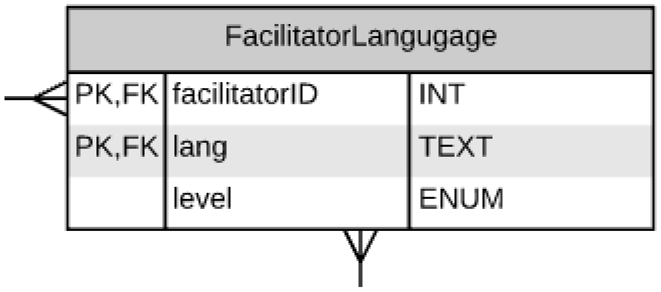
The Facilitators Table stores specific data for a facilitator type of employee. The table has the following attributes:

- facilitatorID (PK)
- program

The Facilitators Table has 3 relationships and they are:

Table	Relationship	Other Table
Facilitators	Zero or One-to-One	Employees
Facilitators	One-to-Many	FacilitatorClassAttendance
Facilitators	One-to-Many	FacilitatorLanguage

FacilitatorLanguage Table



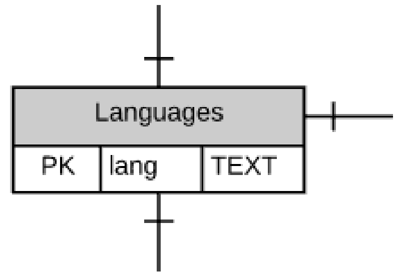
The FacilitatorLanguage Table stores data regarding the languages that a facilitator is able to speak. The table has the following attributes:

- facilitatorID (PK/FK)
- lang (PK/FK)
- level

The FacilitatorLanguage Table has 2 relationships and they are:

Table	Relationship	Other Table
FacilitatorLanguage	Many-to-One	Facilitators
FacilitatorLanguage	Many-to-One	Languages

Languages Table



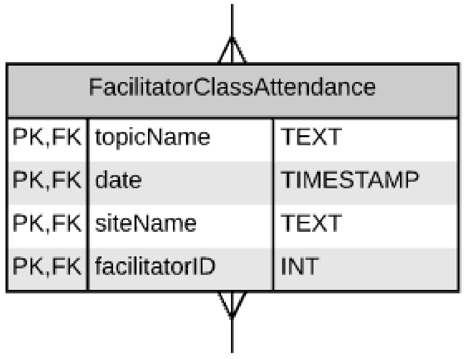
The Languages Table stores various languages. The table has the following attributes:

- lang (PK)

The Languages Table has 3 relationships and they are:

Table	Relationship	Other Table
Languages	One-to-Many	FacilitatorLanguage
Languages	One-to-many	ClassOffering
Languages	One-to-Many	ParticipantsIntakeLanguages

FacilitatorClassAttendance Table



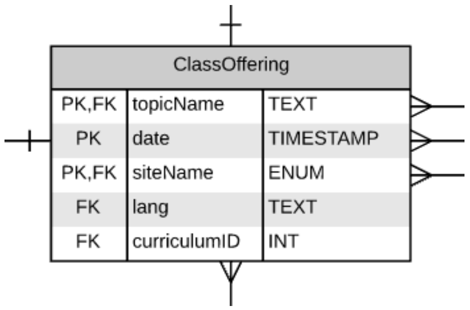
The FacilitatorClassAttendance Table stores data regarding class attendance taken by the facilitator. The table has the following attributes:

- topicName (PK/FK)
- date (PK/FK)
- siteName (PK/FK)
- facilitatorID (PK/FK)

The FacilitatorClassAttendance Table has 2 relationships and they are:

Table	Relationship	Other Table
FacilitatorClassAttendance	Many-to-One	Facilitators
FacilitatorClassAttendance	Many-to-One	ClassOffering

ClassOffering Table



ClassOffering		
PK,FK	topicName	TEXT
PK	date	TIMESTAMP
PK,FK	siteName	ENUM
FK	lang	TEXT
FK	curriculumID	INT

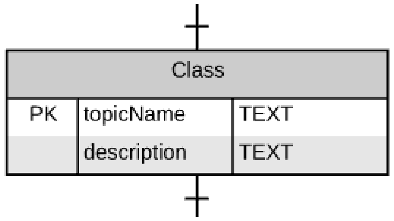
The ClassOffering Table stores data regarding class attendance taken by the facilitator. The table has the following attributes:

- topicName (PK/FK)
- date (PK)
- siteName (PK/FK)
- lang (FK)
- curriculumID (FK)

The ClassOffering Table has 6 relationships and they are:

Table	Relationship	Other Table
ClassOffering	Many-to-One	Languages
ClassOffering	Many-to-One	Curricula
ClassOffering	Many-to-One	Class
ClassOffering	Many-to-One	Sites
ClassOffering	One-to-Many	FacilitatorClassAttendance
ClassOffering	One-to-Many	ParticipantClassAttendance

Class Table



Class		
PK	topicName	TEXT
	description	TEXT

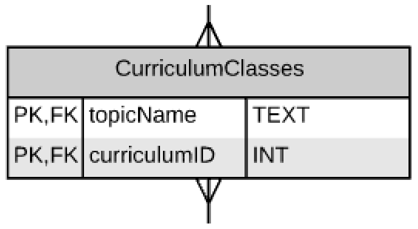
The Class Table stores basic data about classes. The table has the following attributes:

- topicName (PK)
- description

The Class Table has 2 relationships and they are:

Table	Relationship	Other Table
Class	One-to-Many	ClassOffering
Class	One-to-many	CurriculumClasses

CurriculumClasses Table



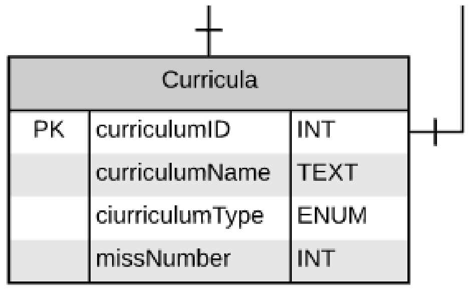
The CurriculumClass Table stores data connecting a class with a specific curriculum. The table has the following attributes:

- topicName (PK/FK)
- curriculumID (PK/FK)

The CurriculumClasses Table has 2 relationships and they are:

Table	Relationship	Other Table
CurriculumClass	Many-to-One	Class
CurriculumClass	Many-to-One	Curricula

Curricula Table



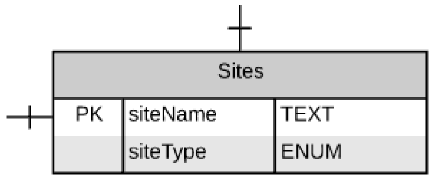
The Curricula Table stores data about specific curricula. The table has the following attributes:

- curriculumID (PK)
- curriculumName
- curriculumType
- missNumber

The Curricula Table has 2 relationships and they are:

Table	Relationship	Other Table
Curricula	One-to-Many	CurriculumClasses
ClassOffering	One-to-Many	ClassOffering

Sites Table



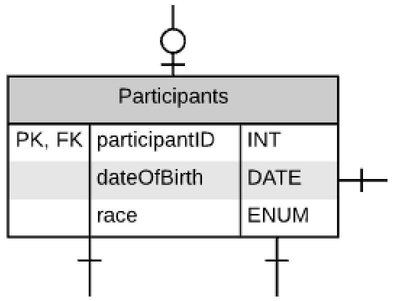
The Sites Table stores data about the type of various sites. The table has the following attributes:

- siteName (PK)
- siteType

The Sites Table has 2 relationships and they are:

Table	Relationship	Other Table
Sites	One-to-Many	ClassOffering
Sites	One-to-Many	ParticipantOutOfHouseSite

Participants Table



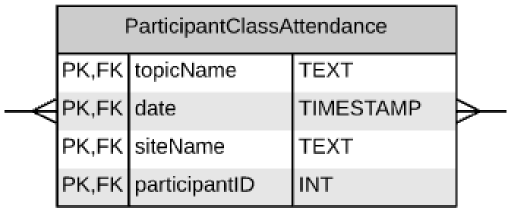
The Participants Table stores data about participants. The table has the following attributes:

- participantID (PK/FK)
- dateOfBirth
- race

The Participants Table has 4 relationships and they are:

Table	Relationship	Other Table
Participants	Zero or One-to-One	People
Participants	One-to-Many	ParticipantClassAttendance
Participants	One-to-Zero or One	OutOfHouse
Participants	One-to-Many	ParticipantsFormDetails

ParticipantClassAttendance Table



ParticipantClassAttendance		
PK,FK	topicName	TEXT
PK,FK	date	TIMESTAMP
PK,FK	siteName	TEXT
PK,FK	participantID	INT

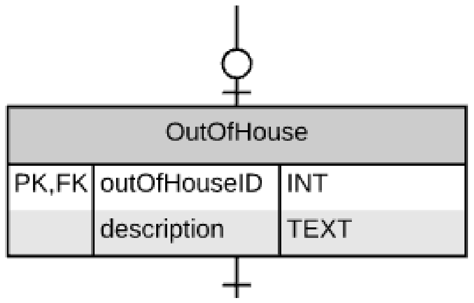
The ParticipantClassAttendance Table stores data about attendance as recorded by participants. The table has the following attributes:

- topicName (PK/FK)
- date (PK/FK)
- siteName (PK/FK)
- participantID (PK/FK)

The ParticipantClassAttendance Table has 2 relationships and they are:

Table	Relationship	Other Table
ParticipantClassAttendance	Many-to-One	Participants
ParticipantClassAttendance	Many-to-One	ClassOffering

OutOfHouse Table



OutOfHouse		
PK,FK	outOfHouseID	INT
	description	TEXT

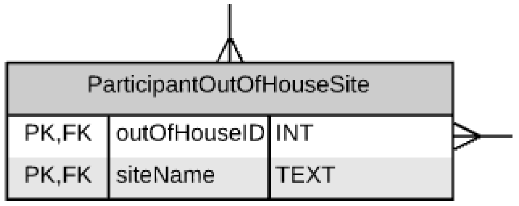
The OutOfHouse Table stores basic data about out-of-house sites descriptions. The table has the following attributes:

- outOfHouseID (PK/FK)
- description

The OutOfHouse Table has 2 relationships and they are:

Table	Relationship	Other Table
OutOfHouse	Zero or One-to-One	Participants
OutOfHouse	One-to-Many	ParticipantOutOfHouseSite

ParticipantOutOfHouseSite Table



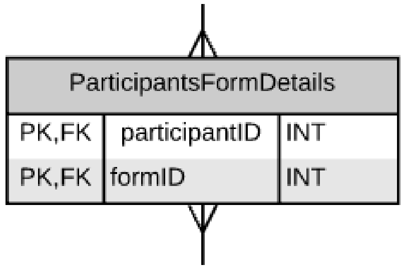
The ParticipantOutOfHouseSite Table stores basic data about out-of-house sites names. The table has the following attributes:

- outOfHouseID (PK/FK)
- siteName (PK/FK)

The ParticipantOutOfHouseSite Table has 2 relationships and they are:

Table	Relationship	Other Table
ParticipantOutOfHouseSite	Many-to-One	OutOfHouse
ParticipantOutOfHouseSite	Many-to-One	Sites

ParticipantsFormDetails Table



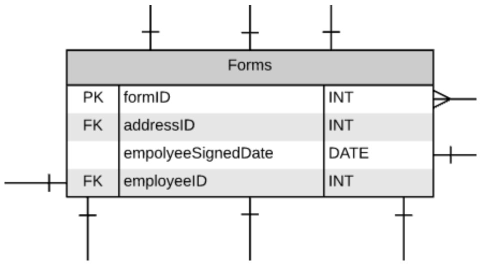
The ParticipantsFormDetails Table stores basic data connecting participants to forms. The table has the following attributes:

- participantID (PK/FK)
- formID (PK/FK)

The ParticipantsFormDetails Table has 2 relationships and they are:

Table	Relationship	Other Table
ParticipantsFormDetails	Many-to-One	Participants
ParticipantsFormDetails	Many-to-One	Forms

Forms Table



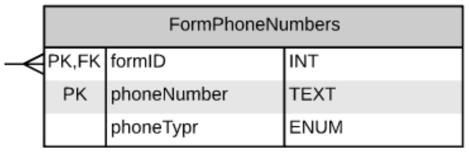
The Forms Table stores regarding forms. The table has the following attributes:

- formID (PK)
- addressID (FK)
- employeeSignedDate
- employeeID (FK)

The Forms Table has 9 relationships and they are:

Table	Relationship	Other Table
Forms	One-to-Many	ParticipantsFormDetails
Forms	One-to-Many	FormPhoneNumbers
Forms	Many-to-One	Employees
Forms	One-to-One	Addresses
Forms	One-to-Zero or One	SelfReferral
Forms	One-to-Zero or One	IntakeInformation
Forms	One-to-Zero or One	Surveys
Forms	One-to-Zero or One	AgencyReferral
Forms	One-to-Many	Family

FormPhoneNumbers Table



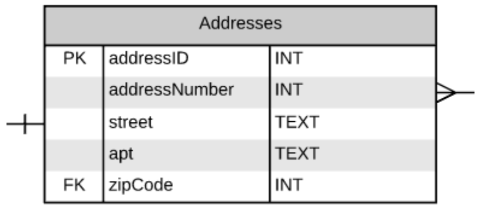
The FormPhoneNumbers Table stores regarding forms. The table has the following attributes:

- formID (PK/FK)
- phoneNumber (PK)
- phoneType

The Forms Table has 1 relationship and it is:

Table	Relationship	Other Table
FormPhoneNumbers	Many-to-One	Forms

Addresses Table



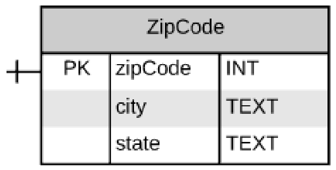
The Addresses Table stores basic data about addresses. The table has the following attributes:

- addressID (PK)
- addressNumber
- street
- apt
- zipCode (FK)

The Addresses Table has 2 relationships and they are:

Table	Relationship	Other Table
Addresses	Many-to-One	ZipCode
Addresses	One-to-One	Forms

ZipCode Table



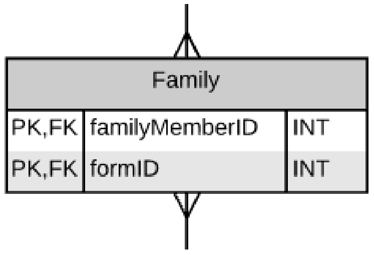
The ZipCode Table stores basic data about zip codes. The table has the following attributes:

- zipCode (PK)
- city
- state

The ZipCode Table has 1 relationships and it is:

Table	Relationship	Other Table
ZipCode	One-to-Many	Addresses

Family Table



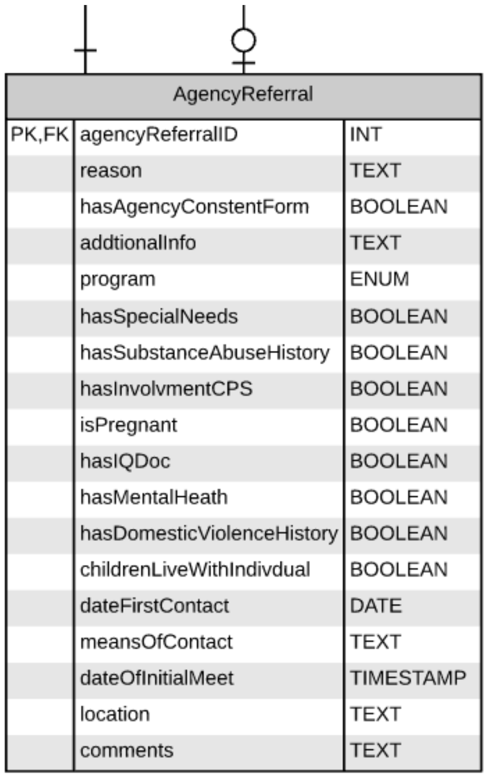
The Family Table a connects a generated ID to forms which mention family with additional data. The table has the following attributes:

- familyMemberID (PK/FK)
- formID (PK/FK)

The Family Table has 2 relationships and they are:

Table	Relationship	Other Table
Family	Many-to-One	Forms
Family	Many-to-One	FamilyMembers

AgencyReferral Table



AgencyReferral		
PK,FK	agencyReferralID	INT
	reason	TEXT
	hasAgencyConstentForm	BOOLEAN
	addtionalInfo	TEXT
	program	ENUM
	hasSpecialNeeds	BOOLEAN
	hasSubstanceAbuseHistory	BOOLEAN
	hasInvolvmentCPS	BOOLEAN
	isPregnant	BOOLEAN
	hasIQDoc	BOOLEAN
	hasMentalHeath	BOOLEAN
	hasDomesticViolenceHistory	BOOLEAN
	childrenLiveWithIndivdual	BOOLEAN
	dateFirstContact	DATE
	meansOfContact	TEXT
	dateOfInitialMeet	TIMESTAMP
	location	TEXT
	comments	TEXT

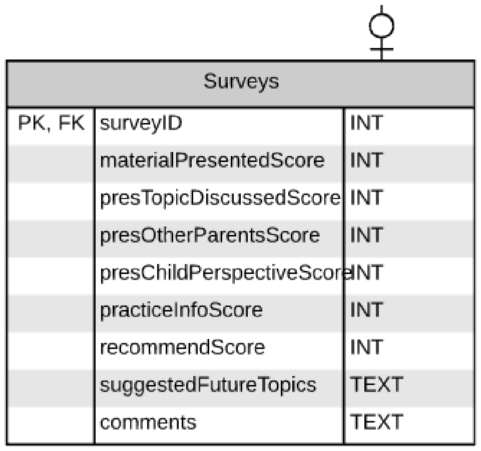
The AgencyReferral Table stores basic data from the referral form for a referral by an agency. The table has the following attributes:

- agencyReferralID (PK/FK)
- reason
- hasAgencyConsentForm
- additionalInfo
- program
- hasSpecialNeeds
- hasSubstanceAbuseHistory
- hasInvolvementCPS
- isPregnant
- hasIQDoc
- hasMentalHealth
- hasDomesticAbuseHistory
- childrenLiveWithIndividual
- dateFirstContact
- meansofContact
- dateOfInitialMeet
- location
- comments

The AgencyReferral Table has 2 relationships and they are:

Table	Relationship	Other Table
AgencyReferral	Zero or One-to-One	Forms
AgencyReferral	One-to-Many	ContactAgencyAssociatedWithReferred

Surveys Table



Surveys		
PK, FK	surveyID	INT
	materialPresentedScore	INT
	presTopicDiscussedScore	INT
	presOtherParentsScore	INT
	presChildPerspectiveScore	INT
	practiceInfoScore	INT
	recommendScore	INT
	suggestedFutureTopics	TEXT
	comments	TEXT

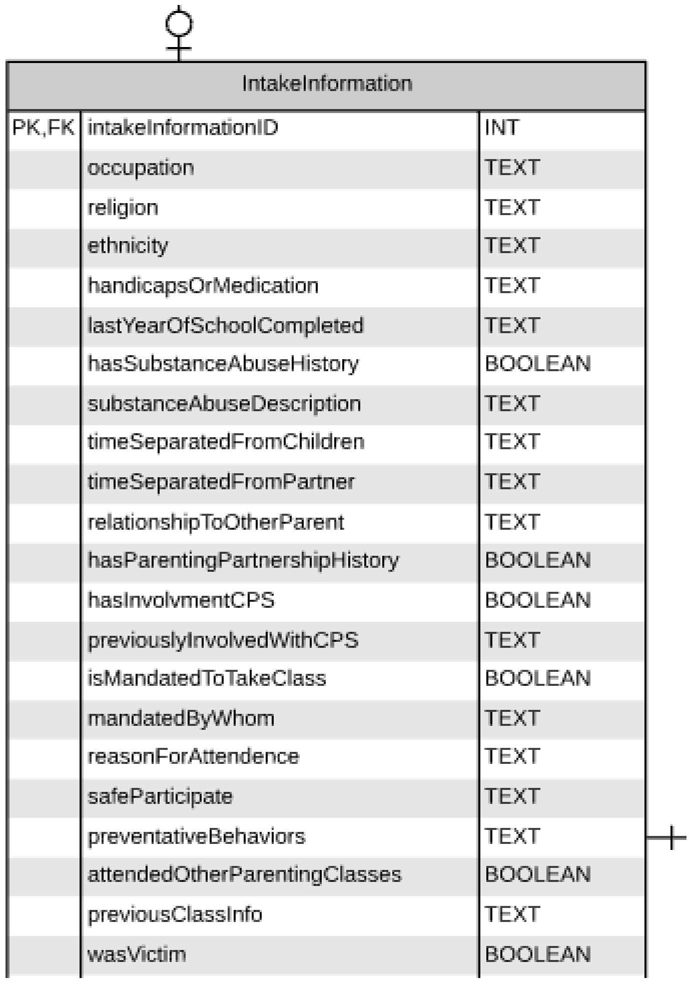
The Surveys Table stores data from the class surveys. The table has the following attributes:

- surveyID (PK)
- materialPresentedScore
- presTopicDiscussedScore
- presOtherParentsScore
- presChildPerspectiveScore
- practiceInfoScore
- recommendScore
- suggestedFutureTopics
- comments

The Surveys Table has 1 relationships and it is:

Table	Relationship	Other Table
Surveys	Zero or One-to-One	Forms

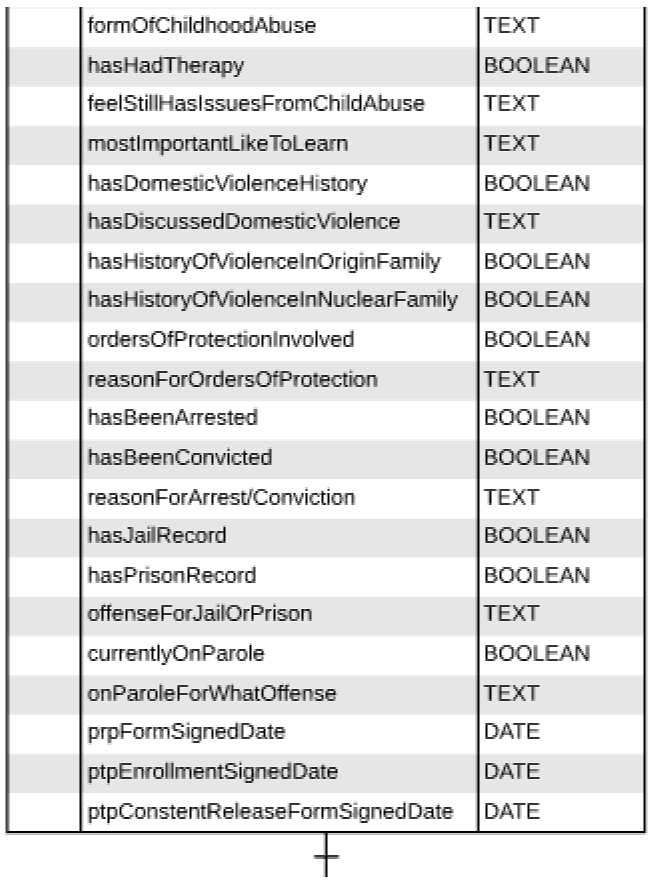
IntakeInformation Table



IntakeInformation		
PK,FK	intakeInformationID	INT
	occupation	TEXT
	religion	TEXT
	ethnicity	TEXT
	handicapsOrMedication	TEXT
	lastYearOfSchoolCompleted	TEXT
	hasSubstanceAbuseHistory	BOOLEAN
	substanceAbuseDescription	TEXT
	timeSeparatedFromChildren	TEXT
	timeSeparatedFromPartner	TEXT
	relationshipToOtherParent	TEXT
	hasParentingPartnershipHistory	BOOLEAN
	hasInvolvmentCPS	BOOLEAN
	previouslyInvolvedWithCPS	TEXT
	isMandatedToTakeClass	BOOLEAN
	mandatedByWhom	TEXT
	reasonForAttendence	TEXT
	safeParticipate	TEXT
	preventativeBehaviors	TEXT
	attendedOtherParentingClasses	BOOLEAN
	previousClassInfo	TEXT
	wasVictim	BOOLEAN

The IntakeInformation Table stores data from the intake forms. The table has the following attributes:

- intakeInformationID(PK/FK)
- occupation
- religion
- ethnicity
- handicapsOrMedication
- lastYearOfSchoolCompleted
- hasSubstanceAbuseHistory
- substanceAbuseDescription
- timeSeparatedFromChildren
- timeSeparatedFromPartner
- relationshipToOtherParent
- hasParentingPartnershipHistory
- hasInvolvementCPS
- previouslyInvolvedWithCPS
- isMandatedToTakeClass
- mandatedByWhom
- reasonForAttendance
- safeParticipate
- preventativeBehaviors
- attendedOtherParentingClasses
- previousClassInfo
- wasVictim



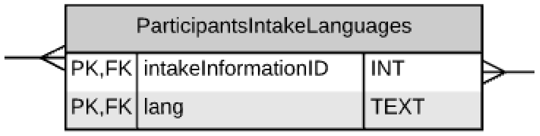
	formOfChildhoodAbuse	TEXT
	hasHadTherapy	BOOLEAN
	feelStillHasIssuesFromChildAbuse	TEXT
	mostImportantLikeToLearn	TEXT
	hasDomesticViolenceHistory	BOOLEAN
	hasDiscussedDomesticViolence	TEXT
	hasHistoryOfViolenceInOriginFamily	BOOLEAN
	hasHistoryOfViolenceInNuclearFamily	BOOLEAN
	ordersOfProtectionInvolved	BOOLEAN
	reasonForOrdersOfProtection	TEXT
	hasBeenArrested	BOOLEAN
	hasBeenConvicted	BOOLEAN
	reasonForArrest/Conviction	TEXT
	hasJailRecord	BOOLEAN
	hasPrisonRecord	BOOLEAN
	offenseForJailOrPrison	TEXT
	currentlyOnParole	BOOLEAN
	onParoleForWhatOffense	TEXT
	prpFormSignedDate	DATE
	ptpEnrollmentSignedDate	DATE
	ptpConstentReleaseFormSignedDate	DATE

- formOfChildhoodAbuse
- hasHadTherapy
- feelStillHasIssuesFromChildAbuse
- mostImportantLikeToLearn
- hasDomesticViolenceHistory
- hasDiscussedDomesticViolence
- hasHistoryOfViolenceInOriginFamily
- hasHistoryOfViolenceInNuclearFamily
- ordersOfProtectionInvolved
- reasonForOrdersOfProtection
- hasBeenArrested
- hasBeenConvicted
- reasonForArrest/Conviction
- hasJailRecord
- hasPrisonRecord
- offenseForJailOrPrison
- currentlyOnParole
- onParoleForWhatOffense
- hasOtherFamilyMembersTakingClass
- ptpFormSignedDate
- ptpEnrollmentSignedDate
- ptpConsentReleaseFormSignedDate

The IntakeInformation Table has 3 relationships and they are:

Table	Relationship	Other Table
IntakeInformation	Zero or One-to-One	Forms
IntakeInformation	One-to-Many	ParticipantsIntakeLanguages
IntakeInformation	One-to-Many	EmergencyContactDetail

ParticipantsIntakeLanguages Table



ParticipantsIntakeLanguages		
PK,FK	intakeInformationID	INT
PK,FK	lang	TEXT

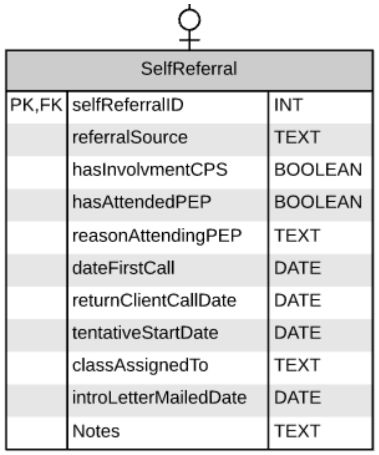
The ParticipantsIntakeLanguage Table stores basic data connecting intake forms to a language. The table has the following attributes:

- intakeInformationID (PK/FK)
- lang (PK/FK)

The ParticipantsIntakeLanguages Table has 2 relationships and they are:

Table	Relationship	Other Table
ParticipantsIntakeLanguage	Many-to-One	IntakeInformation
ParticipantsIntakeLanguage	Many-to-One	Languages

SelfReferral Table



SelfReferral		
PK,FK	selfReferralID	INT
	referralSource	TEXT
	hasInvolvmentCPS	BOOLEAN
	hasAttendedPEP	BOOLEAN
	reasonAttendingPEP	TEXT
	dateFirstCall	DATE
	returnClientCallDate	DATE
	tentativeStartDate	DATE
	classAssignedTo	TEXT
	introLetterMailedDate	DATE
	Notes	TEXT

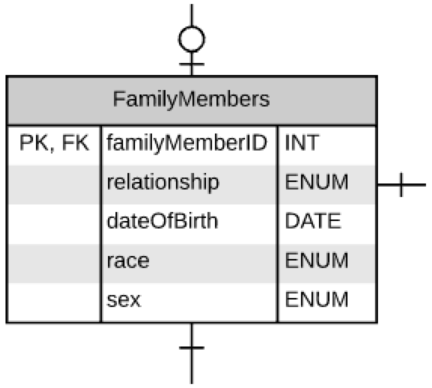
The SelfReferral Table stores data about the self-referral form. The table has the following attributes:

- selfReferralID (PK/FK)
- referralSource
- hasInvolvementCPS
- hasAttendedPEP
- dateFirstCall
- returnClientCallDate
- tentativeStartDate
- classAssignedTo
- introLetterMailedDate
- notes

The SelfReferral Table has 1 relationship and it is:

Table	Relationship	Other Table
SelfReferral	Zero or One-to-One	Forms

FamilyMembers Table



FamilyMembers		
PK, FK	familyMemberID	INT
	relationship	ENUM
	dateOfBirth	DATE
	race	ENUM
	sex	ENUM

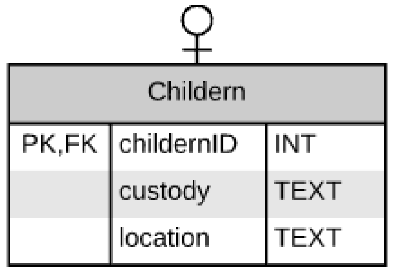
The FamilyMembers Table stores basic data about family members. The table has the following attributes:

- memberID (PK/FK)
- relationship
- dateOfBirth
- race
- sex

The FamilyMembers Table has 3 relationships and they are:

Table	Relationship	Other Table
FamilyMembers	Zero or One-to-One	People
FamilyMembers	One-to-Zero or One	Children
FamilyMembers	One-to-Many	Family

Children Table



Childern		
PK,FK	childernID	INT
	custody	TEXT
	location	TEXT

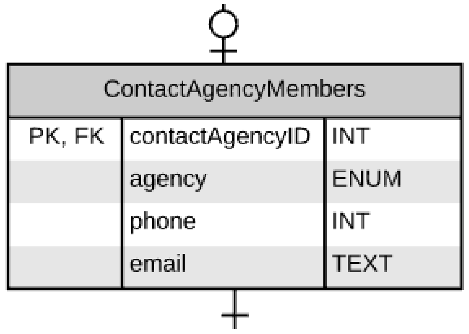
The Children Table stores data regarding custody and location of children. The table has the following attributes:

- childrenID (PK/FK)
- custody
- location

The Children Table has 1 relationship and it is:

Table	Relationship	Other Table
Children	Zero or One-to-One	FamilyMembers

ContactAgencyMembers Table



ContactAgencyMembers		
PK, FK	contactAgencyID	INT
	agency	ENUM
	phone	INT
	email	TEXT

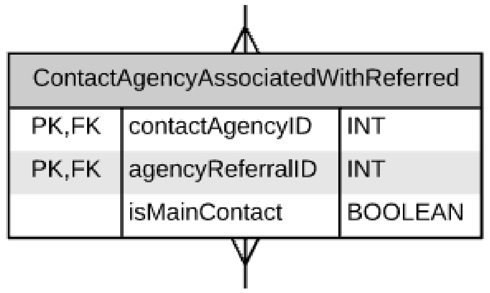
The ContactAgencyMembers Table stores basic data about a contact agency. The table has the following attributes:

- contactAgencyID (PK/FK)
- agency
- phone
- email

The ContactAgencyMembers Table has 2 relationships and they are:

Table	Relationship	Other Table
ContactAgencyMembers	Zero or One-to-Many	People
ContactAgencyMembers	One-to-Many	ContactAgencyAssociatedWithReferred

ContactAgencyAssociatedWithReferred Table



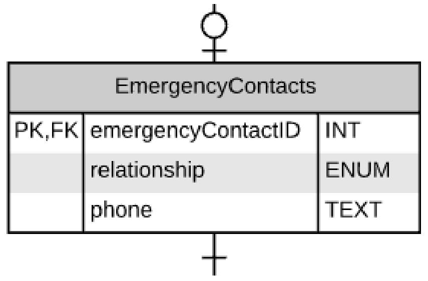
The ContactAgencyAssociatedWithReferred Table stores basic data connecting a contact agency with an agency referral. The table has the following attributes:

- contactAgencyID (PK/FK)
- agencyReferralID (PK/FK)
- isMainContact

The ContactAgencyAssociatedWithReferred Table has 2 relationships and they are:

Table	Relationship	Other Table
ContactAgencyAssociatedWithReferred	Many-to-One	ContactAgencyMembers
ContactAgencyAssociatedWithReferred	Many-to-One	AgencyReferral

EmergencyContacts Table



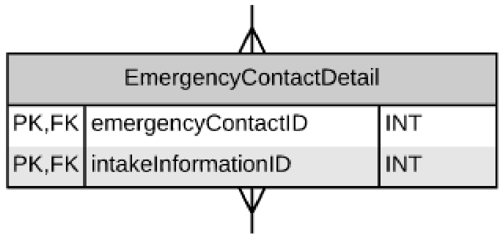
The EmergencyContacts Table stores basic data about emergency contacts. The table has the following attributes:

- emergencyContactID (PK/FK)
- relationship
- phone

The EmergencyContacts Table has 2 relationships and they are:

Table	Relationship	Other Table
EmergencyContacts	Zero or One-to-One	People
EmergencyContacts	One-to-Many	EmergencyContactDetail

EmergencyContactDetail Table



The EmergencyContactDetail Table stores basic data about emergency contacts. The table has the following attributes:

- emergencyContactID (PK/FK)
- relationship
- phone

The EmergencyContactDetail Table has 2 relationships and they are:

Table	Relationship	Other Table
EmergencyContactDetail	Many-to-One	IntakeInformation
EmergencyContactDetail	Many-to-One	EmergencyContacts

SQL Statements

"DROP TABLE"

The "`DROP TABLE IF EXISTS`" statement in Structured Query Language (SQL) removes a table, if it exists with the provided name. The table would have previously been created with the "`CREATE TABLE`" statement. The dropped table is completely removed from the database schema and disk file. Once the table is removed, the file cannot be recovered—so use this statement with care.

```
DROP TABLE IF EXISTS
EmergencyContactDetail;

DROP TABLE IF EXISTS
ParticipantsIntakeLanguages;

DROP TABLE IF EXISTS Family;

DROP TABLE IF EXISTS
ContactAgencyAssociatedWithReferred
;

DROP TABLE IF EXISTS
IntakeInformation;

DROP TABLE IF EXISTS Surveys;

DROP TABLE IF EXISTS
AgencyReferral;

DROP TABLE IF EXISTS SelfReferral;

DROP TABLE IF EXISTS
ParticipantsFormDetails;

DROP TABLE IF EXISTS
FormPhoneNumbers;

DROP TABLE IF EXISTS Forms;

DROP TABLE IF EXISTS Addresses;

DROP TABLE IF EXISTS ZipCode;
```

```
DROP TABLE IF EXISTS
ParticipantOutOfHouseSite;

DROP TABLE IF EXISTS
FacilitatorLangugage;

DROP TABLE IF EXISTS
ParticipantClassAttendance;

DROP TABLE IF EXISTS
FacilitatorClassAttendance;

DROP TABLE IF EXISTS ClassOffering;

DROP TABLE IF EXISTS
CurriculumClasses;

DROP TABLE IF EXISTS Classes;

DROP TABLE IF EXISTS Curricula;

DROP TABLE IF EXISTS Sites;

DROP TABLE IF EXISTS Languages;

DROP TABLE IF EXISTS
ContactAgencyMembers;

DROP TABLE IF EXISTS
EmergencyContacts;

DROP TABLE IF EXISTS Children;

DROP TABLE IF EXISTS FamilyMembers;
```

```
DROP TABLE IF EXISTS OutOfHouse;

DROP TABLE IF EXISTS Participants;

DROP TABLE IF EXISTS Facilitators;

DROP TABLE IF EXISTS Employees;

DROP TABLE IF EXISTS People;

DROP TYPE IF EXISTS RELATIONSHIP;

DROP TYPE IF EXISTS PARENTINGPROGRAM;

DROP TYPE IF EXISTS PROGRAMTYPE;

DROP TYPE IF EXISTS PHONETYPE;

DROP TYPE IF EXISTS PERMISSION;

DROP TYPE IF EXISTS STATES;

DROP TYPE IF EXISTS LEVELTYPE;

DROP TYPE IF EXISTS CURRICULUMTYPE;

DROP TYPE IF EXISTS REFERRALTYPE;

DROP TYPE IF EXISTS FORM;

DROP TYPE IF EXISTS SEX;

DROP TYPE IF EXISTS RACE;
```

"CREATE"

Create statements in SQL are used to create tables, types, and other objects that are a part of an ER Diagram.

"CREATE TYPE AS ENUM"

The "`CREATE TYPE AS ENUM`" statement in SQL creates an enumerated data type with the name provided. Enumerated (enum) types are data types that comprise a static, ordered set of values.

```
CREATE TYPE RACE AS ENUM('Asian', 'Black', 'Latino', 'Native American', 'Pacific Islander', 'White');

CREATE TYPE SEX AS ENUM ('Male', 'Female');

CREATE TYPE FORM AS ENUM ('Intake', 'Referral');

CREATE TYPE REFERRALTYPE AS ENUM ('Self', 'Court', 'Agency', 'Friend', 'Family');

CREATE TYPE CURRICULUMTYPE AS ENUM ('FULL', 'MINI');

CREATE TYPE LEVELTYPE AS ENUM ('PRIMARY', 'SECONDARY');

CREATE TYPE STATES AS ENUM('Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut', 'Delaware',

'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',

'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',

'New Jersey', 'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania',

'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',

'West Virginia', 'Wisconsin', 'Wyoming');

CREATE TYPE PERMISSION AS ENUM('User', 'Facilitator', 'Administator', 'Superuser');

CREATE TYPE PHONETYPE AS ENUM('Primary', 'Secondary', 'Day', 'Evening', 'Home', 'Cell');

CREATE TYPE PROGRAMTYPE AS ENUM('In-House', 'Jail', 'Rehab');

CREATE TYPE PARENTINGPROGRAM AS ENUM('TPP', 'SNPP', 'PEP');

CREATE TYPE RELATIONSHIP AS ENUM ('Mother', 'Father', 'Daughter', 'Son', 'Sister', 'Brother', 'Aunt', 'Uncle', 'Niece',
'Nephew', 'Cousin', 'Grandmother', 'Grandfather', 'Granddaughter', 'Grandson', 'Stepsister', 'Stepbrother', 'Stepmother',
'Stepfather', 'Stepdaughter', 'Stepson', 'Sister-in-law', 'Brother-in-law', 'Mother-in-law', 'Daughter-in-law', 'Son-in-law',
'Friend', 'Other');
```

“CREATE TABLE”

The “`CREATE TABLE`” statement in SQL defines a new table with the given parameters. The following “`CREATE TABLE`” statements are used to create the various tables in the database, corresponding with the tables in the ER Diagram.

People and Subtypes

```
CREATE TABLE IF NOT EXISTS People (
  peopleID                                                 SERIAL              NOT NULL     UNIQUE,
  firstName                                                TEXT                NOT NULL,
  lastName                                                 TEXT                NOT NULL,
  PRIMARY KEY (peopleID)
);

CREATE TABLE IF NOT EXISTS Employees (
  employeeID                                               INT,
  email                                                    TEXT                NOT NULL,
  primaryPhone                                             TEXT,
  permisionLevel                                           PERMISSION          NOT NULL,
  PRIMARY KEY (employeeID),
  FOREIGN KEY (employeeID) REFERENCES People(peopleID)
);

CREATE TABLE IF NOT EXISTS Facilitators (
  facilitatorID                                            INT,
  program                                                  PARENTINGPROGRAM,
  PRIMARY KEY (facilitatorID),
  FOREIGN KEY (facilitatorID) REFERENCES Employees(employeeID)
);
```

```
CREATE TABLE IF NOT EXISTS Participants (
  participantID                                            INT,
  dateOfBirth                                              DATE                NOT NULL,
  race                                                     RACE                NOT NULL,
  PRIMARY KEY (participantID),
  FOREIGN KEY (participantID) REFERENCES People(peopleID)
);

CREATE TABLE IF NOT EXISTS OutOfHouse (
  outOfHouseID                                             INT,
  description                                              TEXT,
  PRIMARY KEY (outOfHouseID),
  FOREIGN KEY (outOfHouseID) REFERENCES Participants(participantID)
);

CREATE TABLE IF NOT EXISTS FamilyMembers (
  familyMemberID                                           INT,
  relationship                                             RELATIONSHIP        NOT NULL,
  dateOfBirth                                              DATE                NOT NULL,
  race                                                     RACE,
  sex                                                      SEX,
  PRIMARY KEY (familyMemberID),
  FOREIGN KEY (familyMemberID) REFERENCES People(peopleID)
);
CREATE TABLE IF NOT EXISTS Children (
  childrenID                                               INT,
  custody                                                  TEXT                NOT NULL,
  location                                                 TEXT                NOT NULL,
  PRIMARY KEY (childrenID),
  FOREIGN KEY (childrenID) REFERENCES FamilyMembers(familyMemberID)
);
```

```
CREATE TABLE IF NOT EXISTS EmergencyContacts (
  emergencyContactID                                          INT,
  relationship                                                RELATIONSHIP        NOT NULL,
  primaryPhone                                                TEXT                NOT NULL,
  PRIMARY KEY (emergencyContactID),
  FOREIGN KEY (emergencyContactID) REFERENCES People(peopleID)
);

CREATE TABLE IF NOT EXISTS ContactAgencyMembers (
  contactAgencyID                                             INT,
  agency                                                      REFERRALTYPE        NOT NULL,
  phone                                                       TEXT                NOT NULL,
  email                                                       TEXT                NOT NULL,
  PRIMARY KEY (contactAgencyID),
  FOREIGN KEY (contactAgencyID) REFERENCES People(peopleID)
);
```

Curricula and Class

```
CREATE TABLE IF NOT EXISTS Languages (
  lang                                                        TEXT                NOT NULL     UNIQUE,
  PRIMARY KEY (lang)
);

CREATE TABLE IF NOT EXISTS Sites (
  siteName                                                    TEXT                NOT NULL     UNIQUE,
  programType                                                 PROGRAMTYPE         NOT NULL,
  PRIMARY KEY (siteName)
);
```

```
CREATE TABLE IF NOT EXISTS Curricula (
  curriculumID                                         SERIAL              NOT NULL      UNIQUE,
  curriculumName                                       TEXT                NOT NULL,
  curriculmType                                        PROGRAMTYPE         NOT NULL,
  missNumber                                           INT                 DEFAULT 2,
  PRIMARY KEY (curriculumID)
);

CREATE TABLE IF NOT EXISTS Classes (
  topicName                                            TEXT                NOT NULL      UNIQUE,
  description                                          TEXT,
  PRIMARY KEY (topicName)
);

CREATE TABLE IF NOT EXISTS CurriculumClasses (
  topicName                                            TEXT,
  curriculumID                                         INT,
  PRIMARY KEY (topicName, curriculumID),
  FOREIGN KEY (topicName) REFERENCES Classes(topicName),
  FOREIGN KEY (curriculumID) REFERENCES Curricula(curriculumID)
);

CREATE TABLE IF NOT EXISTS ClassOffering (
  topicName                                            TEXT,
  date                                                 TIMESTAMP           NOT NULL,
  siteName                                             TEXT,
  lang                                                 TEXT                DEFAULT 'English',
  curriculumID                                         INT,
  PRIMARY KEY (topicName, date, siteName),
  FOREIGN KEY (topicName) REFERENCES Classes(topicName),
  FOREIGN KEY (siteName) REFERENCES Sites(siteName),
  FOREIGN KEY (lang) REFERENCES Languages(lang),
  FOREIGN KEY (curriculumID) REFERENCES Curricula(curriculumID)
);
```

```
CREATE TABLE IF NOT EXISTS FacilitatorClassAttendance (
  topicName                                           TEXT,
  date                                                TIMESTAMP,
  siteName                                            TEXT,
  facilitatorID                                       INT,
  PRIMARY KEY (topicName, date, siteName, facilitatorID),
  FOREIGN KEY (topicName, date, siteName) REFERENCES ClassOffering(topicName, date, siteName),
  FOREIGN KEY (facilitatorID) REFERENCES Facilitators(facilitatorID)
);

CREATE TABLE IF NOT EXISTS ParticipantClassAttendance (
  topicName                                           TEXT,
  date                                                TIMESTAMP,
  siteName                                            TEXT,
  participantID                                       INT,
  PRIMARY KEY (topicName, date, siteName, participantID),
  FOREIGN KEY (topicName, date, siteName) REFERENCES ClassOffering(topicName, date, siteName),
  FOREIGN KEY (participantID) REFERENCES Participants(participantID)
);

CREATE TABLE IF NOT EXISTS FacilitatorLangugage (
  facilitatorID                                       INT,
  lang                                                TEXT,
  level                                               LEVELTYPE                 NOT NULL,
  PRIMARY KEY (facilitatorID, lang, level),
  FOREIGN KEY (facilitatorID) REFERENCES Facilitators(facilitatorID),
  FOREIGN KEY (lang) REFERENCES Languages(lang)
);
```

```
CREATE TABLE IF NOT EXISTS ParticipantOutOfHouseSite (
  outOfHouseID INT,
  siteName TEXT,
  PRIMARY KEY (outOfHouseID, siteName),
  FOREIGN KEY (outOfHouseID) REFERENCES OutOfHouse(outOfHouseID),
  FOREIGN KEY (siteName) REFERENCES Sites(siteName)
);
```

Forms and Related Tables

```
CREATE TABLE IF NOT EXISTS ZipCode (
  zipCode                                                     INT                       UNIQUE,
  city                                                        TEXT                      NOT NULL,
  state                                                       STATES                    NOT NULL,
  PRIMARY KEY (zipCode)
);

CREATE TABLE IF NOT EXISTS Addresses (
  addressID                                                   SERIAL                    NOT NULL      UNIQUE,
  addressNumber                                               INT,
  aptInfo                                                     TEXT,
  street                                                      TEXT                      NOT NULL,
  zipCode                                                     INT                       NOT NULL,
  PRIMARY KEY (addressID),
  FOREIGN KEY (zipCode) REFERENCES ZipCode(zipCode)
);
```

```
CREATE TABLE IF NOT EXISTS Forms (
  formID                                            SERIAL                    NOT NULL      UNIQUE,
  addressID                                         INT,
  empolyeeSignedDate                                DATE                      NOT NULL      DEFAULT NOW(),
  employeeID                                        INT,
  PRIMARY KEY (formID),
  FOREIGN KEY (addressID) REFERENCES Addresses(addressID),
  FOREIGN KEY (employeeID) REFERENCES Employees(EmployeeID)
);

CREATE TABLE IF NOT EXISTS FormPhoneNumbers (
  formID INT,
  phoneNumber TEXT,
  phoneType PHONETYPE,
  PRIMARY KEY (formID, phoneNumber),
  FOREIGN KEY (formID) REFERENCES Forms(formID)
);

CREATE TABLE IF NOT EXISTS ParticipantsFormDetails (
  participantID                                     INT,
  formID                                            INT,
  PRIMARY KEY (participantID, formID),
  FOREIGN KEY (participantID) REFERENCES Participants(participantID),
  FOREIGN KEY (formID) REFERENCES Forms(formID)
);
```

```
CREATE TABLE IF NOT EXISTS SelfReferral (
  selfReferralID                                      INT,
  referralSource                                      TEXT,
  hasInvolvmentCPS                                    BOOLEAN,
  hasAttendedPEP                                      BOOLEAN,
  reasonAttendingPEP                                  TEXT,
  dateFirstCall                                       DATE,
  returnClientCallDate                                DATE,
  tentativeStartDate                                  DATE,
  classAssignedTo                                     TEXT,
  introLetterMailedDate                               DATE,
  Notes                                               TEXT,
  PRIMARY KEY (selfReferralID),
  FOREIGN KEY (selfReferralID) REFERENCES Forms(formID)
);
CREATE TABLE IF NOT EXISTS Surveys (
  surveyID                                            INT,
  materialPresentedScore                              INT,
  presTopicDiscussedScore                             INT,
  presOtherParentsScore                               INT,
  presChildPerspectiveScore                           INT,
  practiceInfoScore                                   INT,
  recommendScore                                      INT,
  suggestedFutureTopics                               TEXT,
  comments                                            TEXT,
  PRIMARY KEY (surveyID),
  FOREIGN KEY (surveyID) REFERENCES Forms(formID)
);
```

```
CREATE TABLE IF NOT EXISTS AgencyReferral (
  agencyReferralID                                    INT,
  secondaryPhone                                      TEXT,
  reason                                              TEXT,
  hasAgencyConstentForm                               BOOLEAN,
  addtionalInfo                                       TEXT,
  program                                             PARENTINGPROGRAM,
  hasSpecialNeeds                                     BOOLEAN,
  hasSubstanceAbuseHistory                            BOOLEAN,
  hasInvolvmentCPS                                    BOOLEAN,
  isPregnant                                          BOOLEAN,
  hasIQDoc                                            BOOLEAN,
  hasMentalHeath                                      BOOLEAN,
  hasDomesticViolenceHistory                          BOOLEAN,
  childrenLiveWithIndivdual                           BOOLEAN,
  dateFirstContact                                    DATE,
  meansOfContact                                      TEXT,
  dateOfInitialMeet                                   TIMESTAMP,
  location                                            TEXT,
  comments                                            TEXT,
  PRIMARY KEY (agencyReferralID),
  FOREIGN KEY (agencyReferralID) REFERENCES Forms(formID)
);
```

```
CREATE TABLE IF NOT EXISTS IntakeInformation (
  intakeInformationID                           INT,
  secondaryPhone                                TEXT,
  occupation                                    TEXT,
  religion                                      TEXT,
  ethnicity                                     TEXT,
  handicapsOrMedication                         TEXT,
  lastYearOfSchoolCompleted                     TEXT,
  hasSubstanceAbuseHistory                      BOOLEAN,
  substanceAbuseDescription                     TEXT,
  timeSeparatedFromChildren                     TEXT,
  timeSeparatedFromPartner                      TEXT,
  relationshipToOtherParent                     TEXT,
  hasParentingPartnershipHistory                BOOLEAN,
  hasInvolvmentCPS                              BOOLEAN,
  previouslyInvolvedWithCPS                     TEXT,
  isMandatedToTakeClass                         BOOLEAN,
  mandatedByWhom                                TEXT,
  reasonForAttendence                           TEXT,
  safeParticipate                               TEXT,
  preventativeBehaviors                         TEXT,
  attendedOtherParentingClasses                 BOOLEAN,
  previousClassInfo                             TEXT,
  wasVictim                                     BOOLEAN,
  formOfChildhoodAbuse                          TEXT,
  hasHadTherapy                                 BOOLEAN,
  feelStillHasIssuesFromChildAbuse              TEXT,
  mostImportantLikeToLearn                      TEXT,
  hasDomesticViolenceHistory                    BOOLEAN,
  hasDiscussedDomesticViolence                  TEXT,
  hasHistoryOfViolenceInOriginFamily            BOOLEAN,
  hasHistoryOfViolenceInNuclearFamily           BOOLEAN,
  ordersOfProtectionInvolved                    BOOLEAN,
  reasonForOrdersOfProtection                   TEXT,
  hasBeenArrested                               BOOLEAN,
  hasBeenConvicted                              BOOLEAN,
  reasonForArrestOrConviction                   TEXT,
  hasJailRecord                                 BOOLEAN,
  hasPrisonRecord                               BOOLEAN,
  offenseForJailOrPrison                        TEXT,
  currentlyOnParole                             BOOLEAN,
  onParoleForWhatOffense                        TEXT,
  prpFormSignedDate                             DATE,
  ptpEnrollmentSignedDate                       DATE,
  ptpConstentReleaseFormSignedDate              DATE,
  PRIMARY KEY (intakeInformationID),
  FOREIGN KEY (intakeInformationID) REFERENCES
Forms(formID)
);
```

```
CREATE TABLE IF NOT EXISTS ContactAgencyAssociatedWithReferred (
  contactAgencyID   INT,
  agencyReferralID  INT,
  isMainContact     BOOLEAN,
  PRIMARY KEY (contactAgencyID, agencyReferralID),
  FOREIGN KEY (contactAgencyID) REFERENCES ContactAgencyMembers(contactAgencyID),
  FOREIGN KEY (agencyReferralID) REFERENCES AgencyReferral(agencyReferralID)
);

CREATE TABLE IF NOT EXISTS Family (
  familyMembersID INT,
  formID INT,
  PRIMARY KEY (familyMembersID, formID),
  FOREIGN KEY (familyMembersID) REFERENCES FamilyMembers(familyMemberID),
  FOREIGN KEY (formID) REFERENCES Forms(formID)
);


CREATE TABLE IF NOT EXISTS ParticipantsIntakeLanguages (
  intakeInformationID INT,
  lang TEXT,
  PRIMARY KEY (intakeInformationID, lang),
  FOREIGN KEY (intakeInformationID) REFERENCES IntakeInformation(intakeInformationID),
  FOREIGN KEY (lang) REFERENCES Languages(lang)
);

CREATE TABLE IF NOT EXISTS EmergencyContactDetail (
  emergencyContactID INT,
  intakeInformationID INT,
  PRIMARY KEY (emergencyContactID, intakeInformationID),
  FOREIGN KEY (emergencyContactID) REFERENCES EmergencyContacts(emergencyContactID),
  FOREIGN KEY (intakeInformationID) REFERENCES IntakeInformation(intakeInformationID)
);
```

“INSERT INTO”

Insert statements in SQL are used to populate tables with data. The following is **sample data**.

People

```
INSERT INTO People(firstName, lastName, middleInit) VALUES ('James', 'Crowley', 'D');
INSERT INTO People(firstName, lastName) VALUES ('Marcos', 'Barbieri');
INSERT INTO People(firstName, lastName) VALUES ('Carson', 'Badame');
INSERT INTO People(firstName, lastName, middleInit) VALUES ('Jesse', 'Opitz', 'P');
INSERT INTO People(firstName, lastName, middleInit) VALUES ('Rachel', 'Ulicni', 'M');
INSERT INTO People(firstName, lastName) VALUES ('John', 'Randis');
INSERT INTO People(firstName, lastName) VALUES ('Christopher', 'Algozzine');
INSERT INTO People(firstName, lastName) VALUES ('Dan', 'Grogan');
INSERT INTO People(firstName, lastName) VALUES ('Michlle', 'Opitz');
INSERT INTO People(firstName, lastName) VALUES ('Michlle', 'Crawley');
```

Employees

```
INSERT INTO Employees(employeeID, email, primaryPhone, permissionLevel) VALUES (5,
'Rachel@thecpca.com', '845-867-5309', 'Facilitator');
INSERT INTO Employees(employeeID, email, primaryPhone, permissionLevel) VALUES (3,
'Carson@thecpca.com', '845-234-4567', 'User');
INSERT INTO Employees(employeeID, email, permissionLevel) VALUES (6, 'John@thecpca.com',
'Superuser');
INSERT INTO Employees(employeeID, email, permissionLevel) VALUES (7, 'Christopher@thecpca.com',
'Facilitator');
INSERT INTO Employees(employeeID, email, permissionLevel) VALUES (8, 'Dan@thecpca.com',
'Administrator');
```

Facilitators

```
INSERT INTO Facilitators(facilitatorID, program) VALUES (5, 'PEP');
INSERT INTO Facilitators(facilitatorID, program) VALUES (7, 'TPP');
```

Participants

```
INSERT INTO Participants(participantID, dateOfBirth, race) VALUES (2, '1996-04-03', 'Pacific
Islander');
INSERT INTO Participants(participantID, dateOfBirth, race) VALUES (4, '1878-01-01', 'White');
```

Out Of House

```
INSERT INTO OutOfHouse(outOfHouseID, description) VALUES (4, 'Jailed for beating wife and kids');
```

Family Members

```
INSERT INTO FamilyMembers(familyMemberID, relationship, dateOfBirth, race, sex) VALUES (9,
'Daughter', '2001-07-23', 'Black', 'Female');
```

Children

```
INSERT INTO Children(childrenID, custody, location) VALUES (9, 'NO', 'Mothers');
```

Emergency Contacts

```
INSERT INTO EmergencyContacts(emergencyContactID, relationship, primaryPhone) VALUES (9,
'Daughter', '518-347-0303');
```

Contact Agency Members

```
INSERT INTO ContactAgencyMembers(contactAgencyID, agency, phone) VALUES (10, 'Court', '845-100-
2324');
```

Languages

```
INSERT INTO Languages(lang) VALUES ('English');
INSERT INTO Languages(lang) VALUES ('Spanish');
```

Sites

```
INSERT INTO Sites(siteName, programType) VALUES ('Fox Run', 'Rehab');
INSERT INTO Sites(siteName, programType) VALUES ('Dutchess County Jail', 'Jail');
INSERT INTO Sites(siteName, programType) VALUES ('Poughkeepsie Site', 'In-House');
```

Curricula

```
INSERT INTO Curricula(curriculumName, curriculumType, missNumber) VALUES ('DC Womens Jail',
'Jail', 2);
INSERT INTO Curricula(curriculumName, curriculumType) VALUES ('In-House Poughkeepsie', 'In-
House');
INSERT INTO Curricula(curriculumName, curriculumType) VALUES ('In-House Men', 'In-House');
INSERT INTO Curricula(curriculumName, curriculumType) VALUES ('Fox Run', 'Rehab');
```

Class

```
INSERT INTO Classes(topicName, description) VALUES ('How to be a good parent', 'A class that does
what it says');
INSERT INTO Classes(topicName) VALUES ('How to be Cool');
INSERT INTO Classes(topicName, description) VALUES ('Parenting 101', 'Intro Class');
INSERT INTO Classes(topicName) VALUES ('Nurtuing/Culture/Spirituality');
INSERT INTO Classes(topicName) VALUES ('Developing Empathy/Getting Needs Met');
INSERT INTO Classes(topicName) VALUES ('Recognizing & Undering Feelings');
INSERT INTO Classes(topicName) VALUES ('Problem Solving & Decision Making');
INSERT INTO Classes(topicName) VALUES ('Communication/Listening/Criticism/Confrontation/Fair
Fighting');
INSERT INTO Classes(topicName) VALUES ('Understanding & Expressing Anger');
INSERT INTO Classes(topicName) VALUES ('Talking about Effects of Drugs/Alcohol/Smoking on
Family');
INSERT INTO Classes(topicName) VALUES ('Recap classes 1-8');
INSERT INTO Classes(topicName) VALUES ('Relationships/Personal Space');
INSERT INTO Classes(topicName) VALUES ('Children''s Brain Development');
INSERT INTO Classes(topicName) VALUES ('Male/Female Brain/Quiz');
INSERT INTO Classes(topicName) VALUES ('Ages & Stages: Appropriate Expectations');
INSERT INTO Classes(topicName) VALUES ('Ages & Stages: Infants to Toddler');
INSERT INTO Classes(topicName) VALUES ('Ages & Stages: Preschool to Adolescence');
INSERT INTO Classes(topicName) VALUES ('Establishing Nurturing Parenting Routines');
INSERT INTO Classes(topicName) VALUES ('Child Proofing Home/Safety Checklist/Safety Reminders');
INSERT INTO Classes(topicName) VALUES ('Recap classes 10-17');
INSERT INTO Classes(topicName) VALUES ('Feeding Young Children Nutritious Foods');
INSERT INTO Classes(topicName) VALUES ('Keeping Children Safe/Child Abuse & Neglect');
INSERT INTO Classes(topicName) VALUES ('Improving Self-Worth/Children''s Self Worth');
INSERT INTO Classes(topicName) VALUES ('Developing Personal Power Adults/Children');
INSERT INTO Classes(topicName) VALUES ('Helping Children Manage Behavior');
INSERT INTO Classes(topicName) VALUES ('Attachment/Sepration & Loss');
INSERT INTO Classes(topicName) VALUES ('Understanding Discipline/Developing Family
Morals/Values/Rights');
INSERT INTO Classes(topicName) VALUES ('Using Rewards & Punishment to Guide/Teach
Children/Praise');
INSERT INTO Classes(topicName) VALUES ('Alternatives to Physical Punishment');
INSERT INTO Classes(topicName) VALUES ('Guest Speaker');
INSERT INTO Classes(topicName) VALUES ('Understanding & handling Stress');
INSERT INTO Classes(topicName) VALUES ('Keeping Children Keeping/Child Abuse & Neglect');
```

CurriculumClasses

```
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('How to be a good parent', 1);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('How to be a good parent', 2);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('How to be Cool', 1);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Parenting 101', 1);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Nurtuing/Culture/Spirituality',
7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Developing Empathy/Getting Needs
Met', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Recognizing & Undering Feelings',
7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Problem Solving & Decision
Making', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES
('Communication/Listening/Criticism/Confrontation/Fair Fighting', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Understanding & Expressing
Anger', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Talking about Effects of
Drugs/Alcohol/Smoking on Family', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Recap classes 1-8', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Relationships/Personal Space',
7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Children''s Brain Development',
7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Male/Female Brain/Quiz', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Ages & Stages: Appropriate
Expectations', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Ages & Stages: Infants to
Toddler', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Ages & Stages: Preschool to
Adolescence', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Establishing Nurturing Parenting
Routines', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Child Proofing Home/Safety
Checklist/Safety Reminders', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Recap classes 10-17', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Feeding Young Children Nutritious
Foods', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Keeping Children Safe/Child Abuse
& Neglect', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Improving Self-Worth/Children''s
Self Worth', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Developing Personal Power
Adults/Children', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Helping Children Manage
Behavior', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Attachment/Sepration & Loss', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Understanding
Discipline/Developing Family Morals/Values/Rights', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Using Rewards & Punishment to
Guide/Teach Children/Praise', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Alternatives to Physical
Punishment', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Guest Speaker', 7);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Developing Empathy/Getting Needs
Met', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Recognizing & Undering Feelings',
5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES
('Communication/Listening/Criticism/Confrontation/Fair Fighting', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Understanding & handling Stress',
5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Understanding & Expressing
Anger', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Relationships/Personal Space',
5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Establishing Nurturing Parenting
Routines', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Keeping Children Keeping/Child
Abuse & Neglect', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Improving Self-Worth/Children''s
Self Worth', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Developing Personal Power
Adults/Children', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Helping Children Manage
Behavior', 5);
INSERT INTO CurriculumClasses(topicName, curriculumID) VALUES ('Understanding
Discipline/Developing Family Morals/Values/Rights', 5);
```

Class Offering

```
INSERT INTO ClassOffering(topicName, date, siteName, lang, curriculumID) VALUES ('How to be a
good parent', '2017-09-23 05:22:21.649491', 'Dutchess County Jail', 'English', 1);
INSERT INTO ClassOffering(topicName, date, siteName, lang, curriculumID) VALUES ('How to be
Cool', '2017-09-23 05:22:21.649491', 'Dutchess County Jail', 'English', 1);
INSERT INTO ClassOffering(topicName, date, siteName, lang, curriculumID) VALUES ('Parenting 101',
'2017-09-23 05:22:21.649491', 'Dutchess County Jail', 'English', 1);
```

Facilitator Class Attendance

```
INSERT INTO FacilitatorClassAttendance(topicName, date, siteName, facilitatorID) VALUES ('How to
be a good parent', '2017-09-23 05:22:21.649491', 'Dutchess County Jail', 5);
```

Participant Class Attendance

```
INSERT INTO ParticipantClassAttendance(topicName, date, siteName, participantID) VALUES ('How to
be a good parent', '2017-09-23 05:22:21.649491', 'Dutchess County Jail', 4);
```

Facilitator Language

```
INSERT INTO FacilitatorLanguage(facilitatorID, lang, level) VALUES (5, 'English', 'PRIMARY');
```

Participant Out of House Site

```
INSERT INTO ParticipantOutOfHouseSite(outOfHouseID, siteName) VALUES (4, 'Dutchess County Jail');
```

“STORED PROCEDURES”

Stored Procedures are sets of SQL statements with an assigned name, which are stored in the RDBMS as a group, so it can be reused and shared. Stored procedures can reduce traffic, because SQL statements can be executed in batches rather than sending multiple requests, ultimately expediting everyday data transactions.

PeopleInsert

Inserts information into a Person object

Author: John Randis

```
CREATE OR REPLACE FUNCTION peopleInsert(fname TEXT DEFAULT NULL::text,
        lname TEXT DEFAULT NULL::text,
        mInit VARCHAR DEFAULT NULL::varchar)
        RETURNS VOID AS
$BODY$
    BEGIN
        INSERT INTO People(firstName, lastName, middleInit) VALUES (fname, lname, mInit);
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

ZipCodeSafeInsert

Inserts information into a Zip Code object

Author: Marcos Barbieri

```
CREATE OR REPLACE FUNCTION zipCodeSafeInsert(INT, TEXT, StATES) RETURNS VOID AS
$func$
    DECLARE
        zip     INT    := $1;
        city    TEXT   := $2;
        state   STATES   := $3;
    BEGIN
        IF NOT EXISTS (SELECT ZipCodes.zipCode FROM ZipCodes WHERE ZipCodes.zipCode = zip) THEN
            INSERT INTO ZipCodes VALUES (zip, city, CAST(state AS STATES));
        END IF;
    END;
$func$ LANGUAGE plpgsql;
```

RegisterParticipantIntake

Registers the participate intake form

Author: Marcos Barbieri

```
CREATE OR REPLACE FUNCTION registerParticipantIntake(
    fname text DEFAULT NULL::text,
    lname text DEFAULT NULL::text,
    dob date DEFAULT NULL::date,
    race text DEFAULT NULL::text,
    housenum integer DEFAULT NULL::integer,
    streetaddress text DEFAULT NULL::text,
    apartmentInfo TEXT DEFAULT NULL::TEXT,
    zipcode integer DEFAULT NULL::integer,
    city text DEFAULT NULL::text,
    state text DEFAULT NULL::text,
    occupation text DEFAULT NULL::text,
    religion text DEFAULT NULL::text,
    ethnicity text DEFAULT NULL::text,
    handicapsormedication text DEFAULT NULL::text,
    lastyearschool text DEFAULT NULL::text,
    hasdrugabusehist boolean DEFAULT NULL::boolean,
    substanceabusedescr text DEFAULT NULL::text,
    timeseparatedfromchildren text DEFAULT NULL::text,
    timeseparatedfrompartner text DEFAULT NULL::text,
    relationshiptootherparent text DEFAULT NULL::text,
    hasparentingpartnershiphistory boolean DEFAULT NULL::boolean,
    hasInvolvementCPS boolean DEFAULT NULL::boolean,
    hasprevinvolvmentcps text DEFAULT NULL::text,
    ismandatedtotakeclass boolean DEFAULT NULL::boolean,
    whomandatedclass text DEFAULT NULL::text,
    reasonforattendence text DEFAULT NULL::text,
    safeparticipate text DEFAULT NULL::text,
    preventparticipate text DEFAULT NULL::text,
    hasattendedotherparenting boolean DEFAULT NULL::boolean,
    kindofparentingclasstaken text DEFAULT NULL::text,
    victimchildabuse boolean DEFAULT NULL::boolean,
    formofchildhoodabuse text DEFAULT NULL::text,
    hashadtherapy boolean DEFAULT NULL::boolean,
    stillissuesfromchildabuse boolean DEFAULT NULL::boolean,
    mostimportantliketolearn text DEFAULT NULL::text,
    hasdomesticviolencehistory boolean DEFAULT NULL::boolean,
    hasdiscusseddomesticviolence boolean DEFAULT NULL::boolean,
    hashistorychildabuseoriginfam boolean DEFAULT NULL::boolean,
    hashistoryviolencenuclearfamily boolean DEFAULT NULL::boolean,
    ordersofprotectioninvolved boolean DEFAULT NULL::boolean,
    reasonforordersofprotection text DEFAULT NULL::text,
    hasbeenarrested boolean DEFAULT NULL::boolean,
    hasbeenconvicted boolean DEFAULT NULL::boolean,
    reasonforarrestorconviction text DEFAULT NULL::text,
    hasjailrecord boolean DEFAULT NULL::boolean,
    hasprisonrecord boolean DEFAULT NULL::boolean,
    offensejailprisonrec text DEFAULT NULL::text,
    currentlyonparole boolean DEFAULT NULL::boolean,
    onparoleforwhatoffense text DEFAULT NULL::text,
    ptpmainformsigneddate date DEFAULT NULL::date,
    ptpenrollmentsigneddate date DEFAULT NULL::date,
    ptpconstentreleaseformsigneddate date DEFAULT NULL::date,
    employeeemail text DEFAULT NULL::text)
  RETURNS void AS
$BODY$
    DECLARE
        eID                     INT;
        participantID                  INT;
        adrID                          INT;
        signedDate                     DATE;
        formID                         INT;
    BEGIN
        -- First make sure that the employee is in the database. We don't want to authorize dirty inserts
        PERFORM Employees.employeeID FROM Employees WHERE Employees.email = employeeEmail;
        IF FOUND THEN
            eID := (SELECT Employees.employeeID FROM Employees WHERE Employees.email = employeeEmail);
            RAISE NOTICE 'employee %', eID;
            -- Now check if the participant already exists in the system
            PERFORM Participants.participantID FROM Participants
                      WHERE Participants.participantID = (SELECT People.peopleID
                                                           FROM People
                                                           WHERE People.firstName = fName AND People.lastName =
lName);
            IF FOUND THEN
                participantID := (SELECT Participants.participantID FROM Participants
                                  WHERE Participants.participantID = (SELECT People.peopleID
                                                                       FROM People
                                                                       WHERE People.firstName = fName AND
People.lastName = lName));
                RAISE NOTICE 'participant %', participantID;

                -- Handling anything relating to Address/Location information
                PERFORM zipCodeSafeInsert(registerParticipantIntake.zipCode, city, state);
                RAISE NOTICE 'zipCode %', (SELECT ZipCodes.zipCode FROM ZipCodes WHERE ZipCodes.city =
registerParticipantIntake.city AND
                                                                                    ZipCodes.state =
registerParticipantIntake.state::STATES);
                RAISE NOTICE 'Address info % % % %', houseNum, streetAddress, apartmentInfo,
registerParticipantIntake.zipCode;
                INSERT INTO Addresses(addressNumber, street, aptInfo, zipCode) VALUES (houseNum, streetAddress,
apartmentInfo, registerParticipantIntake.zipCode);
                adrID := (SELECT Addresses.addressID FROM Addresses WHERE Addresses.addressNumber =
registerParticipantIntake.houseNum AND
                                                                               Addresses.street =
registerParticipantIntake.streetAddress AND
                                                                               --Addresses.aptInfo =
registerParticipantIntake.apartmentInfo AND
                                                                               Addresses.zipCode =
registerParticipantIntake.zipCode);

                -- Fill in the actual form information
                RAISE NOTICE 'address %', adrID;
                signedDate := (current_date);
                INSERT INTO Forms(addressID, employeeSignedDate, employeeID) VALUES (adrID, signedDate, eID);
                formID := (SELECT Forms.formID FROM Forms WHERE Forms.addressID = adrID AND
                                                                Forms.employeeSignedDate = signedDate AND
                                                                Forms.employeeID = eID);
                RAISE NOTICE 'formID %', formID;
                INSERT INTO IntakeInformation VALUES (formID,
                                                      occupation,
                                                      religion,
                                                      ethnicity,
                                                      handicapsOrMedication,
                                                      lastYearSchool,
                                                      hasDrugAbuseHist,
                                                      substanceAbuseDescr,
                                                      timeSeparatedFromChildren,
                                                      timeSeparatedFromPartner,
                                                      relationshipToOtherParent,
                                                      hasParentingPartnershipHistory,
                                                      hasInvolvementCPS,
                                                      hasPrevInvolvmentCPS,
                                                      isMandatedToTakeClass,
                                                      whoMandatedClass,
                                                      reasonForAttendence,
                                                      safeParticipate,
                                                      preventParticipate,
                                                      hasAttendedOtherParenting,
                                                      kindOfParentingClassTaken,
                                                      victimChildAbuse,
                                                      formOfChildhoodAbuse,
                                                      hasHadTherapy,
                                                      stillIssuesFromChildAbuse,
                                                      mostImportantLikeToLearn,
                                                      hasDomesticViolenceHistory,
                                                      hasDiscussedDomesticViolence,
                                                      hasHistoryChildAbuseOriginFam,
                                                      hasHistoryViolenceNuclearFamily,
                                                      ordersOfProtectionInvolved,
                                                      reasonForOrdersOfProtection,
                                                      hasBeenArrested,
                                                      hasBeenConvicted,
                                                      reasonForArrestOrConviction,
                                                      hasJailRecord,
                                                      hasPrisonRecord,
                                                      offenseJailPrisonRec,
                                                      currentlyOnParole,
                                                      onParoleForWhatOffense,
                                                      ptpMainFormSignedDate,
                                                      ptpEnrollmentSignedDate,
                                                      ptpConstentReleaseFormSignedDate
                                                  );
                INSERT INTO ParticipantsFormDetails VALUES (participantID, formID);
            ELSE
                RAISE EXCEPTION 'Was not able to find participant';
            END IF;
        ELSE
            RAISE EXCEPTION 'Was not able to find employee';
        END IF;
    END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Employee

Inserts a new person into the Employees table and links them with an ID in the People table

Author: Carson Badame

```
CREATE OR REPLACE FUNCTION employeeInsert(
    fname TEXT DEFAULT NULL::text,
    lname TEXT DEFAULT NULL::text,
    mInit VARCHAR DEFAULT NULL::varchar,
    em TEXT DEFAULT NULL::text,
    pPhone TEXT DEFAULT NULL::text,
    pLevel PERMISSION DEFAULT 'Coordinator'::PERMISSION)
RETURNS VOID AS
$BODY$
    DECLARE
        eID INT;
    BEGIN
        PERFORM Employees.employeeID FROM People, Employees WHERE People.firstName = fname AND People.lastName =
lname AND People.middleInit = mInit AND People.peopleID = Employees.employeeID;
        IF FOUND THEN
            RAISE NOTICE 'Employee already exists.';
        ELSE
            -- Checks to see if new employee already exists in People table
            PERFORM People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname AND
People.middleInit = mInit;
            -- If they do, insert link them to peopleID and insert into Employees table
            IF FOUND THEN
                eID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname
AND People.middleInit= mInit);
                RAISE NOTICE 'people %', eID;
                INSERT INTO Employees(employeeID, email, primaryPhone, permissionLevel) VALUES (eID, em, pPhone, 
pLevel);
            -- Else create new person in People table and then insert them into Employees table
            ELSE
                INSERT INTO People(firstName, lastName, middleInit) VALUES (fname, lname, mInit);
                eID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname 
AND People.middleInit = mInit);
                RAISE NOTICE 'people %', eID;
                INSERT INTO Employees(employeeID, email, primaryPhone, permissionLevel) VALUES (eID, em, pPhone, 
pLevel);
            END IF;
        END IF;
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Facilitator

Inserts a new person into the Facilitators table and links them with an ID in the Employees and People tables

Author: Carson Badame

```
CREATE OR REPLACE FUNCTION facilitatorInsert(
    fname TEXT DEFAULT NULL::text,
    lname TEXT DEFAULT NULL::text,
    mInit VARCHAR DEFAULT NULL::varchar,
    em TEXT DEFAULT NULL::text,
    pPhone TEXT DEFAULT NULL::text,
    pLevel PERMISSION DEFAULT 'Coordinator'::PERMISSION)
RETURNS VOID AS
$BODY$
    DECLARE
        fID INT;
        eReturn TEXT;
    BEGIN
    -- Check to see if the facilitator already exists
        PERFORM Facilitators.facilitatorID FROM People, Employees, Facilitators WHERE People.firstName = fname AND
People.lastName = lname AND People.middleInit = mInit AND People.peopleID = Employees.employeeID AND
Employees.employeeID = Facilitators.facilitatorID;
        -- If they do, do need insert anything
        IF FOUND THEN
            RAISE NOTICE 'Facilitator already exists.';
        ELSE
            -- If they do not, check to see if they exists as an employee
            PERFORM Employees.employeeID FROM Employees, People WHERE People.firstName = fname AND People.lastName =
lname AND People.middleInit = mInit AND People.peopleID = Employees.employeeID;
            -- If they do, then add the facilitator and link them to the employee
            IF FOUND THEN
                fID := (SELECT Employees.employeeID FROM Employees, People WHERE People.firstName = fname AND
People.lastName = lname AND People.middleInit = mInit AND People.peopleID = Employees.employeeID);
                RAISE NOTICE 'employee %', fID;
                INSERT INTO Facilitators(facilitatorID) VALUES (fID);
            -- If they do not, run the employeeInsert function and then add the facilitator
            ELSE
```

```
                SELECT employeeInsert(fname, lname, mInit, em, pPhone, pLevel) INTO eReturn;
                fID := (SELECT Employees.employeeID FROM Employees, People WHERE People.firstName = fname AND
People.lastName = lname AND People.middleInit = mInit AND People.peopleID = Employees.employeeID);
                RAISE NOTICE 'employee %', fID;
                INSERT INTO Facilitators(facilitatorID) VALUES (fID);
            END IF;
        END IF;
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Agency Member

Inserts a new person into the ContactAgencyMembers table and links them with an ID in the People table

Author: John Randis and Carson Badame

```
CREATE OR REPLACE FUNCTION agencyMemberInsert(
    fname TEXT DEFAULT NULL::text,
    lname TEXT DEFAULT NULL::text,
    mInit VARCHAR DEFAULT NULL::varchar,
    agen REFERRALTYPE DEFAULT NULL::referraltype,
    phn INT DEFAULT NULL::int,
    em TEXT DEFAULT NULL::text,
    isMain BOOLEAN DEFAULT NULL::boolean,
    arID INT DEFAULT NULL::int)
RETURNS VOID AS
$BODY$
    DECLARE
        caID INT;
        pReturn TEXT;
    BEGIN
    -- Check to see if the agency member already exists
        PERFORM ContactAgencyMembers.contactAgencyID FROM ContactAgencyMembers, People WHERE People.firstName = fname
AND People.lastName = lname AND People.middleInit = mInit AND People.peopleID = ContactAgencyMembers.contactAgencyID;
        -- If they do, do not insert anything
        IF FOUND THEN
            RAISE NOTICE 'Agency member already exists.';
        ELSE
            -- If they do not, check to see if they exists as an a person
            PERFORM People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname AND
People.middleInit = mInit;
            -- If they do, then add the agency member and link them to the employee
            IF FOUND THEN
                caID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName =
lname AND People.middleInit = mInit);
                RAISE NOTICE 'AgencyMember %', caID;
                INSERT INTO ContactAgencyMembers(contactAgencyID, agency, phone, email) VALUES (caID, agen, phn, em);
                INSERT INTO ContactAgencyAssociatedWithReferred(contactAgencyID, agencyReferralID, isMainContact)
```

```
 (caID, agen, phn, em);
                INSERT INTO ContactAgencyAssociatedWithReferred(contactAgencyID, agencyReferralID, isMainContact)
VALUES (caID, arID, isMain);
            -- If they do not, run create the person and then add them as an agency member
            ELSE
                INSERT INTO People(firstName, lastName, middleInit) VALUES (fname, lname, mInit);
                caID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName =
lname AND People.middleInit = mInit);
                RAISE NOTICE 'AgencyMember %', caID;
                INSERT INTO ContactAgencyMembers(contactAgencyID, agency, phone, email) VALUES (caID, agen, phn, em);
                INSERT INTO ContactAgencyAssociatedWithReferred(contactAgencyID, agencyReferralID, isMainContact)
VALUES (caID, arID, isMain);
            END IF;
        END IF;
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Add Agency Referral

Adds an Agency Referral Form

Author: John Randis and Carson Badame

```
CREATE OR REPLACE FUNCTION addAgencyReferral(
  fName TEXT DEFAULT NULL::TEXT,
  lName TEXT DEFAULT NULL::TEXT,
  mInit VARCHAR DEFAULT NULL::VARCHAR,
  dob DATE DEFAULT NULL::DATE,
  race RACE DEFAULT NULL::RACE,
  gender SEX DEFAULT NULL::SEX,
  housenum INTEGER DEFAULT NULL::INTEGER,
  streetaddress TEXT DEFAULT NULL::TEXT,
  apartmentInfo TEXT DEFAULT NULL::TEXT,
  zipcode INTEGER DEFAULT NULL::INTEGER,
  city TEXT DEFAULT NULL::TEXT,
  state STATES DEFAULT NULL::STATES,
  referralReason TEXT DEFAULT NULL::TEXT,
  hasAgencyConsentForm BOOLEAN DEFAULT FALSE::BOOLEAN,
  referringAgency TEXT DEFAULT NULL::TEXT,
  referringAgencyDate DATE DEFAULT NULL::DATE,
  additionalInfo TEXT DEFAULT NULL::TEXT,
  hasSpecialNeeds BOOLEAN DEFAULT NULL::BOOLEAN,
  hasSubstanceAbuseHistory BOOLEAN DEFAULT NULL::BOOLEAN,
  hasInvolvementCPS BOOLEAN DEFAULT NULL::BOOLEAN,
  isPregnant BOOLEAN DEFAULT NULL::BOOLEAN,
  hasIQDoc BOOLEAN DEFAULT NULL::BOOLEAN,
  mentalHealthIssue BOOLEAN DEFAULT NULL::BOOLEAN,
  hasDomesticViolenceHistory BOOLEAN DEFAULT NULL::BOOLEAN,
  childrenLiveWithIndividual BOOLEAN DEFAULT NULL::BOOLEAN,
  dateFirstContact DATE DEFAULT NULL::DATE,
  meansOfContact TEXT DEFAULT NULL::TEXT,
  dateOfInitialMeeting TIMESTAMP DEFAULT NULL::DATE,
  location TEXT DEFAULT NULL::TEXT,
  comments TEXT DEFAULT NULL::TEXT,
  eID INT DEFAULT NULL::INT)
  RETURNS int AS
      $BODY$
          DECLARE
              participantID             INT;
              agencyReferralID    INT;
              contactAgencyID           INT;
              adrID               INT;
              signedDate          DATE;
              formID                           INT;
              participantReturn TEXT;
          BEGIN
              PERFORM Participants.participantID FROM Participants
                                    WHERE Participants.participantID = (SELECT People.peopleID
                                                                        FROM People
                                                                        WHERE People.firstName = fName AND
People.lastName = lName);
              IF FOUND THEN
                participantID := (SELECT Participants.participantID FROM Participants
                                      WHERE Participants.participantID = (SELECT People.peopleID
                                                                          FROM People
                                                                          WHERE People.firstName = fName AND
People.lastName = lName));
                RAISE NOTICE 'participant %', participantID;

                 -- Handling anything relating to Address/Location information
                PERFORM zipCodeSafeInsert(addAgencyReferral.zipCode, city, state);
                RAISE NOTICE 'zipCode %', addAgencyReferral.zipCode;
                RAISE NOTICE 'Address info % % % %', houseNum, streetAddress, apartmentInfo,
addAgencyReferral.zipCode;
                INSERT INTO Addresses(addressNumber, street, aptInfo, zipCode) VALUES (houseNum, streetAddress,
apartmentInfo, addAgencyReferral.zipCode);
                adrID := (SELECT Addresses.addressID FROM Addresses WHERE Addresses.addressNumber =
```

Create Family Member

Creates a family member in the database

Author: Jesse Opitz

```
CREATE OR REPLACE FUNCTION createFamilyMember(
    fname TEXT DEFAULT NULL::text,
    lname TEXT DEFAULT NULL::text,
    mInit VARCHAR DEFAULT NULL::varchar,
    rel RELATIONSHIP DEFAULT NULL::relationship,
    dob DATE DEFAULT NULL::date,
    rac RACE DEFAULT NULL::race,
    gender SEX DEFAULT NULL::sex,
    -- IF child is set to True
    -- -- Inserts child information
    child BOOLEAN DEFAULT NULL::boolean,
    cust TEXT DEFAULT NULL::text,
    loc TEXT DEFAULT NULL::text,
    fID INT DEFAULT NULL::int)
RETURNS VOID AS
$BODY$
    DECLARE
        fmID INT;
        pReturn TEXT;
    BEGIN
        SELECT peopleInsert(fname, lname, mInit) INTO pReturn;
        fmID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname AND
People.middleInit = mInit);
        RAISE NOTICE 'FamilyMember %', fmID;
        INSERT INTO FamilyMembers(familyMemberID, relationship, dateOfBirth, race, sex) VALUES (fmID, rel, dob, rac,
gender);
        IF child = True THEN
          INSERT INTO Children(childrenID, custody, location) VALUES(fmID, cust, loc);
        END IF;
        INSERT INTO Family(familyMembersID, formID) VALUES (fmID, fID);
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Participants

Creates a participant in the correct order

Author: Jesse Opitz and Marcos Barbieri

```
-- Stored Procedure for Creating Participants
DROP FUNCTION IF EXISTS createParticipants(TEXT, TEXT, VARCHAR, DATE, RACE, SEX);
CREATE OR REPLACE FUNCTION createParticipants(
    fname TEXT DEFAULT NULL::text,
    lname TEXT DEFAULT NULL::text,
    mInit VARCHAR DEFAULT NULL::varchar,
    dob DATE DEFAULT NULL::date,
    rac RACE DEFAULT NULL::race,
    gender SEX DEFAULT NULL::sex)
RETURNS VOID AS
$BODY$
    DECLARE
        partID INT;
        pReturn TEXT;
    BEGIN
        PERFORM peopleInsert(fname, lname, mInit);
        SELECT People.peopleID INTO partID
        FROM People
        WHERE People.firstName = fname AND
              People.lastName = lname AND
              People.middleInit = mInit;
        RAISE NOTICE 'people %', partID;
        INSERT INTO Participants(participantID, dateOfBirth, race, sex) VALUES (partID, dob, rac, gender);
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Participant Attendance Insert

Used for inserting the attendance record for a participant for a specific class offering

Author: Marcos Barbieri

```
CREATE OR REPLACE FUNCTION participantAttendanceInsert(
    attendanceParticipantFName TEXT DEFAULT NULL::TEXT,
    attendanceParticipantMiddleInit TEXT DEFAULT NULL::TEXT,
    attendanceParticipantLName TEXT DEFAULT NULL::TEXT,
    attendanceParticipantSex SEX DEFAULT NULL::SEX,
    attendanceParticipantRace RACE DEFAULT NULL::RACE,
    attendanceParticipantAge INT DEFAULT NULL::INT,
    attendanceTopic TEXT DEFAULT NULL::TEXT,
    attendanceDate TIMESTAMP DEFAULT NULL::TIMESTAMP,
    attendanceSiteName TEXT DEFAULT NULL::TEXT,
    attendanceComments TEXT DEFAULT NULL::TEXT,
    attendanceNumChildren INT DEFAULT NULL::INT,
    isAttendanceNew BOOLEAN DEFAULT NULL::BOOLEAN,
    attendanceParticipantZipCode INT DEFAULT NULL::INT,
    inHouseFlag BOOLEAN DEFAULT FALSE::BOOLEAN
)
RETURNS VOID AS
$BODY$
    DECLARE
        ptpId INT;
    BEGIN
        PERFORM ClassOffering.topicName
        FROM ClassOffering
        WHERE ClassOffering.topicName = attendanceTopic AND
              ClassOffering.date = attendanceDate AND
```

```
            ClassOffering.siteName = attendanceSiteName;
        IF FOUND THEN
            -- Try and see if the information matches any individual
            PERFORM ParticipantInfo.ParticipantID
            FROM ParticipantInfo
            WHERE ParticipantInfo.firstName=attendanceParticipantFName AND
                  ParticipantInfo.middleInit=attendanceParticipantMiddleInit AND
                  ParticipantInfo.lastName=attendanceParticipantLName AND
                  ParticipantInfo.sex=attendanceParticipantSex AND
                  ParticipantInfo.race=attendanceParticipantRace AND
                  date_part('year', ParticipantInfo.dateOfBirth)=CalculateDOB(attendanceParticipantAge);
            IF FOUND THEN
                -- Make sure only one match is found. If two are found we can't do
                -- anything about it. We have to leave it up to the developers to ask
                -- the users
                PERFORM MatchedPeopleCount.count
                FROM (SELECT COUNT(ParticipantInfo.ParticipantID)
                      FROM ParticipantInfo
                      WHERE ParticipantInfo.firstName=attendanceParticipantFName AND
                            ParticipantInfo.middleInit=attendanceParticipantMiddleInit AND
                            ParticipantInfo.lastName=attendanceParticipantLName AND
                            ParticipantInfo.sex=attendanceParticipantSex AND
                            ParticipantInfo.race=attendanceParticipantRace AND
                            date_part('year', ParticipantInfo.dateOfBirth)=CalculateDOB(attendanceParticipantAge)) AS
MatchedPeopleCount
                WHERE MatchedPeopleCount.count = 1;
                IF FOUND THEN
                    SELECT ParticipantInfo.ParticipantID
                    INTO ptpId
                    FROM ParticipantInfo
                    WHERE ParticipantInfo.firstName=attendanceParticipantFName AND
                          ParticipantInfo.middleInit=attendanceParticipantMiddleInit AND
                          ParticipantInfo.lastName=attendanceParticipantLName AND
                          ParticipantInfo.sex=attendanceParticipantSex AND
```

```
                           ParticipantInfo.race=attendanceParticipantRace AND
                           date_part('year', ParticipantInfo.dateOfBirth)=CalculateDOB(attendanceParticipantAge);
                     -- Still need to verify that sitename and topic exist
                     INSERT INTO ParticipantClassAttendance VALUES (attendanceTopic,
                                                                    attendanceDate,
                                                                    attendanceSiteName,
                                                                    ptpId,
                                                                    attendanceComments,
                                                                    attendanceNumChildren,
                                                                    isAttendanceNew,
                                                                    attendanceParticipantZipCode);
                 ELSE
                     RAISE EXCEPTION
                         'Multiple participants with the same information were found'
                         USING HINT = 'Please specify the ID and fields necessary and insert directly into the table';
                 END IF;
             ELSE
                 IF (inHouseFlag IS FALSE) THEN
                     PERFORM createParticipants(fname := attendanceParticipantFName::TEXT,
                         lname := attendanceParticipantLName::text,
                         mInit := attendanceParticipantMiddleInit::VARCHAR,
                         dob := make_date((date_part('year', current_date)-attendanceParticipantAge)::INT, 1,
1)::DATE,
                         rac := attendanceParticipantRace::RACE,
                         gender := attendanceParticipantSex::SEX);
                     PERFORM ParticipantAttendanceInsert(
                         attendanceParticipantFName := attendanceParticipantFName::TEXT,
                         attendanceParticipantMiddleInit := attendanceParticipantMiddleInit::TEXT,
                         attendanceParticipantLName := attendanceParticipantLName::TEXT,
                         attendanceParticipantSex := attendanceParticipantSex::SEX,
                         attendanceParticipantRace := attendanceParticipantRace::RACE,
                         attendanceParticipantAge := attendanceParticipantAge::INT,
                         attendanceTopic := attendanceTopic::TEXT,
                         attendanceDate := attendanceDate::TIMESTAMP,
                         attendanceSiteName := attendanceSiteName::TEXT,
                         attendanceComments := attendanceComments::TEXT,
                         attendanceNumChildren := attendanceNumChildren::INT,
                         isAttendanceNew := isAttendanceNew::BOOLEAN,
                         attendanceParticipantZipCode := attendanceParticipantZipCode::INT,
                         inHouseFlag := inHouseFlag::BOOLEAN
                     );
                 END IF;
             END IF;
         ELSE
             RAISE NOTICE 'Creating class offering with topic name %', attendanceTopic;
             PERFORM CreateClassOffering(
                 offeringTopicName := attendanceTopic::TEXT,
                 offeringTopicDescription := ''::TEXT,
                 offeringTopicDate := attendanceDate::TIMESTAMP,
                 offeringSiteName := attendanceSiteName::TEXT,
                 offeringLanguage := 'English'::TEXT,
                 offeringCurriculumId := NULL::INT);
         END IF;
     END
 $BODY$
     LANGUAGE plpgsql VOLATILE;
```

Create Class Offering

Creates the class (if necessary) and the class offering for a specific course

Author:

```
CREATE OR REPLACE FUNCTION CreateClassOffering(
    offeringTopicName TEXT DEFAULT NULL::TEXT,
    offeringTopicDescription TEXT DEFAULT NULL::TEXT,
    offeringTopicDate TIMESTAMP DEFAULT NULL::TIMESTAMP,
    offeringSiteName TEXT DEFAULT NULL::TEXT,
    offeringLanguage TEXT DEFAULT NULL::TEXT,
    offeringCurriculumId INT DEFAULT NULL::INT)
RETURNS VOID AS
$BODY$
BEGIN
    PERFORM Classes.topicName
    FROM Classes
    WHERE Classes.topicName=offeringTopicName;
    IF FOUND THEN
        PERFORM ClassOffering.topicName
        FROM ClassOffering
        WHERE ClassOffering.topicName=offeringTopicName AND
              ClassOffering.date=offeringTopicDate AND
              ClassOffering.siteName=offeringSiteName;
        IF FOUND THEN
            RAISE EXCEPTION 'Class Offering already exists --> %', offeringTopicName
                 USING HINT = 'Please check topicName, date and siteName';
        ELSE
            INSERT INTO ClassOffering VALUES (offeringTopicName, offeringTopicDate, offeringSiteName,
offeringLanguage, offeringCurriculumId);
        END IF;
    ELSE
        INSERT INTO Classes VALUES (offeringTopicName, offeringTopicDescription);
        PERFORM CreateClassOffering(
            offeringTopicName := offeringTopicName::TEXT,
            offeringTopicDescription := offeringTopicDescription::TEXT,
            offeringTopicDate := offeringTopicDate::TIMESTAMP,
            offeringLanguage := offeringLanguage::TEXT,
            offeringCurriculumId := offeringCurriculumId::INT);
    END IF;
END
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Calculate DOB

Takes the age and subtracts it from the current year to get an age estimate; the reasoning is that the PEP program only asks for age on certain forms, not DOB—however, age should neber be stored (only DOB), so we must calculate this manually

Author: Marcos Barbieri

```
CREATE OR REPLACE FUNCTION calculateDOB(age INT DEFAULT NULL::INT)
 RETURNS INT AS
 $BODY$
    DECLARE
        currentYear INT := date_part('year', CURRENT_DATE);
        dob INT;
    BEGIN
        dob := currentYear - age;
        RETURN dob;
    END
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Self Referral

Inserts a new referral form to the addSelfReferral table and links them with an ID in the Forms, Participants, and People tables

Author: Carson Badame

```
CREATE OR REPLACE FUNCTION addSelfReferral(
    fName TEXT DEFAULT NULL::TEXT,
    lName TEXT DEFAULT NULL::TEXT,
    mInit VARCHAR DEFAULT NULL::VARCHAR,
    dob DATE DEFAULT NULL::DATE,
    raceVal RACE DEFAULT NULL::RACE,
    sexVal SEX DEFAULT NULL::SEX,
    houseNum INTEGER DEFAULT NULL::INTEGER,
    streetAddress TEXT DEFAULT NULL::TEXT,
    apartmentInfo TEXT DEFAULT NULL::TEXT,
    zip INTEGER DEFAULT NULL::INTEGER,
    cityName TEXT DEFAULT NULL::TEXT,
    stateName STATES DEFAULT NULL::STATES,
    refSource TEXT DEFAULT NULL::TEXT,
    hasInvolvement BOOLEAN DEFAULT NULL::BOOLEAN,
    hasAttended BOOLEAN DEFAULT NULL::BOOLEAN,
    reasonAttending TEXT DEFAULT NULL::TEXT,
    firstCall DATE DEFAULT NULL::DATE,
    returnCallDate DATE DEFAULT NULL::DATE,
    startDate DATE DEFAULT NULL::DATE,
    classAssigned TEXT DEFAULT NULL::TEXT,
    letterMailedDate DATE DEFAULT NULL::DATE,
    extraNotes TEXT DEFAULT NULL::TEXT,
    eID INT DEFAULT NULL::INT)
    RETURNS void AS
        $BODY$
            DECLARE
                pID                 INT;
                fID                 INT;
                adrID               INT;
                srID                INT;
                signedDate          DATE;
            BEGIN
                -- Check if the person already exists in the db
                PERFORM People.peopleID FROM People WHERE People.firstName = fname AND People.lastName = lname AND
```

```
People.middleInit = mInit;
                IF FOUND THEN
                    pID := (SELECT People.peopleID FROM People WHERE People.firstName = fname AND People.lastName =
lname AND People.middleInit = mInit);
                    PERFORM * FROM Participants WHERE Participants.participantID = pID;

                    IF FOUND THEN
                        RAISE NOTICE 'participant %', pID;

                         -- Handling anything relating to Address/Location information
                        PERFORM zipcode FROM ZipCodes WHERE ZipCodes.city = cityName AND ZipCodes.state =
stateName::STATES;
                        IF FOUND THEN
                          RAISE NOTICE 'Zipcode already exists.';
                        ELSE
                          INSERT INTO ZipCodes(zipcode, city, state) VALUES (zip, cityName, stateName);
                          RAISE NOTICE 'zipCode %', (SELECT zipcode FROM ZipCodes WHERE ZipCodes.city = cityName AND
ZipCodes.state = stateName::STATES);
                        END IF;
                        RAISE NOTICE 'Address info % % % %', houseNum, streetAddress, apartmentInfo, zip;
                        INSERT INTO Addresses(addressNumber, street, aptInfo, zipCode) VALUES (houseNum,
streetAddress, apartmentInfo, zip);
                        adrID := (SELECT Addresses.addressID FROM Addresses WHERE Addresses.addressNumber = houseNum
AND
                                                                                      Addresses.street =
streetAddress AND
                                                                                      Addresses.zipCode = zip);

                        -- Fill in the actual form information
                        RAISE NOTICE '+ %', adrID;
                        signedDate := (current_date);
                        INSERT INTO Forms(addressID, employeeSignedDate, employeeID, participantID) VALUES (adrID,
signedDate, eID, pID);
                        fID := (SELECT Forms.formID FROM Forms WHERE Forms.addressID = adrID AND
                                                                        Forms.employeeSignedDate = signedDate AND
Forms.employeeID = eID);

                        RAISE NOTICE 'formID %', fID;
                        INSERT INTO SelfReferral VALUES (selfReferralID,
                                                           refSource,
                                                           hasInvolvement,
                                                           hasAttended,
                                                           reasonAttending,
                                                           firstCall,
                                                           returnCallDate,
                                                           startDate,
                                                           classAssigned,
                                                           letterMailedDate,
                                                           extraNotes);

                    ELSE
                        INSERT INTO Participants(participantID, dateOfBirth, race, sex) VALUES (pID, dob, raceVal,
sexVal);
                        PERFORM addSelfReferral(fName, lName, mInit, dob, raceVal, sexVal, houseNum, streetAddress,
apartmentInfo, zip, cityName, stateName, refSource, hasInvolvement,
                            hasAttended, reasonAttending, firstCall, returnCallDate, startDate, classAssigned,
letterMailedDate, extraNotes);
                    END IF;
                ELSE
                    INSERT INTO People(firstName, lastName, middleInit) VALUES (fName, lName, mInit);
                    PERFORM addSelfReferral(fName, lName, mInit, dob, raceVal, sexVal, houseNum, streetAddress,
apartmentInfo, zip, cityName, stateName, refSource, hasInvolvement,
                            hasAttended, reasonAttending, firstCall, returnCallDate, startDate, classAssigned,
letterMailedDate, extraNotes);
                END IF;
            END;
        $BODY$
  LANGUAGE plpgsql VOLATILE;
```

Create Emergency Contact

Used to create an emergency contact by other stored procedures

Author: Jesse Opitz

```
DROP FUNCTION IF EXISTS createEmeregencyContact();
CREATE OR REPLACE FUNCTION createEmergencyContact(
    pID INT DEFAULT NULL::int,
    intInfoID INT DEFAULT NULL::int,
    rel RELATIONSHIP DEFAULT NULL::relationship,
    phon TEXT DEFAULT NULL::text
)
RETURNS VOID AS
$BODY$
    DECLARE
    BEGIN
        INSERT INTO EmergencyContacts(emergencyContactID, relationship, phone) VALUES (pID, rel, phon);
        INSERT INTO  EmergencyContactDetail(emergencyContactID, intakeInformationID) VALUES (pID, intInfoID);
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Create Curriculum

Links topic to a new curriculum

Author: Jesse Opitz

```
DROP FUNCTION IF EXISTS createCurriculum();
CREATE OR REPLACE FUNCTION createCurriculum(
    tnID INT DEFAULT NULL::int,
    currName TEXT DEFAULT NULL::text,
    currType PROGRAMTYPE DEFAULT NULL::programtype,
    missNum INT DEFAULT NULL::int
)
RETURNS INT AS
$BODY$
    DECLARE
    cID INT;
    BEGIN
        INSERT INTO Curricula(curriculumName, curriculumType, missNumber) VALUES (currName, currType, missNum);
        SELECT Curricula.curriculumID FROM Curricula WHERE Curriclua.curriculumName = currName AND
Curricula.curriculumType = currType AND Curricula.missNumber = missNum INTO cID;
        RETURN cID;
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

Create Out of House Participant

Create a new Out of House Participant making sure all information is stored soundly

Author: Marcos Barbieri

```
DROP FUNCTION IF EXISTS createOutOfHouseParticipant(TEXT, TEXT, TEXT, INT, RACE, TEXT);
     CREATE OR REPLACE FUNCTION createOutOfHouseParticipant(
        participantFirstName TEXT DEFAULT NULL::TEXT,
        participantMiddleInit TEXT DEFAULT NULL::TEXT,
        participantLastName TEXT DEFAULT NULL::TEXT,
        participantAge INT DEFAULT NULL::INT,
        participantRace RACE DEFAULT NULL::RACE,
        participantSex SEX DEFAULT NULL::SEX,
        participantDescription TEXT DEFAULT NULL::TEXT)
    RETURNS INT AS
    $BODY$
        DECLARE
            dateOfBirth DATE;
            ptpID INT;
        BEGIN
            PERFORM OutOfHouse.outOfHouseID
            FROM People
            INNER JOIN Participants
            ON People.peopleID=Participants.participantID
            INNER JOIN OutOfHouse
            ON People.peopleID=OutOfHouse.outOfHouseID
            WHERE People.firstName=participantFirstName AND
                  People.middleInit=participantMiddleInit AND
                  People.lastName=participantLastName AND
                  date_part('year', Participants.dateOfBirth)=(date_part('year', CURRENT_DATE)-participantAge) AND
                  Participants.race=participantRace AND
                  Participants.sex=participantSex;
            IF FOUND THEN
                RAISE EXCEPTION 'Participant is already in the system. Cannot duplicate.';
            ELSE
                dateOfBirth := format('%s-%s-%s', (date_part('year', CURRENT_DATE)-participantAge)::TEXT, '01',
'01')::DATE;
                INSERT INTO People(firstName, middleInit, lastName) VALUES (participantFirstName,
participantMiddleInit, participantLastName);
```

```
                ptpID := LASTVAL();
                INSERT INTO Participants VALUES (ptpID, dateOfBirth, participantRace, participantSex);
                INSERT INTO OutOfHouse VALUES (ptpID, participantDescription);
                RETURN ptpID;
            END IF;
        END;
    $BODY$
        LANGUAGE plpgsql VOLATILE;
```

Create Class

Creates a class linking to a curriculum through the createCurriculum stored procedure

Author: Jesse Opitz

```
DROP FUNCTION IF EXISTS createClass();

CREATE OR REPLACE FUNCTION createClass(
    currName TEXT DEFAULT NULL::text,
    currType PROGRAMTYPE DEFAULT NULL::programtype,
    missNum INT DEFAULT NULL::int,
    topName TEXT DEFAULT NULL::text,
    classDesc TEXT DEFAULT NULL::text,
    dat TIMESTAMP DEFAULT NULL::timestamp,
    nameOfSite SITES DEFAULT NULL::sites,
    language TEXT DEFAULT NULL::text,
    currID INT DEFAULT NULL::int
)
RETURNS VOID AS
$BODY$
```

```
    BEGIN
      INSERT INTO Class(topicName, description) VALUES (topName, classDesc);
      RAISE NOTICE 'class %', topName;

      INSERT INTO ClassOffering(topicName, date, siteName, lang, curriculumID) VALUES (topName, dat, nameOfSite, 
language, currID);
    END;
$BODY$
    LANGUAGE plpgsql VOLATILE;
```

"VIEWS"

View statements in SQL are used to compose a virtual table based on the result of multiple SQL statements that call upon various rows and columns from the database. It is used to quickly display large amounts of general information in a more digestible format.

ClassAttendanceDetails

Returns all information related to a participant and their attendance for all classes offered

Author: John Randis and Marcos Barbieri

```
DROP VIEW IF EXISTS ClassAttendanceDetails;
CREATE VIEW ClassAttendanceDetails AS
    SELECT Participants.participantID,
           People.firstName,
           People.middleInit,
           People.lastName,
           Participants.dateOfBirth,
           Participants.race,
           Participants.sex,
           ParticipantClassAttendance.topicName,
           ParticipantClassAttendance.date,
           ParticipantClassAttendance.siteName,
           ParticipantClassAttendance.comments,
           ParticipantClassAttendance.numChildren,
           ParticipantClassAttendance.isNew,
           ParticipantClassAttendance.zipCode,
           Curricula.curriculumName,
           Curricula.curriculumType,
           FacilitatorClassAttendance.facilitatorID
       FROM Participants
       INNER JOIN People
       ON Participants.participantID=People.peopleID
       INNER JOIN ParticipantClassAttendance
       ON Participants.participantID=ParticipantClassAttendance.participantID
       INNER JOIN ClassOffering
       ON ClassOffering.topicName=ParticipantClassAttendance.topicName AND
      ClassOffering.date=ParticipantClassAttendance.date AND
      ClassOffering.siteName=ParticipantClassAttendance.siteName
       INNER JOIN Curricula
       ON Curricula.curriculumID=ClassOffering.curriculumID
       INNER JOIN FacilitatorClassAttendance
       ON FacilitatorClassAttendance.topicName=ClassOffering.topicName AND
      FacilitatorClassAttendance.date=ClassOffering.date AND
      FacilitatorClassAttendance.siteName=ClassOffering.siteName;
```

FacilitatorInfo

Returns all relevant information on facilitators. Since data for employees is scattered across three tables, this will make it easier for app developers to query for facilitator information.

Author: Carson Badame

```
CREATE VIEW FacilitatorInfo AS
 SELECT facilitators.facilitatorid,
    people.firstname,
    people.lastname,
    people.middleinit,
    employees.email,
    employees.primaryphone,
    employees.permissionlevel,
    facilitatorlanguage.lang,
    facilitatorlanguage.level AS langlevel
   FROM people,
    facilitators,
    employees,
    facilitatorlanguage
  WHERE people.peopleid = employees.employeeid AND employees.employeeid =
facilitators.facilitatorid AND facilitators.facilitatorid = facilitatorlanguage.facilitatorid
  ORDER BY facilitators.facilitatorid;
```

FamilyInfo

Returns data about the family related to the person ID

Author: Carson Badame

```
CREATE VIEW FamilyInfo AS
 SELECT family.formid AS familyid,
    people.peopleid,
    people.firstname,
    people.lastname,
    people.middleinit,
    familymembers.relationship,
    familymembers.dateofbirth,
    familymembers.race,
    familymembers.sex
   FROM people,
    familymembers,
    family
  WHERE people.peopleid = familymembers.familymemberid AND familymembers.familymemberid =
family.familymembersid
  ORDER BY family.formid;
```

ParticipantStatus

Returns basic information about a participant and the amount of classes they have attended, including the name of the most recent one

Author: Carson Badame

```
CREATE VIEW ParticipantStatus AS
 SELECT participants.participantid,
    people.firstname,
    people.lastname,
    people.middleinit,
    participants.dateofbirth,
    participants.race,
    participantclassattendance.topicname AS mostrecentclass,
    participantclassattendance.date,
    max(atttotal.totalclasses) AS totalclasses
   FROM people,
    participants,
    participantclassattendance,
    ( SELECT participantclassattendance_1.participantid,
            row_number() OVER (ORDER BY participantclassattendance_1.participantid) AS
totalclasses
           FROM participantclassattendance participantclassattendance_1) atttotal
  WHERE people.peopleid = participants.participantid AND participants.participantid =
participantclassattendance.participantid
  GROUP BY participants.participantid, people.firstname, people.lastname, people.middleinit,
participants.dateofbirth, participants.race, participantclassattendance.topicname,
participantclassattendance.date
  ORDER BY participants.participantid;
```

CurriculumInfo

Gathers curriculum information

Author: Jesse Opitz

```
CREATE VIEW CurriculumInfo AS
  SELECT curricula.curriculumID,
    curricula.curriculumName,
    curricula.curriculumType,
    curricula.missNumber,
    curriculumclasses.topicName,
    classes.description
  FROM curricula,
    curriculumclasses,
    classes
  WHERE curricula.curriculumID = curriculumclasses.curriculumID AND curriculumclasses.topicname =
classes.topicname
  GROUP BY curricula.curriculumID, curriculumclasses.curriculumID, curriculumclasses.topicName,
classes.topicName
  ORDER BY curricula.curriculumID;
```

GetCurricula

Retrieves curriculum ID and curriculum name

Author: John Randis

```
CREATE VIEW GetCurricula AS
    SELECT c.curriculumid, c.curriculumname
    FROM curricula c
    ORDER BY c.curriculumname ASC;
```

GetClasses

Retrieves curriculum ID and topic name to get a class

Author: John Randis

```
CREATE VIEW getClasses AS
    SELECT cc.curriculumid, cc.topicname
    FROM curriculumclasses cc
    ORDER BY cc.curriculumid;
```

ParticipantInfo

Joins together all information about a participant. This view is necessary because the information is spread out across two tables People and Participant

Author: Marcos Barbieri

```
CREATE VIEW ParticipantInfo AS
    SELECT Participants.participantID,
           People.firstName,
           People.middleInit,
           People.lastName,
           Participants.dateOfBirth,
           Participants.race,
           Participants.sex
    FROM Participants
    INNER JOIN People
    ON Participants.participantID=People.peopleID
    INNER JOIN Forms
    ON Participants.participantID=Forms.participantID;
```

“Useful Queries”

The following queries may be useful for common tasks.

Select All

```
SELECT * FROM People;
SELECT * FROM Employees;
SELECT * FROM Facilitators;
SELECT * FROM Participants;
SELECT * FROM OutOfHome;
Select * FROM FamilyMembers;
SELECT * FROM Children;
Select * FROM EmergencyContacts;
SELECT * FROM ContactAgencyMembers;
```

Return Tables with Number of Classes per Curriculum

```sql
SELECT curriculumname, COUNT(DISTINCT topicName)
FROM curriculuminfo
GROUP BY curriculumID, curriculumname;
```