MET CS 669 Database Design and Implementation for Business
Term Project Iteration 5

MyKids-CheckIn Project Direction Overview

I would like to develop an app that automates the Children's Ministry Check in process for each service at my church. When all the parties involved use the app, different parties will be able to have access to different information. Team leaders will be able to have the history of every person that has volunteered and a list of all the children that have attended a specific date and time and in what age groups. Teacher/Volunteers will be able to have access to important information regarding the children checked in to their classroom, like medical alerts, custodial, pickup authorized list information, notes, etc. Team leaders will be able to manage classrooms better by having current information on the total of children in each classroom and limiting the max # of children to stay in compliance with fire code regulations.

Here are some examples of how the application would be used. The team leader will set up the classrooms in the app and assign the available volunteers to have the classroom ready to check in. Once classrooms are setup (usually 15 minutes prior to the service start time), parents will come to a kiosk and type in their previously registered phone number which will display the list of children they have registered. The app will print out the badges for their children, one badge per child and one for the parent. The badge will have the child's name, age group, and QR code. Parents will go to a classroom assigned to their child's age group, and the Teacher/Volunteer will scan the child's badge to check them into their classroom. After service has ended, parents will check out their child by bringing the parent badge to be scanned by the teacher. The app will validate that the QR codes of the parent and child match.

Since all this information is being tracked in the app, Team leaders will be able to pull up the classroom's current list, they will also be able to know the exact location of a child, and the exact time a child was picked up. The database will also store information about the family and children, like home address, phone number, driver's license, children's age, any alert information (medical, custodial, special needs, etc.), and pick-up list. It will store information about the different, age groups, classrooms assigned to a specific age group and volunteers' information. It will save the Check-in and check-out date and time, classroom and any notes entered by the volunteers.

The reason I picked this project is that I have volunteered as a teacher and I am also a parent of 2 precious kids which are part of the children's ministries. The church currently has an app which has a mixture of an automated and manual process. I understand the need as a volunteer and as a parent to have an app that will completely automate the current process.

Project Iteration 2: Revision of the Project Direction Overview

Our church has different campuses in different locations, which creates the need to keep track of what campus a person is attending. Because of this, I added another example of how the application would be used.

The Admin Team will be able to enter information of all the different campuses and register all the service/events happening in a specific campus. They will also maintain information regarding the different buildings and the rooms that can be assigned to an event (e.g., Children's Ministry event) They will be able to schedule specific events allowing a person to register in these events, before a check-in process starts.

MyKids-CheckIn Use Cases and Fields

One important usage of the database is when a person registers their family members.

New Family Registration Use Case

1. The parent/guardian visits the kiosk located where the app is installed.
2. The parent/guardian will click on the "Create New Family" option.
3. The parent/guardian selects the campus they usually attend to, to then proceed to enter all the required information for them and the child/children and the new family is created in the database.
4. The app will ask the parent/guardian if they would like to print out badges for the current service/event. If they select yes, the app will printout the badges that will be used in the check-in and check-out process.

From the database perspective, this use case requires storing information about the new family (from steps #2 and #3) and information about the printed badges for the current service/event (step #4). Steps #1 apply to the user but not the database directly.

Parent/Guardian Fields

Fields	What it stores	Why it's Needed.
FirstName	This is the first name of the parent/guardian.	This is necessary for displaying the parent/guardian's name on the app or used for addressing any communication.
LastName	This is the last name of the parent/guardian.	This is necessary for displaying the parent/guardian's last name on the app or used for addressing any communication.

PhoneNumber	This is the phone number of the parent/guardian.	This is necessary to use it to contact the parent by sending them text or calling them if any issue with their child arises.
Address1	This is the address 1 of the parent/guardian.	This is used for sending out mail to the parent/guardian.
Address2	This is the address 2 of the parent/guardian.	This is used for sending out mail to the parent/guardian.

Fields	What it stores	Why it's Needed.
City	This is the city of the parent/guardian.	This is used for sending out mail to the parent/guardian.
State	This is the state of the parent/guardian.	This is used for sending out mail to the parent/guardian.
ZipCode	This is the zip code of the parent/guardian.	This is used for sending out mail to the parent/guardian.
Email	This is the email of the parent/guardian.	This is used for sending out email to the parent/guardian.
IdentificationNumber	This is the parent/guardian's identification number (Driver's license, state id, etc.)	This is useful when we need to verify the person is the parent/guardian, in the case where they misplaced the pick-up badge.
IdentificationType	This is the type of identification number the parent/guardian has provided. (Driver's license, state id, etc.)	This is useful (in conjunction with the identification number) when we need to verify the person is the parent/guardian, in the case where they misplaced the pick-up badge.
RelationshipToKid	This is the type of relationship they have with the child. Examples are: parent, grandparent, step-parent, foster-parent, aunt, etc.	This is useful when addressing communication.
DateOfBirth	This is the date of birth of the parent/guardian.	This is useful to send out Happy Birthday emails to the parent/guardian or any other communication that can be age specific.
MaritalStatus	This is the marital status of the parent/guardian.	This is very useful for discipleship.
Gender	This is the gender of the parent/guardian.	This is useful to personalize communications and send out emails depending on gender group events.

Fields	What it stores	Why it's Needed.
Campus	This is the campus the parent attends regularly.	This is used to know what campus a parent usually attends, this will help personalize their app view to only display information related to the campus or send mail or email regarding the campus.

Child Fields

Fields	What it stores	Why it's Needed.
FirstName	This field stores the first name of the child	This is necessary to print out their name on the badge or display it on the screen.
LastName	This field stores the last name of the child	This is necessary to print out their name on the badge or display it on the screen.
NickName	This field stores the nick name of the child.	This is useful to know the name a child is used to at home.
DateOfBirth	This field stores the date of birth of the child.	This is used to direct parents to the correct age group classroom.
Gender	This field stores the gender of the child.	This is useful for when the church has events specific to gender, we can send out communications to the correct gender group.

Alert/Warning List fields

Fields	What it stores	Why it's Needed.
TypeOfAlert	This field stores the type of alert a child has. Examples: allergies, custodial, etc.	This is useful for categorizing the alerts to display on the screen.
Description	This field stores the description of the alert.	This is used to display a detailed description of the type of alert.
ExpiratonDate	This field stores the expiration date of the alert.	This is useful for displaying certain alerts only for a period of time or if not stored, it will always display it.

Pick-up List fields

Fields	What it stores	Why it's Needed.
FirstName	This stores the first name of a person that is in the pick-up list of a child	This is useful to display on the screen to verify that the person can pick up the child
LastName	This stores the last name of a person that is in the pick-up list of a child	This is useful to display on the screen to verify that the person can pick up the child
PhoneNumber	This stores the phone number of a person that is in the pick-up list of a child	This is useful to display on the screen.

Attendee's Registration Service/Event fields

Fields	What it stores	Why it's Needed.
EventScheduleId	This stores the unique identifier of the event schedule of the service/Event the parent/guardian is registering their kids in.	This is used in the application to display information related to a specific scheduled event.
AttendeeId	This stores the unique identifier of the child or a person registering for the service/event.	This is used later to relate the information of the event a child/person is registered and their information

RegistrationDate	This stores the date and time the child was registered.	This is used to know the date and time a child was registered.
EventGroupId	This stores the Event Group unique identifier which is the possible age group the child may be checked in.	This is used to relate the information of the event group information
RegistrationCode	This stores the unique registration code for a child.	This is used when generating the QR code to be printed on the child's badge.

Another important usage of the database is when a Team Leader sets up the classes that will be available to check in children.

Setting up classrooms for check in Use Case

1. The team leader accesses the app from their tablets and signs in with their credentials.
2. The team leader will select the service/event and clicks on the "Open Classroom" option.
3. The team leader selects an age group with the classroom and assigns the teacher/volunteers.

From the database perspective, this use case requires storing information about the classroom and volunteer assignment (from step #3).

Classroom fields

Fields	What it stores	Why it's Needed.
AgeGroup	This stores the name of the age group. Examples: 1st grade, 2nd grade, etc.	This is useful to display on the screen.
RoomNumber	This stores the room number.	This is useful to display on screen to know where the room is located.
ClassName	This stores the room name. Example: 1st grade A, 1st grade B, 1st grade C.	This is useful because it is easier for parents to remember a letter than a 4 or more-digit number.
ScheduleDate	This stores the date the class was scheduled for check-in.	This is useful for reporting purposes and also to display only the current class schedule.

MaxNumber	This stores the max number of people allowed in the room.	This is used when applying restrictions in check in.
EventScheduleId	This stores the unique identifier of the event schedule of the service/Event the parent/guardian is registering their kids in.	This is used in the application to display information related to a specific scheduled event.

Volunteer fields

These fields are not stored by this use case but are used to assign the volunteers to a classroom.

Fields	What it stores	Why it's Needed.
FirstName	This store the volunteers first name.	This is useful to display on the screen.
LastName	This stores the volunteers last name.	This is useful to display on the screen.
StartDate	This stores the date they became volunteers.	This
LastBackgroundCheck	This stores the date of the last background checked.	This helps team leaders to make sure they only assign volunteers that have passed a background check.

Another important usage of the database is where the teacher/volunteer starts the check in process.

Classroom check-in/check-out

1. The teacher/volunteer accesses the app from their tablets and signs in with their credentials.
2. They will see the classroom they are assigned to.
3. When a parent and child print out the badges, they will present the badge with the QR code to the teacher/volunteer, who will scan the QR code.
4. The teacher/volunteer will add any comments to the child's record that parents might share (diaper bag, bottle to be given at certain time, etc.)
5. Once service ends, parent brings the badge to be scanned by the teacher/volunteer,
6. The app will verify that the parent badge matches with the child QR code; if it does it will check-out the child from the classroom.

From the database perspective, this use case requires storing information about volunteers sign in (from step#1), the check-in and check-out process (from steps #3, #4, #5 and #6). Steps #2 displays the classroom information but does not store any information.

Check-in fields

Fields	What it stores	Why it's Needed.
CheckIn	This field stores the date and time that a child was checked in.	This is information that can be used for reporting.
CheckOut	This field stores the date and time that a child was checked out.	This is information that can be used for reporting.
TypeVerificationCheckIn	This field stores the type of verification in the check-in process. Example: If it was scanned or manual.	This information is used to know if the child was checked in scanning the QR code from the badge or the parent did not have the QR code and it was done manually.
TypeVerificationCheckOut	This field stores the type of verification in the check-out process. Example: If it was scanned or manual.	This information is used to know if the child was checked out scanning the QR code from the badge or the parent did not have the QR code and it was done manually.
Notes	This field is used to store notes regarding the child.	This is useful to store any instructions given by the parent at check-in time or to store any information the teacher may like to note.

MyKids-CheckIn Structural Rules

New Family Registration Use Case

1. The parent/guardian visits the kiosk located where the app is installed.
2. The parent/guardian will click on the “Create New Family” option.
3. The parent/guardian selects the campus they usually attend to, to then proceed to enter all the required information for them and the child/children and the new family is created in the database.
4. The app will ask the parent/guardian if they would like to print out badges for the current service/event. If they select yes, the app will printout the badges that will be used in the check-in and check-out process.

For this use case I can see several components involved: the application, the parent using the application and the database. I will focus on what will be stored in the database. From step # 3, I see a Campus entity, Parent entity, and Child entity. From step #4, I see an Event entity. Steps #1 and #2 do not reflect any other entity or relationship.

The application will track what campus a parent and child usually attend. For this use case a Parent or child is associated with a Campus. The application will also track the Event (e.g., “Children’s Ministry Sunday Service 9:00 am”), hosted at a specific Campus, a child is registered. The application also tracks the children that are associated with a parent.

I now have enough information to create some structural database rules. I’ll number them so that they can later be referred to by number.

First rule,

1. ***Each Parent is associated with one Campus; Each Campus may be associated with many Parents.***

   I created this structural rule because I infer from the use cases that the application cannot add a Parent to the database without selecting what campus they attend, which is why I made this association mandatory (“each Parent is…”). I indicated that it is optional for a Campus to associate many Parents, to leave room for the fact that a Campus is created before any Parent is created.

Second rule,

2. ***Each Event may be associated with many Campuses; Each Campus may be associated with many Events.***

   I create this structural rule because I infer from the use cases that each Campus may be associated with many Events. I indicated that it is optional for a Campus to associate many Events, to leave room for the fact that a Campus is created before any Event is created. I made the Event association with the campus optional as well, because the app may let me create an event that is still not associated with a campus.

Third rule,

3. ***Each Parent is associated to one or many children. A Child is associated with one or many Parents*.**
   I created this structural rule because I infer from the use cases that each Parent is entered in the system as a Parent if they have at least one child. The same way a child may not be entered in the system without a parent/guardian assign to.

Forth rule,

4. ***A child may be associated to many events; An event may be associated to many children.***

   I created this structural rule because I infer from the use case that a child does not have to be registered in an event when we add a child in the application for the first time, making associating optional. The same way, we can create an event and not have any child registered. This association is optional.

Setting up classrooms for check in Use Case

1. The team leader accesses the app from their tablets and signs in with their credentials.
2. The team leader will select the service/event and clicks on the "Open Classroom" option.
3. The team leader selects the Event and the event group (e.g. age group) with the classroom and assigns the teacher/volunteers.

For this use case I can see 3 different entities that are involved in storing data. Classroom entity, Event entity (e.g., "Children's Ministry Sunday Service 9:00 am"), Event Group entity (e.g. "2-year old") and the Volunteer entity. According to the use case a classroom cannot be setup without an assigned volunteer and without assigning a specific event and event.

Fifth rule,

5. ***Each Event may be associated with many classrooms; Each Classroom may be associated with many Events.***
   I created this structural rule because I infer from the use cases that each Classroom may be setup in the application without indicating what event this classroom is assigned to. Therefore, I make it optional. On the other end, an Event is optional to have an association with many classrooms, since an Event is created before a classroom is created.

Sixth rule,

6. ***Each Event may be associated to many Event Groups; An Event Group is associated to only one Event.***
   Although it is not explicitly mentioned, it stands to reason that each event group is associated to one Event since we need to store what event and event group is being created for, making the participation mandatory between the event group and the event. However, the association between event and event group is optional since an event can be added to the system without it being associated to an Event group.

Seventh rule,

7. ***A volunteer may be associated with one classroom; A classroom is associated to one or many volunteers.***

   I created this structural rule based on the use case explanation that when the team leader sets up a classroom, the application requires you to select at least one volunteer for a classroom open for the check-in process. Because of this rule, the relation between the classroom and volunteer is mandatory. On the other end, the relationship between the volunteer and the classroom is optional, since a volunteer can be added to the application without being assign to any classroom.

Classroom check-in/check-out

1. The teacher/volunteer accesses the app from their tablets and signs in with their credentials.
2. They will see the classroom they are assigned to.
3. When a parent and child print out the badges, they will present the badge with the QR code to the teacher/volunteer, who will scan the QR code.
4. The teacher/volunteer will add any comments to the child's record that parents might share (diaper bag, bottle to be given at certain time, etc.)
5. Once service ends, parent brings the badge to be scanned by the teacher/volunteer,
6. The app will verify that the parent badge matches with the child QR code; if it does it will check-out the child from the classroom.

From this use case, I see three significant data points: Child entity, Volunteer Entity, Classroom Entity and Check-In Entity. In the previous 2 use cases I described the rules to some of these entities.

Eighth rule,

8. ***A Classroom may be associated to many Check-ins; A Check-in is associated to one classroom.***
   I created this structural rule based on the use case explanation that when a classroom is setup the Volunteer may start the check-in process. The association between the

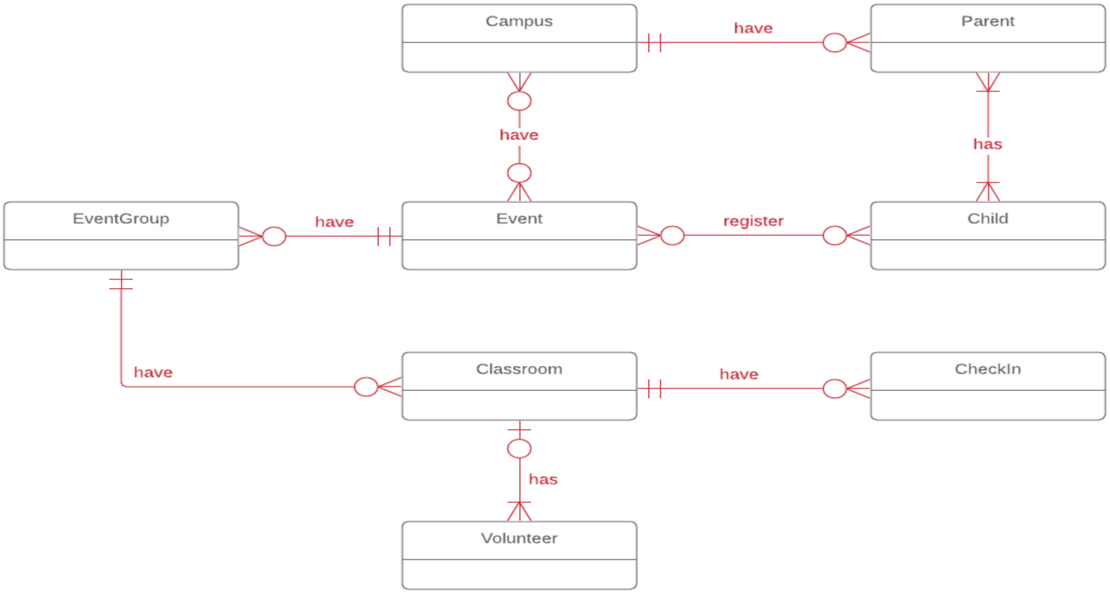
Classroom and the check-in is optional, to leave room for the fact that a classroom is setup before any child checks in. Once a child is checked in, they have an association with a classroom.

Initial MyKids-CheckIn ERD

Here are the structural database rules I came up with, base on my 3 use cases, in Iteration 2.

1. Each Parent is associated with one Campus; Each Campus may be associated with many Parents.
2. Each Event may be associated with many Campuses; Each Campus may be associated with many Events.
3. Each Parent is associated to one or many children. A Child is associated with one or many Parents.
4. A child may be associated to many events; An event may be associated to many children.
5. Each Event may be associated with many classrooms; Each Classroom may be associated with many Events.
6. Each Event may be associated to many Event Groups; An Event Group is associated to only one Event.
7. A volunteer may be associated with one classroom; A classroom is associated to one or many volunteers.
8. A Classroom may be associated to many Attendances; An Attendance is associated to one classroom.

Here is the ERD I came up with for these rules, using Crow's notation.



Adding Specialization-Generalization to MyKids-CheckIn

I reviewed my existing use cases and noticed that the first use case I can depict a specialization hierarchy.

New Family Registration Use Case

1. The parent/guardian visits the kiosk located where the app is installed.
2. The parent/guardian will click on the “Create New Family” option.
3. The parent/guardian selects the campus they usually attend to, to then proceed to enter all the required information for them and the child/children and the new family is created in the database.
4. The app will ask the parent/guardian if they would like to print out badges for the current service/event. If they select yes, the app will printout the badges that will be used in the check-in and check-out process.

I realized that parent/guardian and child share many characteristics and have other characteristics that are specific to a parent or child, which means that they should really be treated as different type or person. I modified the use case, so it reflects it clearly.

New Family Registration Use Case (Updated)

1. The person visits the kiosk located where the app is installed.
2. The person will click on the “Create New Family” option.
3. The person selects the campus they usually attend to; the app will then ask them if the person they are entering information for is a parent or a child. If it is a child they will have to extra information, like medical alerts (e.g., allergies, special needs, medication, etc.) or custodial information. Once all information is entered the new family is created in the database.
4. The app will ask the parent/guardian if they would like to print out badges for the current service/event. If they select yes, the app will printout the badges that will be used in the check-in and check-out process.

Now that #3 mentions that a parent/guardian and child are a person, I came up with a new structural database rule to support the change to the use case as follows. “***A person is a child or a parent.”*** However, after reading through the other use cases, another type of person comes in place and these are the volunteers. I added a new use case, so it would reflect the volunteer type of person clearly.

New Volunteer Registration Use Case (New)

1. The person visits the kiosk where the app is installed.
2. The person will click on the “Registration to Volunteer”.
3. The person selects the campus they usually attend to and enters all the required information. Once all the information is entered, the Volunteer is created in the database as a Volunteer with a pending status.
4. The volunteer will be contacted in a few days for finger printing for background check. If the person passes the background check, he will be approved to volunteer, and his status will be changed to approved.

Now that I have added this new use case “***New Volunteer Registration Use Case”*** and with the changes I made to “***New Family Registration Use Case”,*** *step* #3, I created this new structural database rule to support the change:

“Each person is a parent, child or volunteer, or several of these”

My database for now has only 3 types of person: parent, child and volunteer. For this specialization-generalization rule, the relationship is totally complete. I have decided to have it as a **totally complete**, because of the current purpose of the app: keep track of the child check-in process for the different services. In the future, if this app is used for other events, then I will change it to partially complete, since we can have a person registered that does not fall in to any of the current types of person. A person can be a parent and a volunteer, so the relationship is **overlapping**.

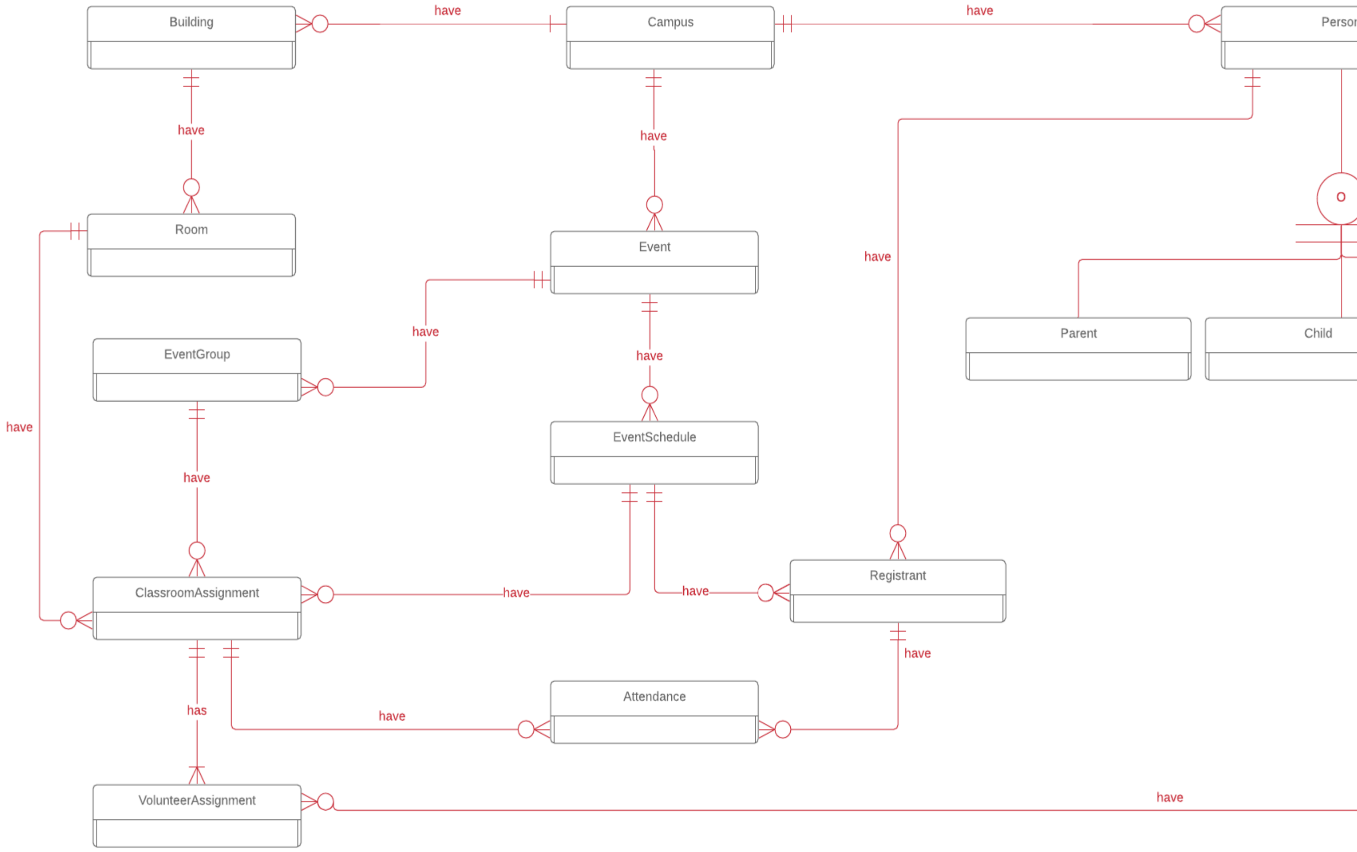
I have 9 structural database rules, including my original 8 plus the one I just created.

1. Each Parent is associated with one Campus; Each Campus may be associated with many Parents.
2. Each Event may be associated with many Campuses; Each Campus may be associated with many Events.
3. Each Parent is associated to one or many children. A Child is associated with one or many Parents.
4. A child may be associated to many events; An event may be associated to many children.
5. Each Event may be associated with many classrooms; Each Classroom may be associated with many Events.
6. Each Event may be associated to many Event Groups; An Event Group is associated to only one Event.
7. A volunteer may be associated with one classroom; A classroom is associated to one or many volunteers.
8. A Classroom may be associated to many Attendances; An Attendance is associated to one classroom.
9. Each person is a parent, child or volunteer, or several of these.

I also added

I then added changes to my Initial ERD to support the following changes:

- Support the one additional structural database rule.
- The change I made to the structural database rule #5
- Change of the entity name “CheckIn” to “Attendance”


Building
have
Campus
have
Perso
have
have
Room
Event
have
o
have
EventGroup
have
Parent
Child
have
have
EventSchedule
have
ClassroomAssignment
have
have
Registrant
have
has
have
Attendance
VolunteerAssignment
have

MyKids-CheckIn Relationship Classification and Associative Mapping

The associative relationships in my conceptual ERD are:

1. **Campus/Person:** A Campus may be associated with many Persons; Each Person is associated to a Campus.
2. **Campus/Event:** A Campus may be associated with many Events; Each Event is associated to a Campus.
3. **Event/Event Group:** An Event may be associated with many Event Groups; Each Event Group is associated to an Event.
4. **Event/Classroom:** An Event may be associated with many Classrooms; A Classroom may be associated to many Events.
5. **Classroom/Person:** A Classroom may be associated with many persons; A Person is associated to a classroom.
6. **Classroom/Attendance:** A classroom may be associated to many Attendances; An Attendance may be associated to a Classroom.

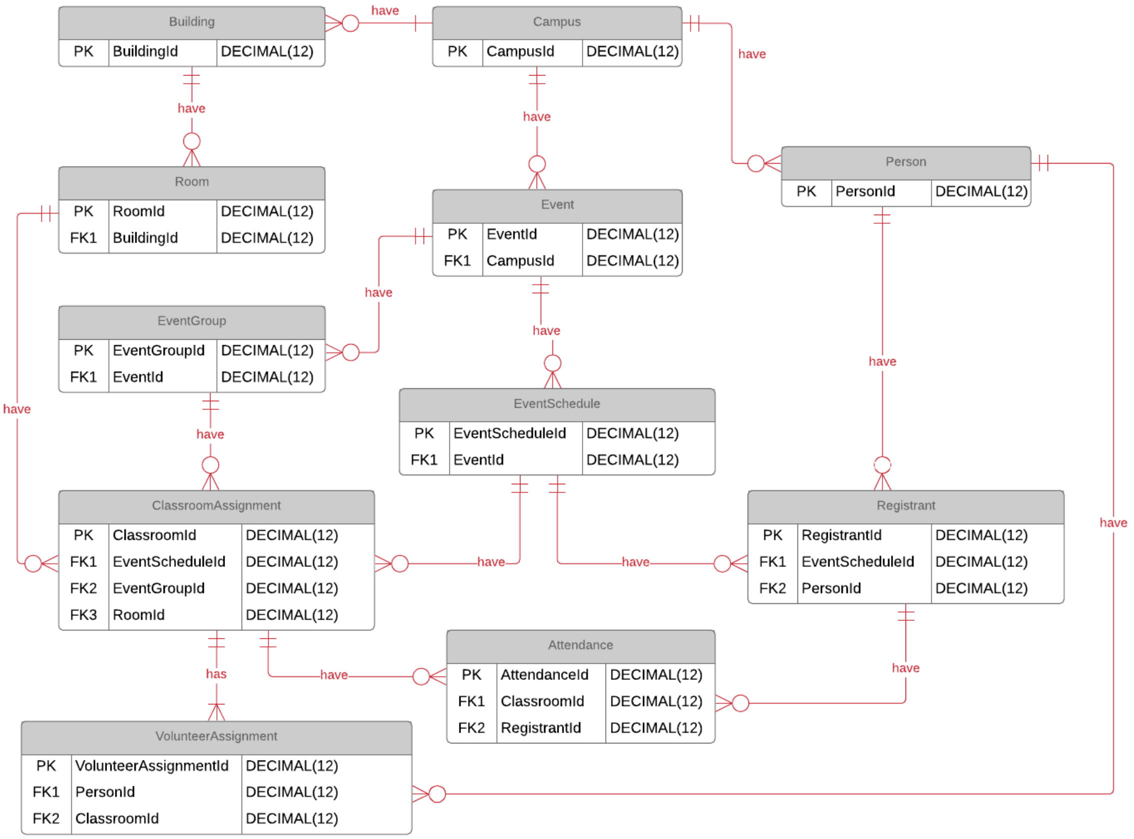
I changed the name of the entity **Classroom** to **ClassroomAssignment** to better describe its purpose. I added 2 new entities, **Building** and **Room**, to be able to assign a specific room when creating a new classroom assignment. I added a third entity, **Registrant**, that will have a relationship with the Person who will register for an event. The reason I added this entity is because a Person can register and not necessarily check in to a classroom, since they can change their mind and leave. With this I can keep track of the persons who registered versus the ones that attend.

Since Event/Classroom is a M:N relationship, it was necessary to create a bridge entity to support the relationship. I named the entity **EventSchedule**.

Because of all these changes, new associative relationships appear which I describe below:

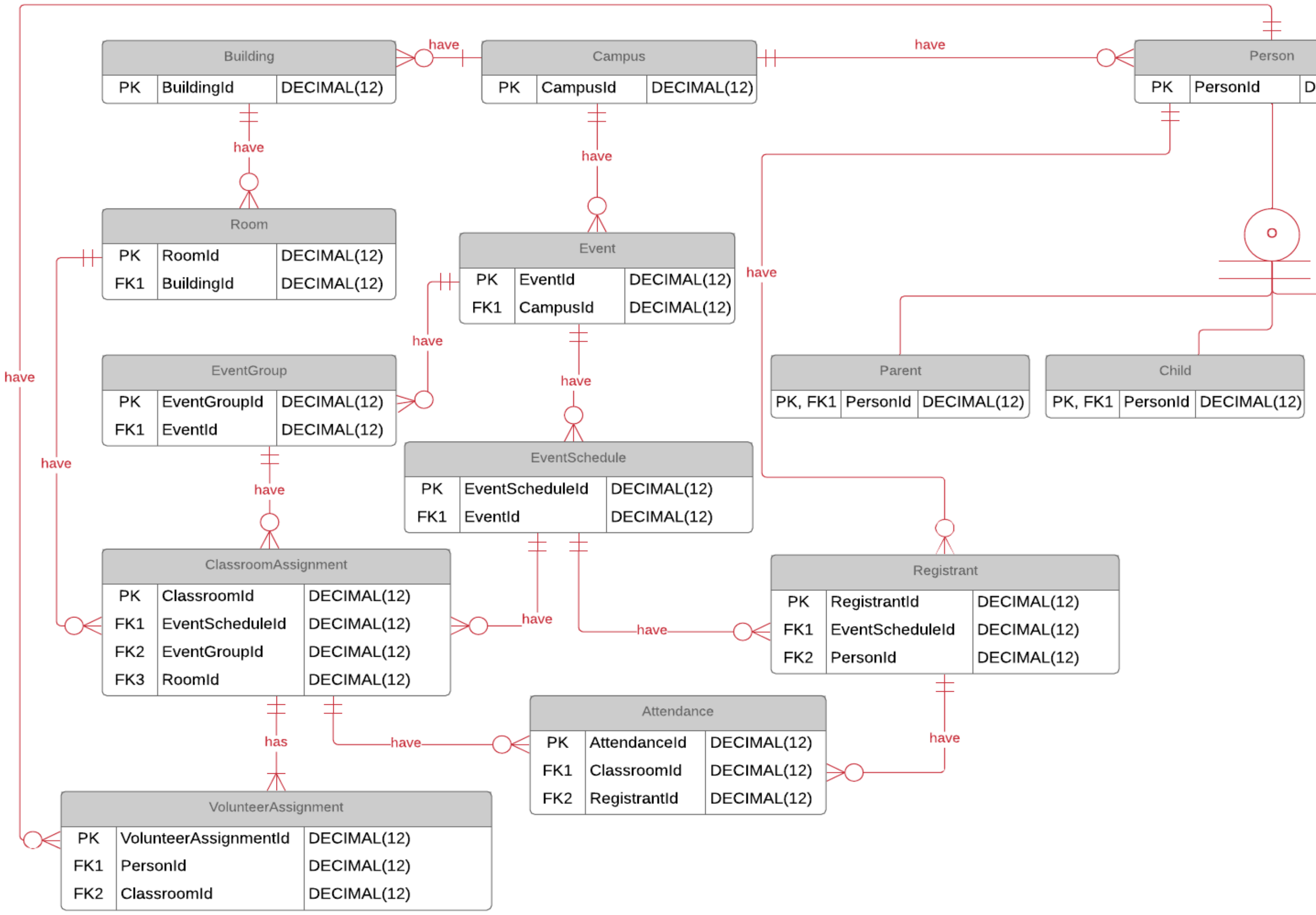
7. **Building/Room:** A building may be associated with many rooms; A room is associated with one Building
8. **Campus/Building:** A Campus may be associated with many Building; A Building is associated to one Campus.
9. **Event/EventSchedule:** An Event may be associated with many Event Schedules; An Event Schedule is associated to one Event.
10. **EventSchedule/ClassroomAssignment:** An Event Schedule may be associated to many Classroom Assignments; A Classroom Assignment is associated to one Event Schedule.
11. **EventSchedule/RegistrantAttendance;** An Event Schedule may be associated to many RegistrantAttendance; A RegistrantAttendance is associated to one Event Schedule.

I created surrogate keys for all my tables and used the datatype DECIMAL when creating my primary keys.



MyKids-CheckIn Specialization-Generalization Mapping

I have one specialization-generalization relationships in my conceptual ERD, for the Person entity. Here is my DBMS physical EERD with these relationships mapped into them.



Building
PK BuildingId DECIMAL(12)
have
Campus
PK CampusId DECIMAL(12)
have
Person
PK PersonId D
have
have
Room
PK RoomId DECIMAL(12)
FK1 BuildingId DECIMAL(12)
Event
PK EventId DECIMAL(12)
FK1 CampusId DECIMAL(12)
have
o
have
Parent
PK, FK1 PersonId DECIMAL(12)
Child
PK, FK1 PersonId DECIMAL(12)
have
EventGroup
PK EventGroupId DECIMAL(12)
FK1 EventId DECIMAL(12)
have
have
have
EventSchedule
PK EventScheduleId DECIMAL(12)
FK1 EventId DECIMAL(12)
have
ClassroomAssignment
PK ClassroomId DECIMAL(12)
FK1 EventScheduleId DECIMAL(12)
FK2 EventGroupId DECIMAL(12)
FK3 RoomId DECIMAL(12)
have
have
Registrant
PK RegistrantId DECIMAL(12)
FK1 EventScheduleId DECIMAL(12)
FK2 PersonId DECIMAL(12)
Attendance
PK AttendanceId DECIMAL(12)
FK1 ClassroomId DECIMAL(12)
FK2 RegistrantId DECIMAL(12)
has
have
have
VolunteerAssignment
PK VolunteerAssignmentId DECIMAL(12)
FK1 PersonId DECIMAL(12)
FK2 ClassroomId DECIMAL(12)

The additional entities under Person are Parent, Child and Volunteer, each of which have a primary and foreign key of PersonId which reference the primary key of Person. With these additional mappings, this DBMS physical now has all the relationships in the conceptual ERD.

MyKids-CheckIn Attributes

When I started adding the attributes for each one of my tables, I realized that the ***Registrant*** table could be combined with the ***Attendance*** table. I removed these 2 tables and combined them in the ***RegistrantAttendance*** table.

I also noticed, that my ***Parent*** and ***Volunteer*** subtype entities had common attributes, which then I decided to create a subtype ***Adult*** as the supertype entity of the ***Volunteer*** subtype entity.

Because of these changes my structural database rules changed as follows:

1. Each *Person* is associated with one *Campus*; Each *Campus* may be associated with many *Persons*.
2. Each *Event* is associated with one *Campus*; Each *Campus* may be associated with many *Events*.
3. Each *Adult* may be associated to one or many *children*. A *Child* is associated with one or many *Adults*.
4. Each *Event* may be associated to many *Event Schedule*; An *Event Schedule* is associated to one *Event*.
5. An *Event Schedule* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *Event Schedule*.
6. A *child* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *child*.
7. Each *Event Schedule* may be associated with many *Classroom Assignments*; Each *Classroom Assignment* is associated with one *Event Schedule*.
8. Each *Event* may be associated to many *Event Groups*; An *Event Group* is associated to only one *Event*.
9. A *Volunteer* may be associated with many *Volunteer Assignments*; A *Volunteer Assignment* is associated to one *Volunteer*.
10. A *Classroom Assignment* may be associated to many *Registrant Attendances*; A *Registrant Attendance* is associated to one *Classroom Assignment*.
11. An *Event Group* may be associated to many *Classroom Assignment*; A *Classroom Assignment* is associated to one *Event Group*.
12. Each *Campus* may be associated to many *Buildings*; A Building is associated to one *Campus*.
13. Each *Building* may be associated to many *Rooms*; A *Room* is associated to one *Building*.

14. A *Room* may be associated to many *Classroom Assignments*; A *Classroom Assignment* is associated to one *Room*.
15. Each *person* is an *adult* or *child*.
16. Each *adult* is a *volunteer*, or none.

Below I will describe all my attributes with their datatypes for each table in my database and I will explain the reason of my choices.

Table	Attribute	Datatype	Reasoning
Building	Name	VARCHAR(100)	Every building has a name which acts like the identifier for the building when the user is looking it up in the app. I allow for up to 100 characters.
Building	Description	VARCHAR(1000)	Every building may have a description. People may want to describe the building more than just with the name. I allow for 1,000 characters so that people can type in something long if they need to.
Building	NumberFloor	DECIMAL(2)	Every building has a number of floors. I allow for up to 2 digit number.
Building	Notes	VARCHAR(1000)	Every building may have notes. People may want to add extra notes of the building.
Building	Active	DECIMAL(1)	This flag will let the user Inactivate a building that we no longer have. The values it will have will be 1 for Active and 0 for inactive.
Room	Name	VARCHAR(100)	Every room has a name which acts like the identifier of the room and how it will be searched in the app. I allow for up to 100 characters. Example: 2-year-old room "A" 2-year-old room "B"
Room	RoomNumber	VARCHAR(10)	Every room has a room number. I created it as a VARCHAR(10) since some room numbers may be alphanumeric and I allow up to 10 characters.
Room	Capacity	DECIMAL(5)	Each room has a number of people capacity. I allow up to 5-digit number which is 99999 which gives enough space in case something extraordinary happens.
Room	Floor	DECIMAL(2)	Every room has a floor number where it is located. I allow up to 2-digit number.
Room	Note	VARCHAR(1000)	Every room may have notes. In some cases, people would like to add notes like, this room has a projector or bathroom, or if

			volunteers should be very careful with specific items in the room, etc.
Room	Active	DECIMAL(1)	This flag will let the user Inactivate a room that we no longer have. The values it will have will be 1 for Active and 0 for inactive.
Campus	Name	VARCHAR(255)	Every campus has a name that serves as the identifier of the campus. I allow up to 255 characters.
Campus	Addess1	VARCHAR(255)	Every campus has an address which is the street. I allow up to 255 characters.
Campus	Address2	VARCHAR(255)	Every campus may have extra information on the address. I allow up to 255 characters.
Campus	City	VARCHAR(255)	Every campus is in a city. I allow up to 255 characters.
Campus	State	VARCHAR(2)	Every campus is in a state. I allow 2 characters.
Campus	PostalCode	VARCHAR(10)	Every campus has a postal code. I allow up to 10 characters.
Campus	Active	DECIMAL(1)	This flag will let the user Inactivate a campus that we no longer have. The values it will have will be 1 for Active and 0 for inactive.
Event	Name	VARCHAR(255)	Every event has a name that acts like an identifier and how people will search it in the app. I allow up to 255 characters.
Event	Description	VARCHAR(1000)	Every event may have a detailed description of the event. I allow up to 1000 characters.
Event	RegistrationCheckInType	CHAR(1)	Every event has a type of check in, whether a registrant checks in the kiosk with their phone number (P) or with their ID (I). The 2 distinct values this attribute will have are 'P' and 'I'. This helps to setup the kiosk with the correct screen.
Event	Active	DECIMAL(1)	This flag will let the user Inactivate an event that we no longer have. The values it will have will be 1 for Active and 0 for inactive.
EventSchedule	EventStartDate	DATETIME	Each event scheduled has a start date and time.
EventSchedule	EventEndDate	DATETIME	Each event scheduled has an end date and time.
EventSchedule	RegistrationStartDate	DATETIME	Each event schedule has a registration start date and time. For example, and Event is scheduled for the Sunday 9:00 am service, but we can start registering kids starting at 8:30 am so the app should be available to start generating the badges at this time.
EventSchedule	RegistrationEndDate	DATETIME	Each event schedule has a registration end date and time. For

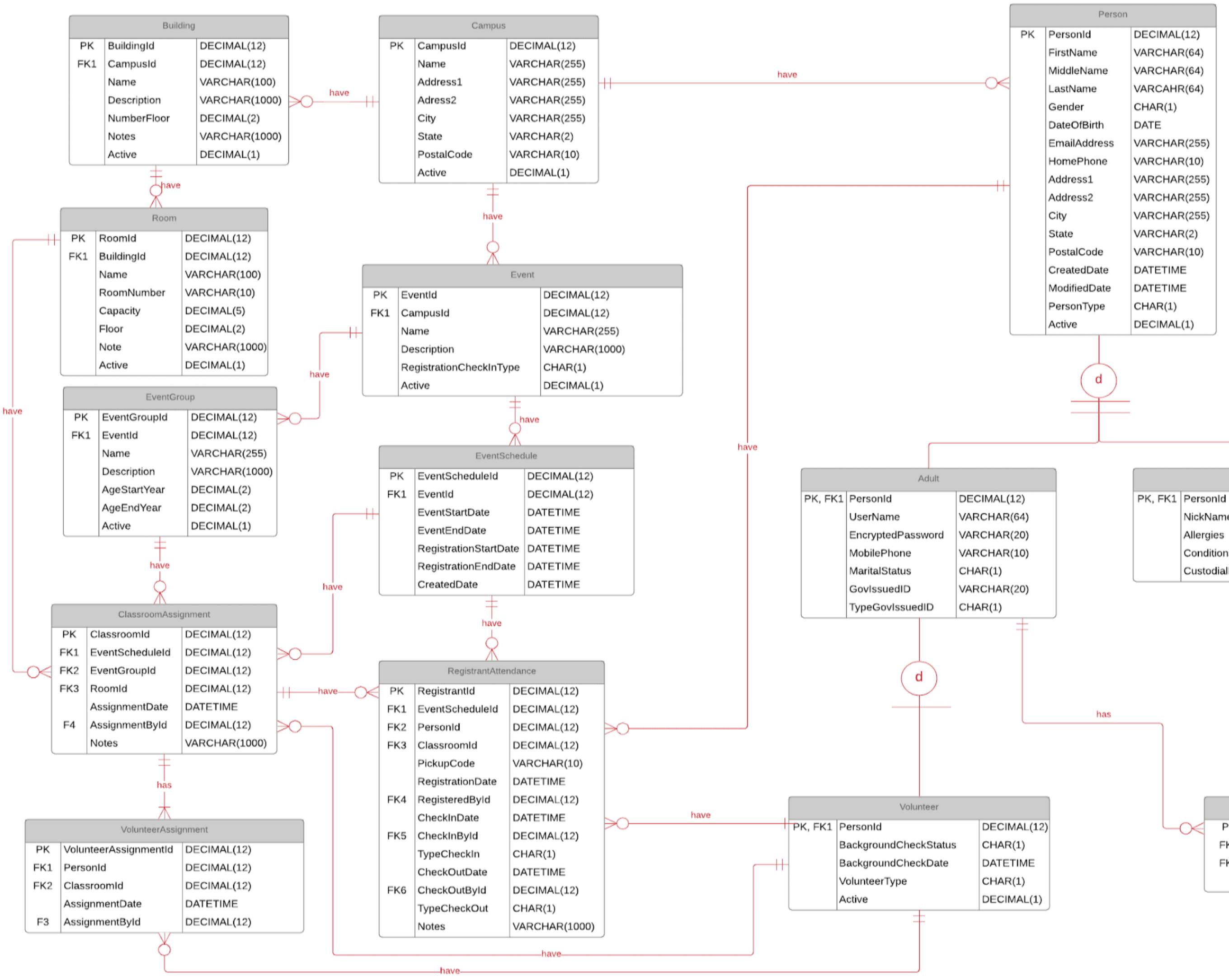
			example, for the 9:00 am service the parents can start printing the badges (registering) at 8:30 am and a parent that comes in late can print out badges up until 10: 00 am.
EventSchedule	CreatedDate	DATETIME	Each event schedule has a created date.
EventGroup	Name	VARCHAR(255)	Each event group has a name the will be like the identifier. I allow up to 255 characters. Example: 0 to 3-month-old 4 to 6-month-old 7 to 12-month-old 13 to 24-month-old 2-year-old 3-year-old PreK 4 Kindergarten, etc.
EventGroup	Description	VARCHAR(1000)	Each event group may have a description. I allow up to 1000 characters
EventGroup	AgeStartYear	DECIMAL(4,2)	Each event group has an age range. The app will convert it into year's even if it is months. This will be used by the app to suggest what age group a child should check in to. I allow up 2 digit and 2 decimal points.
EventGroup	AgeEndYear	DECIMAL(2)	Each event group has an age range. The app will convert it into year's even if it is months. This will be used by the app to suggest what age group a child should check in to. I allow up 2 digit and 2 decimal points.
EventGroup	Active	DECIMAL(1)	This flag will let the user Inactivate an event group that we no longer have. The values it will have will be 1 for Active and 0 for inactive.
ClassroomAssignment	AssignmentDate	DATETIME	Every classroom assignment will have an assignment date and time which is the date the class was opened to start checking in.
ClassroomAssignment	AssignmentById	DECIMAL(12)	Every classroom assignment will have an assignmentById which is the volunteer that setup the classroom in the app.
ClassroomAssignment	Notes	VARCHAR(1000)	Every classroom assignment may have notes, that could be used by the team leader volunteer or by the teacher volunteers.
RegistrantAttendance	PickupCode	VARCHAR(10)	Every registrant has a pickup code that the app will generate. I allow up to 10 characters.
RegistrantAttendance	RegistrationDate	DATETIME	Every registrant (child) has a registration date when parents sign them in in the kiosk to printout badges.

RegistrantAttendance	RegisteredById	DECIMAL(12)	Every registrant is registered by the parents, this attribute will save the parent id.
RegistrantAttendance	CheckInDate	DATETIME	Every registrant may have a check-in date and time when they are checked in to their classrooms.
RegistrantAttendance	CheckInById	DECIMAL(12)	Every registrant may have a volunteer that checks them in to the classroom.
RegistrantAttendance	TypeCheckIn	CHAR(1)	Every registrant may have been checked-in with scanning QR code or manually. The attribute values would be ‘C’ for QR code or ‘M’ for manual check in.
RegistrantAttendance	CheckOutDate	DATETIME	Every registrant may have a check out date and time when they are checked out.
RegistrantAttendance	CheckOutById	DECIMAL(12)	Every registrant may have a volunteer that checked them out.
RegistrantAttendance	TypeCheckOut	CHAR(1)	Every registrant may have been checked-out with scanning QR code or manually. The attribute values would be ‘C’ for QR code or ‘M’ for manual check out.
VolunteerAssignment	AssignmentDate	DATETIME	Every volunteer assignment has an assignment date and time to their classroom.
VolunteerAssignment	AssigmentById	DECIMAL(12)	Every volunteer assignment is done by a team leader volunteer.
Person	FirstName	VARCHAR(64)	Every person has a first name. I allow up to 64 characters.
Person	MiddleName	VARCHAR(64)	Every person may have a middle name. I allow up to 64 characters.
Person	LastName	VARCHAR(64)	Every person has a last name. I allow up to 64 characters.
Person	Gender	CHAR(1)	Every person has a gender. ‘F’ for female and ‘M’ for male. I allow one character.
Person	DateOfBirth	DATE	Each person has a date of birth.
Person	EmailAddress	VARCHAR(255)	Each person may have an email address.
Person	HomePhone	VARCHAR(10)	Each person may have a home phone.
Person	Address1	VARCHAR(255)	Each person has an address 1.
Person	Address2	VARCHAR(255)	Each person may have an address 2.
Person	City	VARCHAR(255)	Each person lives in a city.
Person	State	VARCHAR(2)	Each person lives in a state.
Person	PostalCode	VARCHAR(10)	Each person has a postal code.
Person	CreatedDate	DATETIME	This is the date and time when the record was created in the table.
Person	ModifiedDate	DATETIME	This is the date and time the record was last modified.
Person	PersonType	CHAR(1)	This is the subtype discriminator: It will have ‘A’ for adult and ‘C’ for child.
Person	Active	DECIMAL(1)	This is a flag that will be used to inactivate a person.

Adult	UserName	VARCHAR(64)	Every adult has a username. I allow up to 64 characters.
Adult	EncryptedPassword	VARCHAR(20)	Every adult has an encrypted password that can be used to log in to the app.
Adult	MobilePhone	VARCHAR(10)	Every adult may have a mobile phone.
Adult	MaritalStatus	CHAR(1)	Every adult has a marital status. ● Divorce ● Married ● Partnered ● Separated ● Single ● Widowed
Adult	GovIssuedID	VARCHAR(20)	Every adult has a government issued Id. I allow up to 20 characters.
Adult	TypeGovIssuedID	CHAR(1)	Every adult's government issued id can be a state id or a driver's license. Attribute values: 'S' for state id or 'D' for driver's license.
Child	NickName	VARCHAR(64)	Every child may have a specific name they are called at home. I allow up to 64 characters.
Child	Allegries	VARCHAR(1000)	Every child may have allergies. I allow up to 1000 characters.
Child	Conditions	VARCHAR(1000)	Every child may have some medical conditions. I allow up to 1000 characters.
Child	CustodialInfo	VARCHAR(1000)	Every child may have specific custodial information. I allow up to 1000 characters.
Volunteer	BackgroundCheckStatus	CHAR(1)	Every volunteer must pass a background check. There are 3 different status for the results: ● Clear ● Consider ● Pending
Volunteer	BackgroundCheckDate	DATETIME	Every volunteer must have the date of his last background check since they have to repeat it every 5 or 7 years.
Volunteer	VolunteerType	CHAR(1)	Volunteers can be categorized in: ● Teachers ● Team Leaders ● Admin
Volunteer	Active	DECIMAL(1)	A volunteer has a status of 'Active' to volunteer. This flag will be used to Inactivate a volunteer.
ParentChild	ChildId	DECIMAL(12)	Every child has a parent. And every Parent has a child.
ParentChild	ChildRelationship	VARCHAR(20)	Every child has a specific relationship with the parent. They can be

			● Foster Child ● Guest Child ● Minor Child

Here is my ERD with the attributes included.



Building

PK	BuildingId	DECIMAL(12)
FK1	CampusId	DECIMAL(12)
	Name	VARCHAR(100)
	Description	VARCHAR(1000)
	NumberFloor	DECIMAL(2)
	Notes	VARCHAR(1000)
	Active	DECIMAL(1)

Campus

PK	CampusId	DECIMAL(12)
	Name	VARCHAR(255)
	Address1	VARCHAR(255)
	Adress2	VARCHAR(255)
	City	VARCHAR(255)
	State	VARCHAR(2)
	PostalCode	VARCHAR(10)
	Active	DECIMAL(1)

Person

PK	PersonId	DECIMAL(12)
	FirstName	VARCHAR(64)
	MiddleName	VARCHAR(64)
	LastName	VARCAHR(64)
	Gender	CHAR(1)
	DateOfBirth	DATE
	EmailAddress	VARCHAR(255)
	HomePhone	VARCHAR(10)
	Address1	VARCHAR(255)
	Address2	VARCHAR(255)
	City	VARCHAR(255)
	State	VARCHAR(2)
	PostalCode	VARCHAR(10)
	CreatedDate	DATETIME
	ModifiedDate	DATETIME
	PersonType	CHAR(1)
	Active	DECIMAL(1)

Room

PK	RoomId	DECIMAL(12)
FK1	BuildingId	DECIMAL(12)
	Name	VARCHAR(100)
	RoomNumber	VARCHAR(10)
	Capacity	DECIMAL(5)
	Floor	DECIMAL(2)
	Note	VARCHAR(1000)
	Active	DECIMAL(1)

Event

PK	EventId	DECIMAL(12)
FK1	CampusId	DECIMAL(12)
	Name	VARCHAR(255)
	Description	VARCHAR(1000)
	RegistrationCheckInType	CHAR(1)
	Active	DECIMAL(1)

EventGroup

PK	EventGroupId	DECIMAL(12)
FK1	EventId	DECIMAL(12)
	Name	VARCHAR(255)
	Description	VARCHAR(1000)
	AgeStartYear	DECIMAL(2)
	AgeEndYear	DECIMAL(2)
	Active	DECIMAL(1)

EventSchedule

PK	EventScheduleId	DECIMAL(12)
FK1	EventId	DECIMAL(12)
	EventStartDate	DATETIME
	EventEndDate	DATETIME
	RegistrationStartDate	DATETIME
	RegistrationEndDate	DATETIME
	CreatedDate	DATETIME

Adult

PK, FK1	PersonId	DECIMAL(12)
	UserName	VARCHAR(64)
	EncryptedPassword	VARCHAR(20)
	MobilePhone	VARCHAR(10)
	MaritalStatus	CHAR(1)
	GovIssuedID	VARCHAR(20)
	TypeGovIssuedID	CHAR(1)

PK, FK1	PersonId
	NickNam
	Allergies
	Condition
	Custodial

ClassroomAssignment

PK	ClassroomId	DECIMAL(12)
FK1	EventScheduleId	DECIMAL(12)
FK2	EventGroupId	DECIMAL(12)
FK3	RoomId	DECIMAL(12)
	AssignmentDate	DATETIME
F4	AssignmentById	DECIMAL(12)
	Notes	VARCHAR(1000)

RegistrantAttendance

PK	RegistrantId	DECIMAL(12)
FK1	EventScheduleId	DECIMAL(12)
FK2	PersonId	DECIMAL(12)
FK3	ClassroomId	DECIMAL(12)
	PickupCode	VARCHAR(10)
	RegistrationDate	DATETIME
FK4	RegisteredById	DECIMAL(12)
	CheckInDate	DATETIME
FK5	CheckInById	DECIMAL(12)
	TypeCheckIn	CHAR(1)
	CheckOutDate	DATETIME
FK6	CheckOutById	DECIMAL(12)
	TypeCheckOut	CHAR(1)
	Notes	VARCHAR(1000)

Volunteer

PK, FK1	PersonId	DECIMAL(12)
	BackgroundCheckStatus	CHAR(1)
	BackgroundCheckDate	DATETIME
	VolunteerType	CHAR(1)
	Active	DECIMAL(1)

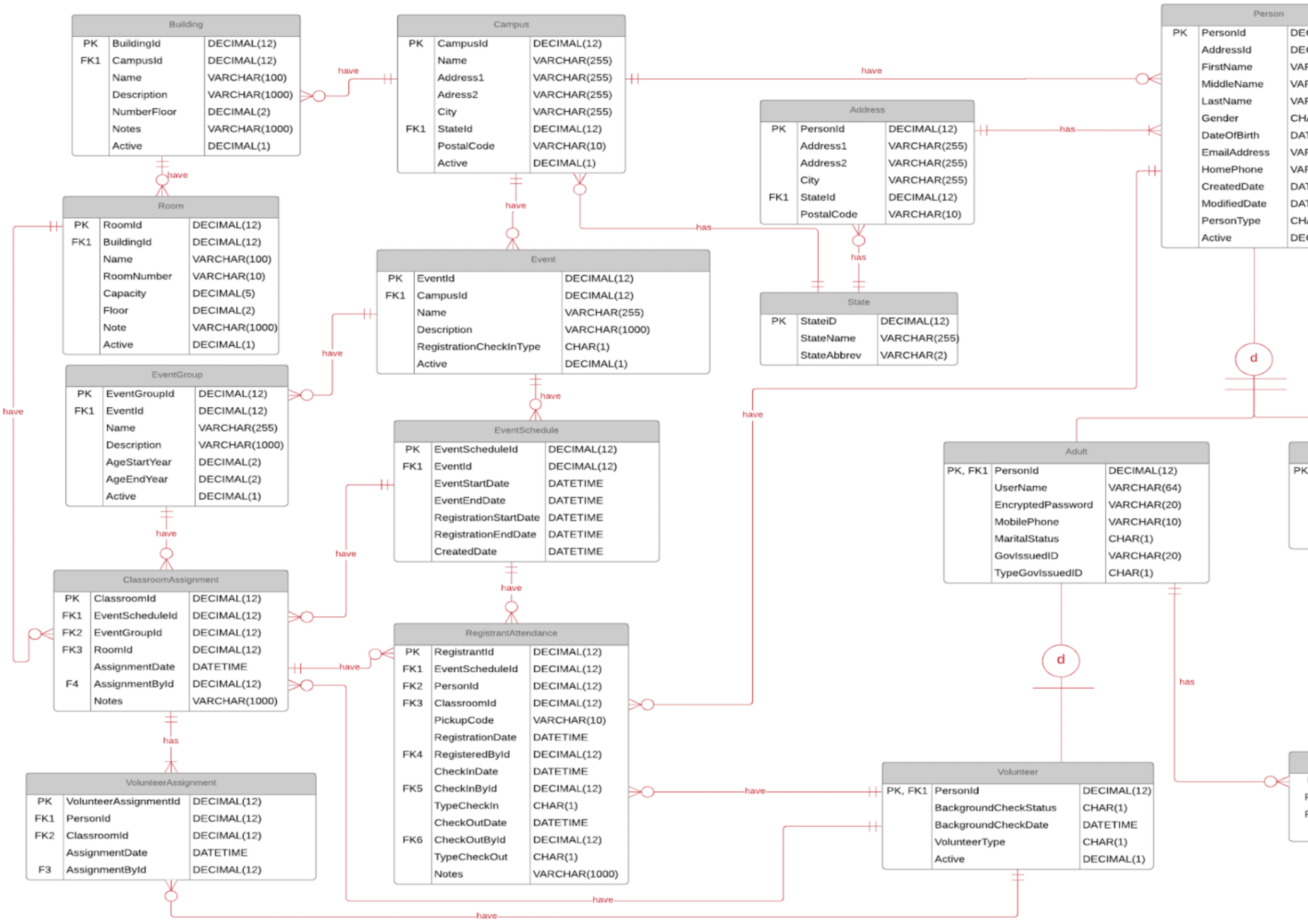
P
FK
FK

VolunteerAssignment

PK	VolunteerAssignmentId	DECIMAL(12)
FK1	PersonId	DECIMAL(12)
FK2	ClassroomId	DECIMAL(12)
	AssignmentDate	DATETIME
F3	AssignmentById	DECIMAL(12)

MyKids-CheckIn Normalization

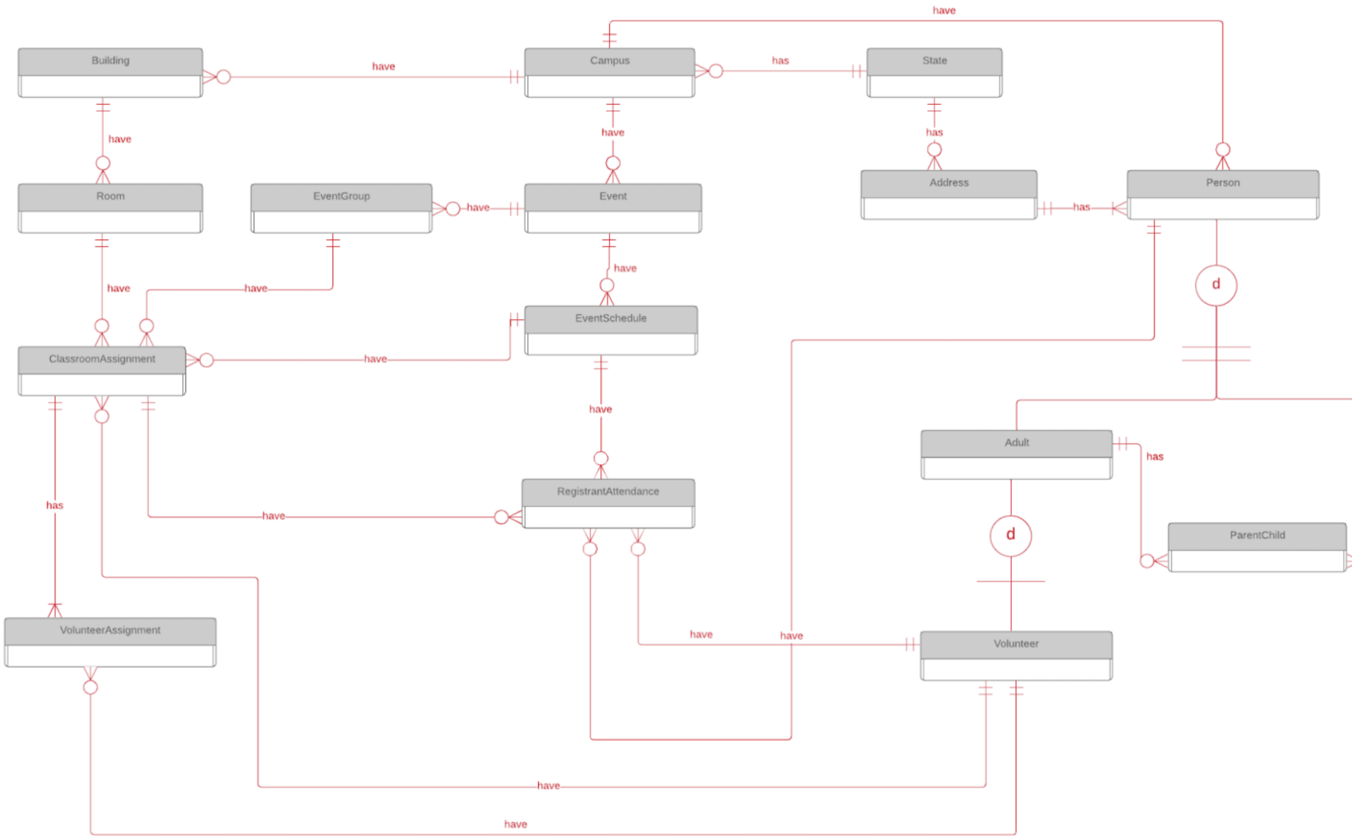
I notice only one place where there is redundancy in my physical ERD, and that is with the address information in the Person entity. If different people are registered and the are all living in the same address, this information is redundant. Here is my ERD with the address information normalized.



Below are my structural database rules modified to reflect the new entities. The new ones are italicized.

1. Each *Person* is associated with one *Campus*; Each *Campus* may be associated with many *Persons*.
2. Each *Event* is associated with one *Campus*; Each *Campus* may be associated with many *Events*.
3. Each *Adult* may be associated to one or many *children*. A *Child* is associated with one or many *Adults*.
4. Each *Event* may be associated to many *Event Schedule*; An *Event Schedule* is associated to one *Event*.
5. An *Event Schedule* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *Event Schedule*.
6. A *Person* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *Person*.
7. Each *Event Schedule* may be associated with many *Classroom Assignments*; Each *Classroom Assignment* is associated with one *Event Schedule*.
8. Each *Event* may be associated to many *Event Groups*; An *Event Group* is associated to only one *Event*.
9. A *Volunteer* may be associated with many *Volunteer Assignments*; A *Volunteer Assignment* is associated to one *Volunteer*.
10. A *Classroom Assignment* may be associated to many *Registrant Attendances*; A *Registrant Attendance* is associated to one *Classroom Assignment*.
11. An *Event Group* may be associated to many *Classroom Assignment*; A *Classroom Assignment* is associated to one *Event Group*.
12. Each *Campus* may be associated to many *Buildings*; A Building is associated to one *Campus*.
13. Each *Building* may be associated to many *Rooms*; A *Room* is associated to one *Building*.
14. A *Room* may be associated to many *Classroom Assignments*; A *Classroom Assignment* is associated to one *Room*.
15. Each *person* is an *adult* or *child*.
16. Each *adult* is a *volunteer*, or none.
17. A *person* lives at an *address*; Each *address* is associated with one or many persons.
18. Each *address* has a *state*; Each *state* may be associated with many *addresses*.

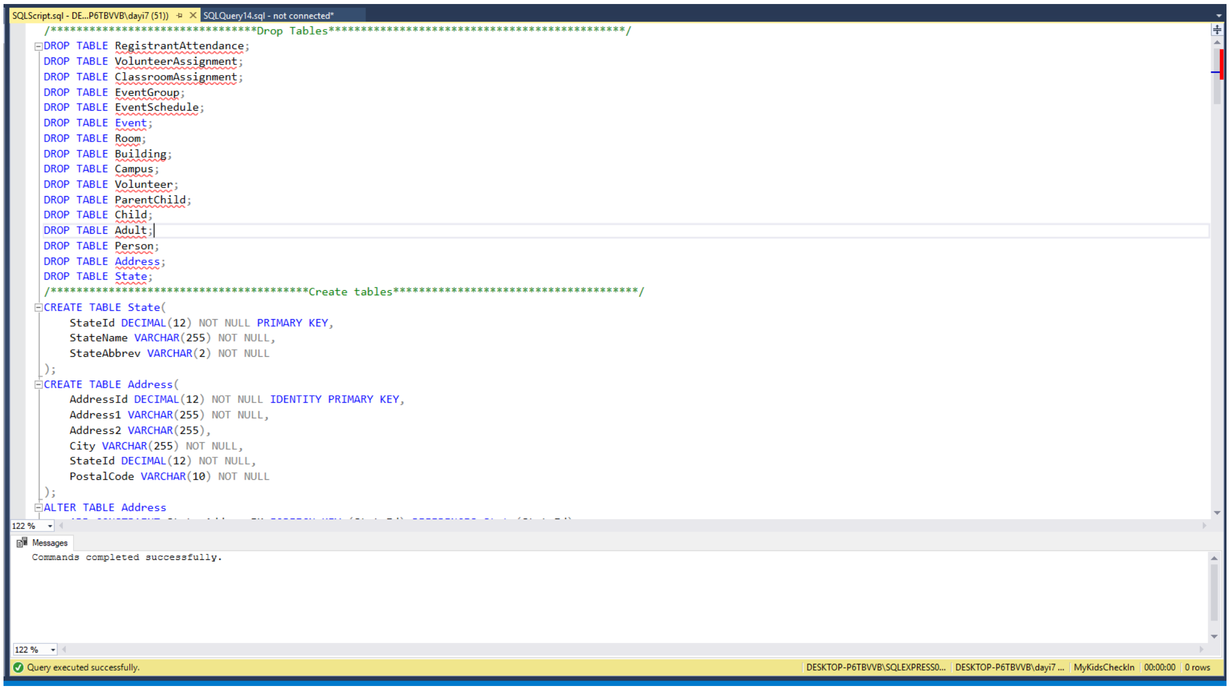
Below is my new conceptual ERD to reflect the new entities.


have
Building
have
Campus
has
State
have
have
has
Room
EventGroup
have
Event
Address
has
Person
have
have
have
d
EventSchedule
ClassroomAssignment
have
have
Adult
has
has
RegistrantAttendance
have
d
ParentChild
have
have
VolunteerAssignment
Volunteer
have
have

MyKids-CheckIn Create Script

I created a script file named SQLScript.sql which includes all the DROP TABLE commands at the top s that script is rerunnable and then I added the CREATE TABLE command with all the columns and constraints. I just included a few screens shots of the scripts and the execution.

Drop statement for all the tables and create table State and Address.

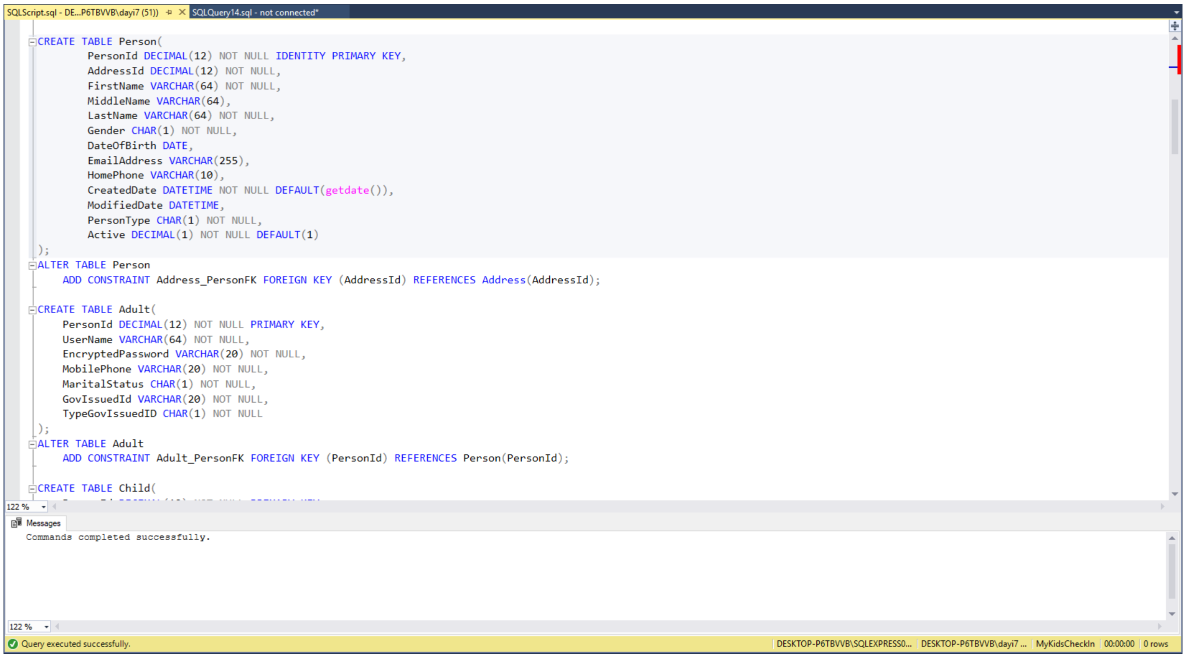

SQLScript.sql - DE...P6TBVVB\dayi7 (51))
SQLQuery14.sql - not connected*
122 %
Messages
122 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```
/*******************************Drop Tables*********************************************/
DROP TABLE RegistrantAttendance;
DROP TABLE VolunteerAssignment;
DROP TABLE ClassroomAssignment;
DROP TABLE EventGroup;
DROP TABLE EventSchedule;
DROP TABLE Event;
DROP TABLE Room;
DROP TABLE Building;
DROP TABLE Campus;
DROP TABLE Volunteer;
DROP TABLE ParentChild;
DROP TABLE Child;
DROP TABLE Adult;
DROP TABLE Person;
DROP TABLE Address;
DROP TABLE State;
/**************************************Create tables*************************************/
CREATE TABLE State(
    StateId DECIMAL(12) NOT NULL PRIMARY KEY,
    StateName VARCHAR(255) NOT NULL,
    StateAbbrev VARCHAR(2) NOT NULL
);
CREATE TABLE Address(
    AddressId DECIMAL(12) NOT NULL IDENTITY PRIMARY KEY,
    Address1 VARCHAR(255) NOT NULL,
    Address2 VARCHAR(255),
    City VARCHAR(255) NOT NULL,
    StateId DECIMAL(12) NOT NULL,
    PostalCode VARCHAR(10) NOT NULL
);
ALTER TABLE Address
```

```
Commands completed successfully.
```

Query executed successfully.

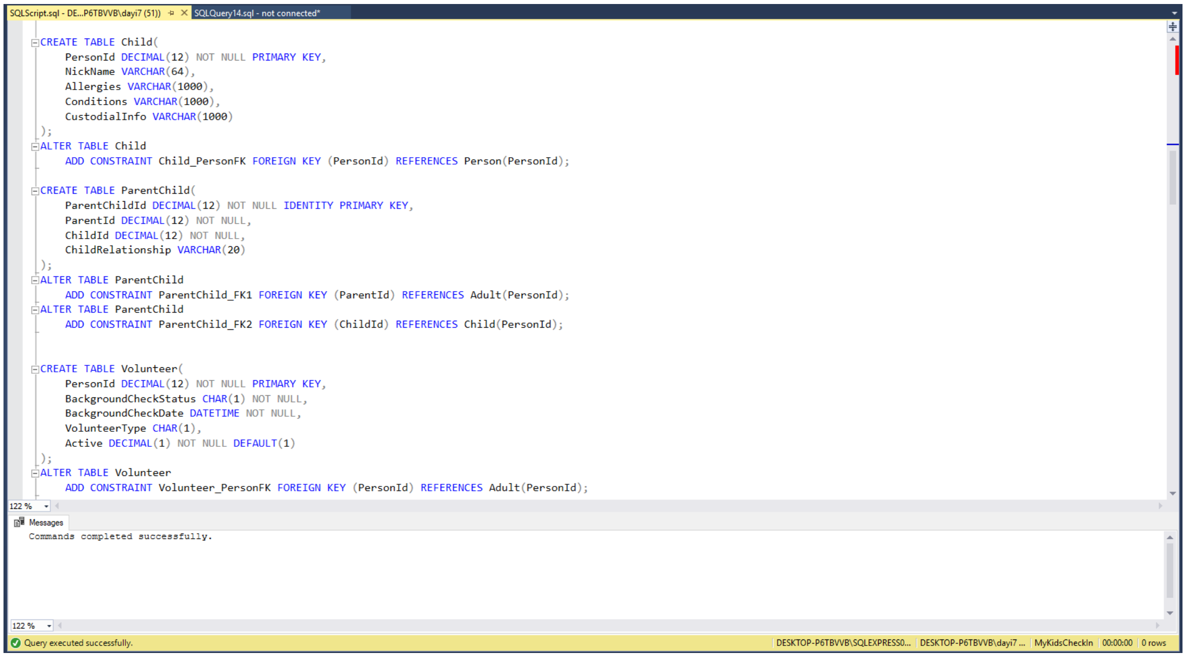
Create table Person Supertype and Adult subtype


SQLScript.sql - DE...P6TBVVB\dayi7 (51))
SQLQuery14.sql - not connected*
122 %
Messages
122 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```
CREATE TABLE Person(
        PersonId DECIMAL(12) NOT NULL IDENTITY PRIMARY KEY,
        AddressId DECIMAL(12) NOT NULL,
        FirstName VARCHAR(64) NOT NULL,
        MiddleName VARCHAR(64),
        LastName VARCHAR(64) NOT NULL,
        Gender CHAR(1) NOT NULL,
        DateOfBirth DATE,
        EmailAddress VARCHAR(255),
        HomePhone VARCHAR(10),
        CreatedDate DATETIME NOT NULL DEFAULT(getdate()),
        ModifiedDate DATETIME,
        PersonType CHAR(1) NOT NULL,
        Active DECIMAL(1) NOT NULL DEFAULT(1)
);
ALTER TABLE Person
    ADD CONSTRAINT Address_PersonFK FOREIGN KEY (AddressId) REFERENCES Address(AddressId);

CREATE TABLE Adult(
    PersonId DECIMAL(12) NOT NULL PRIMARY KEY,
    UserName VARCHAR(64) NOT NULL,
    EncryptedPassword VARCHAR(20) NOT NULL,
    MobilePhone VARCHAR(20) NOT NULL,
    MaritalStatus CHAR(1) NOT NULL,
    GovIssuedId VARCHAR(20) NOT NULL,
    TypeGovIssuedID CHAR(1) NOT NULL
);
ALTER TABLE Adult
    ADD CONSTRAINT Adult_PersonFK FOREIGN KEY (PersonId) REFERENCES Person(PersonId);

CREATE TABLE Child(
```

```
Commands completed successfully.
```

Query executed successfully.



SQLScript.sql - DE...P6TBVVB\dayi7 (51))
SQLQuery14.sql - not connected*
122 %
Messages
122 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```sql
CREATE TABLE Child(
    PersonId DECIMAL(12) NOT NULL PRIMARY KEY,
    NickName VARCHAR(64),
    Allergies VARCHAR(1000),
    Conditions VARCHAR(1000),
    CustodialInfo VARCHAR(1000)
);
ALTER TABLE Child
    ADD CONSTRAINT Child_PersonFK FOREIGN KEY (PersonId) REFERENCES Person(PersonId);

CREATE TABLE ParentChild(
    ParentChildId DECIMAL(12) NOT NULL IDENTITY PRIMARY KEY,
    ParentId DECIMAL(12) NOT NULL,
    ChildId DECIMAL(12) NOT NULL,
    ChildRelationship VARCHAR(20)
);
ALTER TABLE ParentChild
    ADD CONSTRAINT ParentChild_FK1 FOREIGN KEY (ParentId) REFERENCES Adult(PersonId);
ALTER TABLE ParentChild
    ADD CONSTRAINT ParentChild_FK2 FOREIGN KEY (ChildId) REFERENCES Child(PersonId);


CREATE TABLE Volunteer(
    PersonId DECIMAL(12) NOT NULL PRIMARY KEY,
    BackgroundCheckStatus CHAR(1) NOT NULL,
    BackgroundCheckDate DATETIME NOT NULL,
    VolunteerType CHAR(1),
    Active DECIMAL(1) NOT NULL DEFAULT(1)
);
ALTER TABLE Volunteer
    ADD CONSTRAINT Volunteer_PersonFK FOREIGN KEY (PersonId) REFERENCES Adult(PersonId);
```

```
Commands completed successfully.
```

Query executed successfully.

MyKids-CheckIn Indexes

Below is a table identifying each foreign key column that I will create an index for, whether the index should be unique or not, and why.

Column	Unique?	Description
Address.StateId	Not unique	The FK in Address referencing State is not unique because there can be many addresses with the same state.
Person.AddressId	Not unique	The FK in Person referencing Address is not unique because there can be many person with the same address.
ParentChild.ParentId	Not unique	The FK in ParentChild referencing Adult is not unique because there can be many child of the same Adult.
ParentChild.ChildId	Not unique	The FK in ParentChild referencing Child is not unique because there can be many Parents of the same Child.

Campus.StateId	Not unique	The FK in Campus referencing State is not unique because there can be many Campuses with the same state.
Building.CampusId	Not unique	The FK in Building referencing Campus is not unique because there can be many Buildings with the same campus.
Room.BuildingId	Not unique	The FK in Room referencing Building is not unique because there can be many rooms in the same building.
Event.CampusId	Not unique	The FK in Event referencing Campus is not unique because there can be many Events in the same campus.
EventSchedule.EventId	Not unique	The FK in EventSchedule referencing Event is not unique because there can be many EventSchedule for the same event.
EventGroup.EventId	Not unique	The FK in EventGroup referencing Event is not unique because there can be many EventGroup for the same Event.
ClassroomAssignment.EventScheduleId	Not unique	The FK in ClassroomAssignment referencing EventSchedule is not unique because there can be many ClassroomAssignments for the same EventSchedule.

ClassroomAssignment.EventGroupId	Not unique	The FK in ClassroomAssignment referencing EventGroup is not unique because there can be many ClassroomAssignments for the same EventGroup.
ClassroomAssignment.RoomId	Not unique	The FK in ClassroomAssignment referencing Room is not unique because there can be many ClassroomAssignments with the same room.
ClassroomAssignment.AssignmentById	Not unique	The FK in ClassroomAssignment referencing Volunteer is not unique because there can be many ClassroomAssignments with the same volunteer Id.
VolunteerAssignment.PersonId	Not unique	The FK in VolunteerAssignment referencing Volunteer is not unique because there can be many VolunteerAssignment with the same person(volunteer).
VolunteerAssignment.ClassroomId	Not unique	The FK in VolunteerAssignment referencing ClassroomAssignment is not unique because there can be many VolunteerAssignment for the same ClassroomAssignment.
VolunteerAssignment.AssignmentById	Not unique	The FK in VolunteerAssignment referencing Volunteer is

		not unique because there can be many VolunteerAssignment for the same volunteer id.
RegistrantAttendance.EventScheduleId	Not unique	The FK in RegistrantAttendance referencing EventSchedule is not unique because there can be many RegistrantAttendance for the same EventSchedule.
RegistrantAttendance.PersonId	Not unique	The FK in RegistrantAttendance referencing Person is not unique because there can be many RegistrantAttendance for the same Person.
RegistrantAttendance.ClassroomId	Not unique	The FK in RegistrantAttendance referencing ClassroomAssignment is not unique because there can be many RegistrantAttendance for the same ClassroomAssignment.
RegistrantAttendance.RegisteredById	Not unique	The FK in RegistrantAttendance referencing Person is not unique because there can be many RegistrantAttendance for the same Person.
RegistrantAttendance.CheckInById	Not unique	The FK in RegistrantAttendance referencing Volunteer is not unique because there can be many RegistrantAttendance for the same Volunteer.

RegistrantAttendance.CheckOutById	Not unique	The FK in RegistrantAttendance referencing Volunteer is not unique because there can be many RegistrantAttendance for the same Volunteer.

I also found 9 query driven indexes by predicting what columns will be commonly queried.

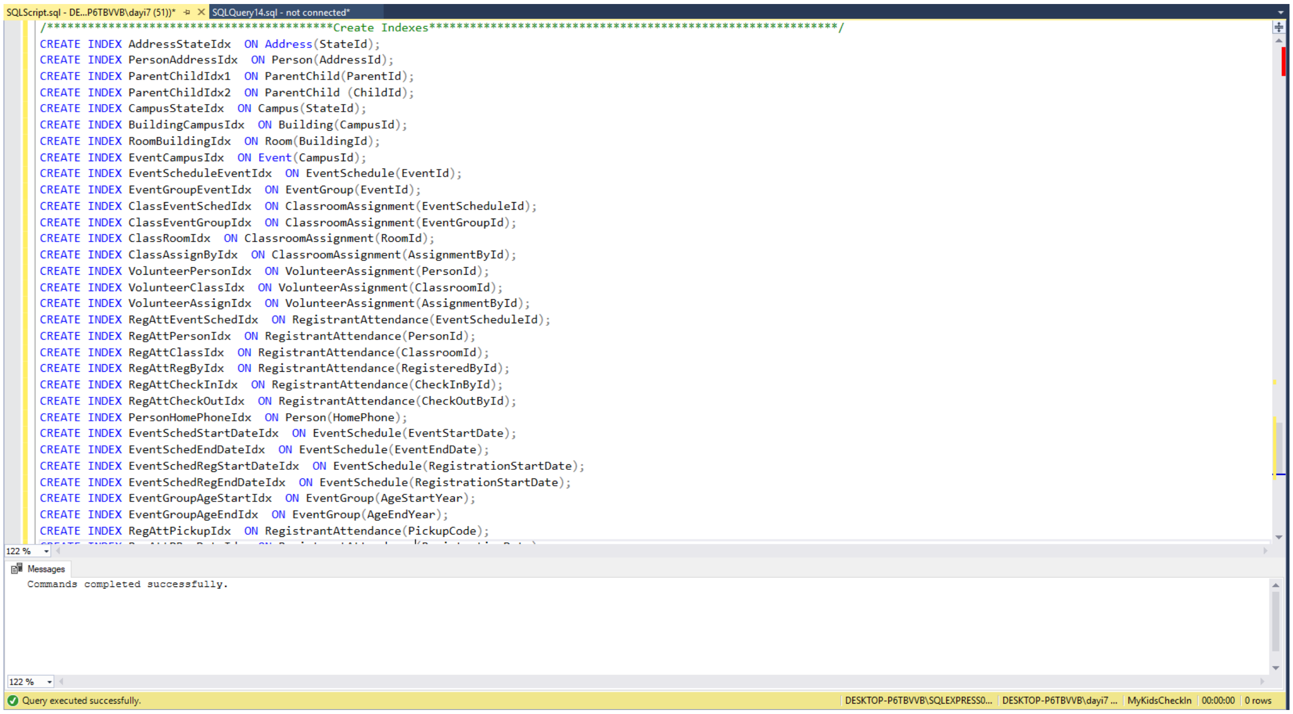
In the following table I will indicate the name of the column and the reason why I considered necessary to index them.

Column	Unique?	Reason
Person.HomePhone	Not unique	When parents register a child, they will have to search their child in the app by the phone number.
EventSchedule.EventStartDate	Not unique	This field will be used in reports and also when a child is being registered it will be used to pull current events.
EventSchedule.EventEndDate	Not unique	This field will be used in reports and also when a child is being registered it will be used to pull current events.
EventGgroup.AgeStartYear	Not unique	When a child is registered to an event the app will suggest the age group a child should check in; this field indicates the start age range value.
EventGgroup.AgeEndYear	Not unique	When a child is registered to an event the app will suggest the age group a child should check in; this field indicates the End age range value.
RegistrantAttendance.PickupCode	Not unique	When a child is either checked-in or checked-out the app has to scan the QR code which contains the PickupCode.
RegistrantAttendance.RegistrationDate	Not unique	When a child is checked-out the app has to scan the QR code which contains the PickupCode and it will also use the registration Date, and classroom to validate that they are checking out the correct child.

EventSchedule.RegistrationStartDate	Not unique	When a parent registers their child in a specific event the app will query the EventSchedule table asking if the current time is in between the Registration StartDate and Registration End date.
EventSchedule.RegistrationEndDate	Not unique	When a parent registers their child in a specific event the app will query the EventSchedule table asking if the current time is in between the Registration StartDate and Registration End date.

MyKids-CheckIn Index Creation

Here is a screenshot demonstrating creation of all my foreign key indexes and the query driven indexes. I added the CREATE INDEX commands to the end of the SQLScript.sql file.


SQLScript.sql - DE...P6TBVVB\dayi7 (51))*
SQLQuery14.sql - not connected*
122 %
Messages
Query executed successfully.
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```sql
/*****************************************Create Indexes***********************************************************/
CREATE INDEX AddressStateIdx  ON Address(StateId);
CREATE INDEX PersonAddressIdx  ON Person(AddressId);
CREATE INDEX ParentChildIdx1  ON ParentChild(ParentId);
CREATE INDEX ParentChildIdx2  ON ParentChild (ChildId);
CREATE INDEX CampusStateIdx  ON Campus(StateId);
CREATE INDEX BuildingCampusIdx  ON Building(CampusId);
CREATE INDEX RoomBuildingIdx  ON Room(BuildingId);
CREATE INDEX EventCampusIdx  ON Event(CampusId);
CREATE INDEX EventScheduleEventIdx  ON EventSchedule(EventId);
CREATE INDEX EventGroupEventIdx  ON EventGroup(EventId);
CREATE INDEX ClassEventSchedIdx  ON ClassroomAssignment(EventScheduleId);
CREATE INDEX ClassEventGroupIdx  ON ClassroomAssignment(EventGroupId);
CREATE INDEX ClassRoomIdx  ON ClassroomAssignment(RoomId);
CREATE INDEX ClassAssignByIdx  ON ClassroomAssignment(AssignmentById);
CREATE INDEX VolunteerPersonIdx  ON VolunteerAssignment(PersonId);
CREATE INDEX VolunteerClassIdx  ON VolunteerAssignment(ClassroomId);
CREATE INDEX VolunteerAssignIdx  ON VolunteerAssignment(AssignmentById);
CREATE INDEX RegAttEventSchedIdx  ON RegistrantAttendance(EventScheduleId);
CREATE INDEX RegAttPersonIdx  ON RegistrantAttendance(PersonId);
CREATE INDEX RegAttClassIdx  ON RegistrantAttendance(ClassroomId);
CREATE INDEX RegAttRegByIdx  ON RegistrantAttendance(RegisteredById);
CREATE INDEX RegAttCheckInIdx  ON RegistrantAttendance(CheckInById);
CREATE INDEX RegAttCheckOutIdx  ON RegistrantAttendance(CheckOutById);
CREATE INDEX PersonHomePhoneIdx  ON Person(HomePhone);
CREATE INDEX EventSchedStartDateIdx  ON EventSchedule(EventStartDate);
CREATE INDEX EventSchedEndDateIdx  ON EventSchedule(EventEndDate);
CREATE INDEX EventSchedRegStartDateIdx  ON EventSchedule(RegistrationStartDate);
CREATE INDEX EventSchedRegEndDateIdx  ON EventSchedule(RegistrationStartDate);
CREATE INDEX EventGroupAgeStartIdx  ON EventGroup(AgeStartYear);
CREATE INDEX EventGroupAgeEndIdx  ON EventGroup(AgeEndYear);
CREATE INDEX RegAttPickupIdx  ON RegistrantAttendance(PickupCode);
```

```
Commands completed successfully.
```

MyKids-CheckIn Transactions

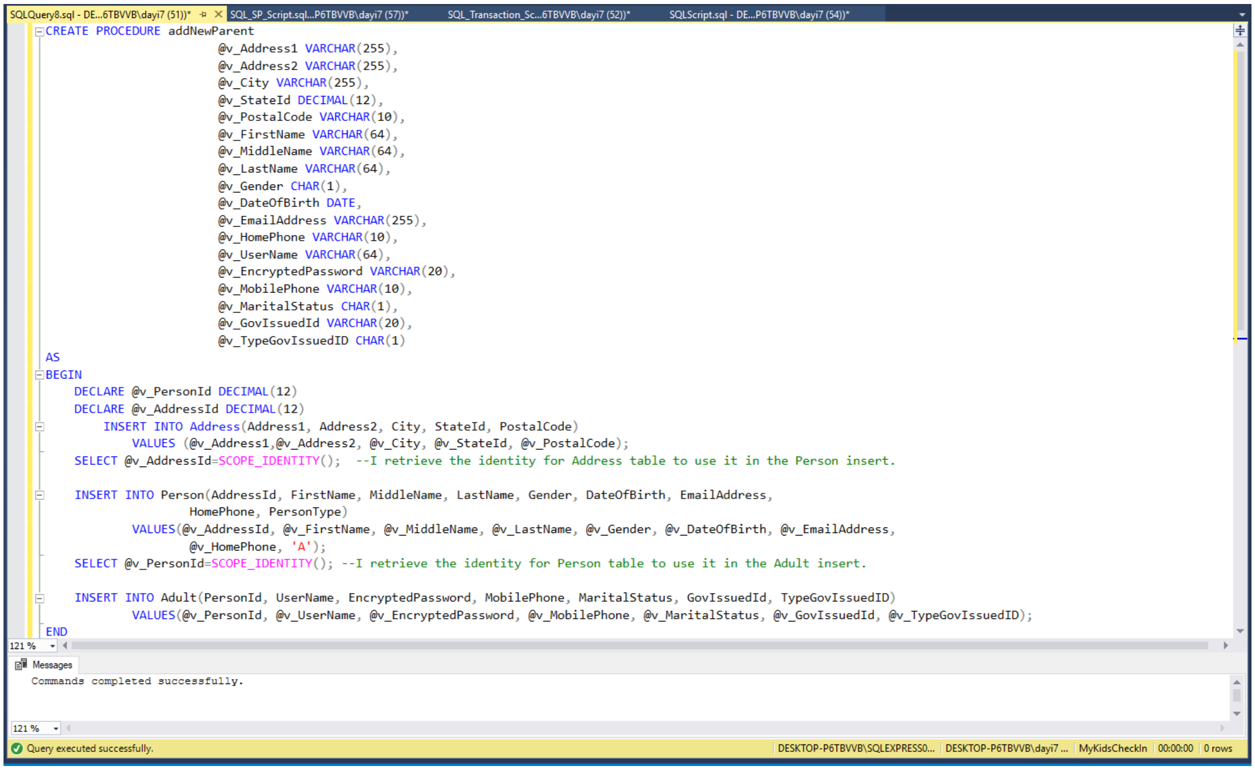
The first use case for MyKids-CheckIn is the “New Family Registration” use case listed below.

New Family Registration Use Case

1. The person visits the kiosk located where the app is installed.
2. The person will click on the “Create New Family” option.
3. The person selects the campus they usually attend to; the app will then ask them if the person they are entering information for is a parent or a child. If it is a child they will have to extra information, like medical alerts (e.g., allergies, special needs, medication, etc.) or custodial information. Once all information is entered the new family is created in the database.
4. The app will ask the parent/guardian if they would like to print out badges for the current service/event. If they select yes, the app will printout the badges that will be used in the check-in and check-out process.

For this use case, I will implement a transaction that creates new family record, using SQL Server. I created 2 stored procedures: “addNewParent”, which adds a new parent and “addNewChild”, that will be used to add a child or children a parent may have.

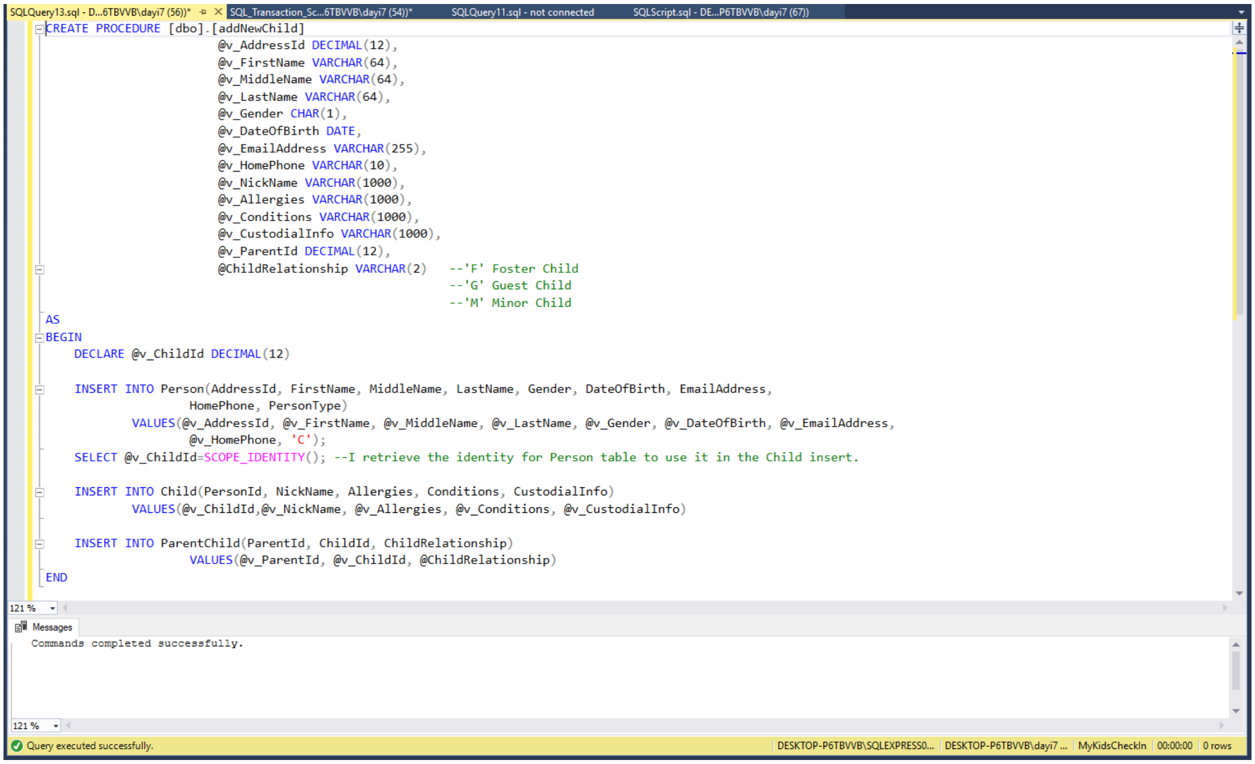
Here are screenshots of my 2 stored procedure definition.


SQLQuery8.sql - DE...6TBVVB\dayi7 (51))*
SQL_SP_Script.sql...P6TBVVB\dayi7 (57))*
SQL_Transaction_Sc...6TBVVB\dayi7 (52))*
SQLScript.sql - DE...P6TBVVB\dayi7 (54))*
121 %
Messages
121 %

```sql
CREATE PROCEDURE addNewParent
                              @v_Address1 VARCHAR(255),
                              @v_Address2 VARCHAR(255),
                              @v_City VARCHAR(255),
                              @v_StateId DECIMAL(12),
                              @v_PostalCode VARCHAR(10),
                              @v_FirstName VARCHAR(64),
                              @v_MiddleName VARCHAR(64),
                              @v_LastName VARCHAR(64),
                              @v_Gender CHAR(1),
                              @v_DateOfBirth DATE,
                              @v_EmailAddress VARCHAR(255),
                              @v_HomePhone VARCHAR(10),
                              @v_UserName VARCHAR(64),
                              @v_EncryptedPassword VARCHAR(20),
                              @v_MobilePhone VARCHAR(10),
                              @v_MaritalStatus CHAR(1),
                              @v_GovIssuedId VARCHAR(20),
                              @v_TypeGovIssuedID CHAR(1)
AS
BEGIN
    DECLARE @v_PersonId DECIMAL(12)
    DECLARE @v_AddressId DECIMAL(12)
        INSERT INTO Address(Address1, Address2, City, StateId, PostalCode)
            VALUES (@v_Address1,@v_Address2, @v_City, @v_StateId, @v_PostalCode);
    SELECT @v_AddressId=SCOPE_IDENTITY();  --I retrieve the identity for Address table to use it in the Person insert.

    INSERT INTO Person(AddressId, FirstName, MiddleName, LastName, Gender, DateOfBirth, EmailAddress,
                HomePhone, PersonType)
            VALUES(@v_AddressId, @v_FirstName, @v_MiddleName, @v_LastName, @v_Gender, @v_DateOfBirth, @v_EmailAddress,
                @v_HomePhone, 'A');
    SELECT @v_PersonId=SCOPE_IDENTITY(); --I retrieve the identity for Person table to use it in the Adult insert.

    INSERT INTO Adult(PersonId, UserName, EncryptedPassword, MobilePhone, MaritalStatus, GovIssuedId, TypeGovIssuedID)
            VALUES(@v_PersonId, @v_UserName, @v_EncryptedPassword, @v_MobilePhone, @v_MaritalStatus, @v_GovIssuedId, @v_TypeGovIssuedID);
END
```

```
Commands completed successfully.
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows



SQLQuery13.sql - D...6TBVVB\dayi7 (56))*
SQL_Transaction_Sc...6TBVVB\dayi7 (54))*
SQLQuery11.sql - not connected
SQLScript.sql - DE...P6TBVVB\dayi7 (67))
121 %
Messages
121 %

```
CREATE PROCEDURE [dbo].[addNewChild]
                        @v_AddressId DECIMAL(12),
                        @v_FirstName VARCHAR(64),
                        @v_MiddleName VARCHAR(64),
                        @v_LastName VARCHAR(64),
                        @v_Gender CHAR(1),
                        @v_DateOfBirth DATE,
                        @v_EmailAddress VARCHAR(255),
                        @v_HomePhone VARCHAR(10),
                        @v_NickName VARCHAR(1000),
                        @v_Allergies VARCHAR(1000),
                        @v_Conditions VARCHAR(1000),
                        @v_CustodialInfo VARCHAR(1000),
                        @v_ParentId DECIMAL(12),
                        @ChildRelationship VARCHAR(2)   --'F' Foster Child
                                                        --'G' Guest Child
                                                        --'M' Minor Child
AS
BEGIN
    DECLARE @v_ChildId DECIMAL(12)

    INSERT INTO Person(AddressId, FirstName, MiddleName, LastName, Gender, DateOfBirth, EmailAddress,
                HomePhone, PersonType)
            VALUES(@v_AddressId, @v_FirstName, @v_MiddleName, @v_LastName, @v_Gender, @v_DateOfBirth, @v_EmailAddress,
                @v_HomePhone, 'C');
    SELECT @v_ChildId=SCOPE_IDENTITY(); --I retrieve the identity for Person table to use it in the Child insert.

    INSERT INTO Child(PersonId, NickName, Allergies, Conditions, CustodialInfo)
            VALUES(@v_ChildId,@v_NickName, @v_Allergies, @v_Conditions, @v_CustodialInfo)

    INSERT INTO ParentChild(ParentId, ChildId, ChildRelationship)
                VALUES(@v_ParentId, @v_ChildId, @ChildRelationship)
END
```

```
Commands completed successfully.
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

Stored Procedure: “addNewParent”

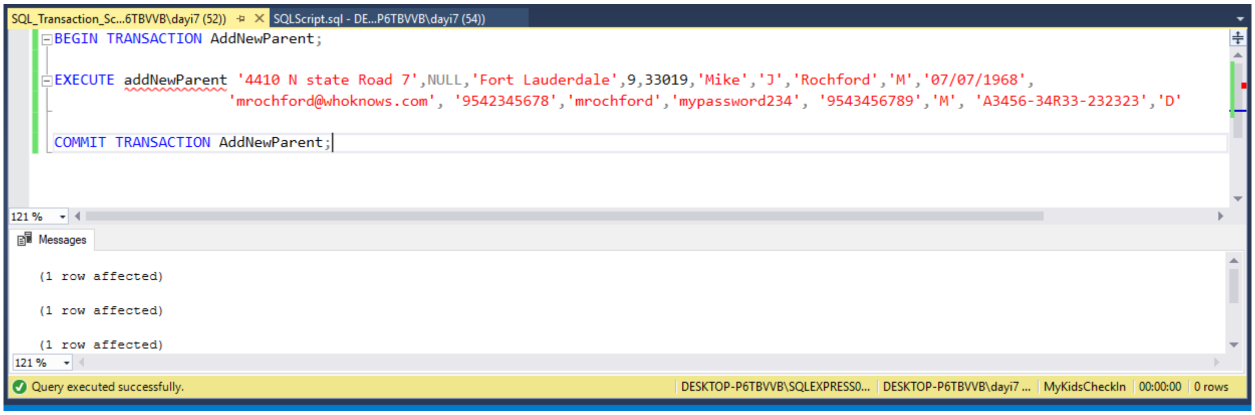
I gave it parameters that correspond to the Address, Person and Adult

The column Person.CreatedDate is always the current date, for which I added a default constraint for this column which will use the getdate() function from SQL Server so I do not need a parameter for it. I also do not need a parameter for Person .PersonType since this procedure will always be used to add a new parent so I hardcode the character ‘A’. I added a default constraint to Adult.IsVolunteer of a value of 0. Since this procedure will not be used to add a volunteer, I will insert the Adult data using the default value.

The primary keys for Address and Person table are Identity fields, which Is why do not need to pass these values

Inside the stored procedure, there are 3 insert statements to insert into the 3 respective tables.

Here is my screenshot of my stored procedure execution.



SQL_Transaction_Sc...6TBVVB\dayi7 (52))
SQLScript.sql - DE...P6TBVVB\dayi7 (54))
121 %
Messages
121 %

```
BEGIN TRANSACTION AddNewParent;

EXECUTE addNewParent '4410 N state Road 7',NULL,'Fort Lauderdale',9,33019,'Mike','J','Rochford','M','07/07/1968',
                     'mrochford@whoknows.com', '9542345678','mrochford','mypassword234', '9543456789','M', 'A3456-34R33-232323','D'

COMMIT TRANSACTION AddNewParent;
```

```
(1 row affected)

(1 row affected)

(1 row affected)
```

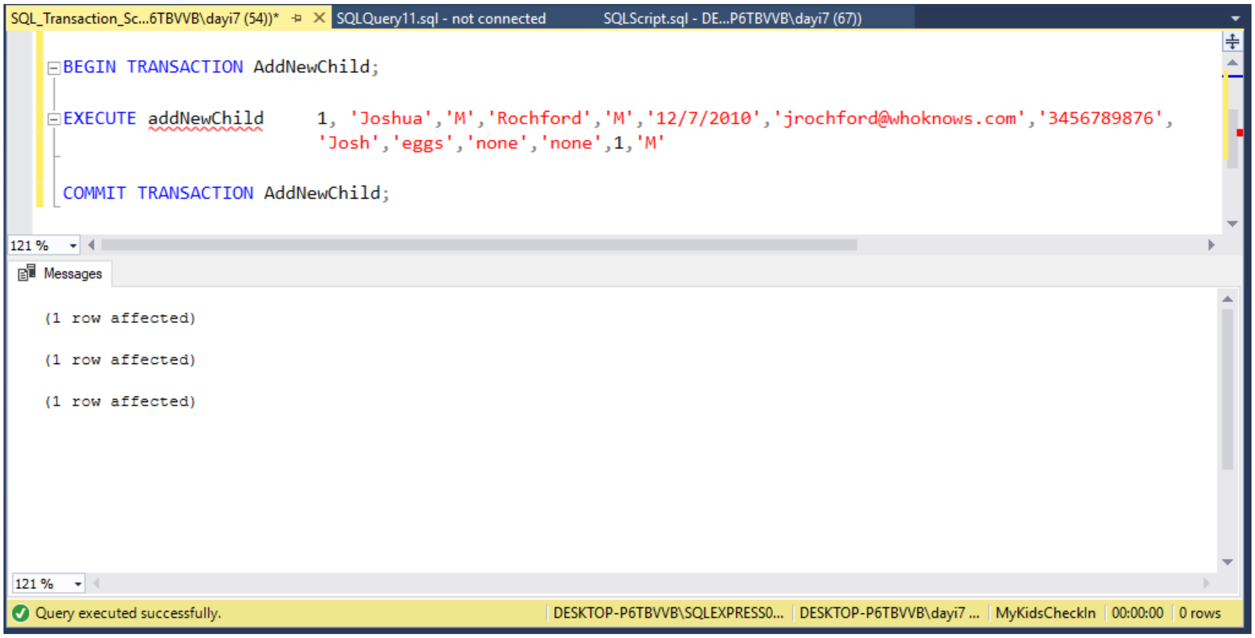
Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

Stored Procedure: “addNewChild”

I gave it parameters that correspond to the Person, Child and ParentChild tables.

Inside the stored procedure, there are 3 insert statements to insert into the 3 respective tables.

Here is a screenshot of my stored procedure execution.



SQL_Transaction_Sc...6TBVVB\dayi7 (54))*
SQLQuery11.sql - not connected
SQLScript.sql - DE...P6TBVVB\dayi7 (67))
121 %
Messages
121 %

```
BEGIN TRANSACTION AddNewChild;

EXECUTE addNewChild     1, 'Joshua','M','Rochford','M','12/7/2010','jrochford@whoknows.com','3456789876',
                        'Josh','eggs','none','none',1,'M'

COMMIT TRANSACTION AddNewChild;
```

```
(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

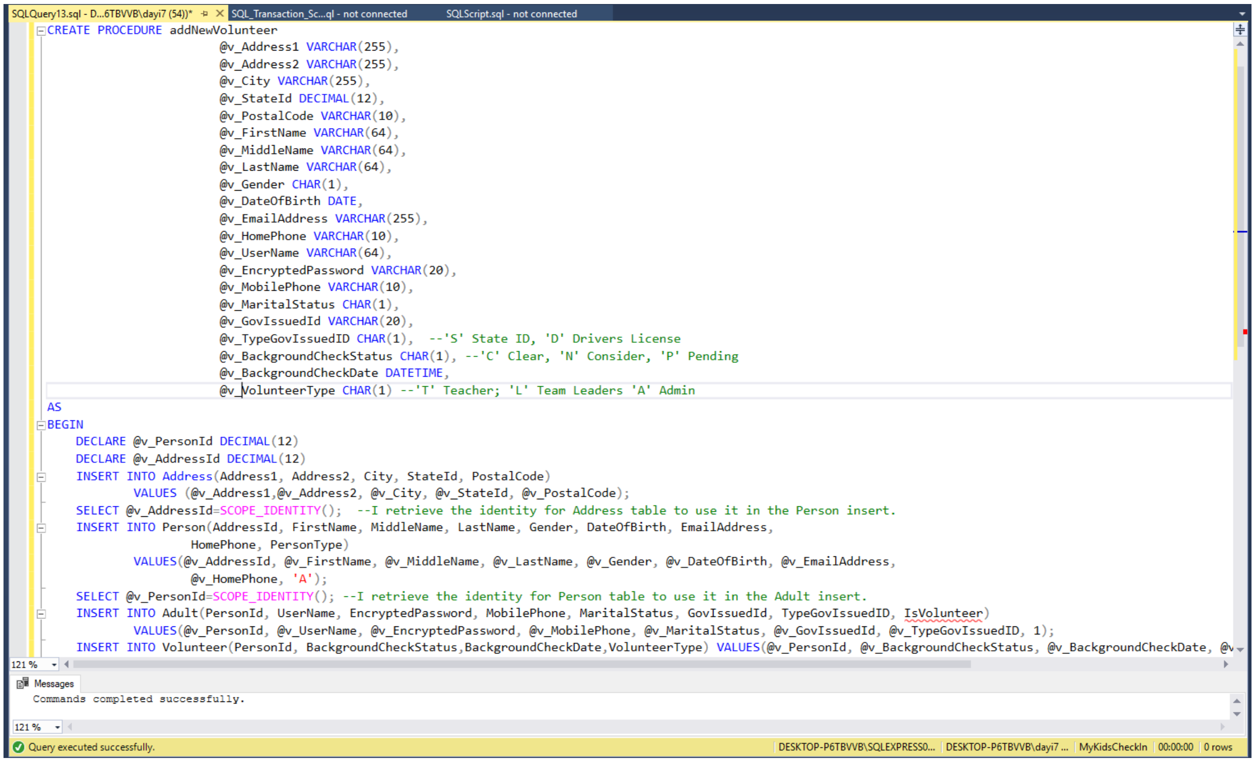
The second use case for MyKids-CheckIn is the "*New Volunteer Registration*" use case listed below.

New Volunteer Registration Use Case

1. The person visits the kiosk where the app is installed.
2. The person will click on the "Registration to Volunteer".
3. The person selects the campus they usually attend to and enters all the required information. Once all the information is entered, the Volunteer is created in the database as a Volunteer with a pending status.
4. The volunteer will be contacted in a few days for finger printing for background check. If the person passes the background check, he will be approved to volunteer, and his status will be changed to approved.

For this use case, I will implement a transaction that creates new volunteer record, using SQL Server. I created a stored procedure: "addNewVolunteer", which adds a new volunteer.

Here is a screenshot of my stored procedure definition.


SQLQuery13.sql - D...6TBVVB\dayi7 (54))*
SQL_Transaction_Sc...ql - not connected
SQLScript.sql - not connected
121 %
Messages
121 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```
CREATE PROCEDURE addNewVolunteer
                        @v_Address1 VARCHAR(255),
                        @v_Address2 VARCHAR(255),
                        @v_City VARCHAR(255),
                        @v_StateId DECIMAL(12),
                        @v_PostalCode VARCHAR(10),
                        @v_FirstName VARCHAR(64),
                        @v_MiddleName VARCHAR(64),
                        @v_LastName VARCHAR(64),
                        @v_Gender CHAR(1),
                        @v_DateOfBirth DATE,
                        @v_EmailAddress VARCHAR(255),
                        @v_HomePhone VARCHAR(10),
                        @v_UserName VARCHAR(64),
                        @v_EncryptedPassword VARCHAR(20),
                        @v_MobilePhone VARCHAR(10),
                        @v_MaritalStatus CHAR(1),
                        @v_GovIssuedId VARCHAR(20),
                        @v_TypeGovIssuedID CHAR(1),  --'S' State ID, 'D' Drivers License
                        @v_BackgroundCheckStatus CHAR(1), --'C' Clear, 'N' Consider, 'P' Pending
                        @v_BackgroundCheckDate DATETIME,
                        @v_VolunteerType CHAR(1) --'T' Teacher; 'L' Team Leaders 'A' Admin
AS
BEGIN
    DECLARE @v_PersonId DECIMAL(12)
    DECLARE @v_AddressId DECIMAL(12)
    INSERT INTO Address(Address1, Address2, City, StateId, PostalCode)
            VALUES (@v_Address1,@v_Address2, @v_City, @v_StateId, @v_PostalCode);
    SELECT @v_AddressId=SCOPE_IDENTITY();  --I retrieve the identity for Address table to use it in the Person insert.
    INSERT INTO Person(AddressId, FirstName, MiddleName, LastName, Gender, DateOfBirth, EmailAddress,
                HomePhone, PersonType)
            VALUES(@v_AddressId, @v_FirstName, @v_MiddleName, @v_LastName, @v_Gender, @v_DateOfBirth, @v_EmailAddress,
                @v_HomePhone, 'A');
    SELECT @v_PersonId=SCOPE_IDENTITY(); --I retrieve the identity for Person table to use it in the Adult insert.
    INSERT INTO Adult(PersonId, UserName, EncryptedPassword, MobilePhone, MaritalStatus, GovIssuedId, TypeGovIssuedID, IsVolunteer)
            VALUES(@v_PersonId, @v_UserName, @v_EncryptedPassword, @v_MobilePhone, @v_MaritalStatus, @v_GovIssuedId, @v_TypeGovIssuedID, 1);
    INSERT INTO Volunteer(PersonId, BackgroundCheckStatus,BackgroundCheckDate,VolunteerType) VALUES(@v_PersonId, @v_BackgroundCheckStatus, @v_BackgroundCheckDate, @v
```

```
Commands completed successfully.
```

Query executed successfully.

Stored Procedure: "addNewVolunteer"

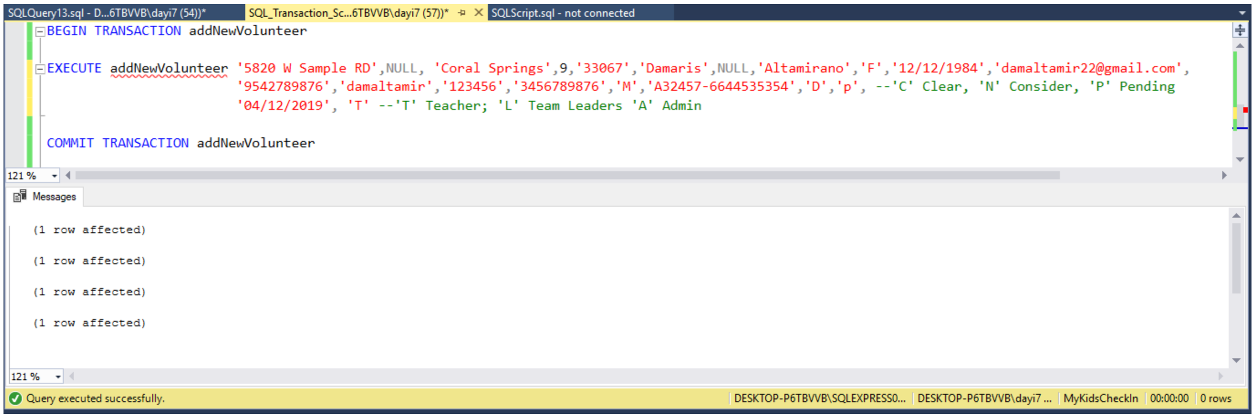
I gave it parameters that correspond to the Address, Person, Adult and Volunteer.

The column Person.CreatedDate is always the current date, for which I added a default constraint for this column which will use the getdate() function from SQL Server so I do not need a parameter for it. I also do not need a parameter for Person .PersonType since this procedure will always be used to add a new volunteer so I hardcode the character ‘A’. I added a default constraint to Adult.IsVolunteer of a value of 0. Since this procedure will be used to add a volunteer I will assign the value of 1.

The primary keys for Address and Person table are Identity fields, which Is why do not need to pass these values

Inside the stored procedure, there are 4 insert statements to insert into the 4 respective tables.

Here is a screenshot of my stored procedure execution.



SQLQuery13.sql - D...6TBVVB\dayi7 (54))*
SQL_Transaction_Sc...6TBVVB\dayi7 (57))*
SQLScript.sql - not connected

```sql
BEGIN TRANSACTION addNewVolunteer

EXECUTE addNewVolunteer '5820 W Sample RD',NULL, 'Coral Springs',9,'33067','Damaris',NULL,'Altamirano','F','12/12/1984','damaltamir22@gmail.com',
                        '9542789876','damaltamir','123456','3456789876','M','A32457-6644535354','D','p', --'C' Clear, 'N' Consider, 'P' Pending
                        '04/12/2019', 'T' --'T' Teacher; 'L' Team Leaders 'A' Admin

COMMIT TRANSACTION addNewVolunteer
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

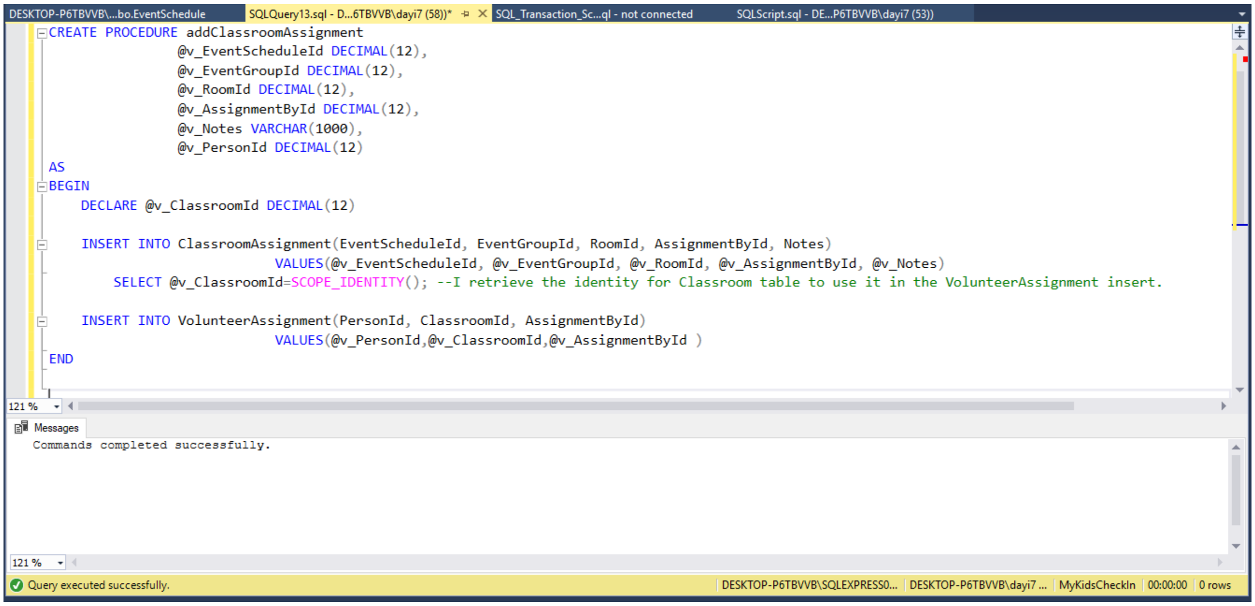
The third use case for MyKids-CheckIn is the “*Setting up classrooms for check-in*” use case listed below.

Setting up classrooms for check in Use Case

1. The team leader accesses the app from their tablets and signs in with their credentials.
2. The team leader will select the service/event and clicks on the “Open Classroom” option.
3. The team leader selects the Event and the event group (e.g. age group) with the classroom and assigns the teacher/volunteers.

For this use case, I will implement a transaction that creates new classroom assignment record, using SQL Server. I created a stored procedure: “addClassroomAssignment”.

Here is a screenshot of my stored procedure definition.


DESKTOP-P6TBVVB\...bo.EventSchedule
SQLQuery13.sql - D...6TBVVB\dayi7 (58))*
SQL_Transaction_Sc...ql - not connected
SQLScript.sql - DE...P6TBVVB\dayi7 (53))
121 %
Messages
121 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

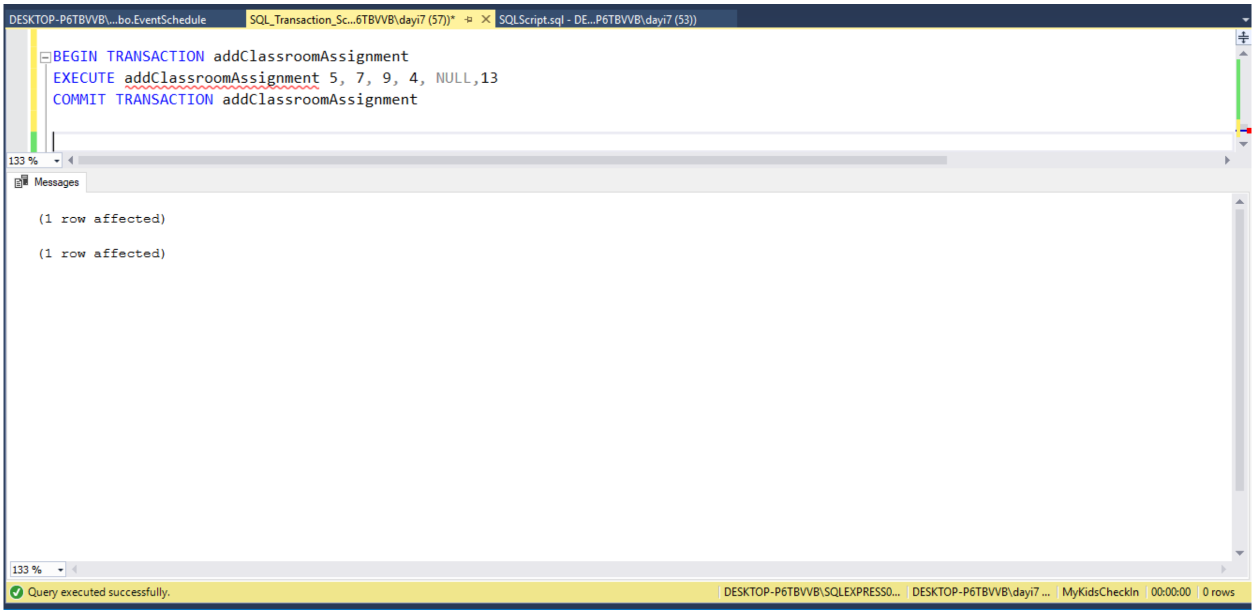
```sql
CREATE PROCEDURE addClassroomAssignment
                @v_EventScheduleId DECIMAL(12),
                @v_EventGroupId DECIMAL(12),
                @v_RoomId DECIMAL(12),
                @v_AssignmentById DECIMAL(12),
                @v_Notes VARCHAR(1000),
                @v_PersonId DECIMAL(12)
AS
BEGIN
    DECLARE @v_ClassroomId DECIMAL(12)

    INSERT INTO ClassroomAssignment(EventScheduleId, EventGroupId, RoomId, AssignmentById, Notes)
                        VALUES(@v_EventScheduleId, @v_EventGroupId, @v_RoomId, @v_AssignmentById, @v_Notes)
        SELECT @v_ClassroomId=SCOPE_IDENTITY(); --I retrieve the identity for Classroom table to use it in the VolunteerAssignment insert.

    INSERT INTO VolunteerAssignment(PersonId, ClassroomId, AssignmentById)
                        VALUES(@v_PersonId,@v_ClassroomId,@v_AssignmentById )
END
```

```
Commands completed successfully.
```

Query executed successfully.

Inside the stored procedure, there are 2 insert statements to insert into the 2 respective tables.

Here is a screenshot of my stored procedure execution.


DESKTOP-P6TBVVB\...bo.EventSchedule
SQL_Transaction_Sc...6TBVVB\dayi7 (57))*
SQLScript.sql - DE...P6TBVVB\dayi7 (53))
133 %
Messages
133 %
DESKTOP-P6TBVVB\SQLEXPRESS0...
DESKTOP-P6TBVVB\dayi7 ...
MyKidsCheckIn
00:00:00
0 rows

```sql
BEGIN TRANSACTION addClassroomAssignment
EXECUTE addClassroomAssignment 5, 7, 9, 4, NULL,13
COMMIT TRANSACTION addClassroomAssignment
```

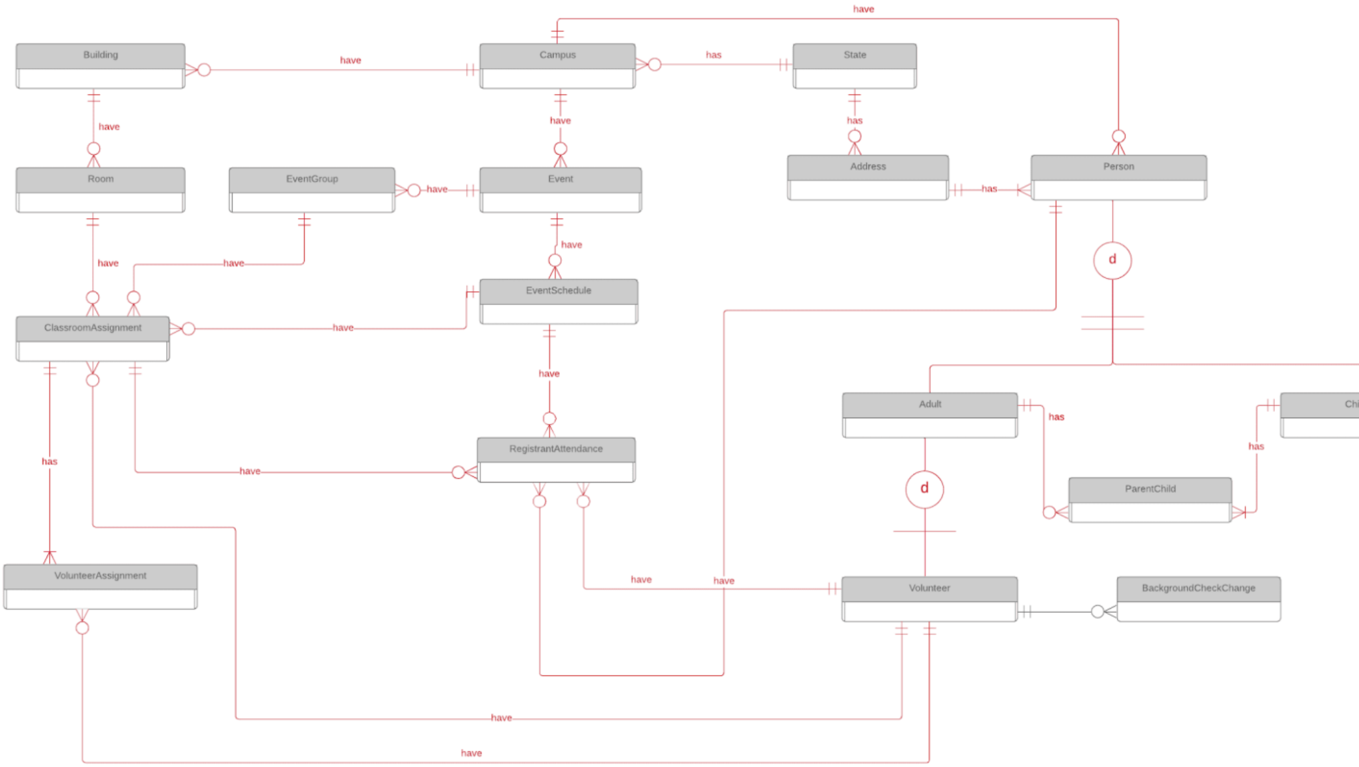
```
(1 row affected)

(1 row affected)
```

Query executed successfully.

MyKids-CheckIn History

In reviewing my DBMS physical ERD, one piece of data that would obviously benefit from a historical record is the volunteer's background check date change in the Volunteers' table. Such a history would help me determine the number of background checks done in a specific period. My new structural database rule is: Each volunteer may have many background check date changes; each back-ground check date change is for a volunteer.

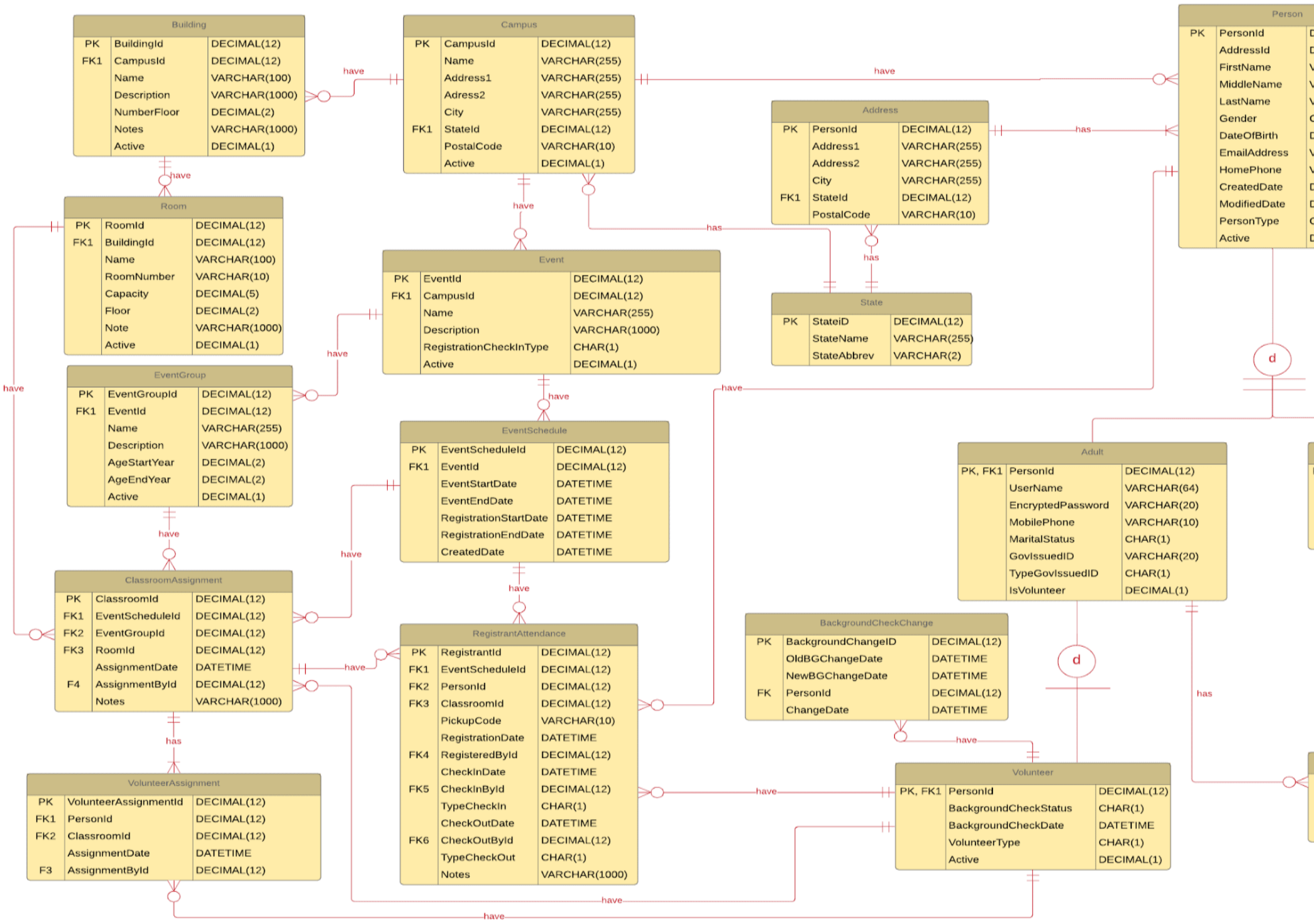
My updated conceptual ERD and structural rules including my new database rule are below:



Below are my structural database rules modified to reflect the new entities. The new ones are italicized.

1. Each *Person* is associated with one *Campus*; Each *Campus* may be associated with many *Persons*.
2. Each *Event* is associated with one *Campus*; Each *Campus* may be associated with many *Events*.
3. Each *Adult* may be associated to one or many *children*. A *Child* is associated with one or many *Adults*.
4. Each *Event* may be associated to many *Event Schedule*; An *Event Schedule* is associated to one *Event*.
5. An *Event Schedule* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *Event Schedule*.
6. A *Person* may be associated to many *Registrant Attendance*; A *Registrant Attendance* is associated to one *Person*.
7. Each *Event Schedule* may be associated with many *Classroom Assignments*; Each *Classroom Assignment* is associated with one *Event Schedule*.
8. Each *Event* may be associated to many *Event Groups*; An *Event Group* is associated to only one *Event*.
9. A *Volunteer* may be associated with many *Volunteer Assignments*; A *Volunteer Assignment* is associated to one *Volunteer*.
10. A *Classroom Assignment* may be associated to many *Registrant Attendances*; A *Registrant Attendance* is associated to one *Classroom Assignment*.
11. An *Event Group* may be associated to many *Classroom Assignment*; A *Classroom Assignment* is associated to one *Event Group*.
12. Each *Campus* may be associated to many *Buildings*; A Building is associated to one *Campus*.
13. Each *Building* may be associated to many *Rooms*; A *Room* is associated to one *Building*.
14. A *Room* may be associated to many *Classroom Assignments*; A *Classroom Assignment* is associated to one *Room*.
15. Each *person* is an *adult* or *child*.
16. Each *adult* is a *volunteer*, or none.
17. A *person* lives at an *address*; Each *address* is associated with one or many persons.
18. Each *address* has a *state*; Each *state* may be associated with many *addresses*.
19. ***Each volunteer may have many background check date changes; each back-ground check date change is for a volunteer.***

I added the BackgroundCheckChange entity and related it to Volunteer table. My updated DBMS physical ERD is below.



Building

PK	BuildingId	DECIMAL(12)
FK1	CampusId	DECIMAL(12)
	Name	VARCHAR(100)
	Description	VARCHAR(1000)
	NumberFloor	DECIMAL(2)
	Notes	VARCHAR(1000)
	Active	DECIMAL(1)

Campus

PK	CampusId	DECIMAL(12)
	Name	VARCHAR(255)
	Address1	VARCHAR(255)
	Adress2	VARCHAR(255)
	City	VARCHAR(255)
FK1	StateId	DECIMAL(12)
	PostalCode	VARCHAR(10)
	Active	DECIMAL(1)

Person

PK	PersonId	D
	AddressId	D
	FirstName	V
	MiddleName	V
	LastName	V
	Gender	C
	DateOfBirth	D
	EmailAddress	V
	HomePhone	V
	CreatedDate	D
	ModifiedDate	D
	PersonType	C
	Active	D

Address

PK	PersonId	DECIMAL(12)
	Address1	VARCHAR(255)
	Address2	VARCHAR(255)
	City	VARCHAR(255)
FK1	StateId	DECIMAL(12)
	PostalCode	VARCHAR(10)

Room

PK	RoomId	DECIMAL(12)
FK1	BuildingId	DECIMAL(12)
	Name	VARCHAR(100)
	RoomNumber	VARCHAR(10)
	Capacity	DECIMAL(5)
	Floor	DECIMAL(2)
	Note	VARCHAR(1000)
	Active	DECIMAL(1)

Event

PK	EventId	DECIMAL(12)
FK1	CampusId	DECIMAL(12)
	Name	VARCHAR(255)
	Description	VARCHAR(1000)
	RegistrationCheckInType	CHAR(1)
	Active	DECIMAL(1)

State

PK	StateiD	DECIMAL(12)
	StateName	VARCHAR(255)
	StateAbbrev	VARCHAR(2)

EventGroup

PK	EventGroupId	DECIMAL(12)
FK1	EventId	DECIMAL(12)
	Name	VARCHAR(255)
	Description	VARCHAR(1000)
	AgeStartYear	DECIMAL(2)
	AgeEndYear	DECIMAL(2)
	Active	DECIMAL(1)

EventSchedule

PK	EventScheduleId	DECIMAL(12)
FK1	EventId	DECIMAL(12)
	EventStartDate	DATETIME
	EventEndDate	DATETIME
	RegistrationStartDate	DATETIME
	RegistrationEndDate	DATETIME
	CreatedDate	DATETIME

Adult

PK, FK1	PersonId	DECIMAL(12)
	UserName	VARCHAR(64)
	EncryptedPassword	VARCHAR(20)
	MobilePhone	VARCHAR(10)
	MaritalStatus	CHAR(1)
	GovIssuedID	VARCHAR(20)
	TypeGovIssuedID	CHAR(1)
	IsVolunteer	DECIMAL(1)

ClassroomAssignment

PK	ClassroomId	DECIMAL(12)
FK1	EventScheduleId	DECIMAL(12)
FK2	EventGroupId	DECIMAL(12)
FK3	RoomId	DECIMAL(12)
	AssignmentDate	DATETIME
F4	AssignmentById	DECIMAL(12)
	Notes	VARCHAR(1000)

BackgroundCheckChange

PK	BackgroundChangeID	DECIMAL(12)
	OldBGChangeDate	DATETIME
	NewBGChangeDate	DATETIME
FK	PersonId	DECIMAL(12)
	ChangeDate	DATETIME

RegistrantAttendance

PK	RegistrantId	DECIMAL(12)
FK1	EventScheduleId	DECIMAL(12)
FK2	PersonId	DECIMAL(12)
FK3	ClassroomId	DECIMAL(12)
	PickupCode	VARCHAR(10)
	RegistrationDate	DATETIME
FK4	RegisteredById	DECIMAL(12)
	CheckInDate	DATETIME
FK5	CheckInById	DECIMAL(12)
	TypeCheckIn	CHAR(1)
	CheckOutDate	DATETIME
FK6	CheckOutById	DECIMAL(12)
	TypeCheckOut	CHAR(1)
	Notes	VARCHAR(1000)

Volunteer

PK, FK1	PersonId	DECIMAL(12)
	BackgroundCheckStatus	CHAR(1)
	BackgroundCheckDate	DATETIME
	VolunteerType	CHAR(1)
	Active	DECIMAL(1)

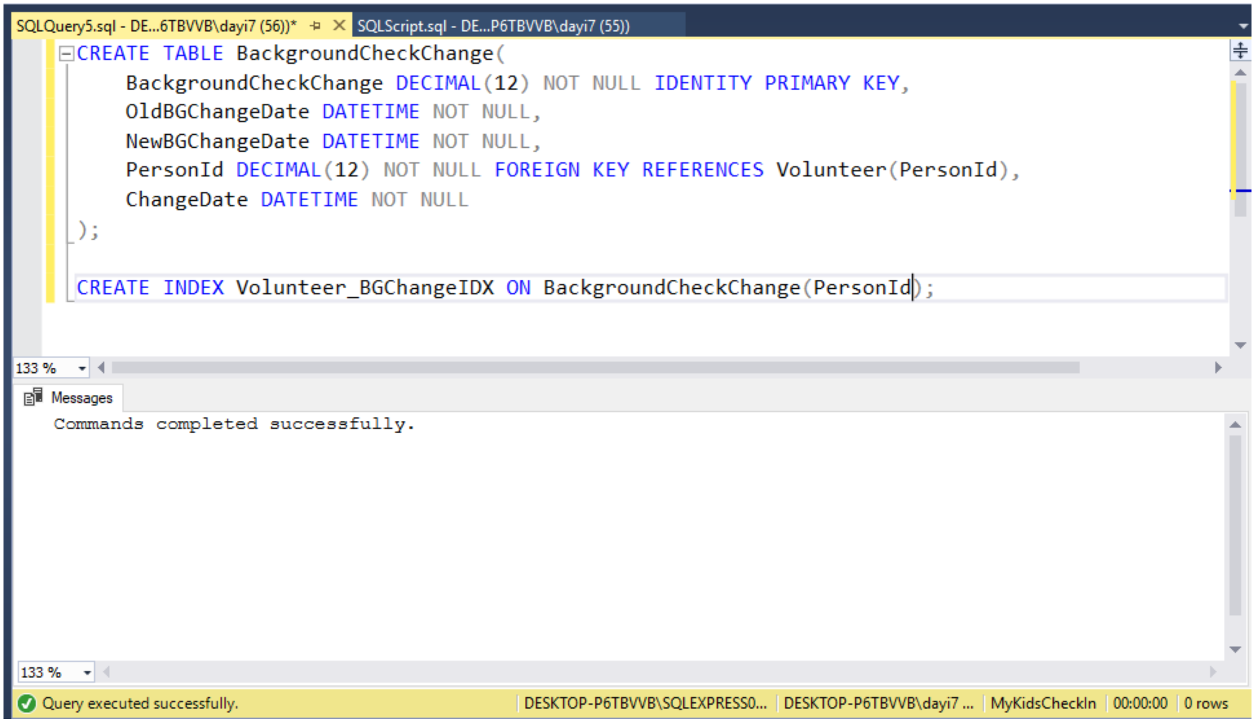
VolunteerAssignment

PK	VolunteerAssignmentId	DECIMAL(12)
FK1	PersonId	DECIMAL(12)
FK2	ClassroomId	DECIMAL(12)
	AssignmentDate	DATETIME
F3	AssignmentById	DECIMAL(12)

The BackgroundCheckChange entity is present and linked to Volunteer entity. Below are the attributes I added and why.

Attribute	Description
BackgroundCheckChangeId	This is the primary key of the history table. It is a DECIMAL(12) to allow many values.
OldBGChangeDate	This is the BackgroundCheckDate before the change. The data type mirrors the BackgroundCheckDate datatype in the Volunteer table.
NewBGChangeDate	This is the BackgrounCheckDate after the changes. The data type mirroes the BackgroundChackDate data type in the volunteer table.
PersonId	This is a foreign key to the Volunteer table, a reference to the Volunteer that had the change in the backgroundcheckdate.
ChangeDate	This is the date the nacground chec change occurred, with a DATETIME data type.

Here is a screenshot of my table creation, which has all of the same attributes and datatypes as indicated in the DBMS physical ERD.



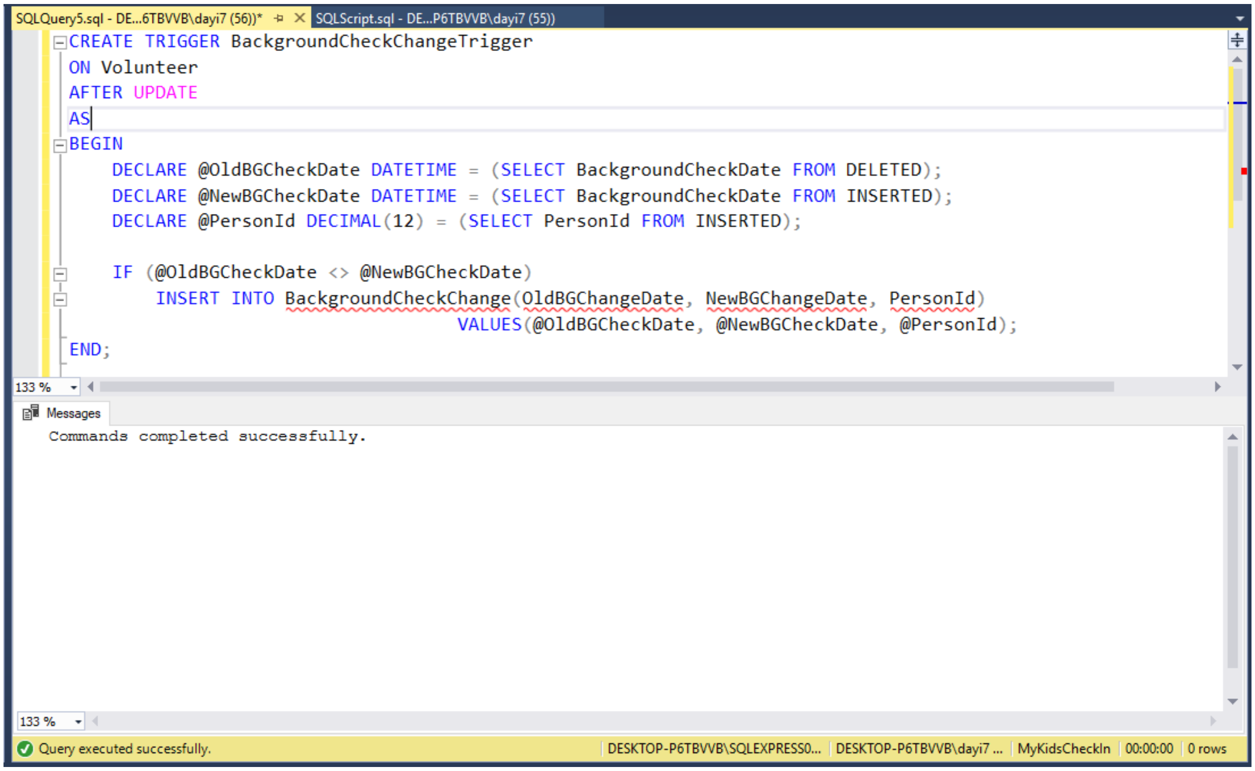
SQLQuery5.sql - DE...6TBVVB\dayi7 (56))*
SQLScript.sql - DE...P6TBVVB\dayi7 (55))
133 %
Messages
133 %

```
CREATE TABLE BackgroundCheckChange(
    BackgroundCheckChange DECIMAL(12) NOT NULL IDENTITY PRIMARY KEY,
    OldBGChangeDate DATETIME NOT NULL,
    NewBGChangeDate DATETIME NOT NULL,
    PersonId DECIMAL(12) NOT NULL FOREIGN KEY REFERENCES Volunteer(PersonId),
    ChangeDate DATETIME NOT NULL
);

CREATE INDEX Volunteer_BGChangeIDX ON BackgroundCheckChange(PersonId);
```

```
Commands completed successfully.
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

Here is a screenshot of my trigger creation which will maintain the BackgroundCheckChange table.


SQLQuery5.sql - DE...6TBVVB\dayi7 (56))*
SQLScript.sql - DE...P6TBVVB\dayi7 (55))
133 %
Messages
133 %

```
CREATE TRIGGER BackgroundCheckChangeTrigger
ON Volunteer
AFTER UPDATE
AS
BEGIN
    DECLARE @OldBGCheckDate DATETIME = (SELECT BackgroundCheckDate FROM DELETED);
    DECLARE @NewBGCheckDate DATETIME = (SELECT BackgroundCheckDate FROM INSERTED);
    DECLARE @PersonId DECIMAL(12) = (SELECT PersonId FROM INSERTED);

    IF (@OldBGCheckDate <> @NewBGCheckDate)
        INSERT INTO BackgroundCheckChange(OldBGChangeDate, NewBGChangeDate, PersonId)
                            VALUES(@OldBGCheckDate, @NewBGCheckDate, @PersonId);
END;
```

```
Commands completed successfully.
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

Code	Description
CREATE TRIGGER BackgroundCheckChangeTrigger ON Volunteer AFTER UPDATE	This starts the definition of the trigger and names it “BackgroundCheckChangeTrigger”. The trigger is linked to the Volunteer table and is executed after any update to that table.
AS BEGIN	This is the part of the syntax starting the trigger block.

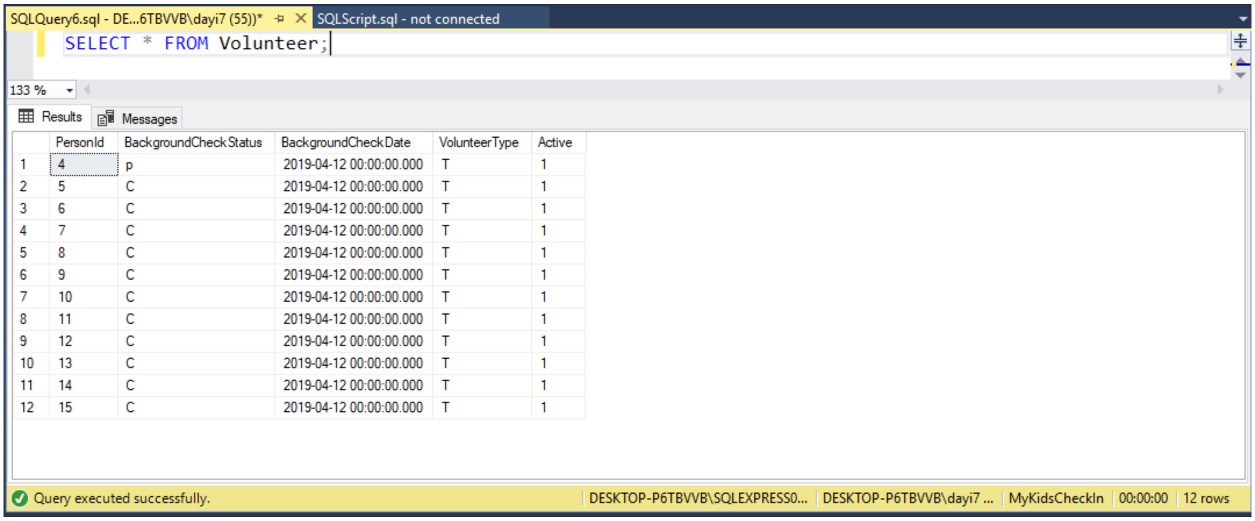
DECLARE @OldBGCheckDate DATETIME = (SELECT BackgroundCheckDate FROM DELETED);
DECLARE @NewBGCheckDate DATETIME = (SELECT BackgroundCheckDate FROM INSERTED);
DECLARE @PersonId DECIMAL(12) = (SELECT PersonId FROM INSERTED);
	This saves the old and new background check date referencing the DELETED and INSERTED pseudo tables, respectively.

IF (@OldBGCheckDate <> @NewBGCheckDate)	This check ensures action is only taken if the background check date has been updated.
INSERT INTO BackgroundCheckChange(OldBGChangeDate, NewBGChangeDate, PersonId)	This inserts the record into the BackgroundCheckChange table. The primary ket is set by the IDENTITY. The old and new BGCheckdates are used from the variables.

`VALUES(@OldBGCheckDate,`
`@NewBGCheckDate, @PersonId);`
	The PersonId is obtained from the INSERTED pseudo table. The ChangeDate field has a default constraint that is assigned the getdate() value.

`END;`	This ends the trigger definition.

First, I select all the records from my Volunteer table.

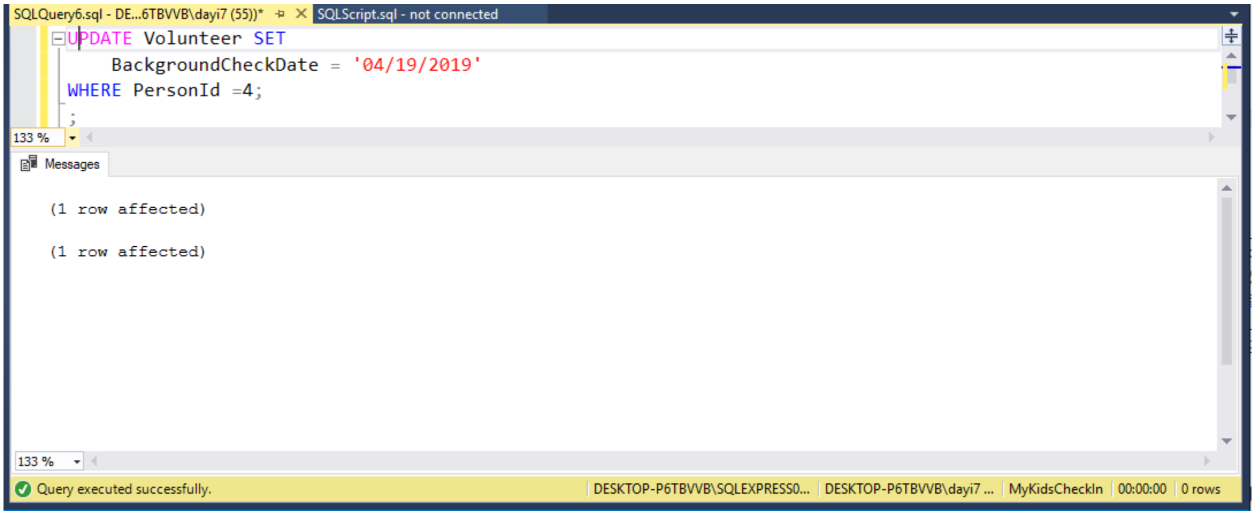

SQLQuery6.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - not connected
133 %
Results
Messages

```
SELECT * FROM Volunteer;
```

	PersonId	BackgroundCheckStatus	BackgroundCheckDate	VolunteerType	Active
1	4	p	2019-04-12 00:00:00.000	T	1
2	5	C	2019-04-12 00:00:00.000	T	1
3	6	C	2019-04-12 00:00:00.000	T	1
4	7	C	2019-04-12 00:00:00.000	T	1
5	8	C	2019-04-12 00:00:00.000	T	1
6	9	C	2019-04-12 00:00:00.000	T	1
7	10	C	2019-04-12 00:00:00.000	T	1
8	11	C	2019-04-12 00:00:00.000	T	1
9	12	C	2019-04-12 00:00:00.000	T	1
10	13	C	2019-04-12 00:00:00.000	T	1
11	14	C	2019-04-12 00:00:00.000	T	1
12	15	C	2019-04-12 00:00:00.000	T	1

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 12 rows

Next, I will update the BackgroundCheckDate of the PersonId 4.

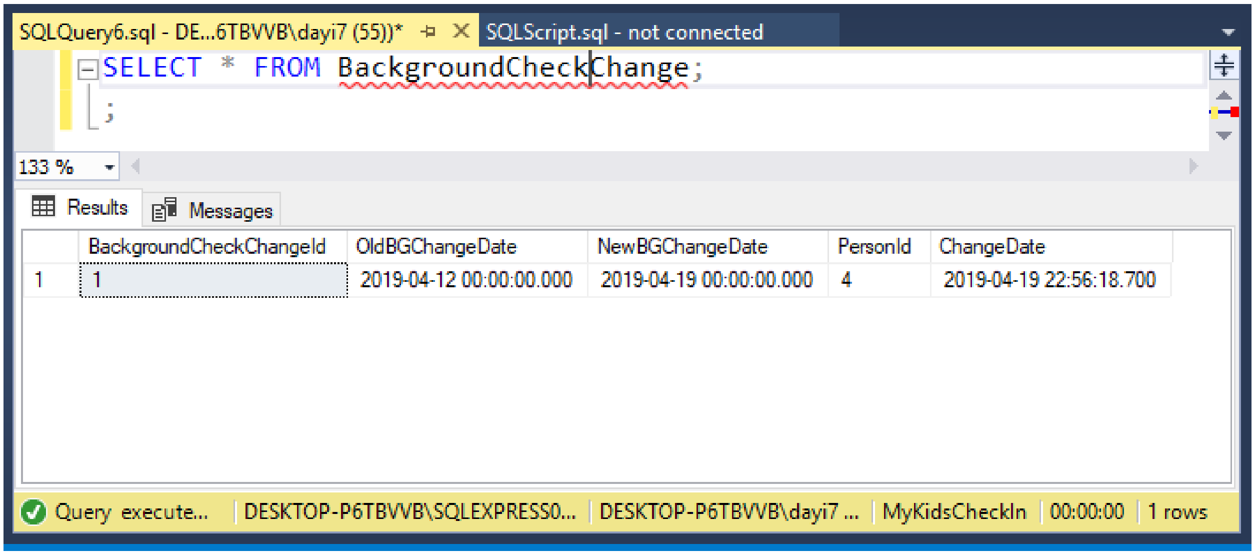

SQLQuery6.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - not connected
133 %
Messages
133 %

```
UPDATE Volunteer SET
    BackgroundCheckDate = '04/19/2019'
WHERE PersonId =4;
;
```

```
(1 row affected)

(1 row affected)
```

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 0 rows

Last, I verify that the BackgroundCheckChange table has a record for tha change done to the Volunteer table.



SQLQuery6.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - not connected
133 %
Results
Messages

```
SELECT * FROM BackgroundCheckChange;
```

	BackgroundCheckChangeId	OldBGChangeDate	NewBGChangeDate	PersonId	ChangeDate
1	1	2019-04-12 00:00:00.000	2019-04-19 00:00:00.000	4	2019-04-19 22:56:18.700

```
Query execute... | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 1 rows
```

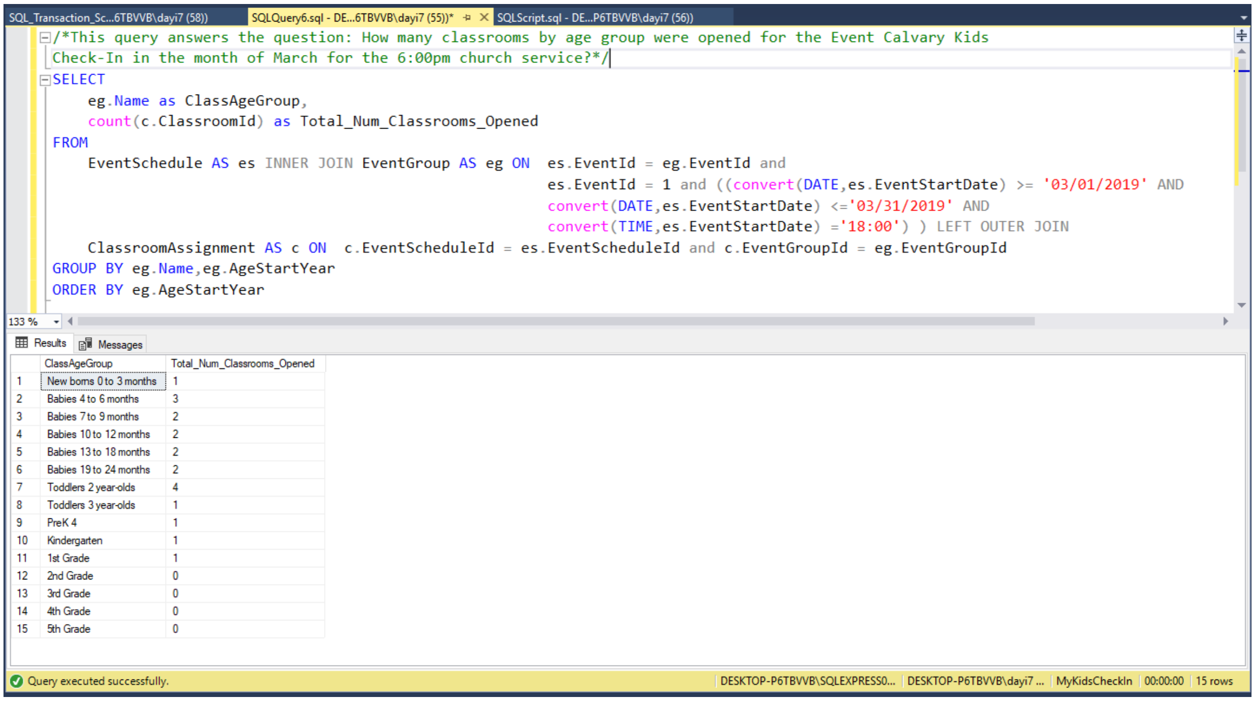
MyKids-CheckIn Question and Query

Question #1

Here is a question useful for the Childrens Minsitry: How many classrooms by age group were opened for the Event Calvary Kids Check-In in the month of March of 2019 for the 6:00pm church service?

First, I explain why this question is useful.

The answer can be used to determine how many Volunteers we need to recruit for certain age groups on a specific service. This will help the team leaders be better prepared with the required # of volunteers. Here is a screenshot of the query I use.



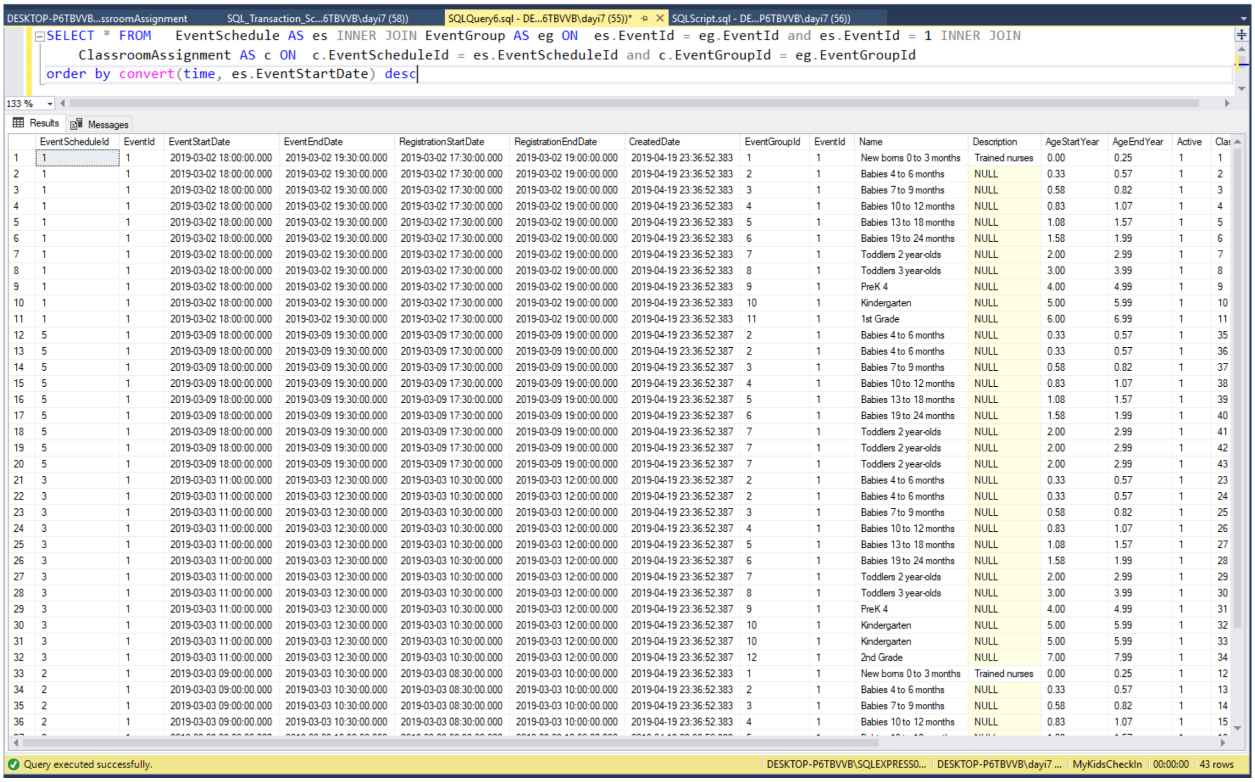
SQL_Transaction_Sc...6TBVVB\dayi7 (58))
SQLQuery6.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - DE...P6TBVVB\dayi7 (56))
133 %
Results
Messages

```sql
/*This query answers the question: How many classrooms by age group were opened for the Event Calvary Kids
Check-In in the month of March for the 6:00pm church service?*/
SELECT
    eg.Name as ClassAgeGroup,
    count(c.ClassroomId) as Total_Num_Classrooms_Opened
FROM
    EventSchedule AS es INNER JOIN EventGroup AS eg ON  es.EventId = eg.EventId and
                                                        es.EventId = 1 and ((convert(DATE,es.EventStartDate) >= '03/01/2019' AND
                                                        convert(DATE,es.EventStartDate) <='03/31/2019' AND
                                                        convert(TIME,es.EventStartDate) ='18:00') ) LEFT OUTER JOIN
    ClassroomAssignment AS c ON  c.EventScheduleId = es.EventScheduleId and c.EventGroupId = eg.EventGroupId
GROUP BY eg.Name,eg.AgeStartYear
ORDER BY eg.AgeStartYear
```

	ClassAgeGroup	Total_Num_Classrooms_Opened
1	New borns 0 to 3 months	1
2	Babies 4 to 6 months	3
3	Babies 7 to 9 months	2
4	Babies 10 to 12 months	2
5	Babies 13 to 18 months	2
6	Babies 19 to 24 months	2
7	Toddlers 2 year-olds	4
8	Toddlers 3 year-olds	1
9	PreK 4	1
10	Kindergarten	1
11	1st Grade	1
12	2nd Grade	0
13	3rd Grade	0
14	4th Grade	0
15	5th Grade	0

```
Query executed successfully.    DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 15 rows
```

To get the results, I join the EventSchedule to the EventGroup table, and limit the results to those with the EventId equal to 1 (that corresponds to the “Calvary Kids Check-In” event) and which EventStartDate was in the month of March. I then do a LEFT OUTER JOIN with the ClassroomAssignment table which has the records of my classes opened for a specific EventSchedule and EventGroup (age group). I use a LEFT OUTER JOIN to be able to get all EventGoups, even if there were no classes opened for a specific EventGroup. I order the results by AgeStartYear field from the EventGroup table. To help prove that the query is working properly, I show the full contents of the ClassroomAssignment and EventSchedule and EventGroup tables with a simple query:



DESKTOP-P6TBVVB...ssroomAssignment
SQL_Transaction_Sc...6TBVVB\dayi7 (58))
SQLQuery6.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - DE...P6TBVVB\dayi7 (56))
133 %
Results
Messages

```
SELECT * FROM   EventSchedule AS es INNER JOIN EventGroup AS eg ON  es.EventId = eg.EventId and es.EventId = 1 INNER JOIN
    ClassroomAssignment AS c ON  c.EventScheduleId = es.EventScheduleId and c.EventGroupId = eg.EventGroupId
order by convert(time, es.EventStartDate) desc
```

	EventScheduleId	EventId	EventStartDate	EventEndDate	RegistrationStartDate	RegistrationEndDate	CreatedDate	EventGroupId	EventId	Name	Description	AgeStartYear	AgeEndYear	Active	Clas
1	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	1	1	New borns 0 to 3 months	Trained nurses	0.00	0.25	1	1
2	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	2
3	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	3
4	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	4
5	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	5
6	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	6
7	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	7
8	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	8	1	Toddlers 3 year-olds	NULL	3.00	3.99	1	8
9	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	9	1	PreK 4	NULL	4.00	4.99	1	9
10	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	10	1	Kindergarten	NULL	5.00	5.99	1	10
11	1	1	2019-03-02 18:00:00.000	2019-03-02 19:30:00.000	2019-03-02 17:30:00.000	2019-03-02 19:00:00.000	2019-04-19 23:36:52.383	11	1	1st Grade	NULL	6.00	6.99	1	11
12	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	35
13	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	36
14	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	37
15	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	38
16	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	39
17	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	40
18	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	41
19	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	42
20	5	1	2019-03-09 18:00:00.000	2019-03-09 19:30:00.000	2019-03-09 17:30:00.000	2019-03-09 19:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	43
21	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	23
22	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	24
23	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	25
24	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	26
25	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	27
26	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	28
27	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	29
28	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	8	1	Toddlers 3 year-olds	NULL	3.00	3.99	1	30
29	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	9	1	PreK 4	NULL	4.00	4.99	1	31
30	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	10	1	Kindergarten	NULL	5.00	5.99	1	32
31	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	10	1	Kindergarten	NULL	5.00	5.99	1	33
32	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	12	1	2nd Grade	NULL	7.00	7.99	1	34
33	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	1	1	New borns 0 to 3 months	Trained nurses	0.00	0.25	1	12
34	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	13
35	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	14
36	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	15

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 43 rows

Upon inspection, you see that there are 43 ClassroomAssignment rows in my database. I ordererd the records by Time in descendant order so the ones at 6pm would be at the top for better visibility. There are only 20 classrooms assignments for the month of March at 6:00pm. If you add up the numbers of my Total_Num_Classrooms_Opened it will equal 20. So as is demonstrated, the query appears to be returning the correct results based upon the question.

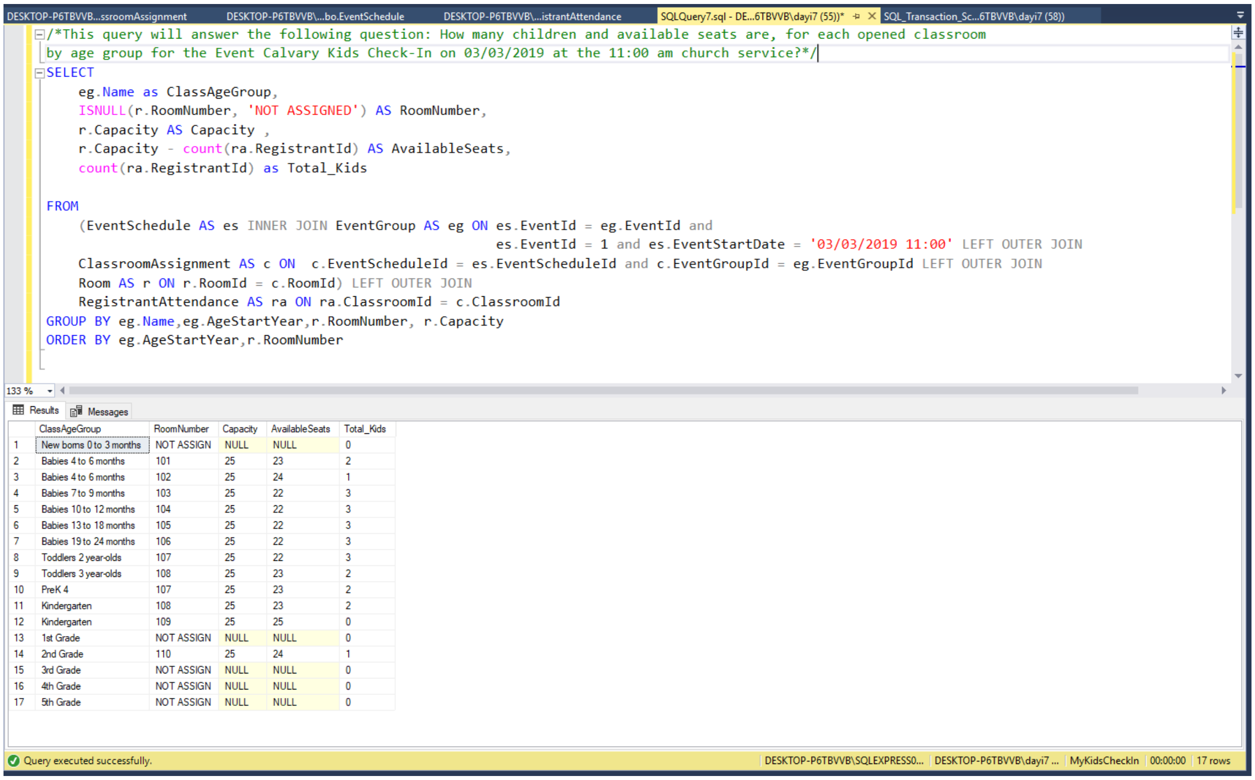
Question #2

Here is another question useful for the Childrens Minsitry: How many children and available seats are, for each opened classroom by age group for the Event Calvary Kids Check-In on 03/03/2019 at the 11:00 am church service?

First, I explain why this question is useful.

The answer can be used to determine what classrooms have been opened for a specific age group and also to see a total number of children that have checked in a classroom and also the available seats so far. This way the team leaders will know quickly which classrooms are getting filled up quickly so they can start preparing a new classroom for that specific age group.

Here is a screenshot of the query I use.



DESKTOP-P6TBVVB...ssroomAssignment
DESKTOP-P6TBVVB\...bo.EventSchedule
DESKTOP-P6TBVVB\...istrantAttendance
SQLQuery7.sql - DE...6TBVVB\dayi7 (55))*
SQL_Transaction_Sc...6TBVVB\dayi7 (58))

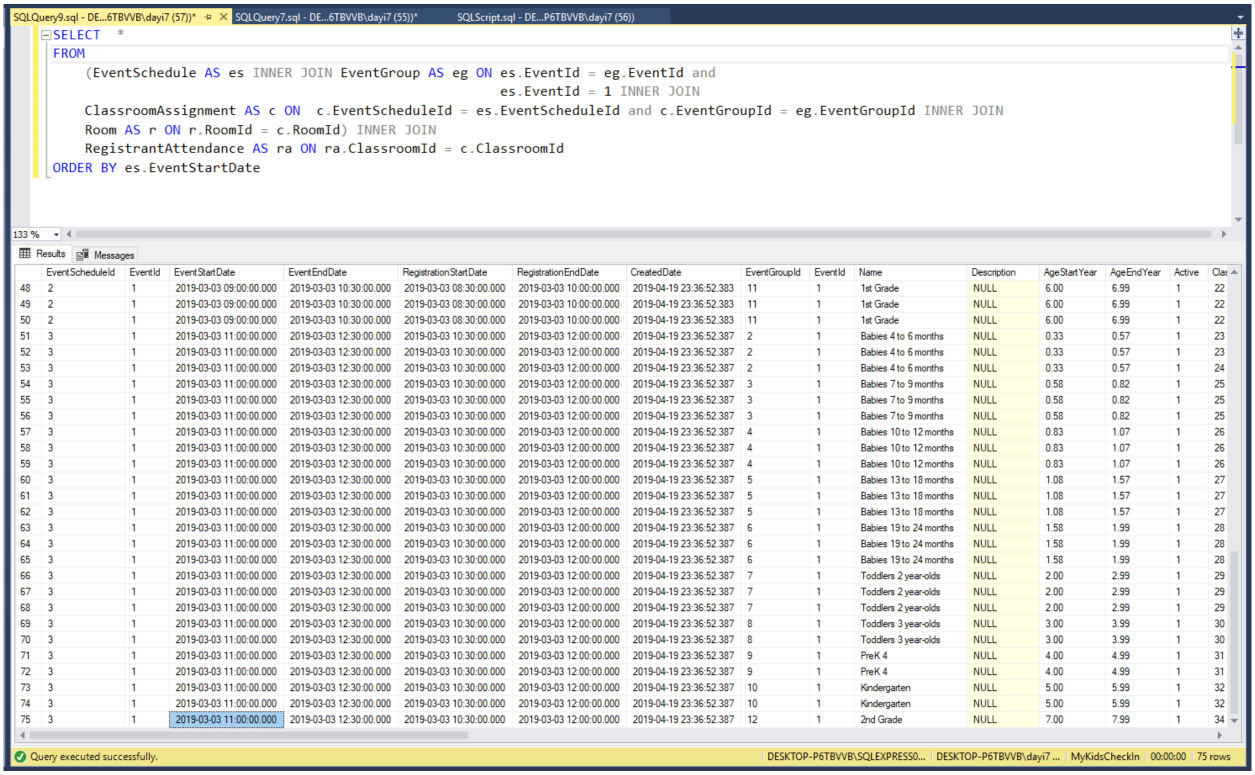
```sql
/*This query will answer the following question: How many children and available seats are, for each opened classroom
by age group for the Event Calvary Kids Check-In on 03/03/2019 at the 11:00 am church service?*/
SELECT
    eg.Name as ClassAgeGroup,
    ISNULL(r.RoomNumber, 'NOT ASSIGNED') AS RoomNumber,
    r.Capacity AS Capacity ,
    r.Capacity - count(ra.RegistrantId) AS AvailableSeats,
    count(ra.RegistrantId) as Total_Kids

FROM
    (EventSchedule AS es INNER JOIN EventGroup AS eg ON es.EventId = eg.EventId and
                                                        es.EventId = 1 and es.EventStartDate = '03/03/2019 11:00' LEFT OUTER JOIN
    ClassroomAssignment AS c ON  c.EventScheduleId = es.EventScheduleId and c.EventGroupId = eg.EventGroupId LEFT OUTER JOIN
    Room AS r ON r.RoomId = c.RoomId) LEFT OUTER JOIN
    RegistrantAttendance AS ra ON ra.ClassroomId = c.ClassroomId
GROUP BY eg.Name,eg.AgeStartYear,r.RoomNumber, r.Capacity
ORDER BY eg.AgeStartYear,r.RoomNumber
```

	ClassAgeGroup	RoomNumber	Capacity	AvailableSeats	Total_Kids
1	New borns 0 to 3 months	NOT ASSIGN	NULL	NULL	0
2	Babies 4 to 6 months	101	25	23	2
3	Babies 4 to 6 months	102	25	24	1
4	Babies 7 to 9 months	103	25	22	3
5	Babies 10 to 12 months	104	25	22	3
6	Babies 13 to 18 months	105	25	22	3
7	Babies 19 to 24 months	106	25	22	3
8	Toddlers 2 year-olds	107	25	22	3
9	Toddlers 3 year-olds	108	25	23	2
10	PreK 4	107	25	23	2
11	Kindergarten	108	25	23	2
12	Kindergarten	109	25	25	0
13	1st Grade	NOT ASSIGN	NULL	NULL	0
14	2nd Grade	110	25	24	1
15	3rd Grade	NOT ASSIGN	NULL	NULL	0
16	4th Grade	NOT ASSIGN	NULL	NULL	0
17	5th Grade	NOT ASSIGN	NULL	NULL	0

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 17 rows

To get the results I joined the EventSchedule, EventGroup, ClassroomAssignment, Room and RegistrantAttendance tables, and limit the results to those with the EventId equal to 1 (that corresponds to the “Calvary Kids Check-In” event) and which EventStartDate equals 03/03/2019 at 11:00 am. I then do a LEFT OUTER JOIN with the ClassroomAssignment table which has the records of my classes opened for a specific EventSchedule and EventGroup (age group). I use a LEFT OUTER JOIN to be able to get all EventGoups, even if there were no classes opened for a specific EventGroup. I then join with the room to be able to get the room number and finally I join with the RegstrantAttendance table which has the records of all the children registered in the event. I order the results by AgeStartYear field from the EventGroup table and RoomNumber from the Room table.

To help prove that the query is working properly, I first added more registrants to the RegistrantAttendance table and I show the full contents of the ClassroomAssignment and EventSchedule, EventGroup, Room and RegistrantAttendance tables with a simple query:


SQLQuery9.sql - DE...6TBVVB\dayi7 (57))*
SQLQuery7.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - DE...P6TBVVB\dayi7 (56))
133 %
Results
Messages

```sql
SELECT  *
FROM
    (EventSchedule AS es INNER JOIN EventGroup AS eg ON es.EventId = eg.EventId and
                                                        es.EventId = 1 INNER JOIN
    ClassroomAssignment AS c ON  c.EventScheduleId = es.EventScheduleId and c.EventGroupId = eg.EventGroupId INNER JOIN
    Room AS r ON r.RoomId = c.RoomId) INNER JOIN
    RegistrantAttendance AS ra ON ra.ClassroomId = c.ClassroomId
ORDER BY es.EventStartDate
```

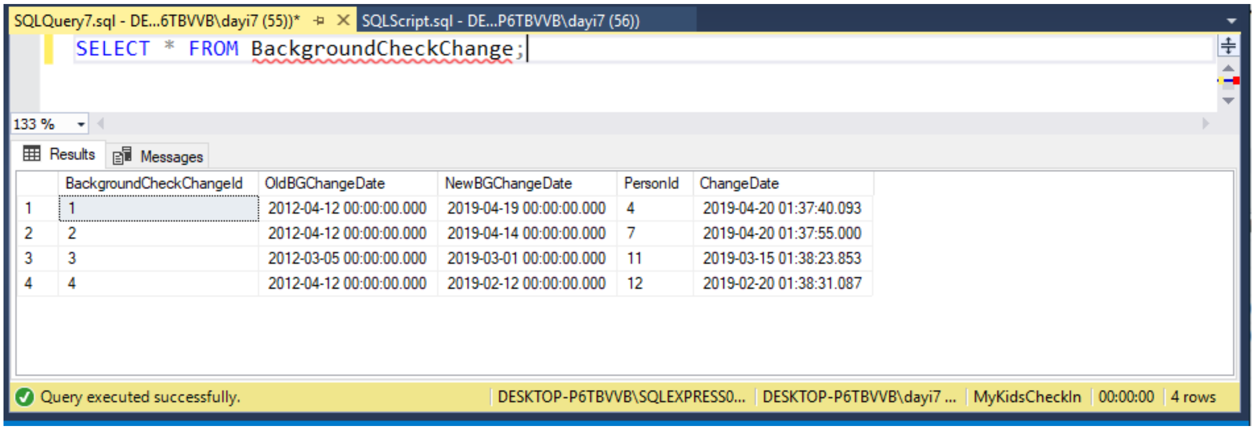
	EventScheduleId	EventId	EventStartDate	EventEndDate	RegistrationStartDate	RegistrationEndDate	CreatedDate	EventGroupId	EventId	Name	Description	AgeStartYear	AgeEndYear	Active	Clas
48	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	11	1	1st Grade	NULL	6.00	6.99	1	22
49	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	11	1	1st Grade	NULL	6.00	6.99	1	22
50	2	1	2019-03-03 09:00:00.000	2019-03-03 10:30:00.000	2019-03-03 08:30:00.000	2019-03-03 10:00:00.000	2019-04-19 23:36:52.383	11	1	1st Grade	NULL	6.00	6.99	1	22
51	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	23
52	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	23
53	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	2	1	Babies 4 to 6 months	NULL	0.33	0.57	1	24
54	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	25
55	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	25
56	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	3	1	Babies 7 to 9 months	NULL	0.58	0.82	1	25
57	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	26
58	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	26
59	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	4	1	Babies 10 to 12 months	NULL	0.83	1.07	1	26
60	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	27
61	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	27
62	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	5	1	Babies 13 to 18 months	NULL	1.08	1.57	1	27
63	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	28
64	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	28
65	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	6	1	Babies 19 to 24 months	NULL	1.58	1.99	1	28
66	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	29
67	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	29
68	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	7	1	Toddlers 2 year-olds	NULL	2.00	2.99	1	29
69	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	8	1	Toddlers 3 year-olds	NULL	3.00	3.99	1	30
70	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	8	1	Toddlers 3 year-olds	NULL	3.00	3.99	1	30
71	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	9	1	PreK 4	NULL	4.00	4.99	1	31
72	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	9	1	PreK 4	NULL	4.00	4.99	1	31
73	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	10	1	Kindergarten	NULL	5.00	5.99	1	32
74	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	10	1	Kindergarten	NULL	5.00	5.99	1	32
75	3	1	2019-03-03 11:00:00.000	2019-03-03 12:30:00.000	2019-03-03 10:30:00.000	2019-03-03 12:00:00.000	2019-04-19 23:36:52.387	12	1	2nd Grade	NULL	7.00	7.99	1	34

```
Query executed successfully.    DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 75 rows
```

Upon inspection, you see that there are 75 ClassroomAssignment rows in my database. I ordered the records by EventStartDate. There are only 25 records for the EventStartDate 03/03/2019 at 11:00 am. If you add up the numbers of my Total_Kids it will equal 25. So as is demonstrated, the query appears to be returning the correct results based upon the question.

Question #3

A useful question from the BackgroundCheckChange history table is: How many volunteer's background check date change for specific period by the month? Since background checks have to be repeated every 7 years, this will help plan ahead of time. In order to provide more data for this, I changed some backgroundcheckDates. Here is what the BackgroundCheckChange table looks like after these changes.

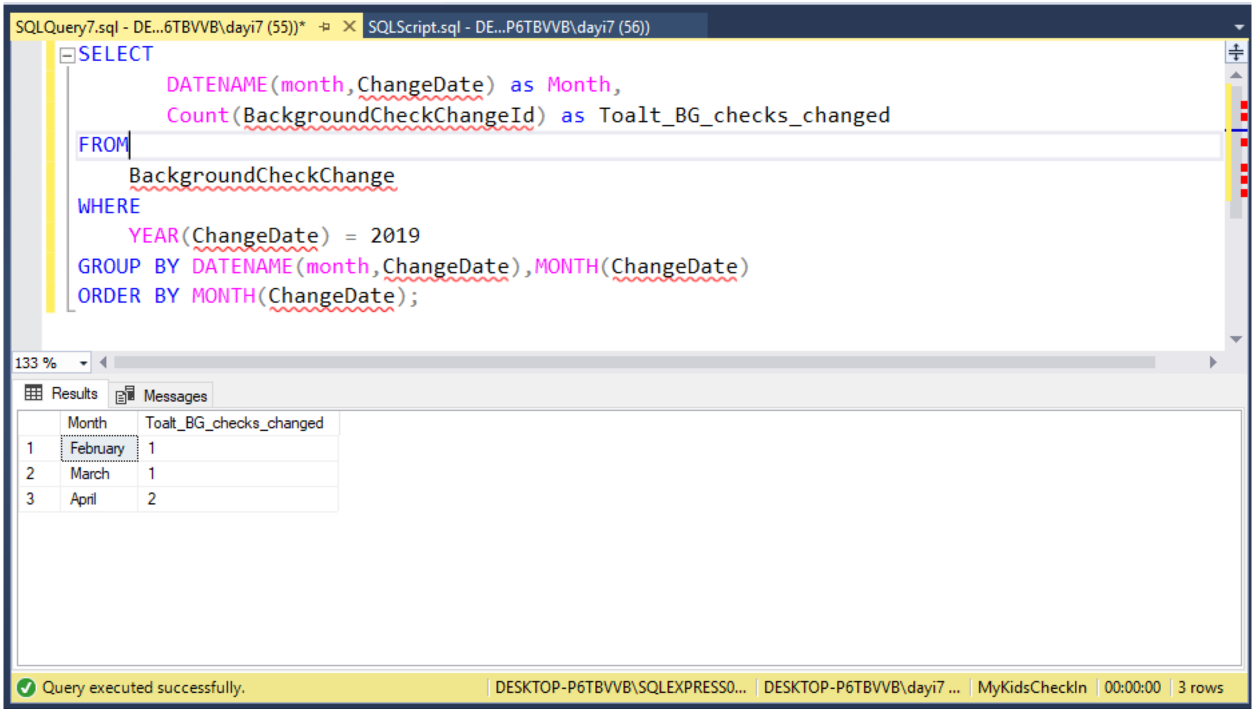

SQLQuery7.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - DE...P6TBVVB\dayi7 (56))
133 %
Results
Messages

```sql
SELECT * FROM BackgroundCheckChange;
```

	BackgroundCheckChangeId	OldBGChangeDate	NewBGChangeDate	PersonId	ChangeDate
1	1	2012-04-12 00:00:00.000	2019-04-19 00:00:00.000	4	2019-04-20 01:37:40.093
2	2	2012-04-12 00:00:00.000	2019-04-14 00:00:00.000	7	2019-04-20 01:37:55.000
3	3	2012-03-05 00:00:00.000	2019-03-01 00:00:00.000	11	2019-03-15 01:38:23.853
4	4	2012-04-12 00:00:00.000	2019-02-12 00:00:00.000	12	2019-02-20 01:38:31.087

```
Query executed successfully.    DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 4 rows
```

There are a total of 4 records but 2 were changed in the month of April, 1 in March and 1 in February. Here is a screen shot of my query and the results:


SQLQuery7.sql - DE...6TBVVB\dayi7 (55))*
SQLScript.sql - DE...P6TBVVB\dayi7 (56))
133 %
Results
Messages

```
SELECT
       DATENAME(month,ChangeDate) as Month,
       Count(BackgroundCheckChangeId) as Toalt_BG_checks_changed
FROM
    BackgroundCheckChange
WHERE
    YEAR(ChangeDate) = 2019
GROUP BY DATENAME(month,ChangeDate),MONTH(ChangeDate)
ORDER BY MONTH(ChangeDate);
```

	Month	Toalt_BG_checks_changed
1	February	1
2	March	1
3	April	2

Query executed successfully. | DESKTOP-P6TBVVB\SQLEXPRESS0... | DESKTOP-P6TBVVB\dayi7 ... | MyKidsCheckIn | 00:00:00 | 3 rows

I use the DATENAME function to display the month name and I use the Month function to use it in my ORDER BY. Since I am running this query for the year 2019, I use the function YEAR to specify just the year 2019 in the where clause.

MyKids-CheckIn Summary and Reflection

My database is for an app that will automate the parent/child registration process and the check-in and check-out process for the Saturday and Sunday services at my church. When I talked to the people involved in this process, I noticed that the church would want this same or similar check-in process for all events. They have different types of events, for the different types of ministries the church offers. After thinking through, I realized this may become a big database with a lot of business rules involved, which is why I have narrow it down to the Children's ministry check-in process.

This week I was able to identify the structural database rules, based on the use cases; with these rules it was so easy to identify all the entities involved and the relationships between them.
I made minor changes to the use cases so that it would be clearer to recognize the entities. I was able to identify the following entities: Campus, Parent, Child, Event, Event Group, Classroom, Volunteer and Check-in, as well as relationships between them.

This exercise helped me understand the importance of well-defined business rules in an organization, which is an important part when designing a database.

Week 3 Summary and Reflection Update

This week, I have done many changes to my project. I changed 2 uses cases to be able to reflect the Specialization-Generalization Hierarchy. I also made changes to the names of a few entities to better describe its purpose. I change the relation I had of EventGroup/Classroom to Event/Classroom. I also created a bridge entity to eliminate the M:N relationship between Event/Classroom.

After making these changes, creating the DBMS Physical EERD was not difficult. I can honestly say that I feel much better this week with how my database design looks. At the beginning of the course I was a little bit worried that this project was going to have to many tables and it was going to be too complex, but after this week, I am excited that this will not be my case.

Week 4 Summary and Reflection Update

I spent more time working on this iteration compared to the last 3. While identifying all the possible attributes that my entities have, I realized I had new entities to create and others that were not necessary.   Although it was a lot of work, I am happy with what I have accomplished so far, and to finally see the database I had in paper, implemented in SQL server.

Week 5 Summary and Reflection Update

It’s incredible how I can finally see my database fully functioning. It really took a lot of time putting together the scripts and populating my tables with somewhat real data to be able to generate correct results with my queries. Although it was a lot of work, I can say I have really enjoyed this project. While working on it I have come up with more ideas and changes that hopefully I will be able to complete as a personal goal.