Marlboro Swim Club Day Camp

Database Design Specifications

April 25, 2014

Christopher Belmonte

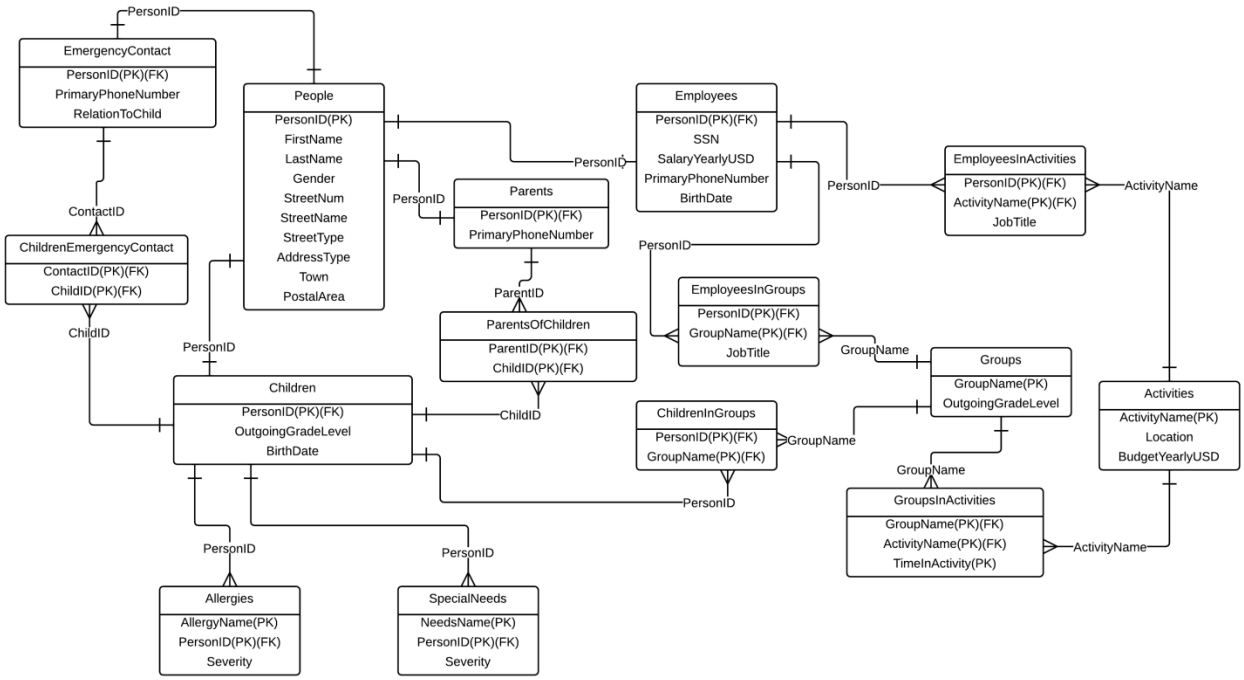
TABLE OF CONTENTS

Executive Summary

The Marlboro Swim Club is a private pool recreational area with a baby pool, a child's pool, an Olympic pool, and a dive tank. This is a place for families who do not own their own pools to come hang out and enjoy their day during the summer. They also have a day camp that runs Monday to Friday. Currently, everything is running on paper system. While this has worked in the past, it is time to upgrade and speed up the process of everything.

This document is the design of a database for the Marlboro Swim Club. Using this database, we can keep track of everyone involved in the camp whether that is parents, children, employees, or emergency contacts. We can also find out where everyone inside the camp is, who has special needs or allergies, and more. This database will be much faster and more efficient than the current paper system that is used now. The users of this database will include all employees of the camp.

ENTITY RELATIONSHIP DIAGRAM



Link if it is too hard to read: http://www.lucidchart.com/invitations/accept/535869b0-c488-485d-8cee-6bcf0a00d8cc

Types

Gender

- As a check constraint, I made a type called Gender. The only valid inputs for gender are ‘M’, ‘F’, or ‘O’ for male, female, and other respectively.

```
CREATE TYPE GenderMFO as ENUM('M', 'F', 'O');
```

Severity

- As with Gender, I made a check constraint for the SpecialNeeds and Allergies table.

```
CREATE TYPE Severity as ENUM('Low', 'Medium', 'High', 'Extreme');
```

TABLES

People

- The people table holds anyone that is an employee, camper, or parent. It is a strong entity and its attributes are inherited by the tables Employees, Children, and Parents.

```
CREATE TABLE People(
PersonID          integer not null,
FirstName         varchar(50) not null,
LastName          varchar(50) not null,
Gender            GenderMFO not null,
StreetNum         integer not null,
StreetName        varchar(50) not null,
StreetType        varchar(50) not null,
AddressType       varchar(50) not null,
Town              varchar(50) not null,
PostalArea  integer not null,
PRIMARY KEY(PersonID)
);
```

Functional Dependencies:

- PersonID → FirstName, LastName, Gender, StreetNum, StreetName, StreetType, AddressType, Town, PostalArea

This sample data is for the first 10 employees. The zip codes all have 0s in front of them but the database filters them out.

	personid integer	firstname character varying(50)	lastname character varying(50)	gender gendermfo	streetnum integer	streetname character varying(50)	streettype character varying(50)	addresstype character varying(50)	town character varying(50)	postalarea integer
1	1	Christopher	Belmonte	M	147	Rolling Hill	Drive	Home	Morganville	7751
2	2	Ralph	Macchio	M	2	Karate Kid	Street	Office	Old Bridge	8857
3	3	Justin	Bieber	F	80	Purple	Lane	Aparment	Freehold	7728
4	4	Thad	Castle	M	1	Blue Mountain Stat	Avenue	Home	Freehold	7728
5	5	Pusha	T	M	10	Rapper	Lane	Home	Morganville	7751
6	6	Killer	Mike	M	3	Run The Jewels	Way	Home	Old Bridge	8857
7	7	El	P	M	4	Run The Jewels	Way	Home	Old Bridge	8857
8	8	Hot	Rod	O	999	Main	Street	Home	Morganville	7751
9	9	Geddy	Lee	M	2112	Rush	Avenue	Home	Marlboro	7746
10	10	Sonny	Moore	M	2	OWSLA	Lane	Home	Marlboro	7746

Children

- The children table takes attritbutes from the People table and ChildrenOfGroups, ParentsOfChildren, EmergencyContact, Allergies, and SpecialNeeds take attributes from it.

```
CREATE TABLE Children(
PersonID        integer not null references People(PersonID) ON
                DELETE CASCADE,
OutgoingGradeLevel  varchar(4) not null,
BirthDate       date not null,
PRIMARY KEY(PersonID)
);
```

Functional Dependencies:

- PersonID → OutGoingGradeLevel, BirthDate

	firstname character varying(50)	lastname character varying(50)	outgoinggradelevel integer	birthdate date
1	Hot	Rod	4	2004-01-01
2	Geddy	Lee	4	2004-08-08
3	Sonny	Moore	2	2006-12-01

Allergies

- This table shows all allergies that children have. This is incredibly important information. This table takes an attribute from the Children table.

```
CREATE TABLE Allergies(
AllergyName varchar(20),
PersonID    integer not null references Children(PersonID) ON DELETE CASCADE,
Severity    severity,
PRIMARY KEY(AllergyName, PersonID)
);
```

Functional Dependencies:

- AllergyName, PersonID → Severity

	allergyname character varying(20)	personid integer	severity severity
1	Peanuts	8	Extreme
2	Penicillin	10	Low

SpecialNeeds

- This table shows any special needs that children have. This table also takes an attribute from the Children table.

```
CREATE TABLE SpecialNeeds(
NeedsName   varchar(20),
PersonID    integer not null references Children(PersonID) ON DELETE CASCADE,
Severity    severity,
PRIMARY KEY(NeedsName, PersonID)
);
```

Functional Dependencies:

- NeedsName, PersonID → Severity

	needsname character varying(20)	personid integer	severity severity
1	ADHD	9	Medium

EmergencyContact

- This table shows all emergency contacts. This table takes an attribute from the Children table.

```
CREATE TABLE EmergencyContact(
PersonID                integer not null references People(PersonID) ON
                        DELETE CASCADE,
PrimaryNumberofContact  varchar(12),
RelationToChild         varchar(20),
PRIMARY KEY(PersonID)
);
```

Functional Dependencies:

- PersonID → PrimaryNumberOfContact, RelationToChild

	personid integer	primarynumberofcontact character varying(12)	relationtochild character varying(20)
1	18	5557778888	Aunt
2	17	8459321801	Uncle
3	16	3247850191	Cousin

Parents

- This table lists all of the parents of children. This table takes attributes from People and gives attributes to ParentsOfChildren

```
CREATE TABLE Parents(
PersonID           integer not null references People(PersonID) ON DELETE
                   CASCADE,
PrimaryPhoneNumber varchar(12),
PRIMARY KEY(PersonID)
);
```

Functional Dependencies:

- PersonID → PrimaryPhoneNumber

	personid integer	primaryphonenumber character varying(12)
1	5	1112223333
2	6	4848484848
3	7	8008675309

Employees

- This table lists any employees that work for the camp. This table takes attributes from People and gives attributes to EmployeesInActivities and EmployeesInGroups

```
CREATE TABLE Employees(
PersonID                integer not null references People(PersonID) ON
                        DELETE CASCADE,
SSN                     varchar(8) not null,
SalaryYearlyUSD         integer not null,
PrimaryPhoneNumber      varchar(12),
BirthDate               date not null,
PRIMARY KEY(PersonID)
);
```

Functional Dependencies:

- EmployeeID → SSN, SalaryYearlyUSD, PrimaryPhoneNumber, BirthDate

	personid integer	ssn integer	salaryyearlyusd integer	primaryphonenumber character varying(12)	birthdate date
1	1	125327219	1100	7328891010	1994-08-08
2	2	801378539	890	7321938102	1992-02-13
3	3	123456789	950	9086121001	1997-05-21
4	4	654938223	1400	9087432012	1996-02-29

Groups

- Each group has students and employees in them. This table will show all of the groups. It is a strong entity and gives attributes to EmployeesInGroups, ChildrenInGroups, and GroupsInActivities.

```
CREATE TABLE Groups(
GroupName           varchar(20) not null,
OutGoingGradeLevel  varchar(4),
PRIMARY KEY(GroupName)
);
```

Functional Dependencies:

- GroupName → OutgoingGradeLevel

	groupname character varying(20)	outgoinggradelevel character varying(4)
1	Toy Story	4
2	Kung Fu Panda	2
3	High School Musica	PreK

Activities

- Every group has activities to do throughout the day. This table shows all of the activities. It is a strong entity and gives attributes to GroupsInActivities.

```
CREATE TABLE Activities(
ActivityName      varchar(20) not null,
Location          varchar(20),
BudgetYearlyUSD   integer,
PRIMARY KEY(ActivityName)
);
```

Functional Dependencies:

- ActivityName → Location, BudgetYearlyUSD

	activityname character varying(20)	location character varying(20)	budgetyearlyusd integer
1	Art	Pavillion	2000
2	Karate/Dance	Pavillion	5000
3	Music	Tree	1000
4	Playground	Playground	500
5	Swim	Pools	500
6	Snack	Tents	800
7	Sports	Grassy Knoll	1000

EmployeesInGroups

- Employees need groups to belong to. This is a table to connect Employees and Groups through a many-to-many relationship. It takes attributes from both Employees and Groups.

```
CREATE TABLE EmployeesInGroups(
PersonID    integer not null references Employees(PersonID) ON
            DELETE CASCADE,
GroupName   varchar(20) not null references Groups(GroupName) ON
            DELETE CASCADE,
JobTitle    varchar(10),
PRIMARY KEY(PersonID, GroupName)
);
```

Functional Dependencies:

- PersonID, GroupName → Job Title

	personid integer	groupname character varying(20)	jobtitle character varying(30)
1	1	Toy Story	Supervisor
2	2	Toy Story	Counselor
3	3	Kung Fu Panda	Assistant Supervisor
4	4	Kung Fu Panda	Counselor

EmloyeesInActivities

- Employees that aren't in groups are in activities. This table connects Employees and Activities through a many-to-many relationship. It takes attributes from both Employees and Activities.

```
CREATE TABLE EmployeesInActivities(
PersonID          integer not null references Employees(PersonID) ON
                  DELETE CASCADE,
ActivityName      varchar(20) not null references Activities(ActivityName) ON
                  DELETE CASCADE,
JobTitle          varchar(30),
PRIMARY KEY(PersonID, ActivityName)
);
```

Functional Dependencies:

- PersonID, ActivityName → JobTitle

Sample data next page →

	personid integer	activityname character varying(20)	jobtitle character varying(30)
1	11	Art	Supervisor
2	12	Karate/Dance	Supervisor
3	13	Music	Supervisor
4	14	Sports	Assistant Supervisor
5	15	Sports	Supervisor

GroupsInActivities

- Groups need to be activities from the time the camp opens at 11:00 till closing time at 4:00. This table is a many-to-many relationship between Groups and Activities and also shows what time the groups are in which area. It takes attributes from both Groups and Activities.

```
CREATE TABLE GroupsInActivities(
GroupName          varchar(20) not null references Groups(GroupName) ON
                   DELETE CASCADE,
ActivityName       varchar(20) not null references Activities(ActivityName) ON
                   DELETE CASCADE,
TimeInActivity     time not null,
PRIMARY KEY(GroupName, ActivityName, TimeInActivity)
);
```

Functional Dependencies:

- GroupName, ActivityName → TimeInActivity

This test data shows where groups are at 13:30 or rather 1:30 PM

	groupname character varying(20)	activityname character varying(20)	timeinactivity time without time zone
1	Toy Story	Art	13:30:00
2	Kung Fu Panda	Karate/Dance	13:30:00

ChildrenInGroups

- ChildrenInGroups shows what children are in what groups. It is a many-to-many relationship between the Children and Groups table. Each kid will end up in a group. This table takes attributes from Children and Groups.

```
CREATE TABLE ChildrenInGroups(
PersonID     integer not null references Children(PersonID) ON DELETE CASCADE,
GroupName    varchar(20) not null references Groups(GroupName) ON DELETE
             CASCADE,
PRIMARY KEY(PersonID, GroupName)
);
```

Functional Dependencies & Test Data next page →

Functional Dependencies:

- PersonID, GroupName →

	personid integer	groupname character varying(20)
1	8	Toy Story
2	9	Toy Story
3	10	Kung Fu Panda

ChildrenEmergencyContact

- This table links the emergency contacts of children to the children. This is a many-to-many relationship between EmergencyContacts and Children because many children can have many emergency contacts and vice versa. Through a query, we will be able to get the names of the contact and the children they are emergency contacts for.

```
CREATE TABLE ChildrenEmergencyContacts(
ContactID          integer not null references EmergencyContact(PersonID) ON
                   DELETE CASCADE,
ChildID            integer not null references Children(PersonID) ON
                   DELETE CASCADE,
PRIMARY KEY(ContactID, ChildID)
);
```

Functional Dependencies:

- ContactID, ChildID →

Sample data next page →

	contactid integer	childid integer
1	16	8
2	17	9
3	18	10

Views

ToyStorySchedule

- Ideally we would make a view for each group for their schedule. This is what that schedule would look like for the group Toy Story.

```
CREATE OR REPLACE VIEW ToyStorySchedule AS
SELECT GroupName, ActivityName, TimeInActivity
FROM GroupsInActivities
WHERE GroupName = 'Toy Story'
ORDER BY TimeInActivity;
```

	groupname character varying(20)	activityname character varying(20)	timeinactivity time without time zone
1	Toy Story	Playground	11:00:00
2	Toy Story	Swim	11:30:00
3	Toy Story	Lunch	12:00:00
4	Toy Story	Music	12:30:00
5	Toy Story	Sports	13:00:00
6	Toy Story	Art	13:30:00
7	Toy Story	Karate/Dance	14:00:00
8	Toy Story	Snack	14:30:00
9	Toy Story	Swim	15:00:00

EmployeesNamesInGroups

- The name of employees and what groups they are in is very important. This is a view to get that information quickly.

```
CREATE OR REPLACE VIEW EmployeesNamesInGroups AS
SELECT DISTINCT p.FirstName, p.LastName, ge.GroupName
FROM People p,
     EmployeesInGroups ge
WHERE p.PersonID = ge.PersonID
ORDER BY ge.GroupName
```

	firstname character varying(50)	lastname character varying(50)	groupname character varying(20)
1	Justin	Bieber	Kung Fu Panda
2	Thad	Castle	Kung Fu Panda
3	Christopher	Belmonte	Toy Story
4	Ralph	Macchio	Toy Story

EmployeesOfSpecialNeeds

- This view will quickly get the names of all employees that watch children with special needs. The query was way too large to fit comfortably on the page so the query itself will be included in the image.

```
CREATE OR REPLACE VIEW EmployeesOfSpecialNeeds AS
SELECT p.FirstName, p.LastName
FROM People p
WHERE PersonID in (SELECT PersonID
                   FROM Employees
                   WHERE PersonID in (SELECT PersonID
                                      FROM EmployeesInGroups
                                      WHERE GroupName in (SELECT GroupName
                                                          FROM ChildrenInGroups
                                                          WHERE PersonID in (SELECT PersonID
                                                                             FROM Children
                                                                             WHERE PersonID in (SELECT PersonID
                                                                                                FROM SpecialNeeds)
                                                                             )
                                                          )
                                      )
);
```

	firstname character varying(50)	lastname character varying(50)
1	Christopher	Belmonte
2	Ralph	Macchio

Reports and Their Queries

- This query shows the emergency contact(s) for the child with PersonID number 9. I will also show the query we would use if we didn't know the child's ID number and only his name. In the program, there would be a dropdown or textbox to input this information and the program would call for these queries.

```
--with his ID--
SELECT distinct p.FirstName, p.LastName, ec.PrimaryPhoneNumber
FROM People p, EmergencyContact ec
WHERE p.PersonID in (SELECT PersonID
                     FROM EmergencyContact
                     WHERE PersonID in (SELECT ContactID
                                        FROM ChildrenEmergencyContacts
                                        WHERE ChildID = '9'))
AND p.PersonID = ec.PersonID;

--with his name--
SELECT p.FirstName, p.LastName, ec.PrimaryPhoneNumber
FROM People p, EmergencyContact ec
WHERE p.PersonID in (SELECT PersonID
                     FROM EmergencyContact
                     WHERE PersonID in (SELECT ContactID
                                        FROM ChildrenEmergencyContacts
                                        WHERE ChildID in (SELECT PersonID
                                                          FROM Children
                                                          WHERE PersonID in (SELECT PersonID
                                                                             FROM People
                                                                             WHERE FirstName = 'Geddy'
                                                                             AND LastName = 'Lee'))))
AND p.PersonID = ec.PersonID;
```

	firstname character varying(50)	lastname character varying(50)	primaryphonenumber character varying(12)
1	Frank	Reynolds	8459321801

- This will show the parent(s) of a specific child. Again I will show the query if we only knew his name and not the ID. The query with the name was too long so it will be in an image.

```
--Using his ID--
SELECT p.FirstName, p.LastName
FROM People p
WHERE p.PersonID in (SELECT PersonID
                     FROM Parents
                     WHERE PersonID in (SELECT ParentID
                                        FROM ParentsOfChildren
                                        WHERE ChildID in (SELECT PersonID
                                                          FROM Children
                                                          WHERE PersonID = 9)
                                       )
                    );
```

```
SELECT p.FirstName, p.LastName
FROM People p
WHERE p.PersonID in (SELECT PersonID
                     FROM Parents
                     WHERE PersonID in (SELECT ParentID
                                        FROM ParentsOfChildren
                                        WHERE ChildID in (SELECT PersonID
                                                          FROM Children
                                                          WHERE PersonID in (SELECT PersonID
                                                                             FROM People p
                                                                             WHERE p.FirstName = 'Geddy'
                                                                             AND p.LastName = 'Lee')
                                                          )
                                        )
                    );
```

	firstname character varying(50)	lastname character varying(50)
1	Killer	Mike

- This next query shows any employee in a certain activity that makes over $1000. There would be a dropdown menu that allows you to choose the dollar amount and the activity name.

```
SELECT p.FirstName, p.LastName, e.SalaryYearlyUSD
FROM People p, Employees e, EmployeesInActivities ea
WHERE p.PersonID = e.PersonID
AND ea.PersonID = e.PersonID
AND e.SalaryYearlyUSD > 1000
AND ea.ActivityName = 'Art';
```

	firstname character varying(50)	lastname character varying(50)	salaryyearlyusd integer
1	Joel	Zimmerman	2100

Stored Procedures

FindGroupsAtTime

- In this stored procedure, you enter a time and you will quickly get where all groups are at a certain time. Ideally, the program this database is behind will have a drop down bar with all available times. The program will then call this function with the time the user chose from the drop down.

```
CREATE OR REPLACE FUNCTION FindGroupsAtTime(time, REFCURSOR)
RETURNS REFCURSOR AS $$
declare
   input_time        time       := $1;
   current_activity  REFCURSOR  := $2;
BEGIN
   open current_activity for
      SELECT GroupName, ActivityName
      FROM GroupsInActivities
      WHERE timeinactivity = input_time;
   RETURN current_activity;
END;
$$
LANGUAGE PLPGSQL;
```

This example is to show where the groups are at 1:00 PM.

```
select FindGroupsAtTime('13:00:00', 'results');
fetch all from results;
```

	groupname character varying(20)	activityname character varying(20)
1	Toy Story	Sports
2	Kung Fu Panda	Art

SetBeginnerSalary

- This stored procedure is for the trigger that will be found in the next section. It is made so that if a salary is entered that is less than the starter salary of $850/year then it will update so that it is $850/year. Everyone in the camp makes at least this and nobody is paid hourly or less than that so we do not need to worry about anything going awry.

Code on next page →

```
CREATE OR REPLACE FUNCTION SetBeginnerSalary()
RETURNS TRIGGER AS $$
BEGIN
   IF (SalaryYearlyUSD < 850)
   FROM Employees
   WHERE SalaryYearlyUSD < 850
   THEN
   UPDATE Employees
   SET SalaryYearlyUSD = 850
   WHERE SalaryYearlyUSD < 850;
   End if;
   Return new;
END
$$
LANGUAGE PLPGSQL;
```

Triggers

SalaryFix

- This trigger uses the stored procedure SetBeginnerSalary() to fix the salary of a person who's entered with a salary of less than $850.

```
CREATE OR REPLACE TRIGGER SalaryFix
AFTER INSERT OR UPDATE
ON Employees
FOR EACH ROW EXECUTE
PROCEDURE SetBeginnerSalary();
```

```
INSERT INTO People(PersonID, FirstName, LastName, Gender, StreetNum, StreetName, StreetType,
                   AddressType, Town, PostalArea)
  VALUES(17, 'Raydon', 'Randell', 'M', 147, 'Felicia', 'Drive', 'Home', 'Morganville', 07751);

INSERT INTO Employees(PersonID, SSN, SalaryYearlyUSD, PrimaryPhoneNumber, BirthDate)
  VALUES (17, 419012764, 200, 1057381129, '1990-11-30');

select * from employees;
```

Notice how the SalaryYearlyUSD for Raydon Randell is $200. This query will show how the trigger works.

```
select * from employees
```

11	17	012764	850	1057381129	1990-11-

The INSERT statement had the SalaryYearlyUSD as $200 but the trigger automatically made it $850.

Security

- I have chosen three levels of security. The top level is for the administrator who has the power to do everything without changing the tables of the database.

```
CREATE ROLE admin;
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA PUBLIC
TO admin;
```

- The next step is for the managers who have the same abilities as the admin with the exception of being able to delete rows. That is an admin only option.

```
CREATE ROLE manager;
GRANT SELECT, INSERT, UPDATE
ON ALL TABLES IN SCHEMA PUBLIC
TO manager;
```

- Last but not least are the rest of the employees in the camp who have no powers with the exception of being able to call queries.

```
CREATE ROLE employee;
GRANT SELECT
ON ALL TABLES IN SCHEMA PUBLIC
TO employee;
```

Implementation, Known Problems, & Future Enhancements

Implementation

- Implementation was difficult at first but once the ER diagram was finished it was smooth sailing.
- Stringing this together with a front end application should not be difficult at all since the database is independent.

Known Problems

- There is no option for a secondary phone number or there will be nulls if the person does not own a phone
- PostalArea is not in BCNF

Future Enhancements

- Make a phone number table with a one-to-many relationship for people to have multiple phone numbers.
- Keep track of children who do swimming lessons and for how long
- Keep track of how much money an activity spent in the last year and give them a new budget accordingly
- Make a view for every group that is in the camp and show their schedule.
- The schedule changes every 2 weeks. Every group follows the same schedule just at a different time frame. The schedule is the same from Playground to Lunch and then a group could have Karate/Dance and end the day with Art.
    - The schedule goes as follows:
        - Playground
        - Swim
        - Lunch
        - Music
        - Sports
        - Art
        - Karate/Dance
        - Snack
        - Pool