MARIST™

DANCE ENSEMBLE

Database Design Proposal

Kelsey O'Brien

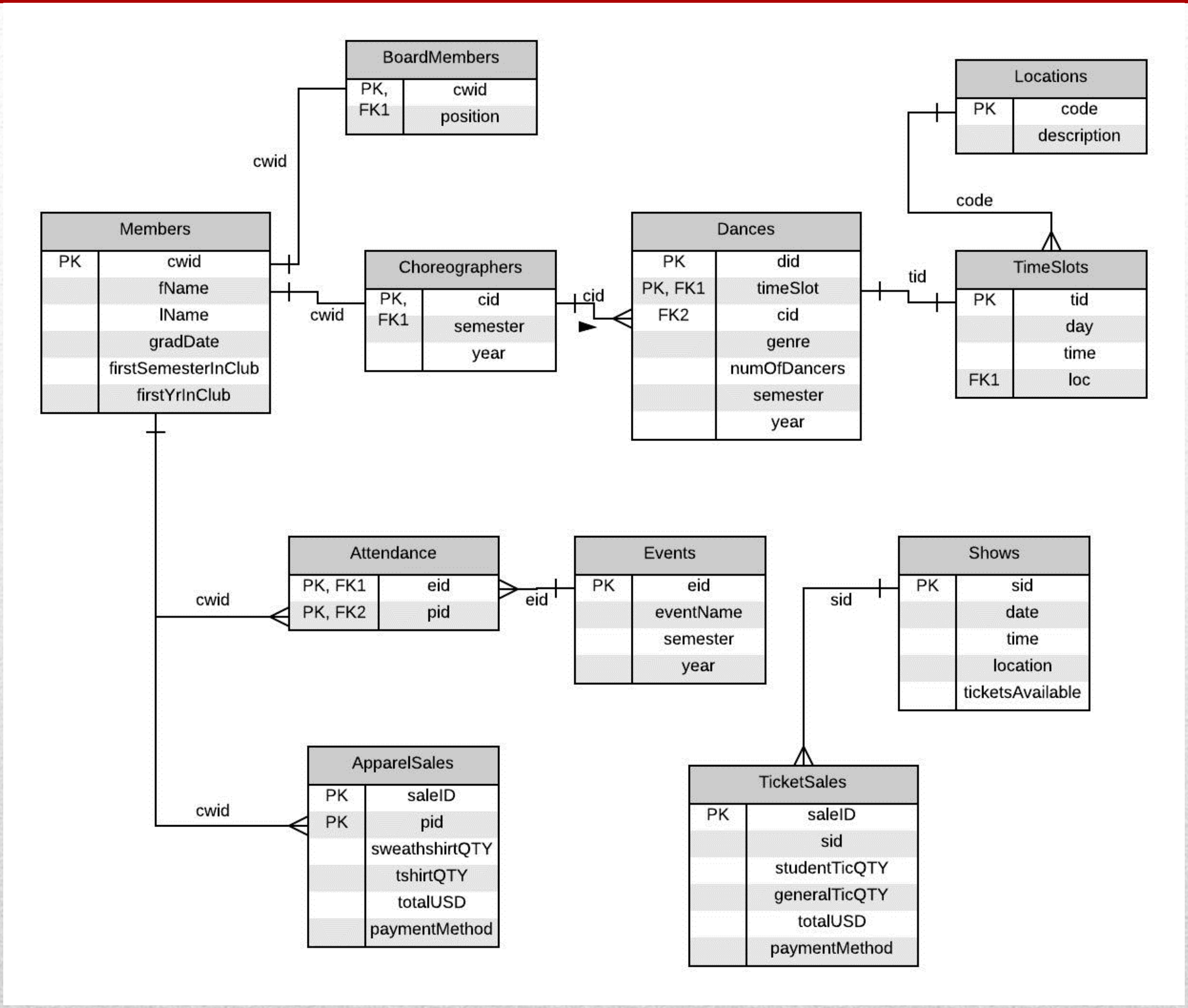
November 29, 2013

Table of Contents

Marist College Dance Ensemble is an entirely student run organization with about four hundred active members. Being one of the largest organizations on the Marist campus poses some problems. First, keeping track of member information is not a simple task; attendance of meetings and community service events is crucial in awarding the appropriate amount of priority points every semester. Second, apparel and tickets to the shows are sold each semester. Information about the buyer, their method of payment and much more must be maintained in order to maintain reliability to the customers, perform proper deposits, and comply to Marist College policies.

Currently, all information is recorded manually on paper, and is then entered in a computer. The creation of a database system to service the dance ensemble would allow the organization to function in a more efficient manner. The system would save time, guarantee reliability, ensure compliance to school policies, and prevent serious mistakes from occurring.

Executive Summary



Entity Relationship Diagram

MARIST DANCE ENSEMBLE

The members table keeps track of all members and information associated with each.

Create Statement:

```
create table Members (
        cwid integer not null check (cwid > 9999999),
        fName char(10) not null,
        lName char(20) not null,
        gradDate integer check (gradDate > 2000),
        firstSemesterInClub varchar not null check (firstSemesterInClub = 'Fall' or firstSemesterInClub = 'Spring'),
        firstYrInClub integer not null check (firstYrInClub > 2000),
 primary key(cwid)
)
```

Functional Dependencies:

Cwid → fName, lName, gradDate, gradDate, firstSemesterInClub, firstYrInClub

Sample Data:

cwid	fName	lName	gradDate	firstSemesterInClub	firstYrInClub
10138349	Kelsey	OBrien	2014	Fall	2010
10283589	Dana	Murano	2014	Fall	2011
10156789	Alana	Brolly	2015	Spring	2011
10238905	Jenn	Ford	2014	Spring	2010
10364839	Jaclyn	Navarro	2016	Fall	2012
10567029	Lauren	Rubis	2016	Spring	2012
10124562	Kate	Green	2015	Fall	2013
10563497	Meghan	Hunt	2016	Spring	2012

Members table

Members can be choreographers. This table keeps track of who the choreographers are and which semester they were a choreographer.

Create Statement:

```
create table Choreographers(
        cid integer not null references Members(cwid),
        firstSemester varchar not null check (semester = 'Fall' or semester = 'Spring'),
        firstYear integer not null check (year > 2000),
 primary key(cid)
)
```

Functional Dependencies:

(cwid) → semester, year

cwid	firstSemester	firstYear
10283589	Spring	2011
10364839	Fall	2011
10238905	Spring	2011
10567029	Fall	2012
10563497	Fall	2013

Choreographers table

Members can be elected board members and take on specific roles to help the organization function. This table keeps track of which members are board members and what their position is.

Create Statement:

```
create table BoardMembers (
        cwid integer references Members(cwid),
        position varchar not null,
 primary key (cwid)
)
```

Functional Dependencies:

cwid → position

Sample Data:

cwid	position
10138349	Website Manager
10283589	President
10364839	Show Manager

BoardMembers table

Every semester MCDE puts on a show of about 30 different dances. The dances table keeps track of all the different dances for each semester.

Create Statement:

```
create table Dances(
            did char(3) not null,
            timeSlot char(3) not null references TimesSlots(tid) unique,
            cid integer not null references Choreographer(cid),
            genre varchar not null,
            numOfDancers integer not null,
            semester varchar not null check (semester = 'Fall' or semester = 'Spring'),
            year integer not null check (year > 2000),
 primary key(did, timeSlot)
)
```

Functional Dependencies:

(did, timeSlot) → cid, genre, numOfDancers, semester, year

Dances table

Sample Data:

did	timeSlot	cid	genre	numOfDancers	semester	year
d01	t02	10563497	Contemporary	32	Fall	2013
d02	t01	10364839	Hip Hop	26	Fall	2013
d03	t03	10238905	Ballet	15	Fall	2013
d04	t06	10567029	Tap	35	Fall	2013
d05	t05	10563497	Lyrical	35	Fall	2013
d06	t04	10283589	Contemporary	34	Fall	2013
d07	t07	10364839	Hip Hop	28	Fall	2013
d08	t09	10238905	Ballet	17	Fall	2013
d09	t08	10567029	Tap	25	Fall	2013
d10	t10	10563497	Lyrical	39	Fall	2013

Dances Table (Cont.)

Scheduling is a challenge for MCDE, there are only two possible locations to hold practices and only select times when those locations are available. This table hold all possible combinations of day, time and location available for the organizations use. Practices are only held Sundays through Thursdays from 5 pm to 10 pm.

Create Statement:

```
create table TimeSlots(
        tid char(3) not null,
        day char(3) not null check (day = 'Sun' or day = 'Mon' or day = 'Tue' or day = 'Wed'or day = 'Thr'),
        time time not null check (time > '4:59' and time < '10:01'),
        loc char(2) not null references Locations(locCode),
 primary key (tid)
)
```

Functional Dependencies:

(tid) → day, time, loc

Sample Data:

tid	day	time	loc
t01	Sun	6:00	DS
t02	Sun	7:00	AS
t03	Mon	5:00	AS
t04	Mon	8:00	AS
t05	Tue	9:00	DS
t06	Tue	9:00	AS
t07	Wed	10:00	AS
t08	Wed	7:00	AS
t09	Thr	9:00	AS
t10	Thr	8:00	DS

TimeSlots table

MARIST DANCE ENSEMBLE

There are only two possible locations for MCDE to hold practices. This table holds the locations for use in the TimeSlots table. Due to the expansion of Marist College, further locations can be added or removed in the future.

Create Statement:

```
create table Locations(
        locCode char(2) not null,
        description varchar not null,
 primary key(locCode)
)
```

Functional Dependencies:

(locCode) → description

locCode	description
DS	Dance Studio
AS	Aerobic Studio

Locations table

MCDE holds and participates in various events throughout the course of the semester. Members have to attend a certain number of events to get priority points. This table holds all the events.

Create Statement:

```
create table Events(
        eid char(4) not null,
        eventName varchar not null,
        semester varchar not null check (semester = 'Fall' or semester = 'Spring'),
        year integer not null check (year > 2000),
primary key(eid)
)
```

Functional Dependencies:

eid → eventName, semester, year

eid	eventName	semester	year
e01	Hunger Walk	Fall	2013
e02	Hunger Meal	Fall	2013
e03	Danceathon	Fall	2013
e04	Heart1 Football Game	Fall	2013
e05	Meeting 1	Fall	2013
e06	Meeting 2	Fall	2013
e07	Meeting 3	Fall	2013

Events table

In order to award the appropriate amount of priority points, MCDE must keep track of which members attended each of the events. This table does just that.

Create Statement:

```
create table Attendance(
        eid char(4) not null references Events(eid),
        pid integer not null references Members(cwid),
primary key(eid, pid)
)
```

Functional Dependencies:

No functional dependencies

Attendance table

Sample Data:

eid	pid
e01	10563497
e01	10364839
e01	10238905
e02	10138349
e02	10563497
e03	10364839
e03	10156789
e03	10124562
e04	10563497
e05	10364839
e05	10238905
e05	10138349
e05	10283589

eid	pid
e06	10138349
e06	10124562
e06	10567029
e06	10238905
e06	10563497
e07	10283589
e07	10364839
e07	10156789
e07	10567029
e07	10563497

Attendance Table (Cont.)

MARIST DANCE ENSEMBLE

MCDE holds two shows every semester. This table holds all the information regarding the time and location of the show.

Create Statement:

```
create table Shows(
        sid char(3) not null,
        date date not null,
        time time not null,
        location varchar not null,
        ticketsAvailable integer not null,
primary key(sid)
)
```

Functional Dependencies:

sid → date, time, location, ticketsAvailable

sid	date	time	location	ticketsAvailable
s01	11-23-2013	2:00	Poughkeepsie High School	1000
s02	11-24-2013	4:30	Poughkeepsie High School	1000
s03	04-19-2014	2:00	McCann Center	640
s04	04-20-2014	4:30	McCann Center	640

Shows table

When selling tickets, the MCDE board members working the ticket sale record the show the tickets are being purchased for, the quantity of each type of ticket, the total of the sale and the buyer payment method. This table holds all of that information.

Create Statement:

```
create table TicketSales(
        saleID serial not null,
        sid char(3) references Shows(sid),
        studentTicQTY integer,
        generalTicQTY integer,
        totalUSD float not null,
        paymentMethod varchar not null check (paymentMethod = 'Cash' or paymentMethod = 'Check'
                                                        or paymentMethod = 'Marist Money'),
primary key(saleID)
)
```

Functional Dependencies:

saleID → sid, studentTicQTY, generalTicQTY, total, paymentMethod

saleID	sid	studentTicQTY	generalTicQTY	totalUSD	paymentMethod
1	s01	1	3	31.00	Cash
2	s02	2	4	62.00	Check
3	s02	1	0	7.00	Marist Money
4	s01	4	1	42.00	Check
5	s01	0	5	60.00	Cash
6	s04	2	3	50.00	Cash
7	s03	1	0	7.00	Marist Money
8	s01	2	2	38.00	Check

TicketSales table

MCDE sells apparel each semester. This table keeps track of who buys apparel, how much they buy, and their method of payment.

Create Statement:

```
create table ApparelSales(
        saleID serial not null,
        pid integer not null references Members(cwid),
        sweatshirtQTY integer,
        tshirtQTY integer not null,
        totalUSD float not null,
        paymentMethod varchar not null check (paymentMethod = 'Cash' or paymentMethod = 'Check'),
primary key(saleID, pid)
)
```

Functional Dependencies:

(saleID, pid) → sweatshirtQTY, tshirtQTY, totalUSD, paymentMethod

saleID	pid	sweatshirtQTY	tshirtQTY	totalUSD	paymentMethod
1	10563497	1	2	45.00	Cash
2	10567029	2	0	50.00	Check
3	10156789	0	4	40.00	Cash
4	10283589	1	1	35.00	Cash
5	10138349	3	1	85.00	Check
6	10238905	0	1	10.00	Cash
7	10364839	1	3	55.00	Check

ApparelSales table

Detailed Apparel Sale Roster

This view, *ApparelSaleRoster*, can be used to see all sales of apparel and the member information associated with each sale. The creation of this view is useful because the *ApparelSale* table only contains the cwid of the member associated with each purchase. Every semester MCDE holds a presale of apparel. Members who buy their apparel at the presale are guaranteed the quantity and sizes that they prefer. The *ApparelSaleRoster* view will be beneficial when distributing all the presale apparel once it comes in.

```
create view ApparelSaleRoster
as
select m.fName, m.lName, m.cwid, a.sweatshirtQTY, a.tshirtQTY, a.totalUSD, a.paymentMethod
            from Members m
            inner join ApparelSales a
            on m.cwid = a.pid;
```

Board members can use to query below to view all apparel sale information at the time of need:

```
            select * from ApparelSaleRoster
```

Choreographer Roster sorted by Semester

Choreographers are not guaranteed a spot in the show every semester. Each choreographer has to audition with the dance they would like to teach at the beginning of each semester. This view shows all the choreographer information and their associated dance sorted by the semester.

```
create view semesterChoreograohers
as
select m.fName, m.lName, m.cwid, d.did, d.semester, d.year
                from Members m
                inner join Choreographers c on m.cwid = c.cid
                inner join Dances d on d.cid = c.cid
                order by semester, year asc
```

Views (Cont.)

Amount of Tickets Sold for each show

This report shows how many tickets have been purchased for each show at a specific point in time. This will be beneficial to MCDE's board members when dealing the employees of Marist's college activities. It is a goal to sell out of tickets for every show to maximize profit. As a result, at the end of every sale they need a report of how many tickets have been for each show so far. In addition, this will allow the board members to notify the rest of the members when the show is getting close to selling out.

```
select t.sid, s.date, s.time, sum(studentTicQTY) as "Students Tickets", sum(generalTicQty) as "General Tickets",
                                                                                                        s.ticketsAvailable
              from TicketSales t
              inner join Shows s on t.sid = s.sid
              group by t.sid, s.date, s.time, s.ticketsAvailable
```

Members graduating in the current year

Every year MCDE recognizes their members who are graduating. Since the organization does not currently have a database system in place, it is always a hassle to get the graduating seniors to come forward so the board members know who to recognize. This report displays all the members who are graduating in the current year.

```
select fName, lName, gradDate
        from Members
        where gradDate = extract(year from now())
```

Board Members who are also Choreographers

```
select m.fName, m.lName
        from Members m
        left join Choreographers c on m.cwid = c.cid
        inner join BoardMembers b on b.cwid = c.cid
```

MARIST DANCE ENSEMBLE

Total ticket sales calculation

This report displays the saleID, quantity of each type of ticket purchased, the total amount entered in manually, and then the total amount when calculated by the database system. Student tickets are sold at the rate of $7 and general tickets are sold for $14. This can be used to make sure the user manually entered in the correct total for the sale. This will help the organization perform safe sales and comply with school policy.

```
select t.saleID, t.studentTicQTy, t.generalTicQTY, totalUSD as "Entered Total",
                    ((t.StudentTicQTY * 7) + (generalTicQTY * 14)) as "Calculated Total"
          from TicketSales t
          group by t.saleID
          having t.studentTicQTY > 0
          or t.generalTicQTY > 0
```

Total apparel sold stored procedure

This stored procedure returns the total amount of money made from selling apparel. This procedure will be useful when making the deposit with college activities at the conclusion of the sale. The board members working the sale can match the records in the database with how much money they have at the end of the sale.

```
create or replace function totalApparelSold()
returns integer as $$
declare
        total float :=0;
begin
        select sum(totalUSD)
        into total
        from ApparelSales;
return total;
end;
$$ language plpgsql;
```

**The same can be done with ticket sales for the same purpose.

decrementTicketsAvailable trigger

The amount of tickets available are stored in the Shows table, while the actual ticket transactions are stored in the *TicketSales* table. When a transaction is entered into the *TicketSales* table the *decrementTicketsAvailable* trigger subtracts the amount of tickets purchased from the *ticketsAvailable* column in the Shows table. This is useful for keeping track of exactly how many tickets are left for sale.

```
create or replace function decrementTicketsAvail()
returns trigger as $$
declare
        --Variable needed to keep track of tickets being purchased
        tixStuQTY integer := 0;
        tixGenQTY integer := 0;
        tixAvail integer :=0;
        showID char(3);
begin
        --Get the amount of student tickets purchased
        select studentTicQTY
        into tixStuQTY
        from TicketSales t
        where t.studentTicQTY > 0;

        --Get the amount of general tickets purchased
        select generalTicQTY
        into tixGenQTY
        from TicketSales t
        where t.studentTicQTY > 0;

        --Get the show id
        select sid
        into showID
        from TicketSales;
```

```
            --Subtract amount of tickets from ticketsAvailable column for each specific show
            update Shows
            set ticketsAvailable = (ticketsAvailable - tixStuQTY)
            where sid = showID;


            update Shows
            set ticketsAvailable = (ticketsAvailable -  tixGenQTY)
            where sid = showID;
return null;
end;
$$ language plpgsql;

create trigger decrementTicketsAvail
            after insert or update on TicketSales
            for each row
            when (pg_trigger_depth() = 0)
            execute procedure decrementTicketsAvail();
```

Triggers (Cont.)

generalMemberRole

The generalMemberRole refers to general members who are not choreographers or board members. This role is allowed to view some tables, but cannot insert or update any table. The tables this role can view are: Dances, Events, Shows, TimeSlots, and Locations. This gives them access to all dance classes and events the club has to offer.

```
CREATE ROLE generalMemberRole

REVOKE ALL PRIVILEGES ON Members FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON BoardMembers FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Choreographers FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Dances FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON TimeSlots FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Locations FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON ApparelSales FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Shows FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON TicketSales FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Events FROM generalMemberRole;
REVOKE ALL PRIVILEGES ON Attendance FROM generalMemberRole;


GRANT SELECT ON Dances FROM generalMemberRole;
GRANT SELECT ON Events FROM generalMemberRole;
GRANT SELECT ON Shows FROM generalMemberRole;
GRANT SELECT ON TimeSlots FROM generalMemberRole;
GRANT SELECT ON Locations FROM generalMemberRole;
```

Security

choreographerRole

This role refers to the members who are also choreographers. They have almost the same access as the generalMemberRole except they have the ability to view the Choreographer table.

```
CREATE ROLE choreographerRole

REVOKE ALL PRIVILEGES ON Members FROM choreographerRole;
REVOKE ALL PRIVILEGES ON BoardMembers FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Choreographers FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Dances FROM choreographerRole;
REVOKE ALL PRIVILEGES ON TimeSlots FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Locations FROM choreographerRole;
REVOKE ALL PRIVILEGES ON ApparelSales FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Shows FROM choreographerRole;
REVOKE ALL PRIVILEGES ON TicketSales FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Events FROM choreographerRole;
REVOKE ALL PRIVILEGES ON Attendance FROM choreographerRole;

GRANT SELECT ON Dances FROM choreographerRole
GRANT SELECT ON Events FROM choreographerRole
GRANT SELECT ON Shows FROM choreographerRole
GRANT SELECT ON TimeSlots FROM choreographerRole
GRANT SELECT ON Locations FROM choreographerRole
GRANT SELECT ON Choreographers FROM choreographerRole
```

boardMemberRole

This role refers to the members who are also a part of the executive board. This role is allowed to view all tables. In addition, the boardMemberRole has access to insert and update all tables that do not hold important member information (i.e.: Members, Choreographers, and BoardMembers tables). In addition, only the President and Vice President deal with which dances are offered each semester so the boardMemberRole cannot alter any values of the Dances, TimeSots and Locations tables.

```
CREATE ROLE boardMemberRole

REVOKE ALL PRIVILEGES ON Members FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON BoardMembers FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Choreographers FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Dances FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON TimeSlots FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Locations FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON ApparelSales FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Shows FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON TicketSales FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Events FROM boardMemberRole;
REVOKE ALL PRIVILEGES ON Attendance FROM boardMemberRole;

GRANT SELECT, INSERT, UPDATE ON Attendance FROM boardMemberRole
GRANT SELECT, INSERT, UPDATE ON Events FROM boardMemberRole
GRANT SELECT, INSERT, UPDATE ON ApparelSales FROM boardMemberRole
GRANT SELECT, INSERT, UPDATE ON Shows FROM boardMemberRole
GRANT SELECT, INSERT, UPDATE ON TicketSales FROM boardMemberRole
```

PVPRole

The PVPRole refers to the president and the vice president of MCDE. Since there are such a large amount of members and just a few board members the president and vice president work hand in hand. This role has access to view, insert, update and delete data from any and all table. They are the only members of they club that have this access.

```
CREATE ROLE PVPRole

REVOKE ALL PRIVILEGES ON Members FROM PVPRole;
REVOKE ALL PRIVILEGES ON BoardMembers FROM PVPRole;
REVOKE ALL PRIVILEGES ON Choreographers FROM PVPRole;
REVOKE ALL PRIVILEGES ON Dances FROM PVPRole;
REVOKE ALL PRIVILEGES ON TimeSlots FROM PVPRole;
REVOKE ALL PRIVILEGES ON Locations FROM PVPRole;
REVOKE ALL PRIVILEGES ON ApparelSales FROM PVPRole;
REVOKE ALL PRIVILEGES ON Shows FROM PVPRole;
REVOKE ALL PRIVILEGES ON TicketSales FROM PVPRole;
REVOKE ALL PRIVILEGES ON Events FROM PVPRole;
REVOKE ALL PRIVILEGES ON Attendance FROM PVPRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON Members FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON BoardMembers FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Choreographers FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Dances FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON TimeSlots FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Locations FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON ApparelSales FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Shows FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON TicketSales FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Events FROM PVPRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Attendance FROM PVPRole;
```

Final Notes

Marist College Dance Ensemble has been an extremely popular and involved organization on the Marist campus for over a decade. MCDE is an entirely student run entity. All work necessary to make the organization function is divided between the eight members of the executive board. Although a database system is not necessary, it would be extremely beneficial to the organization and the board members that work to make everything run smoothly.

The proposal previously stated, is the first revision to a possible MCDE database system. There are many tables, fields, and features that need to be modified or added in order to entirely and accurately model the functions of the organization.

There are a few known problems present in the proposal stated above. The decrementTicketsAvail trigger does not support tickets being refunded. In addition, there are various rules that MCDE has that are not enforced by the database outlined in the proposal. For example, a member must be in the club for at least a semester before they are eligible to become a choreographer. The database holds these values, but does not enforce the rule.

The goal of this proposal was to cover the basic functionality of MCDE. In the future, the database should hold all the rosters for each dance offered. Also, the organization functions by semester. In other words, once the semester is over new members join, there are different choreographers and dances, and new events are offer. In the future, this database should withstand, in an organized manner, across semesters.