

# MARIST

## **Club Management Database**

William Cain

CMPT 308

December 2, 2013

# Table of Contents

---

Executive Summary .....	3
ER Diagram .....	4
Students Table .....	5
Teachers Table .....	6
Semesters Table .....	7
Club_Roles Table .....	8
Clubs Table .....	9
Advisors_Clubs Table .....	10
Students_Clubs Table .....	11
Club_Meetings Table .....	12
Club_Events Table .....	13
Meeting_Attendances Table .....	14
Event_Attendances Table .....	15
Views .....	16
Queries .....	17
Stored Procedures .....	18
Triggers .....	25
Implementation Notes .....	26
Known Issues .....	26
Future Enhancements .....	26

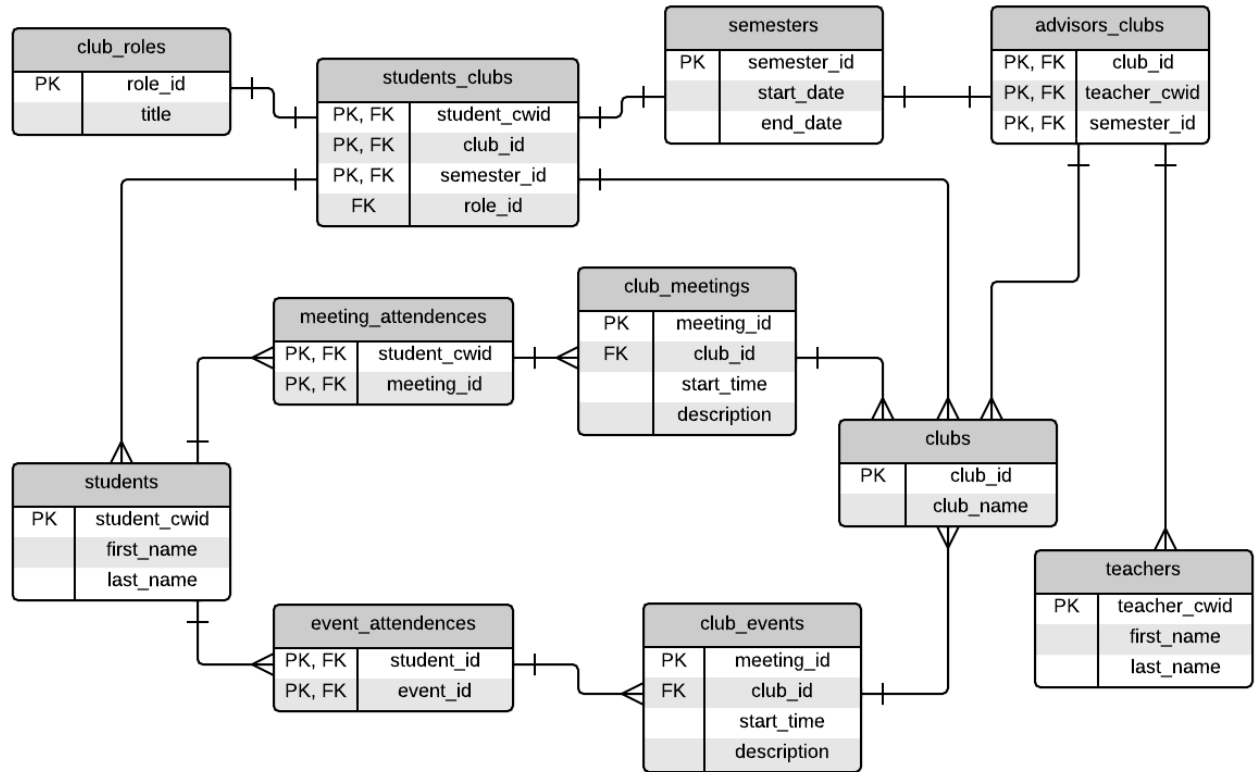
# Executive Summary

---

The Marist College Club Management database keeps track of all the clubs at Marist College. It stores all the clubs along with their accompanying members, officers, and advisors. The database also includes tables for storing club meetings and events along with who attended. Additionally, the database provides several stored procedures to track and calculate the number of priority points each member will receive from a particular club.

# ER Diagram

---



# Students Table

---

The student table stores all student information using the student's CWID to uniquely identify them.

## SQL Create Statement

---

```
CREATE TABLE students (  
    student_cwid VARCHAR(8) NOT NULL PRIMARY KEY,  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255) NOT NULL  
);
```

## Functional Dependencies

---

Student\_cwid → first\_name  
Student\_cwid → last\_name

## Sample Data

---

	student_cwid character varying(8)	first_name character varying(255)	last_name character varying(255)
1	20036153	William	Cain
2	10129460	Bobby	Tables
3	08472532	Jeffrey	Hippo
4	21172544	Alfred	Pennyworth

# Teachers Table

---

The student table stores all student information using the student's CWID to uniquely identify them.

## SQL Create Statement

---

```
CREATE TABLE teachers (  
    teacher_cwid VARCHAR(8) NOT NULL PRIMARY KEY,  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255) NOT NULL  
);
```

## Functional Dependencies

---

teacher\_cwid → first\_name  
teacher\_cwid → last\_name

## Sample Data

---

	teacher_cwid character varying(8)	first_name character varying(255)	last_name character varying(255)
1	12234363	Matthew	Shin
2	10234343	Jeff	Bass
3	10224433	Matthew	Johnson
4	10232333	Julie	Raines
5	10232233	Karen	Schrier
6	10237393	Kristin	Janschewitz
7	10233123	Edward	Smith
8	21234333	Michael	Janofsky
9	10234323	Carolyn	Matheus
10	10234387	Carlos	Rodriguez

# Semesters Table

---

A semester is represented as a range of time with a start and end date. The semester is then identified using an automatically incremented index.

## SQL Create Statement

---

```
CREATE TABLE semesters (  
    semester_id SERIAL NOT NULL PRIMARY KEY,  
    start_date timestamp with time zone NOT NULL,  
    end_date timestamp with time zone NOT NULL  
);
```

## Functional Dependencies

---

semester\_id → start\_date

semester\_id → end\_date

## Sample Data

---

	semester_id integer	start_date timestamp with time zone	end_date timestamp with time zone
1	1	2013-08-26 01:00:00-04	2013-06-20 00:59:59-04
2	2	2014-01-21 00:00:00-05	2014-05-23 00:59:59-04

# Club\_Roles Table

---

Within each club, a student can be either a general member or an officer. An officer typically has one of the following titles: President, Vice President, Treasurer, Webmaster, or Secretary. Typically, clubs follow this standard model; however, occasionally they have additional offices, such as public relations coordinator. Anyone who is not an officer is given the role of “member.”

## SQL Create Statement

---

```
CREATE TABLE club_roles (  
    role_id SERIAL NOT NULL PRIMARY KEY,  
    role_name VARCHAR(255) NOT NULL  
);
```

## Functional Dependencies

---

Role\_id → role\_name

## Sample Data

---

	role_id integer	role_name character varying(255)
1	1	Member
2	2	Treasurer
3	3	Webmaster
4	4	Secretary
5	5	Public Relations
6	6	Vice President
7	7	President



# Clubs Table

---

The club table is responsible for associating each club name with an automatically incremented integer to identify it.

## SQL Create Statement

---

```
CREATE TABLE clubs (  
    club_id SERIAL NOT NULL PRIMARY KEY,  
    club_name VARCHAR(255) NOT NULL  
);
```

## Functional Dependencies

---

club\_id → club\_name

## Sample Data

---

	club_id integer	club_name character varying(255)
1	1	Business Club
2	2	Communication Arts Society
3	3	Computer Society
4	4	Criminal Justice Society
5	5	Marist Game Society
6	6	Psychology Club
7	7	Anime Society
8	8	Habitat for Humanity
9	9	Chess Club

# Advisors\_Clubs Table

---

Advisors\_Clubs is an associative table that tracks who each the club advisor is for a particular semester. Because primary key is a composite based off of the teacher's CWID, the club ID, and the semester, it supports clubs that have more than one advisor, such as the chess club.

## SQL Create Statement

---

```
CREATE TABLE advisors_clubs (  
    teacher_cwid int REFERENCES teachers(teacher_id) NOT NULL,  
    club_id int REFERENCES clubs(club_id) NOT NULL,  
    semester_id int REFERENCES semesters(semester_id) NOT NULL,  
    primary key (teacher_id, club_id, semester_id)  
);
```

## Functional Dependencies

---

N/A

## Sample Data

---

	teacher_cwid character varying(8)	club_id integer	semester_id integer
1	12234363	1	1
2	10234343	2	1
3	10224433	3	1
4	10232333	4	1
5	10232233	5	1
6	10237393	6	1
7	10233123	7	1
8	21234333	8	1
9	10234323	9	1
10	10234387	9	1

# Students\_Clubs Table

---

The students\_clubs table is another associate table that tracks the members of each club for each semester along with what their role in the club was.

## SQL Create Statement

---

```
CREATE TABLE students_clubs (  
    student_cwid VARCHAR(8) REFERENCES students(student_cwid) NOT NULL,  
    club_id int REFERENCES clubs(club_id) NOT NULL,  
    semester_id int REFERENCES semesters(semester_id) NOT NULL,  
    role_id int REFERENCES club_roles(role_id) NOT NULL,  
    primary key (student_cwid, club_id, semester_id)  
);
```

## Functional Dependencies

---

N/A

## Sample Data

---

	student_cwid character varying(8)	club_id integer	semester_id integer	role_id integer
1	20036153	3	1	1
2	20036153	3	2	1
3	20036153	5	1	1
4	20036153	8	1	1
5	08472532	1	2	7
6	08472532	7	2	1
7	10129460	8	1	3
8	10129460	2	1	1
9	10129460	4	2	1

# Club\_Meetings Table

---

The club\_meetings table stores a compilation of all the meetings for each club along with a start time. This table also includes an optional field for club officers to provide a description of what will be discussed at the meeting.

## SQL Create Statement

---

```
CREATE TABLE club_meetings (  
    meeting_id SERIAL NOT NULL PRIMARY KEY,  
    club_id int REFERENCES clubs(club_id) NOT NULL,  
    start_time timestamp with time zone NOT NULL,  
    description VARCHAR(255)  
);
```

## Functional Dependencies

---

meeting\_id → club\_id  
meeting\_id → start\_time  
meeting\_id → description

## Sample Data

---

	meeting_id integer	club_id integer	start_time timestamp with time zone	description character varying(255)
1	1	3	2013-12-08 21:15:00-05	Hack Night in the Linux Lab
2	2	3	2013-12-18 21:15:00-05	Lightning talk about functional programming
3	3	5	2013-12-12 21:30:00-05	Video gamezzzzzzzz
4	4	8	2013-12-03 21:30:00-05	We are going to show another presentation without any sound
5	5	4	2013-12-05 04:05:00-05	

# Club\_Events Table

---

The club\_events table stores a compilation of all the events for each club along with a start time. This table also includes an optional event title and description fields for club officers to provide more detail about the particular event.

## SQL Create Statement

---

```
CREATE TABLE club_events (  
    event_id SERIAL NOT NULL PRIMARY KEY,  
    club_id int REFERENCES clubs(club_id) NOT NULL,  
    start_time timestamp with time zone NOT NULL,  
    event_title VARCHAR(255),  
    description VARCHAR(255)  
);
```

## Functional Dependencies

---

event\_id → club\_id  
event\_id → start\_time  
event\_id → event\_title  
event\_id → description

## Sample Data

---

	event_id integer	club_id integer	start_time timestamp with time zone	event_title character varying(255)	description character varying(255)
1	1	3	2013-10-08 20:00:00-04	Hackathon	
2	2	5	2013-11-26 21:30:00-05	Gaming Day	
3	3	8	2013-12-03 21:30:00-05	Build-A-Home Workshop	
4	4	4	2013-12-05 11:05:00-05		

# Meeting\_Attendances Table

---

Description: The meeting\_attendances table is an associative table that tracks what members attended each club meeting.

## SQL Create Statement

---

```
CREATE TABLE meeting_attendances (  
    meeting_id int REFERENCES club_meetings(meeting_id) NOT NULL,  
    student_cwid VARCHAR(8) REFERENCES students(student_cwid) NOT NULL,  
    primary key (meeting_id, student_cwid)  
);
```

## Functional Dependencies

---

N/A

## Sample Data

---

	meeting_id integer	student_cwid character varying(8)
1	1	20036153
2	2	20036153
3	1	08472532
4	2	08472532

# Event\_Attendances Table

---

The event\_attendances table is an associative table that tracks what members attended each club event.

## SQL Create Statement

---

```
CREATE TABLE event_attendances (  
    event_id int REFERENCES club_events(event_id) NOT NULL,  
    student_cwid VARCHAR(8) REFERENCES students(student_cwid) NOT NULL,  
    primary key (event_id, student_cwid)  
);
```

## Functional Dependencies

---

N/A

## Sample Data

---

	event_id integer	student_cwid character varying(8)
1	1	20036153
2	2	20036153
3	3	08472532
4	4	08472532

# Views

---

This view returns a list of all the clubs that are currently in affect this semester. It calculates the current semester using the current timestamp.

```
DROP VIEW IF EXISTS ClubsThisSemester;

CREATE VIEW ClubsThisSemester AS
SELECT distinct club_name
FROM clubs c, students_clubs sc, semesters sem
WHERE c.club_id = sc.club_id
AND sc.semester_id = sem.semester_id
AND NOW() > sem.start_date
AND NOW() < sem.end_date;
```



# Queries

---

This query returns a list of all the students in Computer Society, along with what role each student plays in the club.

```
SELECT first_name, last_name, role_name
FROM students s, clubs c, students_clubs sc, club_roles cr,
semesters sem
WHERE club_name = 'Computer Society'
AND s.student_cwid = sc.student_cwid
AND c.club_id = sc.club_id
AND cr.role_id = sc.role_id
AND sc.semester_id = sem.semester_id
AND NOW() > sem.start_date
AND NOW() < sem.end_date;
```

# Stored Procedures

---

Returns the number of meetings a student attended for a particular club during a particular semester.

```
CREATE FUNCTION studentForMeetings(clubId int, semId int, cwid text)
RETURNS BIGINT as $$
    SELECT count(cm.meeting_id) as attendances
    FROM club_meetings cm, meeting_attendances ma, semesters sem
    WHERE cm.club_id = $1
    AND cm.start_time >= sem.start_date
    AND cm.start_time <= sem.end_date
    AND sem.semester_id = $2
    AND ma.student_cwid = $3
    AND cm.meeting_id = ma.meeting_id;
$$ language 'sql';

SELECT * FROM studentForMeetings(3, 1, '20036153');
```

# Stored Procedures

---

Returns the number of events a student attended for a particular club during a particular semester.

```
CREATE FUNCTION studentForEvents(eventId int, semId int, cwid text)
RETURNS BIGINT as $$
    SELECT count(ce.event_id) as attendances
    FROM club_events ce, event_attendances ea, semesters sem
    WHERE ce.club_id = $1
    AND ce.start_time >= sem.start_date
    AND ce.start_time <= sem.end_date
    AND sem.semester_id = $2
    AND ea.student_cwid = $3
    AND ce.event_id = ea.event_id;
$$ language 'sql';

SELECT * FROM studentForEvents(3, 1, '20036153');
```

# Stored Procedures

---

Returned the total number of meetings a club held during a particular semester.

```
CREATE FUNCTION totalMeetingsBySemester(clubId int, semId int)
RETURNS BIGINT as $$
    SELECT count(meeting_id) as attendances
    FROM club_meetings ce, semesters sem
    WHERE cm.club_id = $1
    AND cm.start_time >= sem.start_date
    AND cm.start_time <= sem.end_date
    AND sem.semester_id = $2;
$$ language 'sql';

SELECT * FROM totalMeetingsBySemester(3, 1);
```

# Stored Procedures

---

Returned the total number of events a club held during a particular semester.

```
CREATE FUNCTION totalEventsBySemester(eventId int, semId int)
RETURNS BIGINT as $$
    SELECT count(event_id) as attendances
    FROM club_events ce, semesters sem
    WHERE ce.club_id = $1
    AND ce.start_time >= sem.start_date
    AND ce.start_time <= sem.end_date
    AND sem.semester_id = $2;
$$ language 'sql';

SELECT * FROM totalEventsBySemester(3, 1);
```

# Stored Procedures

---

Returned a Boolean whether a particular student earned 3 priority points for a particular semester from the given club.

```
DROP FUNCTION IF EXISTS threePriorityPoints(int, int, text);

CREATE FUNCTION threePriorityPoints(semId int, clubId int, cwid text)
RETURNS BOOLEAN as $$
    SELECT (role_id <> 1 AND

        100 * studentForMeetings(sc.club_id, sc.semester_id, sc.student_cwid) /
        totalMeetingsBySemester(sc.club_id, sc.semester_id) >= 50

        AND studentForMeetings(sc.club_id, sc.semester_id, sc.student_cwid) =
        totalMeetingsBySemester(sc.club_id, sc.semester_id))

    FROM students_clubs sc, semesters sem
    WHERE sc.semester_id = $1
    AND sc.club_id = $2
    AND sc.student_cwid = $3;
$$ language 'sql';

SELECT * FROM threePriorityPoints(1, 3, '20036153');
```

# Stored Procedures

---

Returned a Boolean whether a particular student earned 2 priority points for a particular semester from the given club.

```
DROP FUNCTION IF EXISTS twoPriorityPoints(int, int, text);
CREATE FUNCTION twoPriorityPoints(semId int, clubId int, cwid text)
RETURNS BOOLEAN as $$
    SELECT (100 * studentForMeetings(sc.club_id, sc.semester_id,
sc.student_cwid) / totalMeetingsBySemester(sc.club_id,
sc.semester_id) >= 50
        AND 100 * studentForMeetings(sc.club_id, sc.semester_id,
sc.student_cwid) / totalMeetingsBySemester(sc.club_id,
sc.semester_id) >= 75)
    FROM students_clubs sc, semesters sem
    WHERE sc.semester_id = $1
    AND sc.club_id = $2
    AND sc.student_cwid = $3;
$$ language 'sql';
SELECT * FROM twoPriorityPoints(1, 3, '20036153');
```

# Stored Procedures

---

Returned a Boolean whether a particular student earned 1 priority point for a particular semester from the given club.

```
DROP FUNCTION IF EXISTS onePriorityPoint(int, int, text);
CREATE FUNCTION onePriorityPoint(semId int, clubId int, cwid text)
RETURNS BOOLEAN as $$
    SELECT (100 * studentForMeetings(sc.club_id, sc.semester_id,
sc.student_cwid) / totalMeetingsBySemester(sc.club_id,
sc.semester_id) >= 50
        AND 100 * studentForMeetings(sc.club_id, sc.semester_id,
sc.student_cwid) / totalMeetingsBySemester(sc.club_id,
sc.semester_id) >= 50)
    FROM students_clubs sc, semesters sem
    WHERE sc.semester_id = $1
    AND sc.club_id = $2
    AND sc.student_cwid = $3;
$$ language 'sql';
SELECT * FROM onePriorityPoint(1, 3, '20036153');
```



# Triggers

---

This trigger checks that the end date comes after the start date when inserting into or updating the semesters table. If the end date comes before the start date, it throws an exception.

```
CREATE FUNCTION date_order() RETURNS trigger AS $date_order$
BEGIN
    IF NEW.start_date > NEW.end_date THEN
        RAISE EXCEPTION 'end_date % should come after
            start_date %.', NEW.start_date, NEW.end_date;
    END IF;
    RETURN NEW;
END;
$date_order$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER check_semester_dates
BEFORE INSERT or UPDATE
ON semesters
FOR EACH ROW EXECUTE PROCEDURE date_order();
```

# Implementation Notes

---

The Club Management database consists of 11 interconnected tables. For the system to function properly, the tables must be created in the order that they appear within this document.

## Known Issues

---

As of right now, there are no specified PostgreSQL users with different permissions. It is expected that the security will be handled in the implementation language prior to any SQL queries being made.

In the functions that calculate priority points, the `twoPriorityPoints` will return true even if the student earned 3 priority points. Additionally, the `onePriorityPoint` will return true if the student earned one or more priority points.

According to the revised club rules, club sports cannot give more than 5 priority points per year. This means that captains only earn 2 priority points during the offseason instead of 3.

## Future Enhancements

---

In order to properly calculate the number of priority points, the database should be extended to include a Community Service table with an accompanying `CommunityService_Attendances` associative table. The new priority point guidelines require each club to include this.