MARIST CLUBDASH

DATABASE DESIGN PROPOSAL

David Gunter

Database Systems (CMPT308)

2 December 2013

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The overall goal of this database is to support the Marist College Activities and Student Government offices by reducing report complexity and providing on-demand analytics. With over 80 clubs on campus, concise and accurate data organization is imperative to a club system that aims for seamless operations and event planning.

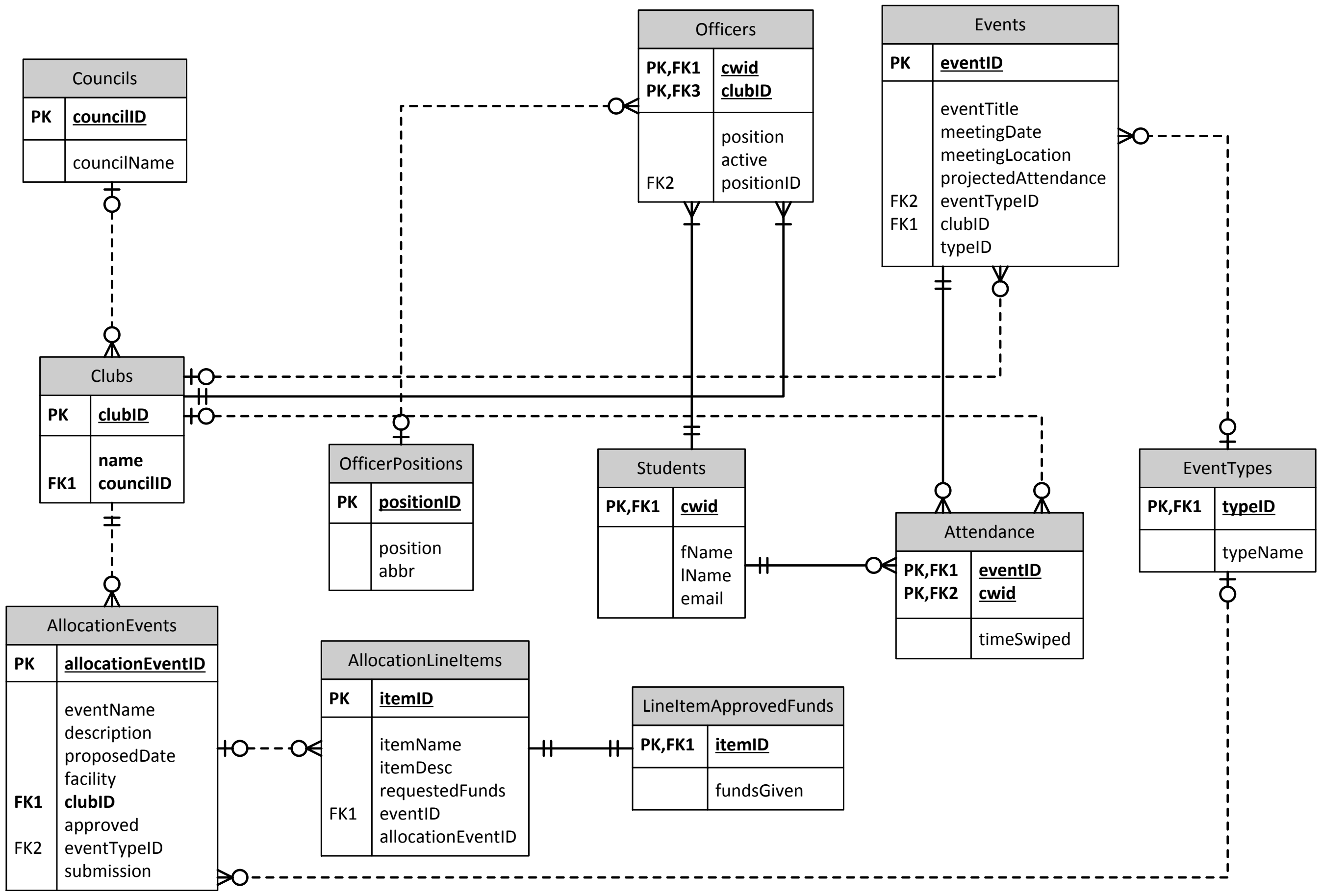
Objectives

ClubDash aims to achieve 3 objectives:

1. **Club Management**
   a. Organize officers and administrators
   b. Collect attendance data for events
   c. Hold club officers accountable
2. **Fund Management & Tracking**
   a. Manage club allocations for each semester
   b. Determine where money is being spent
   c. Create accountability for club spending
   d. Provide analytics into how clubs use their money
3. **Event Management**
   a. Assist with creating and tracking club events/meetings
   b. Give officers and administrators insights into what types of events are most popular
   c. Give administrators the capability to determine if clubs are meeting event requirements

ClubDash creates a central point of information from which club officers and College Activities can benefit. This information will eventually feed into analytics tools, further enhancing the usefulness of the database and allowing for future expansion of the dataset. With the data collected and stored in ClubDash, Marist's club system will benefit through better accountability, reduced administrative overhead, and insights into determining effective club events.

ENTITY RELATIONSHIP DIAGRAM



TABLES

Clubs

Clubs are assigned a name, unique clubID, and a council. The active attribute tells us whether a club is currently operating or is in a decommissioned state (i.e. no members, officers disbanded, has been disbanded by College Activities).

```
CREATE TABLE clubs (

  "clubID" character(15) NOT NULL,
  name character(255) NOT NULL,
  active boolean,
  "councilID" integer NOT NULL,
  CONSTRAINT clubs_pkey PRIMARY KEY ("clubID"),
  CONSTRAINT "clubs_councilID_fkey" FOREIGN KEY ("councilID")
      REFERENCES councils ("councilID") MATCH SIMPLE
  CONSTRAINT "clubs_clubID_key" UNIQUE ("clubID")
);
```

Functional dependencies

clubID → name, active, councilID

clubID	name	active	councilID
CL001	Advertising Club	TRUE	1
CL002	Marist Band	TRUE	3
CL003	Rugby	TRUE	2
CL004	Student Government Association	TRUE	4
CL005	Kappa Kappa Gamma	TRUE	7
CL006	Gaelic Society	FALSE	8
CL007	History Club	FALSE	1
CL008	Beta Alpha Psi	TRUE	5
CL009	SEED	TRUE	6
SPC001	Student Programming Council	TRUE	4
SPC002	Booster Club	TRUE	2

Councils

Councils are groups of clubs that are led by appointed "council managers" to ensure clubs are meeting specific goals relevant to different types of clubs.

```
CREATE TABLE councils (
  "councilID" integer NOT NULL,
  "councilName" character(255) NOT NULL,
  CONSTRAINT councils_pkey PRIMARY KEY ("councilID"),
  CONSTRAINT "councils_councilID_key" UNIQUE ("councilID")
);
```

Functional Dependencies:

councilID → councilName

councilID	councilName
1	Co-Curricular Advisory Council
2	Sports Advisory Council
3	Production-Performance Council
4	Government-Class Council
5	Honorary-Professional Council
6	Social-Service Council
7	Greek Advisory Council
8	Other
9	Intramurals

Attendance

Records all student ID swipes for each event. Students can only swipe once per event.

```
CREATE TABLE attendance (

  "eventID" integer NOT NULL,
  cwid integer NOT NULL,
  "timeSwiped" timestamp without time zone DEFAULT now(),
  CONSTRAINT attendance_pkey PRIMARY KEY ("eventID", cwid),
  CONSTRAINT "attendance_eventID_fkey" FOREIGN KEY ("eventID")
      REFERENCES events ("eventID") MATCH SIMPLE,
  CONSTRAINT swipe_is_unique UNIQUE ("eventID", cwid)
);
```

Functional Dependencies:

eventID, cwid → timeSwiped

eventID	cwid	timeSwiped
1	10121314	1/28/2013 12:04
1	10142581	1/28/2013 12:02
1	10142585	1/28/2013 12:01
1	20011679	1/28/2013 12:03
1	20026881	1/28/2013 12:04
2	10177377	4/25/2013 12:16
2	20011679	4/25/2013 12:16

eventTypes

```
CREATE TABLE "eventTypes"(

  "typeID" integer NOT NULL,
  "typeName" character(255) NOT NULL,
  CONSTRAINT "eventTypes_pkey" PRIMARY KEY ("typeID")
);
```

Functional Dependencies

typeID → typeName

typeID	typeName
1	Academic
2	Athletic
3	Coffee House
4	Community Service
5	Concert
6	Fundraiser
7	Lecture
8	Meeting
9	Performance
10	Social
11	Practice/Rehearsal
12	Sub-Committee Meeting

Events

Contains all attributes for an event including facility, meeting date, and projected attendance. References outside tables for its ***typeID*** *and which club the event belongs to.*

```
CREATE TABLE events (
  "eventID" integer NOT NULL,
  "eventTitle" text NOT NULL,
  "meetingDate" date NOT NULL,
  "meetingLocation" text NOT NULL,
  "projectedAttendance" integer NOT NULL,
  "eventTypeID" integer NOT NULL,
  "clubID" character(255) NOT NULL,
  CONSTRAINT events_pkey PRIMARY KEY ("eventID"),
  CONSTRAINT "events_clubID_fkey" FOREIGN KEY ("clubID")
      REFERENCES clubs ("clubID") MATCH SIMPLE,
  CONSTRAINT "events_eventTypeID_fkey" FOREIGN KEY ("eventTypeID")
      REFERENCES "eventTypes" ("typeID") MATCH SIMPLE,
  CONSTRAINT "events_eventID_key" UNIQUE ("eventID")
);
```

Functional Dependencies

eventID → eventTitle, meetingDate, meetingLocation, projectedAttendance, eventTypeID, clubID

eventID	eventTitle	meetingDate	meetingLocation	projectedAttendance	eventTypeID	clubID
1	Winter Concert	2013-01-28	Nelly Goletti Theatre	200	9	CL002
2	General Meeting #1	2013-04-25	HC2023	400	8	CL009
3	Coffeehouse with Bill Cosby	2013-10-11	MU3102	65	3	SPC001
4	Football v Cowboys	2013-09-25	Tenney Stadium	500	2	SPC002
5	General Meeting #2	2013-10-23	DN101	35	8	CL008
6	Game v Vassar	2013-09-5	Gartland Commons	100	2	CL003
7	Fall Festival	2013-10-21	Jazzmans	150	10	CL004
8	General Member Meeting #5	2011-03-10	SC3003	45	8	CL007
9	Cake Night	2013-05-3	DN205	100	6	CL005
10	Trash Pickup	2012-03-15	DN203	25	4	CL007
11	Fall Concert	2013-09-26	Nelly Goletti	250	9	CL002
12	Fall Bingo	2013-10-1	Cabaret	65	10	SPC001

officerPositions

```
CREATE TABLE "officerPositions" (
  "positionID" integer NOT NULL,
  "position" character(255),
  abbr character(2),
  CONSTRAINT "officerPosition_pkey" PRIMARY KEY ("positionID")
);
```

Functional Dependencies

positionID → position, abbr

positionID	position	abbr
1	President	PR
2	Vice President	VP
3	Secretary	SC
4	Other	OT
5	Treasurer	TR
6	Advisor	AV
10	Administrator	AD

officers

```
CREATE TABLE officers (
  cwid integer NOT NULL,
  "position" integer,
  "clubID" character(255) NOT NULL,
  CONSTRAINT officers_pkey PRIMARY KEY (cwid, "clubID"),
  CONSTRAINT officers_cwid_fkey FOREIGN KEY (cwid)
      REFERENCES students (cwid) MATCH SIMPLE,
  CONSTRAINT officers_position_fkey FOREIGN KEY ("position")
      REFERENCES "officerPositions" ("positionID") MATCH SIMPLE,
  CONSTRAINT one_student_per_position UNIQUE (cwid, "position", "clubID")
);
```

Functional Dependencies:

cwid, clubID → position

cwid	position	clubID
10121314	5	CL009
10142585	4	CL001
10142585	1	CL004
10142585	1	SPC002
10167921	4	CL001
10167921	2	CL003
10167921	2	CL004
10177377	3	SPC002
20056143	4	CL002
20067249	1	CL003

students

```
CREATE TABLE students (
  cwid integer NOT NULL,
  "fName" character(255),
  "lName" character(255),
  email character(255),
  CONSTRAINT students_pkey PRIMARY KEY (cwid)
);
```

Functional Dependencies:

cwid → fName, lName, email

cwid	fName	lName	email
10121314	Dewey	Cheatumanhowe	dewey.cheatumanhowe1@marist.edu
10142581	Busta	Rhymes	busta.rhymes1@marist.edu
10142585	David	Gunter	david.gunter1@marist.edu
10156293	Terry	Daktul	terry.daktul1@marist.edu
10156420	Doctor	Dre	doctor.dre1@marist.edu
10161666	Hans	Zoff	hans.zoff1@marist.edu
10164991	Stoop	Kidd	stoop.kidd1@marist.edu
10167921	Justin	Case	justin.case1@marist.edu
10169410	Ron	Jeremy	ron.jeremy1@marist.edu
10169451	Oliver	Klosoff	oliver.klosoff1@marist.edu
10177377	Brontosaurus	Rex	brontosaurus.rex1@marist.edu
20011679	Alan	Labouseur	alan.labouseur@marist.edu
20012395	Tom	Ato	thomas.ato1@marist.edu
20026881	Bart	Ender	bart.ender1@marist.edu
20045671	Miles	Davis	miles.davis1@marist.edu
20056143	Hugh	Jass	hugh.jass1@marist.edu
20056144	Bart	Simpson	bart.simpson1@marist.edu
20056784	Benjamin	Dover	benjamin.dover1@marist.edu
20067249	Will	Call	will.call1@marist.edu
20079152	Izzy	Able	izzy.able1@marist.edu

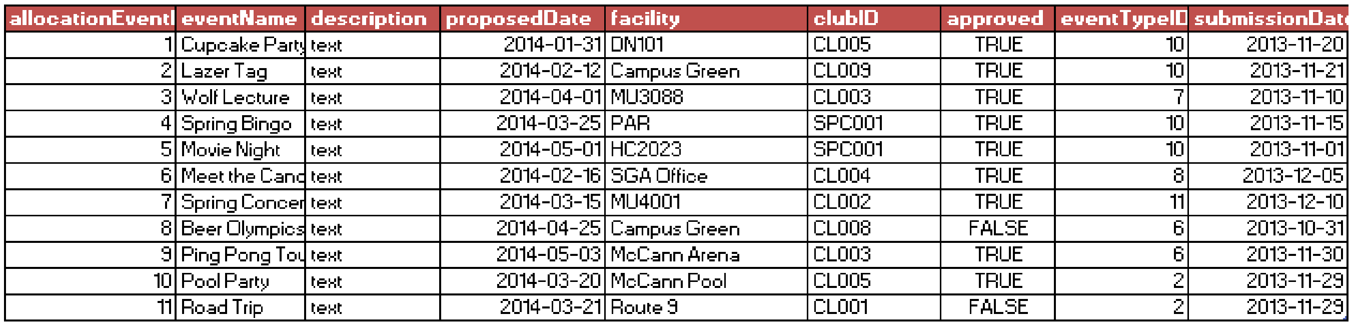
allocationEvents

Allocation events are events that have been submitted to the financial board for approval along with line items, as seen in the ***allocationLineItems*** *table. They are not events that will necessarily end up in the* ***events*** *table.*

```
CREATE TABLE "allocationEvents" (
  "allocationEventID" integer NOT NULL,
  "eventName" character(255) NOT NULL,
  description text,
  "proposedDate" date NOT NULL,
  facility character(255) NOT NULL,
  "clubID" character(15) NOT NULL,
  approved boolean,
  "eventTypeID" integer NOT NULL,
  "submissionDate" date NOT NULL DEFAULT now(),
  CONSTRAINT "allocationEvents_pkey" PRIMARY KEY ("allocationEventID"),
  CONSTRAINT "allocationEvents_clubID_fkey" FOREIGN KEY ("clubID")
      REFERENCES clubs ("clubID") MATCH SIMPLE,
  CONSTRAINT "allocationEvents_eventTypeID_fkey" FOREIGN KEY ("eventTypeID")
      REFERENCES "eventTypes" ("typeID") MATCH SIMPLE
);
```

Functional Dependencies:

allocationEventID → eventName, description, proposedDate, facility, clubID, approved, eventTypeID, submissionDate



allocationEventI	eventName	description	proposedDate	facility	clubID	approved	eventTypeID	submissionDat
1	Cupcake Party	text	2014-01-31	DN101	CL005	TRUE	10	2013-11-20
2	Lazer Tag	text	2014-02-12	Campus Green	CL009	TRUE	10	2013-11-21
3	Wolf Lecture	text	2014-04-01	MU3088	CL003	TRUE	7	2013-11-10
4	Spring Bingo	text	2014-03-25	PAR	SPC001	TRUE	10	2013-11-15
5	Movie Night	text	2014-05-01	HC2023	SPC001	TRUE	10	2013-11-01
6	Meet the Cand	text	2014-02-16	SGA Office	CL004	TRUE	8	2013-12-05
7	Spring Concer	text	2014-03-15	MU4001	CL002	TRUE	11	2013-12-10
8	Beer Olympics	text	2014-04-25	Campus Green	CL008	FALSE	6	2013-10-31
9	Ping Pong Tou	text	2014-05-03	McCann Arena	CL003	TRUE	6	2013-11-30
10	Pool Party	text	2014-03-20	McCann Pool	CL005	TRUE	2	2013-11-29
11	Road Trip	text	2014-03-21	Route 9	CL001	FALSE	2	2013-11-29

allocationLineItems

Line items are specific purchases required of each proposed event. An event can, and should, have multiple line items. Most clubs use these for food purchases, but other purchases can be included here.

```
CREATE TABLE "allocationLineItems" (
  "itemID" integer NOT NULL,
  "itemName" character(255) NOT NULL,
  "itemDesc" text,
  "requestedFunds" numeric(12,2) NOT NULL,
  "eventID" integer NOT NULL,
  CONSTRAINT "allocationLineItems_pkey" PRIMARY KEY ("itemID"),
  CONSTRAINT "allocationLineItems_eventID_fkey" FOREIGN KEY ("eventID")
      REFERENCES "allocationEvents" ("allocationEventID") MATCH SIMPLE
  CONSTRAINT "moneyIsPositive" CHECK ("requestedFunds" > 0::numeric)

);
```

Functional Dependencies:

itemID → itemName, itemDesc, requestedFunds, eventID

itemID	itemName	itemDesc	requestedFunds	eventID
1	Cupcakes	Need 3 dozen cupcakes for the event as food.	250.00	1
2	Tea	Assorted types of tea for the event as drinks	65.00	1
3	Balloons	Balloon decorations	10.00	1
4	Lazer Game	Rental of lazer game from local company	1500.00	2
5	Inflatable Maze	Obstacle course for game	450.00	2
6	Wolf speaker fee	Required fee for speakers	750.00	3
7	Assorted Gift Cards	Prizes for bingo game	115.00	4
8	Movie Rights	Need to pay for rights to show movie	500.00	5
9	Food costs	Popcorn and Drinks for movie	75.00	5
10	Drinks	For refreshment	50.00	6

[table truncated for page layout reasons]

lineItemApprovedFunds

Takes individual line items from ***allocationLineItems*** *and gives administrators the ability to input how much of the requested funds a club will receive for each line item.*

```
CREATE TABLE "LineItemApprovedFunds" (
  "itemID" integer NOT NULL,
  "fundsGiven" numeric(12,2),
  CONSTRAINT "LineItemApprovedFunds_pkey" PRIMARY KEY ("itemID"),
  CONSTRAINT "LineItemApprovedFunds_itemID_fkey" FOREIGN KEY ("itemID")
      REFERENCES "allocationLineItems" ("itemID") MATCH SIMPLE
  CONSTRAINT "moneyIsPositive" CHECK ("fundsGiven" > 0::numeric)

);
```

Functional Dependencies:

itemID → fundsGiven

itemID	fundsGiven
1	200.00
2	65.00
3	10.00
4	1500.00
5	450.00
6	750.00
7	0.00
8	500.00
9	60.00
10	50.00

[table truncated for page layout]

VIEWS

officerDirectory

This report will be helpful when grabbing information about officers for each club. The main use case of this view is for college activities to have a listing, along with contact information, of all club officers. The report will assist administrators in determining who to contact in certain situations, i.e. missing attendance or missing allocation sheets.

```
CREATE OR REPLACE VIEW officerdirectory AS
 SELECT c.name AS "Club", op."position" AS "Position",
    s."fName" AS "First Name", s."lName" AS "Last Name",
    s.email AS "Contact Email", o.cwid, c."clubID" AS "ClubID"
   FROM officers o
   JOIN students s ON s.cwid = o.cwid
   JOIN "officerPositions" op ON op."positionID" = o."position"
   JOIN clubs c ON c."clubID" = o."clubID"
  ORDER BY c."clubID", o."position";
```

clubEvents

This view gives administrators a more human-readable report of all events for a club. Instead of seeing typeIDs and clubIDs, we now see club names and human-readable event types. This makes querying a bit easier in the long-run by removing the need to remember eventType IDs.

```
CREATE OR REPLACE VIEW clubevents AS
 SELECT c.name AS "Club", e."eventTitle" AS "Event", e."meetingDate",
    e."eventID", c."clubID", et."typeName" AS "EventType"
   FROM events e
   JOIN clubs c ON c."clubID" = e."clubID"
   JOIN "eventTypes" et ON e."eventTypeID" = et."typeID"
  ORDER BY e."clubID";
```

club_event_allocation_totals

In order to better collate monetary data, this view collects and sums both the requested allocation funds (aka funds that were propsed for an event) and the given allocation funds (aka the funds that College Activities actually gave a club for an event). The primary purpose of the view is to summarize the financial implications of every event by totaling the cost of each line item associated with an event. The view will give College Activities and clubs an at-a-glance view of how much money was given, in total, to each of its events.

An additional use of this view consists of totaling total funds requested from all clubs and total funds given to all clubs for accounting purposes.

```
CREATE OR REPLACE VIEW club_event_allocation_totals AS
 SELECT DISTINCT ON (l."eventID") l."eventID", ae."clubID", ae."eventName",
    sum(l."requestedFunds") AS "Requested Funds",
    sum(laf."fundsGiven") AS "Funds Given"
   FROM "allocationEvents" ae
   JOIN "allocationLineItems" l ON l."eventID" = ae."allocationEventID"
   JOIN "LineItemApprovedFunds" laf ON laf."itemID" = l."itemID"
  GROUP BY ae."clubID", l."eventID", ae."eventName";
```

STORED PROCEDURES

attendanceForEachEvent

Getting the total attendance for an event is one of the primary functions of this database. To simplify the process of gathering attendance statistics, this function will return the total number of people that attended each event of a specified club, as determined by the ***clubid*** parameter.

```
 CREATE OR REPLACE FUNCTION attendanceforeachevent(IN clubid character varying)

  RETURNS TABLE("Total Attendance" bigint, "Event Name" text) AS
$BODY$
    BEGIN
        RETURN QUERY SELECT count(a.cwid) AS "Total Attendance",
                            e."eventTitle" AS "Event Name"
                     FROM attendance a
                     JOIN events e ON e."eventID" = a."eventID"
                     JOIN clubs c ON c."clubID" = e."clubID"
                     WHERE c."clubID" = clubid
                     GROUP BY e."eventID", e."eventTitle";
    END;
$BODY$
  LANGUAGE plpgsql;
```

listStudentAttendancePerClub

Continuing the trend of reducing complexity when producing reports of attendance, this function returns the total number of events a specific student attended for all applicable clubs. This function does not return rows of clubs for which a student did not swipe in to. The ***cwid*** parameter determines which student we are gathering information for.

```
CREATE OR REPLACE FUNCTION liststudentattendanceperclub(IN studentcwid integer)
  RETURNS TABLE("Events Attended" bigint, "Club Name" character) AS
$BODY$
BEGIN
    RETURN QUERY SELECT count(a.cwid) AS "Events Attended",
                        c.name AS "Club Name"
                 FROM attendance a
                 JOIN events e ON e."eventID" = a."eventID"
                 JOIN clubs c ON c."clubID" = e."clubID"
                 WHERE a.cwid = studentcwid
                 GROUP BY c.name;
END;
$BODY$
  LANGUAGE plpgsql;
```

totalFundsGiven

This function will return the sum of all line item dollar amounts approved for a specific club. The usefulness of this function lies in accounting purposes in that it helps College Activities track where money was flowing out into clubs.

```
CREATE OR REPLACE FUNCTION totalfundsgiven(clubid character varying)
  RETURNS numeric AS
$BODY$
DECLARE
    sumOfGivenFunds numeric;
BEGIN
    SELECT sum("Funds Given")
    FROM club_event_allocation_totals
    WHERE "clubID" = clubid
    INTO sumOfGivenFunds;
    RETURN sumOfGivenFunds;
END;
$BODY$
  LANGUAGE plpgsql;
```

totalFundsRequested

This function is similar to the ***totalFundsGiven*** function, with the exception of returning funds requested, not given. Applications of this function lie in calculating total requested funds and helping a club determine how much money they have requested to better gauge their projected expenses.

```
CREATE OR REPLACE FUNCTION totalfundsrequested(clubid character varying)
  RETURNS numeric AS
$BODY$
DECLARE
    sumOfRequestedFunds numeric;
BEGIN
    SELECT sum("Requested Funds")
    FROM club_event_allocation_totals
    WHERE "clubID" = clubid INTO sumOfRequestedFunds;
    RETURN sumOfRequestedFunds;
END;
$BODY$
  LANGUAGE plpgsql;
```

TRIGGERS

addApprovalEntry

This trigger only activates when a line item is added to ***allocationLineItems***. The trigger automatically creates a new row using the ***itemID*** that was just made.

By automatically making these rows, we reduce the possibility of insert mistakes when creating new rows in *LineItemApprovedFunds*. Instead of manually entering the *itemID*, all the admin needs to do is enter in the approved funds amount that matches with the corresponding *itemID*.

```
CREATE FUNCTION createApprovalEntryForEachItem () RETURNS trigger AS $$

BEGIN
    IF(TG_OP = 'INSERT') THEN
        INSERT INTO "LineItemApprovedFunds" ("itemID")
            VALUES (NEW."itemID");
        RETURN NEW;
    RETURN NULL;
END;

$$ LANGUAGE plpgsql;


CREATE TRIGGER addApprovalEntry
AFTER INSERT
ON "allocationLineItems"
FOR EACH ROW
EXECUTE PROCEDURE createApprovalEntryForEachItem();
```

zeroOutRejectedEventItems

If an allocated event's approval column is set or updated to FALSE, then the function automatically sets all ***fundsGiven*** columns for that event in ***LineItemApprovedFunds*** to 0.00.

This follows the business logic that if an event isn't approved, it won't receive any funds.

```
CREATE FUNCTION zeroOutRejectedEventItems () RETURNS trigger AS $$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        IF (NEW."approved" IS FALSE) THEN
            UPDATE "LineItemApprovedFunds"
            SET "fundsGiven" = 0.00
            WHERE "itemID" IN (
                SELECT ali."itemID"
                FROM "allocationLineItems" ali
                WHERE ali."eventID" =          NEW."allocationEventID");
        END IF;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

[cont. next page]

```
CREATE TRIGGER removeFundsRejectedEvent
AFTER UPDATE
ON "allocationEvents"
FOR EACH ROW
EXECUTE PROCEDURE zeroOutRejectedEventItems();
```

REPORTS

Most Popular Event(s)

The following query generates a report showing which events are most popular (where popularity is determined by number of attendees) through sorting of attendance counts.

```
SELECT
        e."eventTitle" AS "Event Name",
        c."name" AS "Club",
        count(a.cwid) AS "Total Attendance"
FROM attendance a
JOIN "events" e ON e."eventID" = a."eventID"
JOIN "clubs" c ON  c."clubID" = e."clubID"
group by e."eventTitle", c."name", a."eventID"
order by "Total Attendance" DESC
```

Most Popular Event Types

Like the Most Popular Event report, this report gives administrators insight into what types of events are most popular with students. Popularity is again gauged by the total number of attendees.

```
SELECT
        et."typeName" AS "Event Type",
        count(a.cwid) AS "Total Attendance"
FROM attendance a
JOIN "events" e ON e."eventID" = a."eventID"
JOIN "eventTypes" et ON et."typeID" = e."eventTypeID"
group by et."typeID", et."typeName"
order by "Total Attendance" DESC
```

Total Funds Requested & Given Per Club (w/Calculated Difference)

This report gives administrators details regarding how much money a club as a whole requested, how much money the club received, and a calculated difference between the two (“Difference” column).

```
SELECT
         c.name,
         sum("Requested Funds") as "Total Requested",
         sum("Funds Given") as "Funds Given",
         ((sum("Requested Funds") - sum("Funds Given"))* -1) AS "Difference"
FROM club_event_allocation_totals z
JOIN clubs c ON c."clubID" = z."clubID"
GROUP BY c.name
ORDER BY "Total Requested" DESC
```

SECURITY

College_activities_staff

College activities staff are like administrators for the database. They are able to modify, create, and delete data in most tables. They cannot modify database, structure however. This role is responsible for managing all the clubs on campus, and as such it can modify data and data sets as it sees fit.

```
CREATE ROLE college_activities_staff;

REVOKE ALL PRIVILEGES ON "LineItemApprovedFunds" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "allocationEvents" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "allocationLineItems" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "attendance" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "clubs" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "councils" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "eventTypes" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "events" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "officerPositions" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "officers" FROM college_activities_staff;
REVOKE ALL PRIVILEGES ON "students" FROM college_activities_staff;

GRANT SELECT, INSERT, UPDATE, DELETE ON "LineItemApprovedFunds" TO
college_activities_staff;
GRANT SELECT, INSERT, UPDATE, DELETE ON "allocationEvents" TO
college_activities_staff;
GRANT SELECT, INSERT, UPDATE, DELETE ON "allocationLineItems" TO
college_activities_staff;
GRANT SELECT, INSERT, UPDATE, DELETE ON "attendance" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE ON "clubs" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE ON "councils" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE ON "eventTypes" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE, DELETE ON "events" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE ON "officerPositions" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE, DELETE ON "officers" TO college_activities_staff;
GRANT SELECT, INSERT, UPDATE ON "students" TO college_activities_staff;
```

it_council

IT Council will serve as the database administrators for the Club Dash database. Because the members of IT Council rotate, especially as members graduate or choose to no longer participate, this “administrator” role was made a group to better manage changing members.

```
CREATE ROLE it_council;

REVOKE ALL PRIVILEGES ON "LineItemApprovedFunds" FROM it_council;
REVOKE ALL PRIVILEGES ON "allocationEvents" FROM it_council;
REVOKE ALL PRIVILEGES ON "allocationLineItems" FROM it_council;
REVOKE ALL PRIVILEGES ON "attendance" FROM it_council;
REVOKE ALL PRIVILEGES ON "clubs" FROM it_council;
REVOKE ALL PRIVILEGES ON "councils" FROM it_council;
REVOKE ALL PRIVILEGES ON "eventTypes" FROM it_council;
REVOKE ALL PRIVILEGES ON "events" FROM it_council;
REVOKE ALL PRIVILEGES ON "officerPositions" FROM it_council;
REVOKE ALL PRIVILEGES ON "officers" FROM it_council;
REVOKE ALL PRIVILEGES ON "students" FROM it_council;

GRANT ALL ON "LineItemApprovedFunds" TO it_council;
GRANT ALL ON "allocationEvents" TO it_council;
GRANT ALL ON "allocationLineItems" TO it_council;
GRANT ALL ON "attendance" TO it_council;
GRANT ALL ON "clubs" TO it_council;
GRANT ALL ON "councils" TO it_council;
GRANT ALL ON "eventTypes" TO it_council;
GRANT ALL ON "events" TO it_council;
GRANT ALL ON "officerPositions" TO it_council;
GRANT ALL ON "officers" TO it_council;
GRANT ALL ON "students" TO it_council;
```

club_officers

Club officers are in charge of maintaining the club(s) they are assigned to. As a result, these role participants can only create/edit data when dealing with allocation submission, event creation, attendance, and creating new officers. All other data is only selectable by this group in order to protect data integrity.

```
CREATE ROLE club_officers;

REVOKE ALL PRIVILEGES ON "LineItemApprovedFunds" FROM club_officers;
REVOKE ALL PRIVILEGES ON "allocationEvents" FROM club_officers;
REVOKE ALL PRIVILEGES ON "allocationLineItems" FROM club_officers;
REVOKE ALL PRIVILEGES ON "attendance" FROM club_officers;
REVOKE ALL PRIVILEGES ON "clubs" FROM club_officers;
REVOKE ALL PRIVILEGES ON "councils" FROM club_officers;
REVOKE ALL PRIVILEGES ON "eventTypes" FROM club_officers;
REVOKE ALL PRIVILEGES ON "events" FROM club_officers;
REVOKE ALL PRIVILEGES ON "officerPositions" FROM club_officers;
REVOKE ALL PRIVILEGES ON "officers" FROM club_officers;
REVOKE ALL PRIVILEGES ON "students" FROM club_officers;

GRANT SELECT ON "LineItemApprovedFunds" TO club_officers;
GRANT SELECT, INSERT, UPDATE ON "allocationEvents" TO club_officers;
GRANT SELECT, INSERT, UPDATE ON "allocationLineItems" TO club_officers;
GRANT SELECT, INSERT ON "attendance" TO club_officers;
GRANT SELECT ON "clubs" TO club_officers;
GRANT SELECT ON "councils" TO club_officers;
GRANT SELECT ON "eventTypes" TO club_officers;
GRANT SELECT, INSERT, UPDATE ON "events" TO club_officers;
GRANT SELECT ON "officerPositions" TO club_officers;
GRANT SELECT, INSERT, UPDATE ON "officers" TO club_officers;
GRANT SELECT ON "students" TO club_officers;
```

student_role

This role restricts student users, who are not club officers, to simply looking up information about themselves and the events they have attended. This role will only be used to provide an outside interface for ClubDash.

```
CREATE ROLE student_role;

REVOKE ALL PRIVILEGES ON "LineItemApprovedFunds" FROM student_role;
REVOKE ALL PRIVILEGES ON "allocationEvents" FROM student_role;
REVOKE ALL PRIVILEGES ON "allocationLineItems" FROM student_role;
REVOKE ALL PRIVILEGES ON "attendance" FROM student_role;
REVOKE ALL PRIVILEGES ON "clubs" FROM student_role;
REVOKE ALL PRIVILEGES ON "councils" FROM student_role;
REVOKE ALL PRIVILEGES ON "eventTypes" FROM student_role;
REVOKE ALL PRIVILEGES ON "events" FROM student_role;
REVOKE ALL PRIVILEGES ON "officerPositions" FROM student_role;
REVOKE ALL PRIVILEGES ON "officers" FROM student_role;
REVOKE ALL PRIVILEGES ON "students" FROM student_role;

GRANT SELECT ON "attendance" TO student_role;
GRANT SELECT ON "clubs" TO student_role;
GRANT SELECT ON "events" TO student_role;
GRANT SELECT ON "students" TO student_role;
```

IMPLEMENTATION NOTES

This system was designed to reduce the paperwork and use of Excel forms when managing clubs and club allocations. With that in mind, some implementation assumptions were made:

- Dates will be formatted by the application interface before being submitted to the database
- Dollar amounts are in USD (seeing as how Marist College is in the United States).
    - The input for money will be sanitized and reformatted appropriately by the application before being submitted to the database
- For officers: two students ARE able to hold one position (i.e. as co-presidents), so a constraint for this case was not placed on the table.
    - If it was, then we would place co-presidents, co-treasurers, etc. as "other" in terms of officer type categorization.
- No constraint was made on the events table for events occurring on the same date at the same place. This isn't a scheduling system, just a record-keeping database.
- For allocations, we assumed that clubs should be able to edit their allocation submissions. Though this fails to fully reflect the current system of paper allocation submissions, we felt that access can be restricted via the application by checking current/submission dates against a cut-off date.
- Student information is grabbed from the Marist LDAP server via the application interface.

KNOWN ISSUES

- Currently there is no way to “log” who is doing what in the database. This creates an accountability gap and needs to be rectified to help administrators figure out who did what action in case of an error.

- There is no “serialized” data type implemented for itemIDs, eventIDs, allocationEventIDs, and so on. This could potentially create primary key collision.

FUTURE ENHANCEMENTS

- Implement a logging trigger than takes in a CWID and determines an action (update, insert, delete) to attach to that CWID with a timestamp. This will be stored in another table for querying purposes.

- Create an “archive” for officers. This archive table will take officers that were deleted (from when their term ended or they were removed for another reason) and store the information in an officers_archive table.

- Add semester fields (i.e. Fall/Spring) that are either calculated by the application or the database. This can help reduce complexity when creating attendance reports for the semester.

- Create another level of the allocation tables that will handle “re-allocations.”