**College Schedule Planner**

**Group 10**

David Liao, Sid Panjwani, Sashank Nutakki, Bowen Jin

**Introduction**

University students often have reasons to request information regarding the schedules of other students and professors. Rapid access to the academic schedules of the entire student body as well as information related to the teaching faculty offers capabilities such as the ability to efficiently minimize time conflicts when scheduling daytime events or meetings, the ability to quickly request aid from students within the same class sections, and the ability to conveniently determine office hour times that do not conflict with one’s schedule.

Our web application will support lookup of individual schedule information on a protected basis, enrollment by course section, instructor information, and lab schedules. Individuals can choose to provide additional information to the database, such as student organizations they’re participating in and availability outside of class.

**Supported Functionality**

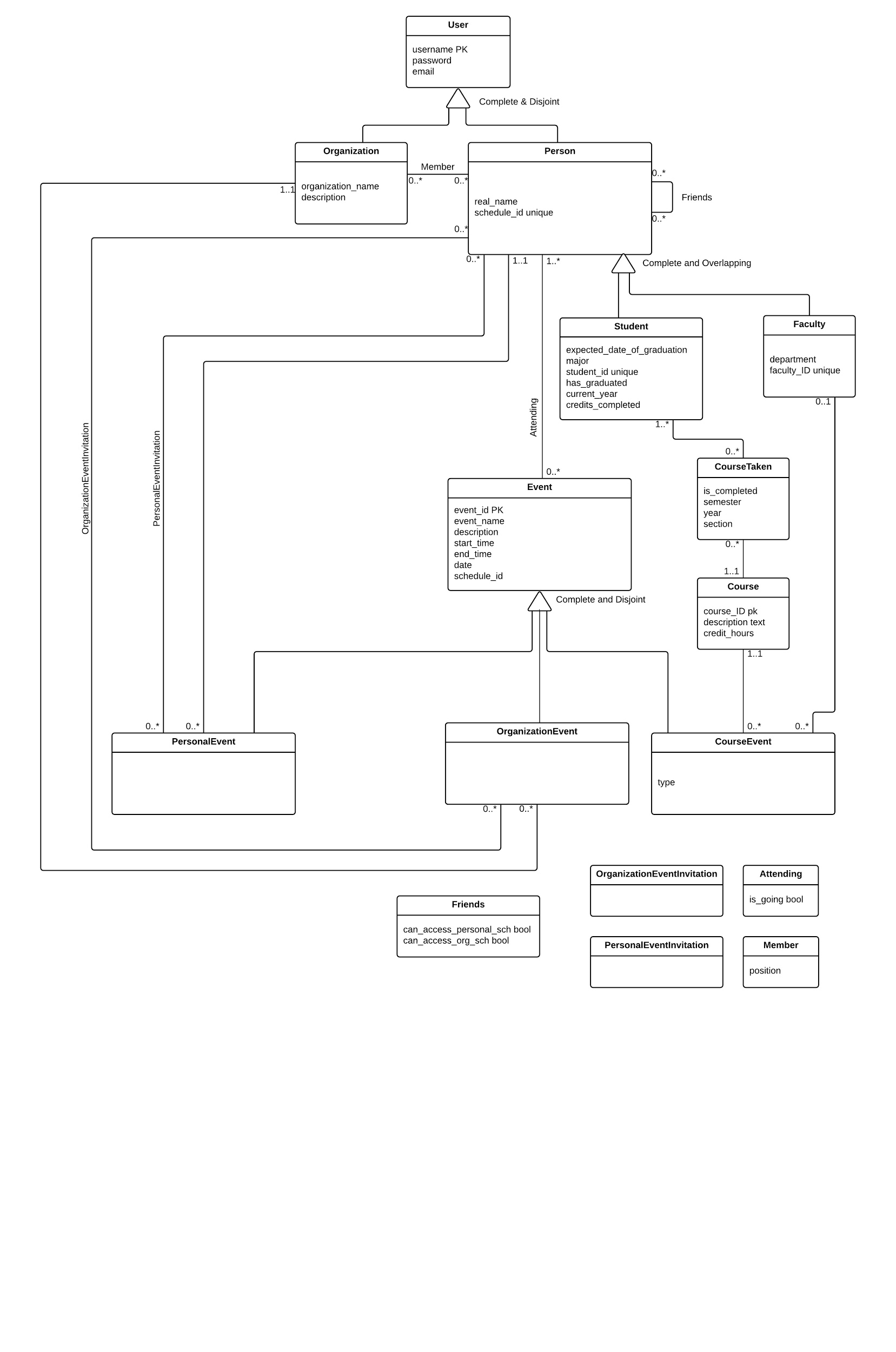
* Information can only be queried from user accounts which are represented within the database. Users with and without accounts may query information from the database.
* Users may grant protected status to personal events and organization events in the database related to themselves, thereby restricting some access to their schedule. Access may be granted to outside parties according to the user’s discretion.
* Users may look up the names and contact information of all students enrolled within a particular class section.
* Users may request the entire weekly class schedule – or class schedule filtered by a particular weekday or time block – of another student (only non-protected information will appear).
* Users may request the class names, class IDs, class sections, professors, and open time blocks in common between two (or more) students.
* Users can provide additional information related to themselves for storage in the database. This information includes additional activities the user is participating in, the specific times when they’re available, etc.

**Description of Major database design elements:**

In our database, we wanted to reduce redundancy and implement a logical and easy to understand schema. To accomplish this, we structured our project around three main table types: Users, Events, and Event Invitations. Users represent the users of the application, such as students, faculty members, and organizations, and includes information such as username, passwords, and emails. Events are the main type of data that a user can create or can respond to. These events can be shared with other users using Event Invitations. Every User must either be a Person or an Organization. A Person can create PersonalEvents, while Organization can create OrganizationEvents. A person can be either a Faculty, a Student, or both, in the example of a teacher’s assistant, with different abilities associated with each of them. We have the Courses table which we use to provide static data about courses that are associated with each CourseEvent. Having a separate table for Courses allows each CourseEvent to share the same data about a Course to prevent redundancy. We used a CoursesTaken table to link Students and Courses while providing additional information about each relationship between a student and a course, such as year taken and section to allow for historical data.

We realized that people wouldn’t want to share all events with everybody and thus to implement our privacy controls we used a Friend table that lists a set of users that are friends with each other along with what one user shares with another user. The friend1 and friend2 are foreign keys of Person. The biggest change we made from this in class was that just because a friend1 chooses to share some information with friend2, does not mean that is reciprocated. The can\_access\_personal\_sch and can\_access\_organization\_sch may be different between the two pairs to allow users to modify the sharing settings for each of their friendships. We came in with the assumption for our program that all students would be okay making their class schedule public.

To facilitate sharing of events among users, we allow users to send invites for their own events. A user could choose to ignore or accept this invitation using the isGoing Boolean.



**Assertions**

/\* Ensures that every User is a Person or an Organization \*/

/\* Enforces completeness \*/

CREATE ASSERTION UserCompleteCoverage (

CHECK (NOT EXISTS ((SELECT U.username FROM User U EXCEPT

SELECT P.username FROM Person P) EXCEPT

SELECT O.username FROM Organization O)

)

);

/\* Ensures that a User is exclusively a Person or an Organization \*/

/\* Enforces disjointedness \*/

CREATE ASSERTION NoOverlapBetweenPersonAndOrganization (

CHECK (NOT EXISTS (SELECT \* FROM Person P, Organization O

WHERE P.username = O.username)

)

);

/\* Ensures that every Event is an OrganizationEvent, PersonalEvent, or CourseEvent \*/

/\* Enforces completeness \*/

CREATE ASSERTION EventCompleteCoverage (

CHECK (NOT EXISTS (((SELECT E.event\_id FROM Event E EXCEPT

SELECT OE.event\_id FROM OrganizationEvent OE) EXCEPT

SELECT PE.event\_id FROM PersonalEvent PE) EXCEPT

SELECT CE.event\_id FROM CourseEvent CE)

)

);

/\* Ensures that an event is exclusively an OrganizationEvent, PersonalEvent, or CourseEvent \*/

/\* Enforces disjointedness \*/

CREATE ASSERTION NoOverlapBetweenOPCEvents (

CHECK (NOT EXISTS (SELECT \* FROM OrganizationEvent OE, PersonalEvent PE, CourseEvent CE

WHERE OE.event\_id = PE.event\_id OR OE.event\_id = CE.event\_id

OR PE.event\_id = CE.event\_id)

)

);

/\* Ensures that every Person is a Student or Faculty \*/

/\* Disjointedness is not enforced because TAs are classified as Students and Faculty \*/

CREATE ASSERTION PersonCompleteCoverage (

CHECK (NOT EXISTS ((SELECT P.username FROM Person P EXCEPT

SELECT username FROM Student S) EXCEPT

SELECT F.username FROM Faculty F)

)

);

/\* Ensures each CourseTaken is associated with at least one Student \*/

CREATE ASSERTION CourseTakenParticipatesStudent (

CHECK (NOT EXISTS (SELECT \*

FROM CourseTaken CT

WHERE CT.student\_id NOT IN

(SELECT S.student\_id FROM Student S)

)

);

**SQL Implementation**

/\* --------------------DROP statements-------------------- \*/

DROP TABLE IF EXISTS Attending;

DROP TABLE IF EXISTS Course;

DROP TABLE IF EXISTS CourseEvent;

DROP TABLE IF EXISTS CourseTaken;

DROP TABLE IF EXISTS Event;

DROP TABLE IF EXISTS Faculty;

DROP TABLE IF EXISTS Friend;

DROP TABLE IF EXISTS Member;

DROP TABLE IF EXISTS Organization;

DROP TABLE IF EXISTS OrganizationEvent;

DROP TABLE IF EXISTS OrganizationEventInvitation;

DROP TABLE IF EXISTS Person;

DROP TABLE IF EXISTS PersonalEvent;

DROP TABLE IF EXISTS PersonalEventInvitation;

DROP TABLE IF EXISTS Student;

DROP TABLE IF EXISTS User;

DROP VIEW IF EXISTS OrgMembers;

DROP VIEW IF EXISTS GraduatedStudents;

DROP VIEW IF EXISTS ClassMembers;

DROP VIEW IF EXISTS AccessiblePersonalEvents;

DROP TRIGGER IF EXISTS DeleteLastStudentOfCourseTaken;

DROP TRIGGER IF EXISTS DeleteEvent;

DROP TRIGGER IF EXISTS InvitedToEvent;

DROP TRIGGER IF EXISTS DeleteOnOrgMembers

/\* --------------------Tables-------------------- \*/

/\* Parent class of Person and Organization \*/

CREATE TABLE User (

username CHAR(30),

password CHAR(30),

email CHAR(40),

PRIMARY KEY (username)

);

/\* Subclass of User \*/

/\* Parent class of Student and Faculty \*/

CREATE TABLE Person (

username CHAR(30),

real\_name CHAR(50),

schedule\_id CHAR(30),

UNIQUE (schedule\_id),

PRIMARY KEY (username),

FOREIGN KEY (username) REFERENCES User

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of User \*/

CREATE TABLE Organization (

username CHAR(30),

organization\_name CHAR(50),

organization\_id CHAR(30),

UNIQUE (organization\_id),

PRIMARY KEY (username),

FOREIGN KEY (username) REFERENCES User

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Association class between Person and Organization \*/

/\* A Person is a Member of an Organization \*/

CREATE TABLE Member (

position CHAR(30),

member\_username CHAR(30),

organization\_username CHAR(30),

PRIMARY KEY (member\_username, organization\_username),

FOREIGN KEY (member\_username) REFERENCES Person (username)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (organization\_username) REFERENCES Organization (username)

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Association class between Person and Person \*/

/\* A Person is a Friend of some other Person \*/

/\* A friend pair (F1, F2) indicates that F1 can(not) access the schedules of F2 \*/

/\* A friend pair (F1, F2) in Friend does not imply (F2, F1) also in Friend \*/

CREATE TABLE Friend (

friend1 CHAR(30),

friend2 CHAR(30),

can\_access\_personal\_sch BOOLEAN,

can\_access\_organization\_sch BOOLEAN,

PRIMARY KEY (friend1, friend2),

FOREIGN KEY (friend1) REFERENCES Person (username)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (friend2) REFERENCES Person (username)

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of Person \*/

CREATE TABLE Faculty (

username CHAR(30),

faculty\_id CHAR(30),

department CHAR(30),

UNIQUE (faculty\_id),

PRIMARY KEY (username),

FOREIGN KEY (username) REFERENCES Person

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of Person \*/

/\* current year: 1 indicates first year, 2 indicates second year, etc. \*/

CREATE TABLE Student (

username CHAR(30),

student\_id CHAR(30),

is\_graduate\_student BOOL,

expected\_date\_of\_graduation DATE,

has\_graduated BOOL,

major CHAR(30),

current\_year INTEGER,

credits\_completed INTEGER,

UNIQUE (student\_id),

PRIMARY KEY (username),

FOREIGN KEY (username) REFERENCES Person

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Parent class of PersonalEvent, OrganizationEvent, and CourseEvent \*/

CREATE TABLE Event (

event\_id CHAR(30),

event\_name CHAR(100),

description CHAR(500),

start\_time TIME,

end\_time TIME,

event\_date DATE,

PRIMARY KEY (event\_id)

);

/\* Association class between Person and Attending\*/

/\* A Person indicates s/he is Attending an Event \*/

CREATE TABLE Attending (

is\_going BOOLEAN,

username CHAR(30),

event\_id CHAR(30),

PRIMARY KEY (username, event\_id),

FOREIGN KEY (username) REFERENCES Person

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (event\_id) REFERENCES Event

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of Event \*/

CREATE TABLE PersonalEvent (

event\_id CHAR(30),

event\_creator CHAR(30) NOT NULL,

PRIMARY KEY (event\_id),

FOREIGN KEY (event\_id) REFERENCES Event

ON DELETE CASCADE ON UPDATE CASCADE

FOREIGN KEY (event\_creator) REFERENCES Person (username)

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of Event \*/

/\* organization\_id is identifier for event creator \*/

CREATE TABLE OrganizationEvent (

event\_id CHAR(30),

organization\_id CHAR(30) NOT NULL,

PRIMARY KEY (event\_id),

FOREIGN KEY (event\_id) REFERENCES Event

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (organization\_id) REFERENCES Organization

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* A Course is not uniquely identified by section, teaching faculty, etc. \*/

CREATE TABLE Course (

course\_id CHAR(30),

description CHAR(1000),

credit\_hours INTEGER,

PRIMARY KEY (course\_id)

);

/\* Provides historical record of all courses taken by all students \*/

/\* Semester can be ‘SPRING’, ‘FALL’, or ‘SUMMER’ \*/

CREATE TABLE CourseTaken (

is\_completed BOOLEAN,

semester CHAR(10),

year INTEGER,

section INTEGER,

student\_id CHAR(30) NOT NULL,

course\_id CHAR(30) NOT NULL,

PRIMARY KEY (student\_id, course\_id, semester, year, section),

FOREIGN KEY (student\_id) REFERENCES Student (student\_id)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (course\_id) REFERENCES Course

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Subclass of Event \*/

/\* type is a short description i.e. lecture, office hours, assessment, etc. \*/

CREATE TABLE CourseEvent (

event\_id CHAR(30),

type CHAR(30),

course\_id CHAR(30),

section INTEGER,

faculty\_id CHAR(30) NOT NULL,

PRIMARY KEY (event\_id),

FOREIGN KEY (event\_id) REFERENCES Event

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (course\_id, section) REFERENCES Course

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (faculty\_id) REFERENCES Faculty

ON DELETE SET NULL ON UPDATE CASCADE

);

/\* Association class between Person and PersonalEvent \*/

/\* A Person receives a PersonalEventInvitation for a PersonalEvent \*/

CREATE TABLE PersonalEventInvitation (

username CHAR(30),

event\_id CHAR(30),

PRIMARY KEY (username, event\_id),

FOREIGN KEY (username) REFERENCES Person

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (event\_id) REFERENCES PersonalEvent

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* Association class between Person and OrganizationEvent \*/

/\* A Person receives an OrganizationEventInvitation for an OrganizationEvent \*/

CREATE TABLE OrganizationEventInvitation (

username CHAR(30),

event\_id CHAR(30),

PRIMARY KEY (username, event\_id),

FOREIGN KEY (Username) REFERENCES Person

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (event\_id) REFERENCES OrganizationEvent

ON DELETE CASCADE ON UPDATE CASCADE

);

/\* -------------------- Views -------------------- \*/

/\* Primary author: Sidhanth Panjwani \*/

/\* creates a view of all persons involved in an organization \*/

CREATE VIEW OrgMembers AS

SELECT O.organization\_name, P.real\_name, M.position, P.username, U.email

FROM User U, Person P, Member M, Organization O

WHERE P.username = M.member\_username

AND O.username = M.organization\_username

AND P.username = U.username;

/\* Primary author: David Liao \*/

/\* creates a view of all graduated students within the database \*/

CREATE VIEW GraduatedStudents AS

SELECT \* FROM Student

WHERE has\_graduated IS 'TRUE';

/\* Primary author: Sashank Nutakki \*/

/\* creates a view of currently enrolled students and the classes they're currently taking \*/

CREATE VIEW ClassMembers AS

SELECT P.real\_name, U.email, CT.course\_id, CT.section

FROM User U, Person P, Student S, CourseTaken CT, Course C

WHERE U.username = P.username

AND P.username = S.username

AND S.student\_id = CT.student\_id

AND CT.is\_completed IS 'FALSE'

AND C.course\_ID = CT.course\_ID;

/\* Primary author: Bowen Jin \*/

/\* creates a view of all people who can access another person's personal schedule \*/

CREATE VIEW AccessiblePersonalEvents AS

SELECT User.username AS accessible\_by, Event.event\_name, Event.description, Event.start\_time, Event.end\_time, Event.event\_date

FROM User, Person, Friend, PersonalEvent, Event

WHERE User.username = Person.username

AND Person.username = Friend.friend1

AND Friend.can\_access\_personal\_sch IS 'TRUE'

AND PersonalEvent.event\_id = Event.event\_id;

/\* -------------------- Populate tables with data -------------------- \*/

/\* User \*/

INSERT INTO User VALUES('amy\_user', 'amy\_pass', 'amy@vanderbilt.edu');

INSERT INTO User VALUES('ben\_user', 'ben\_pass', 'ben@vanderbilt.edu');

INSERT INTO User VALUES('cam\_user', 'cam\_pass', 'cam@vanderbilt.edu');

INSERT INTO User VALUES('dom\_user', 'dom\_pass', 'dom@vanderbilt.edu');

INSERT INTO User VALUES('earl\_user', 'earl\_pass', 'earl@vanderbilt.edu');

INSERT INTO User VALUES('faye\_user', 'faye\_pass', 'faye@vanderbilt.edu');

INSERT INTO User VALUES('gary\_user', 'gary\_pass', 'gary@vanderbilt.edu');

INSERT INTO User VALUES('hugo\_user', 'hugo\_pass', 'hugo@vanderbilt.edu');

INSERT INTO User VALUES('david\_user', 'david\_pass', 'david@vanderbilt.edu');

INSERT INTO User VALUES('org1\_user', 'org1\_pass', 'org1@vanderbilt.edu');

INSERT INTO User VALUES('org2\_user', 'org2\_pass', 'org2@vanderbilt.edu');

INSERT INTO User VALUES('org3\_user', 'org3\_pass', 'org3@vanderbilt.edu');

INSERT INTO User VALUES('grad21','grad21p','grad21e');

INSERT INTO User VALUES('grad31','grad31p','grad31e');

INSERT INTO User VALUES('grad41','grad41p','grad41e');

INSERT INTO User VALUES('grad51','grad51p','grad51e');

INSERT INTO User VALUES('grad61','grad61p','grad61e');

/\* Person \*/

INSERT INTO Person VALUES('amy\_user', 'amy\_real\_name', 'amy1001');

INSERT INTO Person VALUES('ben\_user', 'ben\_real\_name', 'ben1001');

INSERT INTO Person VALUES('cam\_user', 'cam\_real\_name', 'cam1001');

INSERT INTO Person VALUES('dom\_user', 'dom\_real\_name', 'dom1001');

INSERT INTO Person VALUES('earl\_user', 'earl\_real\_name', 'earl1001');

INSERT INTO Person VALUES('faye\_user', 'faye\_real\_name', 'faye1001');

INSERT INTO Person VALUES('gary\_user', 'gary\_real\_name', 'gary1001');

INSERT INTO Person VALUES('hugo\_user', 'hugo\_real\_name', 'hugo1001');

INSERT INTO Person VALUES('david\_user', 'david\_real\_name', 'david1001');

INSERT INTO Person VALUES('grad21','grad21p','grad21e');

INSERT INTO Person VALUES('grad31','grad31p','grad31e');

INSERT INTO Person VALUES('grad41','grad41p','grad41e');

INSERT INTO Person VALUES('grad51','grad51p','grad51e');

INSERT INTO Person VALUES('grad61','grad61p','grad61e');

/\* Organization \*/

INSERT INTO Organization VALUES('org1\_user', 'DBMS appreciation club', 'org1\_1001');

INSERT INTO Organization VALUES('org2\_user', 'Weenie Hut General', 'org2\_1001');

INSERT INTO Organization VALUES('org3\_user', 'Super Weenie Hut General', 'org3\_1001');

/\* Member \*/

INSERT INTO Member VALUES('president', 'earl\_user', 'org1\_user');

INSERT INTO Member VALUES('treasurer', 'faye\_user', 'org1\_user');

/\* Friend \*/

INSERT INTO Friend VALUES('amy\_user', 'ben\_user', 'TRUE', 'TRUE');

INSERT INTO Friend VALUES('ben\_user', 'amy\_user', 'FALSE', 'TRUE');

/\* Student \*/

INSERT INTO Student VALUES('amy\_user', 'amy\_student1001', 'FALSE', '05/20/2020','FALSE', 'Computer Science', 4, 300);

INSERT INTO Student VALUES('ben\_user', 'ben\_student1001', 'FALSE', '05/20/2024', 'FALSE', 'Biology', 1, 45);

INSERT INTO Student VALUES('cam\_user', 'cam\_student1001', 'FALSE', '05/20/2021','FALSE', 'Computer Science', 3, 2);

INSERT INTO Student VALUES('dom\_user', 'dom\_student1001', 'FALSE', '05/20/2017','TRUE', 'Biology', 4, 50);

INSERT INTO Student VALUES('earl\_user', 'earl\_student1001', 'TRUE', '05/20/2022','FALSE', 'Economics', 1, 60);

INSERT INTO Student VALUES('faye\_user', 'faye\_student1001', 'FALSE', '05/20/2022', 'FALSE', 'Economics', 1, 60);

INSERT INTO Student VALUES('david\_user', 'david\_student1001', 'FALSE', '4/20/2017', 'TRUE', 'Chemistry', 2, 55);

INSERT INTO Student VALUES('grad21','grad22','TRUE','01/02/2020','FALSE', 'Computer Science',1,20);

INSERT INTO Student VALUES('grad31','grad32','TRUE','01/02/2020','FALSE', 'Computer Science',1,20);

INSERT INTO Student VALUES('grad41','grad42','TRUE','01/02/2020','FALSE', 'Computer Science',1,20);

INSERT INTO Student VALUES('grad51','grad52','TRUE','01/02/2020','FALSE', 'Computer Science',1,20);

INSERT INTO Student VALUES('grad61','grad62','TRUE','01/02/2020','FALSE', 'Computer Science',1,20);

/\* Faculty \*/

INSERT INTO Faculty VALUES('cam\_user', 'cam\_faculty1001', 'Computer Science');

INSERT INTO Faculty VALUES('dom\_user', 'dom\_faculty1001', 'Biology');

INSERT INTO Faculty VALUES('earl\_user', 'earl\_faculty1001', 'Economics');

INSERT INTO Faculty VALUES('faye\_user', 'faye\_faculty1001', 'Economics');

/\* Course \*/

INSERT INTO Course VALUES('course1001', 'Intro to DBMS', 3);

INSERT INTO Course VALUES('course1002', 'Extreme DBMS', 6);

INSERT INTO Course VALUES('gradCourse1', 'Grad Course', 4);

/\* CourseTaken \*/

INSERT INTO CourseTaken VALUES('TRUE','SPRING','2019',3,'grad22','gradCourse1');

INSERT INTO CourseTaken VALUES('TRUE','SPRING','2019',3,'grad32','gradCourse1');

INSERT INTO CourseTaken VALUES('TRUE','SPRING','2019',3,'grad42','gradCourse1');

INSERT INTO CourseTaken VALUES('TRUE','SPRING','2019',3,'grad52','gradCourse1');

INSERT INTO CourseTaken VALUES('TRUE','SPRING','2019',3,'grad62','gradCourse1');

/\* Event \*/

INSERT INTO Event VALUES('event1001', 'breakfast', 'We gon eat breakfast, my dude', '7:00:00 AM', '7:30:00 AM', '01/01/2020');

INSERT INTO Event VALUES('event1002', 'buy lamps', 'We gon buy lamps, my dude', '3:00:00 PM', '5:00:00 PM', '01/02/2020');

INSERT INTO Event VALUES('event1003', 'lunch', 'im hungry', '12:10:00 PM', '1:00:00 PM', '01/20/2020');

INSERT INTO Event VALUES('org1event1', 'databasePractice', 'you guys need to practice more', '3:00:00 PM', '5:00:00 PM', '01/10/2020');

/\* PersonalEvent \*/

INSERT INTO PersonalEvent VALUES('event1001', 'cam\_user');

INSERT INTO PersonalEvent VALUES('event1002', 'faye\_user');

/\* OrganizationEvent \*/

INSERT INTO OrganizationEvent VALUES('org1event1', 'org1\_user');

/\* Attending \*/

INSERT INTO Attending VALUES('TRUE', 'amy\_user', 'event1001');

INSERT INTO Attending VALUES('TRUE', 'ben\_user', 'event1001');

INSERT INTO Attending VALUES('TRUE', 'cam\_user', 'event1002');

INSERT INTO Attending VALUES('TRUE', 'dom\_user', 'event1002');

/\* CourseTaken \*/

INSERT INTO CourseTaken VALUES('TRUE', 'SPRING', 2017, 1, 'amy\_student1001', 'course1001');

INSERT INTO CourseTaken VALUES('TRUE', 'SPRING', 2020, 2, 'ben\_student1001', 'course1001');

INSERT INTO CourseTaken VALUES('FALSE', 'SPRING', 2019, 1, 'cam\_student1001', 'course1001');

INSERT INTO CourseTaken VALUES('FALSE', 'SPRING', 2019, 1, 'dom\_student1001', 'course1001');

INSERT INTO CourseTaken VALUES('TRUE', 'FALL', 2017, 1, 'earl\_student1001', 'course1002');

INSERT INTO CourseTaken VALUES('TRUE', 'FALL', 2017, 1, 'faye\_student1001', 'course1002');

INSERT INTO CourseTaken VALUES('FALSE', 'SPRING', 2017, 1, 'faye\_student1001', 'course1001');

/\* -------------------- Queries -------------------- \*/

/\* Primary author: Sashank Nutakki \*/

/\* returns the names and usernames of everyone attending a particular event \*/

SELECT P.real\_name, P.username

FROM Person P, Event E, Attending A

WHERE P.username = A.username

AND A.event\_id = E.event\_id

AND E.event\_id = 'event1002';

/\* Primary author: Sashank Nutakki \*/

/\* returns the majors and number of students in each major who have taken or

are taking a particular course, where the number of students who have taken

the course in question exceeds 1 \*/

SELECT S.major, COUNT(DISTINCT S.student\_id)

FROM Student S, CourseTaken CT, Course C

WHERE S.student\_ID = CT.student\_ID

AND CT.course\_ID = C.course\_ID

AND C.course\_ID = 'course1001'

GROUP BY (S.major)

HAVING COUNT(S.student\_id) > 1;

/\* Primary author: Sashank Nutakki \*/

/\* returns all first year undergraduates who are members within an organization \*/

SELECT O.username, O.email

FROM Student S, OrgMembers O

WHERE S.username = O.username

AND S.is\_graduate\_student IS 'FALSE'

AND S.current\_year = 1;

/\* Primary author: Bowen Jin \*/

/\* returns all graduated students within majors currently having

fewer than 20 students \*/

SELECT GS.username, GS.major

FROM GraduatedStudents GS

GROUP BY GS.major

HAVING COUNT(GS.student\_id) < 20;

/\* Primary author: Sidhanth Panjwani \*/

/\* returns all the personal events of a user \*/

SELECT Event.event\_name, Event.description, Event.start\_time, Event.end\_time, Event.event\_date

FROM Event NATURAL JOIN PersonalEvent NATURAL JOIN Attending

WHERE Attending.username = 'ben\_user';

/\* Primary author: David Liao \*/

/\* returns all personal events of a user’s friend \*/

SELECT PE.event\_id, E.event\_name

FROM Event E, PersonalEvent PE, Friend F, Attending A

WHERE F.friend1 = 'amy\_user'

AND F.friend2 = 'ben\_user'

AND F.can\_access\_personal\_sch IS 'TRUE'

AND PE.event\_id = E.event\_id

AND E.event\_id = A.event\_id

AND F.friend2 = A.username;

/\* Primary author: Bowen Jin \*/

/\* returns all members of a particular organization \*/

SELECT username, email

FROM OrgMembers

WHERE organization\_name = 'DBMS appreciation club';

/\* Primary author: Sid Panjwani \*/

/\* returns the student IDs of all students who currently share at least one

class with a particular student\*/

SELECT DISTINCT CT2.student\_id

FROM CourseTaken CT1, CourseTaken CT2

WHERE CT1.student\_id = 'dom\_student1001'

AND CT1.student\_id <> CT2.student\_id

AND CT1.course\_id = CT2.course\_id

AND CT1.year = CT2.year

AND CT1.semester = CT2.semester

AND CT1.section = CT2.section

AND CT1.is\_completed IS 'FALSE';

/\* returns the course ID, course description, and number of students who have

taken or are currently enrolled in a particular course\*/

/\* Primary author: David Liao \*/

SELECT CT.course\_id, C.description, COUNT(CT.student\_id)

FROM Course C, CourseTaken CT, Student S

WHERE CT.student\_id = S.student\_id

AND C.course\_id = CT.course\_id

GROUP BY CT.course\_id

HAVING COUNT(CT.student\_ID) > 1;

/\* students graduated per year ordered in descending order \*/

/\* Primary Author: Sid Panjwani \*/

SELECT expected\_date\_of\_graduation, count(\*)

FROM Student

GROUP BY expected\_date\_of\_graduation

ORDER BY expected\_date\_of\_graduation DESC;

/\* number of students per major in descending order \*/

/\* Primary Author: Sid Panjwani \*/

SELECT major, count(\*)

FROM Student

GROUP BY major

ORDER BY major DESC;

/\* allows a particular student to see alumni who graduated with the same major with email & date graduated \*/

/\*Primary Author: Sid Panjwani \*/

SELECT P.real\_name, U.email, GS.expected\_date\_of\_graduation as year\_graduated

FROM Student S, GraduatedStudents GS, Person P, User U

WHERE S.student\_id = 'ben\_student1001' AND P.username = GS.username and U.username = P.username AND S.major = GS.major;

/\* get all classes with more than 5 graduate students \*/

/\* Primary Author: Sashank Nutakki \*/

SELECT C.course\_ID, C.description, C.credit\_hours, COUNT(S.student\_id) as gradCount

FROM Course C, Student S, CourseTaken CT

WHERE S.is\_graduate\_student = 'TRUE' AND CT.course\_id = C.course\_id AND S.student\_id = CT.student\_id

GROUP BY C.course\_id

HAVING gradCount >= 5;

/\* get all friends of specific user attending a specific event \*/

/\* Primary Author: Bowen Jin \*/

SELECT P2.real\_name

FROM Friend F, Person P1, Person P2, Attending A1, Attending A2

WHERE P1.username = 'amy\_user' AND F.friend1 = P1.username AND F.friend2 = P2.username AND A1.username = P1.username AND A2.username = P2.username AND A1.event\_id = 'event1001' AND A1.event\_id = A2.event\_id;

/\* get all organization events occurring on a specific date for organizations a specific user is registered for \*/

/\* Primary Author: David Liao \*/

SELECT E.event\_name, E.description, E.start\_time, E.end\_time

FROM Person P, OrganizationEvent OE, Event E

WHERE P.username = 'earl\_user' AND E.event\_date = '01/10/2020' AND OE.event\_id = E.event\_id;

/\* get all classes with 5 or more students who have taken it \*/

/\* Primary Author: Sid Panjwani \*/

SELECT C.course\_id, C.description, COUNT(\*)

FROM Course C, CourseTaken CT

WHERE C.course\_id = CT.course\_id

GROUP BY CT.course\_id

HAVING COUNT(\*) >= 5;

/\* get all personal and organization events in a specific schedule of current user\*/

/\* Primary Author: Sid Panjwani \*/

SELECT P.event\_id, O.event\_id

FROM PersonalEvent P, OrganizationEvent O, Person P, Attending A

WHERE P.username = "amy\_user" AND A.username = P.username AND A.event\_id = P.event\_id OR A.event\_id = O.event\_id;

/\* get all personal events of a user’s friend \*/

/\* Primary author: Bowen Jin\*/

SELECT PersonalEvent.event\_id

FROM Event,PersonalEvent,Friend

WHERE Friend.friend2 = 'amy\_user'

AND Friend.friend1 = 'ben\_user'

AND Friend.can\_access\_personal\_sch = 'TRUE'

AND PersonalEvent.event\_creator = Friend.friend1

AND PersonalEvent.event\_id = Event.event\_id;

/\* returns the usernames, names, and departments of all currently enrolled students who are TAs \*/

/\* Primary author: David Liao \*/

/\* returns the usernames, names, and departments of all currently enrolled students who are TAs \*/

/\* Primary author: David Liao \*/

SELECT P.username, P.real\_name, F.department

FROM Person P, Student S, Faculty F

WHERE P.username = S.username

AND S.username = F.username

AND S.has\_graduated = 'FALSE'

ORDER BY ASC;

/\* get all people that are invited to an event \*/

/\*Primary author: David Liao \*/

SELECT Person.real\_name

FROM Person NATURAL JOIN PersonalEventInvitation NATURAL JOIN PersonalEvent WHERE PersonalEvent.event\_id = 'event1001';

/\* deletes an event when the corresponding personal event is deleted \*/

/\*Primary author: Sid Panjwani \*/

CREATE TRIGGER DeleteEvent   
AFTER DELETE ON PersonalEvent  
REFERENCING OLD AS DeletedEvent  
BEGIN  
DELETE FROM Event WHERE event\_id = DeletedEvent.event\_id;

END;

/\* registers all currently enrolled graduate students as TAs within the department

having the same name as their major \*/

/\* Primary author: David Liao\*/

INSERT INTO Faculty(username, faculty\_id, department)

SELECT S.username, S.student\_id, S.major

FROM Student S

WHERE S.is\_graduate\_student IS 'TRUE'

AND S.has\_graduated IS 'FALSE'

/\* following these two delete statements, course with ID 'course1001' should be deleted

from table Course \*/

/\* Primary author: David Liao \*/

DELETE FROM Student

WHERE student\_id = 'earl\_student1001'

DELETE FROM Student

WHERE student\_id = 'faye\_student1001'

/\* insert an event to the persons events when they are invited \*/

/\*Primary author: Bowen Jin\*/

CREATE TRIGGER InvitedToEvent

AFTER INSERT ON

PersonalEventInvitation

BEGIN

INSERT INTO PersonalEvent VALUES(InsertedInvitation.event\_id, InsertedInvitation.username);

END;

/\* If a student is deleted from Student, then if this was the only student associated

with a course from CourseTaken, then that course is deleted from CourseTaken \*/

/\* Primary author: David Liao \*/

CREATE TRIGGER DeleteLastStudentOfCourseTaken

AFTER DELETE ON Student

WHEN (NOT EXISTS (SELECT \*

FROM Student S

WHERE S.student\_id = OLD.student\_id))

BEGIN

DELETE FROM CourseTaken WHERE student\_id = OLD.student\_id;

END;

/\* allows for deletions on OrgMembers

   Primary Author: Sashank Nutakki

   Secondary Author: Sid Panjwani \*/

CREATE TRIGGER DeleteOnOrgMembers

INSTEAD OF DELETE ON OrgMembers

BEGIN

DELETE FROM Member

WHERE member\_username = Old.username AND organization\_username = (SELECT username FROM Organization O where O.organization\_name = Old.organization\_name);

END;

/\* --------------------DROP statements-------------------- \*/

DROP TABLE IF EXISTS Attending;

DROP TABLE IF EXISTS Course;

DROP TABLE IF EXISTS CourseEvent;

DROP TABLE IF EXISTS CourseTaken;

DROP TABLE IF EXISTS Event;

DROP TABLE IF EXISTS Faculty;

DROP TABLE IF EXISTS Friend;

DROP TABLE IF EXISTS Member;

DROP TABLE IF EXISTS Organization;

DROP TABLE IF EXISTS OrganizationEvent;

DROP TABLE IF EXISTS OrganizationEventInvitation;

DROP TABLE IF EXISTS Person;

DROP TABLE IF EXISTS PersonalEvent;

DROP TABLE IF EXISTS PersonalEventInvitation;

DROP TABLE IF EXISTS Student;

DROP TABLE IF EXISTS User;

DROP VIEW IF EXISTS OrgMembers;

DROP VIEW IF EXISTS GraduatedStudents;

DROP VIEW IF EXISTS ClassMembers;

DROP VIEW IF EXISTS AccessiblePersonalEvents;

DROP TRIGGER IF EXISTS DeleteLastStudentOfCourseTaken;

DROP TRIGGER IF EXISTS DeleteEvent;

DROP TRIGGER IF EXISTS InvitedToEvent;

DROP TRIGGER IF EXISTS DeleteOnOrgMembers;