SupportNet
A Social Network for People Struggling with Illness

Group 10

Section 1: Overview

The objective of this project was to design and implement a social networking site for Hospital Patients. Through usage of our project, hospital patients would be able to find and contact various support groups suitable for them. We used a large variety of variables to judge whether a user is suitable to be another user's friend, or whether they were suitable to join a certain support group. These factors include (but aren't limited to) age, gender, illness, treatment, location (city, state), maximum travel distance, diet, mood and exercise patterns. Our vision was to provide patients with an easy and simple to use tool, by which they could find others going through the same things as them.

Major Functions

-Search users by the following criteria:

Disease, Treatment, Diet, Mood, Exercise, Interests/Hobbies

City, Gender, Age bracket, State

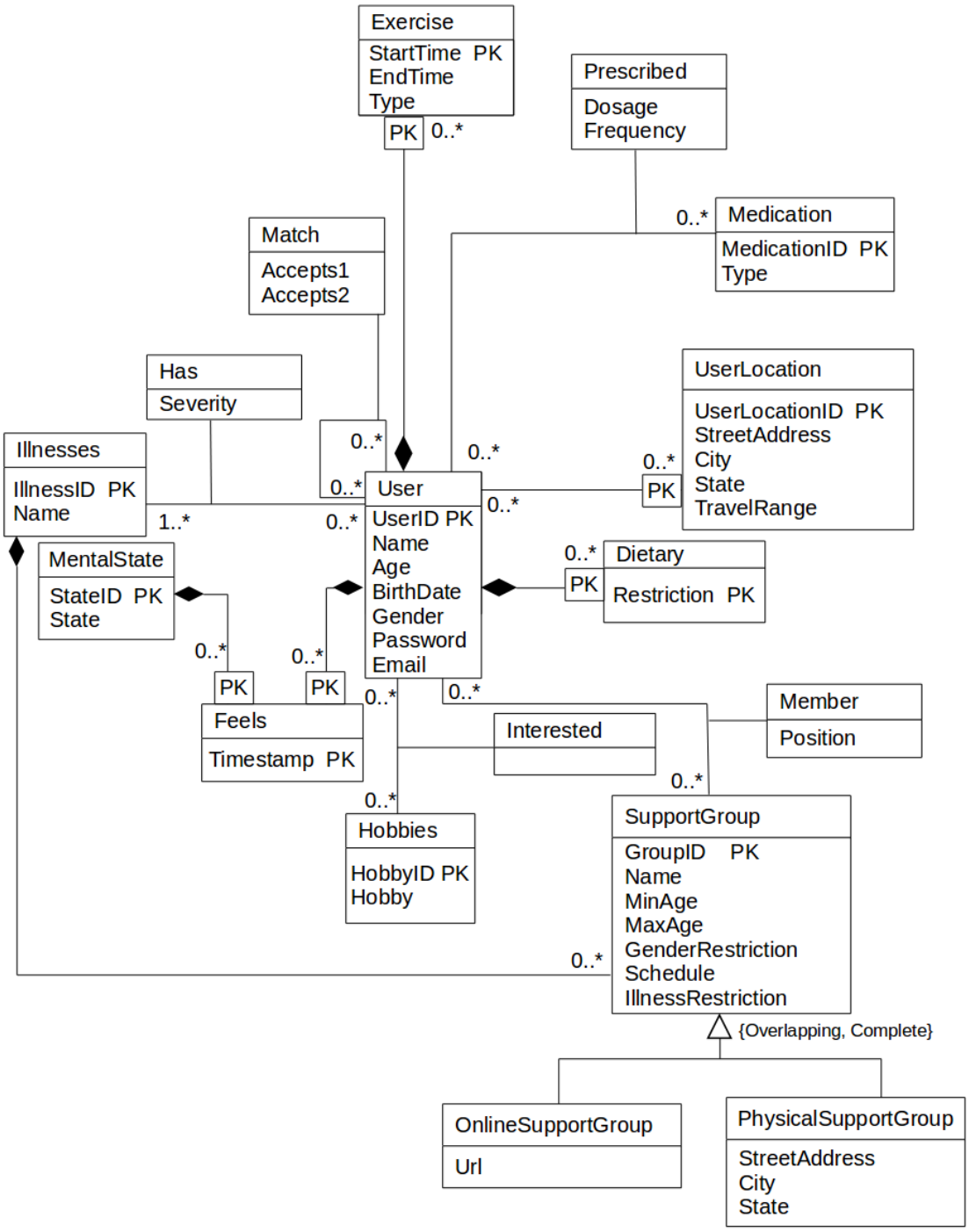
-Pair up users as "friends" based on the above criteria

-Creation of support groups based on common interest/goals/experience.

Design Elements

SQL is used for the storing of relevant user info. The traits which we use to match up patients with other patients/support groups all have their own sql table. These tables all have a relation with the Users table (which contains age, gender, userid) and for the most part have an association class. For example, the Hobbies table has a relation with the central Users table as well as the association class Interested (Pointer to UML). All of our traits tables have a zero to many relation with our Users table (Pointer to UML). For the support groups, we have it broken down into OnlineSupportGroup and PhysicalSupportGroup. (Pointer to UML) We have a trigger implemented that ensures Online and Physical support groups have an entry in the support group. In addition to this trigger, we also have a trigger preventing duplicate entries (Pointer to Triggers). We have written a set of queries based on likely search criteria (Pointer to Queries). These range from simple searches (searching users based on illness) to more complex specific searches such as finding users in city X who have felt a certain mood within the last week. We also implemented an assertion that checks to make sure there are no entries in support groups which aren't in either Online or Physical Support Group (Pointer to Assertion)

Section 2: UML Diagram



Section 3: Functional Dependencies

User: Email > UserID
This is enforced by setting Email as unique.
Illness: Name > IllnessID
This is enforced by setting Name as unique.
Medication: Type > MedicationID
This is enforced by setting Type as unique.
Hobbies: Hobby > HobbyID
This is enforced by setting Hobby as unique.
MentalState: State > StateID
This is enforced by setting State as unique.

The left hand side of each of these dependencies could be used as their respective table's primary key, but this design protects a user's personal information when looking at user data. For example, "jsanchez@gmail.com feels depressed, takes fluoxetine for his Generalized Anxiety Disorder, and likes badminton." would be seen as "3 feels 8, takes 6 for his 12, and likes 11."

Section 4: Assertions

```
/* ASSERTIONS */

/* Asserts that every SupportGroup is in OnlineSupportGroup, PhysicalSupportGroup, or both
        This is approximately supported by the EnsureCompletePhysicalCoverage and
        EnsureCompleteOnlineCoverage triggers */
/*
CREATE ASSERTION SupportGroupCompleteCoverage CHECK
(
        NOT EXISTS (SELECT * FROM SupportGroup
                        WHERE GroupID NOT IN
                                (SELECT GroupID FROM OnlineSupportGroup
                                 UNION SELECT GroupID FROM PhysicalSupportGroup))
);
*/

/* Enforces 1..* cardinality between User and Illness in the association class Has
        This is approximately supported by trigger UserHasAnIllness */
/*
CREATE ASSERTION CHECK
```

```
(
        NOT EXISTS (SELECT * FROM User
                        WHERE UserID NOT IN (SELECT UserID FROM Has)
);
*/

/* ASSERTIONS */

/* Asserts that every SupportGroup is in OnlineSupportGroup,
        PhysicalSupportGroup, or both This is approximately supported by the
        EnsureCompletePhysicalCoverage and EnsureCompleteOnlineCoverage
        triggers*/
/*
CREATE ASSERTION SupportGroupCompleteCoverage CHECK
(
        NOT EXISTS (SELECT * FROM SupportGroup
WHERE GroupID NOT IN
(SELECT GroupID FROM OnlineSupportGroup
 UNION SELECT GroupID FROM PhysicalSupportGroup))
);
*/

/* Enforces 1..* cardinality between User and Illness in the association
        class Has
        This is approximately supported by trigger UserHasAnIllness */
/*
CREATE ASSERTION CHECK
(
        NOT EXISTS (SELECT * FROM User
                        WHERE UserID NOT IN (SELECT UserID FROM Has)
);
*/
```

Section 5: Comprehensive Tables, Inserts, Deletes, Updates, and Queries

```
/* SETUP */
/* Delete tables if they already exist */
DROP TABLE IF EXISTS User;
DROP TABLE IF EXISTS Hobbies;
```

```
DROP TABLE IF EXISTS Interested;
DROP TABLE IF EXISTS MentalState;
DROP TABLE IF EXISTS Feels;
DROP TABLE IF EXISTS Illnesses;
DROP TABLE IF EXISTS Has;
DROP TABLE IF EXISTS Match;
DROP TABLE IF EXISTS Exercise;
DROP TABLE IF EXISTS Medication;
DROP TABLE IF EXISTS Prescribed;
DROP TABLE IF EXISTS UserLocation;
DROP TABLE IF EXISTS Dietary;
DROP TABLE IF EXISTS SupportGroup;
DROP TABLE IF EXISTS OnlineSupportGroup;
DROP TABLE IF EXISTS PhysicalSupportGroup;
DROP TABLE IF EXISTS Member;
/* Delete triggers if they already exist */
DROP TRIGGER IF EXISTS PreventDuplicateMatches;
DROP TRIGGER IF EXISTS EnsureCompletePhysicalCoverage;
DROP TRIGGER IF EXISTS EnsureCompleteOnlineCoverage;
DROP TRIGGER IF EXISTS UserHasAnIllness;
/* Delete indices if they already exist */
DROP INDEX IF EXISTS PersonSearch;
DROP INDEX IF EXISTS SupportGroupIndex;
DROP INDEX IF EXISTS CityAndState;
DROP INDEX IF EXISTS IllnessNames;
DROP INDEX IF EXISTS TreatmentName;
/* Delete views if they already exist */
DROP VIEW IF EXISTS PublicUserData;
DROP VIEW IF EXISTS SuggestedLocalGroupsforUser3;
DROP VIEW IF EXISTS PendingFriendRequests;


/* TABLES */

/* Holds basic information about users of this application */
CREATE TABLE User
(
        UserID INTEGER PRIMARY KEY AUTOINCREMENT,
        Name VARCHAR (255) NOT NULL,
```

```
        Age INT CHECK (Age >= 18),
        BirthDate DATE,
        Gender VARCHAR (255),
        Email VARCHAR (255) UNIQUE,
        Password VARCHAR (255)
);

/* Hobbies table, includes a Hobby ID and the name of the Hobby */
CREATE TABLE Hobbies
(
        HobbyID INTEGER PRIMARY KEY  AUTOINCREMENT,
        Hobby VARCHAR (255) UNIQUE
);

/* Interested; For user and Hobby, Has FK constraint UserID and HobbyID.
        Essentially links Users with hobbies they are interested in*/
CREATE TABLE Interested
(
        UserID INT,
        HobbyID INT,
        PRIMARY KEY (UserID, HobbyID),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (HobbyID) REFERENCES Hobbies(HobbyID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table for the mental state of patients, i.e happy, sad etc…*/
CREATE TABLE MentalState
(
        StateID INTEGER PRIMARY KEY AUTOINCREMENT,
        State VARCHAR (255) UNIQUE
);

/* Table that has info on what User Feels what mental state. This table also
        includes the time at which that user feels that mental state. */
CREATE TABLE Feels
```

```
(
        UserID INT,
        StateID INT,
        Timestamp DATETIME,
        PRIMARY KEY (UserID, StateID, Timestamp),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (StateID) REFERENCES MentalState(StateID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table of possible illnesses that patients can have. Has an association
        class "Has" with Users. The ID of the illness autoincrements with each
        new illness added. */
CREATE TABLE Illnesses
(
        IllnessID INTEGER PRIMARY KEY  AUTOINCREMENT,
        Name VARCHAR (255) UNIQUE
);

/* Association class between Users and Illness. Includes the severity of the
        illness for each patient. */
CREATE TABLE Has
(
        UserID INT,
        IllnessID INT,
        Severity VARCHAR (255),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (IllnessID) REFERENCES Illnesses(IllnessID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        PRIMARY KEY (UserID, IllnessID)
);

/* Details two users who have "matched" and states whether they have accepted
```

```
        eachother as friends or not */
CREATE TABLE Match
(
        ID1 INT,
        ID2 INT,
        Accepts1 BOOLEAN,
        Accepts2 BOOLEAN,
        PRIMARY KEY (ID1, ID2),
        FOREIGN KEY (ID1) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (ID2) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        CHECK (ID1 < ID2)
);

/* Table which holds each users exercise log. Includes the start and end
        time, as well as the type of exercise the patient engages in */
CREATE TABLE Exercise
(
        UserID INT,
        StartTime DATETIME,
        EndTime DATETIME,
        Type VARCHAR (255),
        PRIMARY KEY (UserID, StartTime),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Medicine Type's available for users to use. MedicationID auto-increments */
CREATE TABLE Medication
(
        MedicationID INTEGER PRIMARY KEY AUTOINCREMENT,
        Type VARCHAR (255) UNIQUE
);

/* Prescribed is the association table between the Medication and user
```

```
        relation. This table essentially details which user is prescribed which
        medication and in how low large a frequency and dosage.*/
CREATE TABLE Prescribed
(
        UserID INT,
MedicationID INT,
        Dosage VARCHAR (255),
        Frequency VARCHAR (255),
        Type VARCHAR (255),
        PRIMARY KEY (UserID, MedicationID),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (MedicationID) REFERENCES Medication(MedicationID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table detailing the location of each User. This table includes the City,
        State and StreetAddress of the user. It also includes how far the user
        is willing to travel to meet with support groups or other patients */
CREATE TABLE UserLocation
(
        UserLocationID INTEGER PRIMARY KEY AUTOINCREMENT,
        UserID INT,
        StreetAddress VARCHAR (255),
        City VARCHAR (255),
        State VARCHAR (255),
        TravelRange INT,
        UNIQUE (UserID, StreetAddress, City, State),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table detailing the diet requirements of Users. */
CREATE TABLE Dietary
(
        Restriction VARCHAR (255),
```

```
        UserID INT,
        PRIMARY KEY (Restriction, UserID),
        FOREIGN KEY (UserID) REFERENCES User(UserID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table of Support Groups that Users can join. This encompasses
        OnlineSupportGroup and PhysicalSupportGroup. The info includes
        Min/Max Age, Group Name, Gender/Illness Restriction and a schedule
        of what days the group meets. */
CREATE TABLE SupportGroup
(
        GroupID INTEGER PRIMARY KEY AUTOINCREMENT,
        Name VARCHAR (255),
        MinAge INT,
        MaxAge INT,
        GenderRestriction VARCHAR (255),
        Schedule VARCHAR (255),
        IllnessRestriction INT,
        FOREIGN KEY (IllnessRestriction) REFERENCES Illnesses(IlnessID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Encompassed by SupportGroups, this is a table of Online Support Groups.
        This table has the URL's of the various online support Groups. */
CREATE TABLE OnlineSupportGroup
(
        GroupID INT PRIMARY KEY,
        Url VARCHAR (255),
        FOREIGN KEY (GroupID) REFERENCES SupportGroup(GroupID)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

/* Table of support groups that meet in real life. Table includes the
        StreetAddress, City, State and GroupID of those support groups that meet
        at a physical location. */
```

```
CREATE TABLE PhysicalSupportGroup
(
	GroupID INT PRIMARY KEY,
	StreetAddress VARCHAR (255),
	City VARCHAR (255),
	State VARCHAR (255),
	FOREIGN KEY (GroupID) REFERENCES SupportGroup(GroupID)
		ON UPDATE CASCADE
		ON DELETE CASCADE
);

/* Association class for the User/SupportGroup relation. This table holds what
	position a User holds in the Support Groups they are part of. */
CREATE TABLE Member
(
	Position VARCHAR (255),
	UserID INT,
	GroupID INT,
	PRIMARY KEY (UserID, GroupID),
	FOREIGN KEY (UserID) REFERENCES User(UserID)
		ON UPDATE CASCADE
		ON DELETE CASCADE,
	FOREIGN KEY (GroupID) REFERENCES SupportGroup(GroupID)
		ON UPDATE CASCADE
		ON DELETE CASCADE
);


/* TRIGGERS */

/* Triggers ensuring that Online and PhysicalSupportGroup entries have an
	entry in SupportGroup. Entries in SupportGroup will have to be updated
	to fill null fields */
CREATE TRIGGER EnsureCompletePhysicalCoverage
BEFORE INSERT ON PhysicalSupportGroup
WHEN NOT EXISTS
	(SELECT * FROM SupportGroup WHERE GroupID = NEW.GroupID)
BEGIN
	INSERT INTO SupportGroup
```

```
VALUES(NEW.GroupID, NULL, NULL, NULL, NULL, NULL, NULL);
END;

CREATE TRIGGER EnsureCompleteOnlineCoverage
BEFORE INSERT ON OnlineSupportGroup
WHEN NOT EXISTS
        (SELECT * FROM SupportGroup WHERE GroupID = NEW.GroupID)
BEGIN
        INSERT INTO SupportGroup
VALUES(NEW.GroupID, NULL, NULL, NULL, NULL, NULL, NULL);
END;

/* Enforce the 1..* cardinality between User and Illness in the Has
        association class */
CREATE TRIGGER UserHasAnIllness
AFTER INSERT ON USER
BEGIN
INSERT INTO Has VALUES(NEW.UserID, NULL, NULL);
END;

/* Sample Data Insert Statements */
INSERT INTO User VALUES(NULL, 'Joan Smith', 21, '1994-03-15', 'female',
        'js@fake.com', 'testpsword');
INSERT INTO User VALUES(NULL, 'John Doe', 34, '1980-12-25', 'male',
        'jd@woah.com', '123456');
INSERT INTO User VALUES(NULL, 'Sam Johnson', 58, '1956-08-15', 'male',
        'sj@emailsrfun.com', 'pass');
INSERT INTO User VALUES(NULL, 'Henry Jones', 18, '1997-01-19', 'male',
        'hj@fakepeople.com', 'pupp13s');
INSERT INTO User VALUES(NULL, 'Eddy Dalton', 43, '1972-03-03', 'male',
        'ed@fake.com', 'cats');
INSERT INTO User VALUES(NULL, 'Basil Damion', 30, '1984-12-25', 'male',
        'basil@woah.com', 'agjadfgal');
INSERT INTO User VALUES(NULL, 'Scarlet Tad', 65, '1950-11-01', 'female',
        'scartad@emailme.com', '4hdu3jdiu3');
INSERT INTO User VALUES(NULL, 'Cathy Willa', 26, '1988-06-09', 'female',
        'cw@mail.gov', 'e73f83jck339');
INSERT INTO User VALUES(NULL, 'Howard Christabelle', 28, '1987-04-05', 'male',
        'howchris@pemailme.com', 'howardchrisrocks');
```

```
INSERT INTO User VALUES(NULL, 'Thea Hughie', 63, '1951-08-30', 'female',
        'thea@people.com', 'hugs4fr33');


INSERT INTO Hobbies(Hobby) VALUES('running');
INSERT INTO Hobbies(Hobby) VALUES('golf');
INSERT INTO Hobbies(Hobby) VALUES('music');
INSERT INTO Hobbies(Hobby) VALUES('video gaming');
INSERT INTO Hobbies(Hobby) VALUES('tabletop gaming');
INSERT INTO Hobbies(Hobby) VALUES('yoga');
INSERT INTO Hobbies(Hobby) VALUES('baseball');
INSERT INTO Hobbies(Hobby) VALUES('hiking');
INSERT INTO Hobbies(Hobby) VALUES('kayaking');
INSERT INTO Hobbies(Hobby) VALUES('reading');

INSERT INTO Interested VALUES(1, 1);
INSERT INTO Interested VALUES(4, 1);
INSERT INTO Interested VALUES(5, 1);
INSERT INTO Interested VALUES(6, 1);
INSERT INTO Interested VALUES(2, 2);
INSERT INTO Interested VALUES(3, 3);
INSERT INTO Interested VALUES(4, 4);
INSERT INTO Interested VALUES(5, 5);
INSERT INTO Interested VALUES(6, 6);
INSERT INTO Interested VALUES(7, 7);
INSERT INTO Interested VALUES(8, 8);
INSERT INTO Interested VALUES(9, 9);
INSERT INTO Interested VALUES(10, 10);
INSERT INTO Interested VALUES(1, 10);
INSERT INTO Interested VALUES(1, 8);
INSERT INTO Interested VALUES(2, 3);
INSERT INTO Interested VALUES(7, 4);
INSERT INTO Interested VALUES(8, 4);
INSERT INTO Interested VALUES(3, 6);
INSERT INTO Interested VALUES(2, 7);
INSERT INTO Interested VALUES(8, 9);
INSERT INTO Interested VALUES(5, 10);

INSERT INTO MentalState(State) VALUES('happy');
```

```
INSERT INTO MentalState(State) VALUES('sad');
INSERT INTO MentalState(State) VALUES('lonely');
INSERT INTO MentalState(State) VALUES('worried');
INSERT INTO MentalState(State) VALUES('upset');
INSERT INTO MentalState(State) VALUES('angry');
INSERT INTO MentalState(State) VALUES('hopeful');
INSERT INTO MentalState(State) VALUES('optimistic');
INSERT INTO MentalState(State) VALUES('confused');
INSERT INTO MentalState(State) VALUES('calm');

INSERT INTO Feels(UserID, StateID) VALUES(1, 1);
INSERT INTO Feels(UserID, StateID) VALUES(2, 2);
INSERT INTO Feels(UserID, StateID) VALUES(3, 3);
INSERT INTO Feels(UserID, StateID) VALUES(4, 4);
INSERT INTO Feels(UserID, StateID) VALUES(5, 5);
INSERT INTO Feels(UserID, StateID) VALUES(6, 6);
INSERT INTO Feels(UserID, StateID) VALUES(7, 7);
INSERT INTO Feels(UserID, StateID) VALUES(8, 8);
INSERT INTO Feels(UserID, StateID) VALUES(9, 9);
INSERT INTO Feels(UserID, StateID) VALUES(10, 10);
INSERT INTO Feels(UserID, StateID) VALUES(1, 10);
INSERT INTO Feels(UserID, StateID) VALUES(1, 8);
INSERT INTO Feels(UserID, StateID) VALUES(2, 3);
INSERT INTO Feels(UserID, StateID) VALUES(7, 4);
INSERT INTO Feels(UserID, StateID) VALUES(8, 4);
INSERT INTO Feels(UserID, StateID) VALUES(3, 6);
INSERT INTO Feels(UserID, StateID) VALUES(2, 7);
INSERT INTO Feels(UserID, StateID) VALUES(8, 9);
INSERT INTO Feels(UserID, StateID) VALUES(5, 10);

INSERT INTO Illnesses(Name) VALUES("Crohn's disease");
INSERT INTO Illnesses(Name) VALUES('Diabetes');
INSERT INTO Illnesses(Name) VALUES('Lung cancer');
INSERT INTO Illnesses(Name) VALUES('PTSD');
INSERT INTO Illnesses(Name) VALUES('Postpartum depression');

INSERT INTO Has VALUES(1, 1, 'severe');
INSERT INTO Has VALUES(2, 2, 'moderate');
INSERT INTO Has VALUES(3, 3, 'severe');
```

```
INSERT INTO Has VALUES(4, 4, 'severe');
INSERT INTO Has VALUES(6, 1, 'mild');
INSERT INTO Has VALUES(7, 2, 'mild');
INSERT INTO Has VALUES(8, 3, 'mild');
INSERT INTO Has VALUES(9, 4, 'mild');
INSERT INTO Has VALUES(10, 5, 'severe');
INSERT INTO Has VALUES(1, 5, 'severe');
INSERT INTO Has VALUES(1, 3, 'severe');
INSERT INTO Has VALUES(2, 3, 'moderate');
INSERT INTO Has VALUES(7, 5, 'moderate');
INSERT INTO Has VALUES(8, 1, 'moderate');
INSERT INTO Has VALUES(2, 4, 'moderate');
INSERT INTO Has VALUES(8, 2, 'moderate');
INSERT INTO Has VALUES(5, 1, 'mild');

INSERT INTO Match VALUES(1, 2, 1, 1);
INSERT INTO Match VALUES(1, 4, 1, 1);
INSERT INTO Match VALUES(2, 3, 1, 1);
INSERT INTO Match VALUES(2, 10, 1, 1);
INSERT INTO Match VALUES(3, 7, 0, 1);
INSERT INTO Match VALUES(6, 7, NULL, 1);
INSERT INTO Match VALUES(8, 9, 1, 0);
INSERT INTO Match VALUES(5, 7, 1, NULL);
INSERT INTO Match VALUES(5, 8, 1, NULL);

INSERT INTO Exercise
        VALUES(1, '2015-01-01 10:30:15', '2015-01-01 11:02:34', 'jogging');
INSERT INTO Exercise
        VALUES(2, '2015-02-03 8:04:17', '2015-02-03 8:24:49', 'swimming');
INSERT INTO Exercise
        VALUES(2, '2015-03-11 15:23:55', '2015-03-11 15:43:55', 'walking');
INSERT INTO Exercise
        VALUES(4, '2015-04-01 16:47:31', '2015-04-01 16:48:04', 'pranking');

INSERT INTO Medication(Type) VALUES('prednisone');
INSERT INTO Medication(Type) VALUES('glyburide');
INSERT INTO Medication(Type) VALUES('carboplatin');
INSERT INTO Medication(Type) VALUES('sertraline');
INSERT INTO Medication(Type) VALUES('fluoxetine');
```

```
INSERT INTO Prescribed VALUES(1, 1, '5 doses', 'daily', 'pill');
INSERT INTO Prescribed VALUES(2, 2, '4 doses', 'weekly', 'pill');
INSERT INTO Prescribed VALUES(3, 3, '8 doses', 'twice daily', 'pill');
INSERT INTO Prescribed VALUES(4, 4, '10 doses', 'twice daily', 'pill');
INSERT INTO Prescribed VALUES(5, 5, '4 doses', 'weekly', 'pill');
INSERT INTO Prescribed VALUES(6, 1, '2 doses', 'monthly', 'pill');
INSERT INTO Prescribed VALUES(7, 2, '1 doses', 'monthly', 'injection');
INSERT INTO Prescribed VALUES(8, 3, '4 doses', 'monthly', 'injection');
INSERT INTO Prescribed VALUES(9, 4, '2 doses', 'monthly', 'injection');
INSERT INTO Prescribed VALUES(10, 5, '6 doses', 'twice daily', 'injection');
INSERT INTO Prescribed VALUES(1, 5, '5 doses', 'twice daily', 'injection');
INSERT INTO Prescribed VALUES(1, 3, '10 doses', 'twice daily', 'liquid');
INSERT INTO Prescribed VALUES(2, 3, '4 doses', 'weekly', 'liquid');
INSERT INTO Prescribed VALUES(7, 5, '2 doses', 'weekly', 'liquid');
INSERT INTO Prescribed VALUES(8, 1, '4 doses', 'weekly', 'pill');
INSERT INTO Prescribed VALUES(3, 5, '8 doses', 'weekly', 'liquid');
INSERT INTO Prescribed VALUES(2, 4, '9 doses', 'weekly', 'liquid');
INSERT INTO Prescribed VALUES(8, 2, '4 doses', 'weekly', 'pill');
INSERT INTO Prescribed VALUES(5, 1, '2 doses', 'monthly', 'liquid');

INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(1, '123 Fake St', 'Someville', 'Kansas', 50);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(2, '234 Unreal Blvd', 'Nowhere', 'Georgia', 10);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(3, '345 Decoy Rd', 'Anothertown', 'Florida', 20);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(4, '456 Bogus Ln', 'Nowhere', 'Georgia', 5);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(5, '567 Fabricated Heights', 'Someville', 'Kansas', 6);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(6, '678 Phony Rd', 'Nowhere', 'Georgia', 7);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(7, '789 Fraudulent Ave', 'Someville', 'Kansas', 24);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(8, '890 Artificial St', 'Anothertown', 'Florida', 43);
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
    VALUES(9, '901 Concocted St', 'Somplace', 'New York', 61);
```

```
INSERT INTO UserLocation(UserID, StreetAddress, City, State, TravelRange)
        VALUES(10, '101 Pretend Ln', 'Nowhere', 'Georgia', 17);

INSERT INTO Dietary VALUES('kosher', 1);
INSERT INTO Dietary VALUES('vegan', 6);
INSERT INTO Dietary VALUES('low-carb', 4);

INSERT INTO SupportGroup
        VALUES(NULL, 'Helping Hands', 13, 35, 'male', 'MWF', 4);
INSERT INTO SupportGroup
        VALUES(NULL, 'Stronger Together', 21, 99, 'female', 'TR', 5);
INSERT INTO SupportGroup
        VALUES(NULL, 'For Us All', NULL, NULL, NULL, 'Variable', NULL);

INSERT INTO OnlineSupportGroup
        VALUES(3, 'http://www.forusallthesupporttgroup.com');
INSERT INTO OnlineSupportGroup
        VALUES(1, 'http://www.helpinghandsthesupportgroup.com');

INSERT INTO PhysicalSupportGroup
        VALUES(2, '123 Fake St', 'Someville', 'Kansas');
INSERT INTO PhysicalSupportGroup
        VALUES(1, '234 Unreal Blvd', 'Nowhere', 'Georgia');

INSERT INTO Member VALUES('Leader', 1, 2);
INSERT INTO Member VALUES(NULL, 7, 2);
INSERT INTO Member VALUES('Leader', 2, 1);
INSERT INTO Member VALUES('Webmaster', 4, 1);
INSERT INTO Member VALUES(NULL, 6, 3);
INSERT INTO Member VALUES(NULL, 5, 3);


/* INDICES */

/* 1. Index allowing for quick searching of people based on Name. This is a
        relevant index because most people are unlikely to change their names.
        Searching for another patient by their name is more likely to happen than
        an update to the Name column of Users.  */
```

```
CREATE INDEX PersonSearch
ON User (Name);

/* 2. Index: Finding support Groups based on gender, age range, and illness.
      This information would change very infrequently, but would be accessed
      every time a user's ability to join a group is checked */

CREATE INDEX SupportGroupIndex
ON SupportGroup (GenderRestriction, IllnessRestriction, MinAge, MaxAge);

/* 3. Index for finding UserLocations based on City and State. Though users
      may change their location on occasion, it won't be as often that they'd
      be searched for based on their location. */

CREATE INDEX CityAndState
ON UserLocation (City, State);

/* 4.  Index on the list of illnesses. An index is appropriate here since we
      are unlikely to update the Illnesses table unless we discover a new
      illness.*/

CREATE INDEX IllnessNames
ON Illnesses (Name);

/* 5. Index for the types of Medication available. Same reasoning that applied
      to the Illnesses index applies to this. Searching for a type of medicine
      will occur more frequently than the discovery of new ones. */

CREATE INDEX TreatmentName
ON Medication (Type);


/* VIEWS */

/* 1. View of all Users, but with private information omitted. This lets a
      developer or site admin look at the users in the database without seeing
      their private information, such as email, password, and birthday */
CREATE VIEW PublicUserData AS
SELECT UserId, Name, Age, Gender
```

```
FROM User;

/* 2. View of local support groups a user with UserID = X can go to, but aren't
      a member of. Similar views would exist for every user. Having a view for
      each individual user restricts the data they can see to the data that is
      relevant to them */
CREATE VIEW SuggestedLocalGroupsforUser3 AS
SELECT DISTINCT G.GroupID, G.Name, G.Schedule
FROM (Has NATURAL JOIN User NATURAL JOIN UserLocation) U,
(PhysicalSupportGroup NATURAL JOIN SupportGroup) G
WHERE U.Age >= G.MinAge AND U.Age <= G.MaxAge
      AND (U.Gender = G.GenderRestriction OR G.GenderRestriction IS NULL)
      AND U.IllnessID = G.IllnessRestriction
      AND U.City = G.City AND U.State = G.State
      AND U.UserID = 3
AND G.GroupID NOT IN
      (SELECT GroupID FROM Member NATURAL JOIN User
WHERE UserID = 3);

/* 3. View of all entries in Match where one person has accepted the friend
      request/suggestion and the other hasn't chosen yet. This would be a
      developer/site admin accessible view that would display no important or
      private user information */
CREATE VIEW PendingFriendRequests AS
SELECT * FROM Match
WHERE (Accepts1 IS NULL AND Accepts2 = 1)
OR (Accepts2 IS NULL AND Accepts1 = 1);

/* QUERIES */

/* 1. Query for finding pairs of users with the same illness */
SELECT H1.UserID, H2.UserID, H1.IllnessID
FROM Has H1 JOIN Has H2 USING(IllnessID)
WHERE H1.IllnessID=3 AND H1.IllnessID=H2.IllnessID AND H1.UserID < H2.UserID;

/* 2. Query finding online support groups a User with UserID = X can log on
      to, but isn't a member of
      To make the query runnable, we use UserID = 2 */
SELECT DISTINCT G.GroupID
```

```
FROM (Has JOIN User USING(UserID)) U,
(OnlineSupportGroup JOIN SupportGroup USING(GroupID)) G
WHERE (U.Age >= G.MinAge OR G.MinAge IS NULL)
AND (U.Age <= G.MaxAge OR G.MaxAge IS NULL)
        AND (U.Gender = G.GenderRestriction OR G.GenderRestriction IS NULL)
        AND (U.IllnessID = G.IllnessRestriction OR G.IllnessRestriction IS NULL)
        AND U.UserID = 2
        AND G.GroupID NOT IN
        (SELECT GroupID FROM Member NATURAL JOIN User WHERE UserID = 2);

/* 3. Query verifying that a submitted Password = X and Email = Y are valid
        login info. This would be used on user login To make this query runnable,
        we use Email = js@fake.com and tstpsword*/
SELECT U.Email, U.Password
FROM User U
WHERE Email = 'js@fake.com' AND Password = 'tstpsword';

/* 4. Query finding all users with an Extreme illness
SELECT UserID
FROM Has
WHERE Severity = 'Extreme';

/* 5. Query finding all Users in city X in state Y that have felt worried in
        the last week
        To make the query runnable, we use City, State = Someville, Kansas */
SELECT U.UserID
FROM User U NATURAL JOIN Feels F NATURAL JOIN MentalState M
        NATURAL JOIN UserLocation L
WHERE M.State = 'worried'
AND JULIANDAY(F.timestamp) > (JULIANDAY('now') - 7)
AND L.City = 'Someville' AND L.State = 'Kansas';

/* 6. Query finding users with no friends */
SELECT User.UserID FROM User
WHERE UserID NOT IN
                (SELECT ID1 FROM Match WHERE Accepts1 = 1 AND Accepts2 = 1)
        AND UserID NOT IN
                (SELECT ID2 FROM Match WHERE Accepts1 = 1 AND Accepts2 = 1);
```

```
/* 7. Finds the city, state combinations that contain at least two users
      sharing the same illness */
/* This query uses GROUP BY and aggregation */
SELECT City, State, COUNT(*), IllnessID
FROM UserLocation JOIN User USING(UserID)
      JOIN Has USING(UserID) JOIN Illnesses I USING(IllnessID)
GROUP BY City, State, IllnessID
HAVING COUNT(*) > 1;

/* 8. Query finding Users that share at least 2 hobbies with a User with
      ID = X
      To make the query runnable, we use UserID = 4 */
SELECT DISTINCT I.UserID
FROM Interested I
WHERE I.UserID != 4 AND
      (SELECT COUNT(*)
      FROM Interested I1 JOIN Interested I2 USING (HobbyID)
      WHERE I1.UserID = 4 AND I2.UserID = I.UserID) >= 2;

/* 9. Query finding pairs of users with shared dietary restrictions */
SELECT D1.UserID, D2.UserID, D1.Restriction
FROM Dietary D1 JOIN Dietary D2 USING(Restriction)
WHERE D1.UserID < D2.UserID;

/* 10. Query finding pairs of Users that have done the same exercise in the
      last week */
SELECT E1.UserID, E2.UserID, E1.Type
FROM Exercise E1 JOIN Exercise E2 USING(UserID)
WHERE E1.UserID < E2.UserID AND E1.Type = E2.Type
AND JULIANDAY(E1.EndTime) > (JULIANDAY('now') - 7)
AND JULIANDAY(E2.EndTime) > (JULIANDAY('now') - 7);

/* 11. Query finding leaders of Physical support groups for disease Z in city
      X in state Y
      To make the query runnable, we use IllnessRestriction = 4
      and City, State = Nowhere, Georgia */
SELECT M.UserID, G.GroupID
FROM Member M NATURAL JOIN PhysicalSupportGroup G NATURAL JOIN SupportGroup
      WHERE M.Position = 'Leader'
```

```
AND IllnessRestriction = 4
        AND G.City = 'Nowhere' AND G.State = 'Georgia';

/* 12. Query for finding appropriate SupportGroups for seniors */
SELECT S.Name, S.GroupID
FROM SupportGroup S
WHERE S.MinAge=65;

/* 13. Query for finding a list of people who are interested in Hobby X who
        live in city Y
        We use hobbyID=3 and city='Nowhere'to make this runnable */
SELECT U.UserID
FROM User U NATURAL JOIN Interested I NATURAL JOIN Hobbies H
        NATURAL JOIN UserLocation L
WHERE I.HobbyID=3
AND L.City='Nowhere';

/* 14. Query for finding all people with an illness X ordered by how they
        feel */
SELECT U.UserID
FROM User U NATURAL JOIN MentalState M NATURAL JOIN Illnesses I
WHERE I.Name='PTSD'
ORDER BY M.State;

/* 15. Query finding the publicly accessible information about a user with
        name X'
        To make this runnable, we used Name = Eddy Dalton */
SELECT U.Name, U.Age, U.Gender, L.City, L.State, H.Hobby
FROM User U NATURAL JOIN Interested NATURAL JOIN Hobbies H
NATURAL JOIN UserLocation L
WHERE U.Name = 'Eddy Dalton';

/* 16. Query finding leaders of online support groups for disease X
We use IllnessRestriction=5 to make this runnable */
SELECT M.UserID, G.GroupID
FROM Member M NATURAL JOIN OnlineSupportGroup G NATURAL JOIN SupportGroup
        WHERE M.Position = 'Leader'
AND IllnessRestriction = 5;
```

```
/* 17. Query finding pairs of people that live at the same location */
SELECT L1.UserID, L2.UserID, L1.StreetAddress, L1.City, L1.State
FROM UserLocation L1, UserLocation L2
WHERE L1.StreetAddress = L2.StreetAddress
AND L1.City = L2.City
        AND L1.State = L2.State;

/* 18. Query finding users that have fewer than the average number of friends
        for users in their city */
/* These queries are correlated and use GROUP BY, HAVING, and aggregation */
SELECT M.ID1, COUNT(*) as NumFriends, L.City, L.State
FROM UserLocation L, Match M
WHERE L.UserID = M.ID1
GROUP BY ID1, City, State
HAVING NumFriends < (SELECT AVG(FriendCount) FROM
                        (SELECT M.ID1, COUNT(*) as FriendCount, L.City, L.State
                        FROM UserLocation L0, Match M0
                        WHERE L0.UserID = M0.ID1
                        AND L0.City = L.City
                        AND L0.State = L.State
                        GROUP BY ID1));


/* 19. Insert into Match pairs of users that live in the same city and share
        2 hobbies and illness */
/* This Insert uses correlated queries */
INSERT INTO Match
SELECT H0.UserID, H1.UserID, NULL, NULL
FROM (Has NATURAL JOIN UserLocation) H0, (Has NATURAL JOIN UserLocation) H1
WHERE H0.UserID < H1.UserID
        AND H0.IllnessID = H1.IllnessID
        AND H0.City = H1.City
        AND H0.State = H1.State
AND 2 <= (SELECT COUNT(*)
FROM Interested I0 JOIN Interested I1 USING(HobbyID)
WHERE I0.UserID = H0.UserID
AND I1.UserID = H1.UserID);

/* 20. Update incrementing Age on the User's birthday */
```

```
UPDATE User
SET Age = Age + 1
WHERE STRFTIME("%m-%d", BirthDate) = STRFTIME("%m-%d", 'now');


/* 21. Query finding all SupportGroups that are both Online and Physical */
SELECT S.GroupID, S.Name, P.StreetAddress, P.City, P.State, O.URL
FROM SupportGroup S NATURAL JOIN OnlineSupportGroup O
NATURAL JOIN PhysicalSupportGroup P;


/* 22. Query for finding all users who are taking Medication X
        To make this runnable, we used M.Type = 'Glyburdie' */
SELECT U.UserID, U.Name
From User U NATURAL JOIN Medication M
Where M.Type='Glyburdie';

/* 23. Returns the last user entered into the database*/
SELECT *
FROM User
ORDER BY UserID DESC LIMIT 1;

/* CLEANUP */
/* Delete tables if they already exist */
DROP TABLE IF EXISTS User;
DROP TABLE IF EXISTS Hobbies;
DROP TABLE IF EXISTS Interested;
DROP TABLE IF EXISTS MentalState;
DROP TABLE IF EXISTS Feels;
DROP TABLE IF EXISTS Illnesses;
DROP TABLE IF EXISTS Has;
DROP TABLE IF EXISTS Match;
DROP TABLE IF EXISTS Exercise;
DROP TABLE IF EXISTS Medication;
DROP TABLE IF EXISTS Prescribed;
DROP TABLE IF EXISTS UserLocation;
DROP TABLE IF EXISTS Dietary;
DROP TABLE IF EXISTS SupportGroup;
DROP TABLE IF EXISTS OnlineSupportGroup;
```

```
DROP TABLE IF EXISTS PhysicalSupportGroup;
DROP TABLE IF EXISTS Member;
/* Delete triggers if they already exist */
DROP TRIGGER IF EXISTS PreventDuplicateMatches;
DROP TRIGGER IF EXISTS EnsureCompletePhysicalCoverage;
DROP TRIGGER IF EXISTS EnsureCompleteOnlineCoverage;
DROP TRIGGER IF EXISTS UserHasAnIllness;
/* Delete indices if they already exist */
DROP INDEX IF EXISTS PersonSearch;
DROP INDEX IF EXISTS SupportGroupIndex;
DROP INDEX IF EXISTS CityAndState;
DROP INDEX IF EXISTS IllnessNames;
DROP INDEX IF EXISTS TreatmentName;
/* Delete views if they already exist */
DROP VIEW IF EXISTS PublicUserData;
DROP VIEW IF EXISTS SuggestedLocalGroupsforUser3;
DROP VIEW IF EXISTS PendingFriendRequests;
```