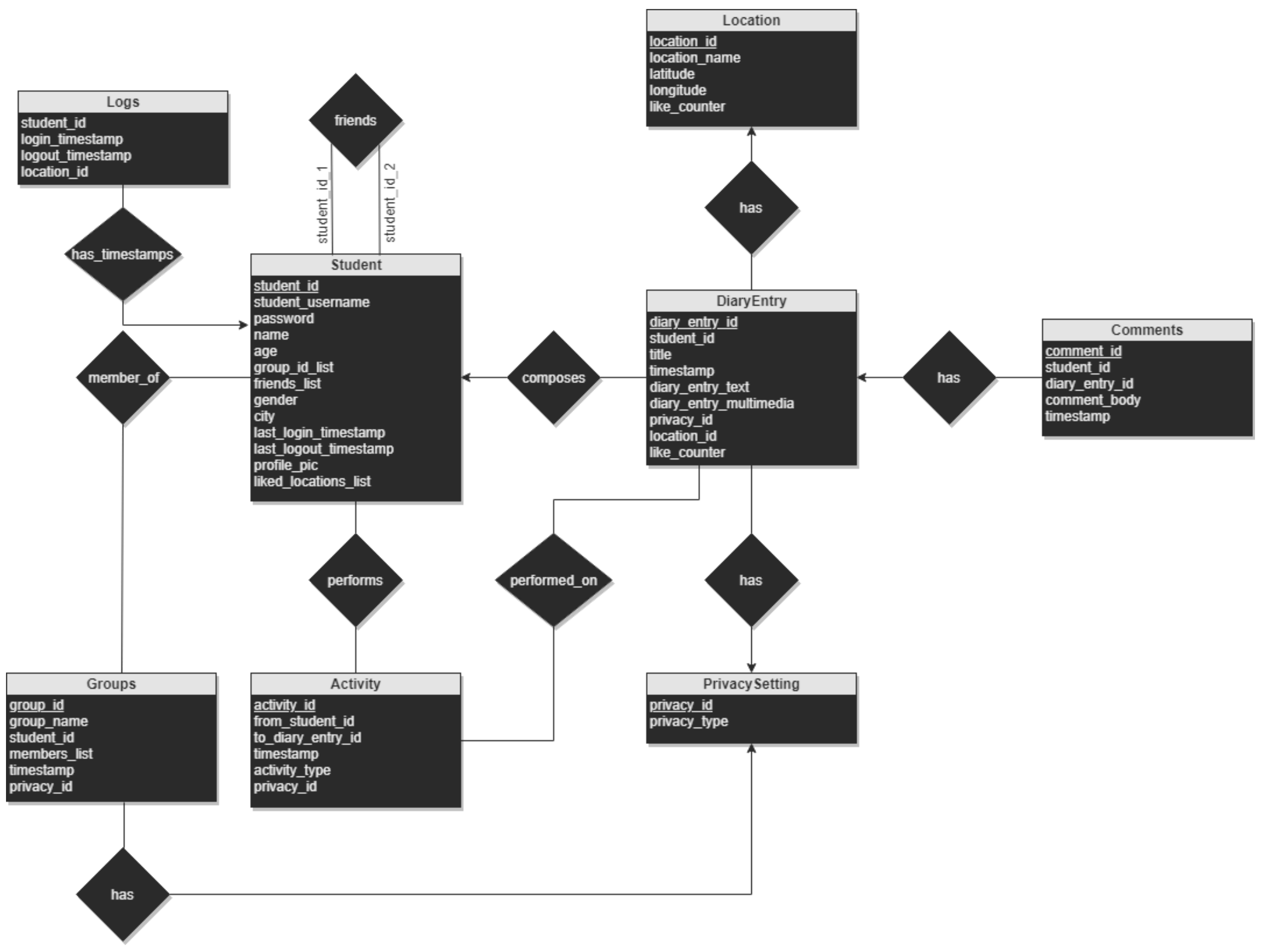
Principles of Database Systems Project Part 1 Documentation

Vikram Sunil Bajaj (vsb259), Ameya Shanbhag (avs431)

We have chosen to create a web application for the **University Students Social Network** use case. Part 1 of this project deals with the designing of a database for the backend of the web application.

TABLE OF CONTENTS:

1. Entity-Relationship Diagram



The diagram above illustrates the ER diagram for our database.
As we know, undirected lines denote many-to-many relationships and directed lines represent one-to-many or many-to-one relationships (with the arrow pointing to the *one* entity).

2. Relational Schemas

- Student(student_id, student_username, password, name, age, group_id_list, friends_list, gender, city, last_login_timestamp, last_logout_timestamp, profile_pic, liked_locations_list)
- Location(location_id, location_name, latitude, longitude, like_counter)
- Logs(student_id, login_timestamp, logout_timestamp, location_id)
- PrivacySetting(privacy_id, privacy_type)
- DiaryEntry(diary_entry_id, student_id, title, timestamp, diary_entry_text, diary_entry_multimedia, privacy_id, location_id, like_counter)
- Comments(comment_id, student_id, diary_entry_id, comment_body, timestamp)
- Friends(student_id_1, student_id_2, timestamp, status)

- Activity(activity_id, from_student_id, to_diary_entry_id, timestamp, activity_type, privacy_id)
- Groups(group_id, group_name, student_id, members_list, timestamp, privacy_id)

2.1. Justification for Schemas

Student
We have used student_id as the primary key for identifying each individual student in our network. Students will have a profile which with attributes such as name, age, gender, city. Also, we have a JSON array called group_id_list which will store the ids of the groups that the student is a part of. The group can be for courses that the students are enrolled in, the organizations they are members of etc. liked_locations_list contains the list of location names *liked* by the student. Student also contains friends_list which consists of all the friends that student has on the social network. Finally, Student also has a field profile_pic to allow students to set a profile picture.

Location
The Location table is used so that we can have location added to any diary entry posted by the student. It also helps us track which city is the most visited among a student's friends. It consists of location_id as the primary key which will be referenced in different tables, and has other attributes including location_name, latitude and longitude.
Location also has a like_counter to count how many times a location was liked by students.

Logs
It helps us keep logs of the duration of each session by a student. Logs also consists of location_id which will help us log the location of the student and can be used to detect any unidentified/suspicious login location.

PrivacySetting
PrivacySetting consists of the attribute privacy_type which will help us identify different types of privacy: private, friends, friends of friends, and public. privacy_id is the primary key.

DiaryEntry
This table is to maintain all the diary entries by the students and consists of the attribute student_id so that we know to which student the entry belongs. Other attributes include title, which will define the title for the diary entry, diary_entry_text for the student to enter large text, diary_entry_multimedia for the student to upload/attach photos and other multimedia with the post, privacy_id to let the student decide to whom does he/she want the diary entry to be visible, location_id to add a location to the diary entry, and like_counter to keep a track of the number likes for a particular entry.

Comments
It consists of student_id which will help us identify the student who has posted the comment, diary_entry_id to denote the diary_entry on which the comment was posted, comment_body which consists of the actual comment text, and timestamp to record the time at which the comment was posted. comment_id is the primary key.

Friends
It is a self-referencing relationship on the Student entity, and is modeled as a separate entity.
It consists of two student_id's of which the first one is the student_id of the student who has sent the friend request and the second is the student_id of the student who has received the friend request from the first student. It also records the timestamp that helps to determine how for long the students have been friends, and also contains the status of the friendship (which can be accepted, pending etc.).

Activity
It consists of from_student_id to show which student is performing the activity, to_diary_entry_id to denote the diary entry on which the activity is being performed, timestamp to record the time of the activity, activity_type to identify if the activity is a like or a comment activity, and privacy_id to show the privacy setting of the activity. activity_id is the primary key.

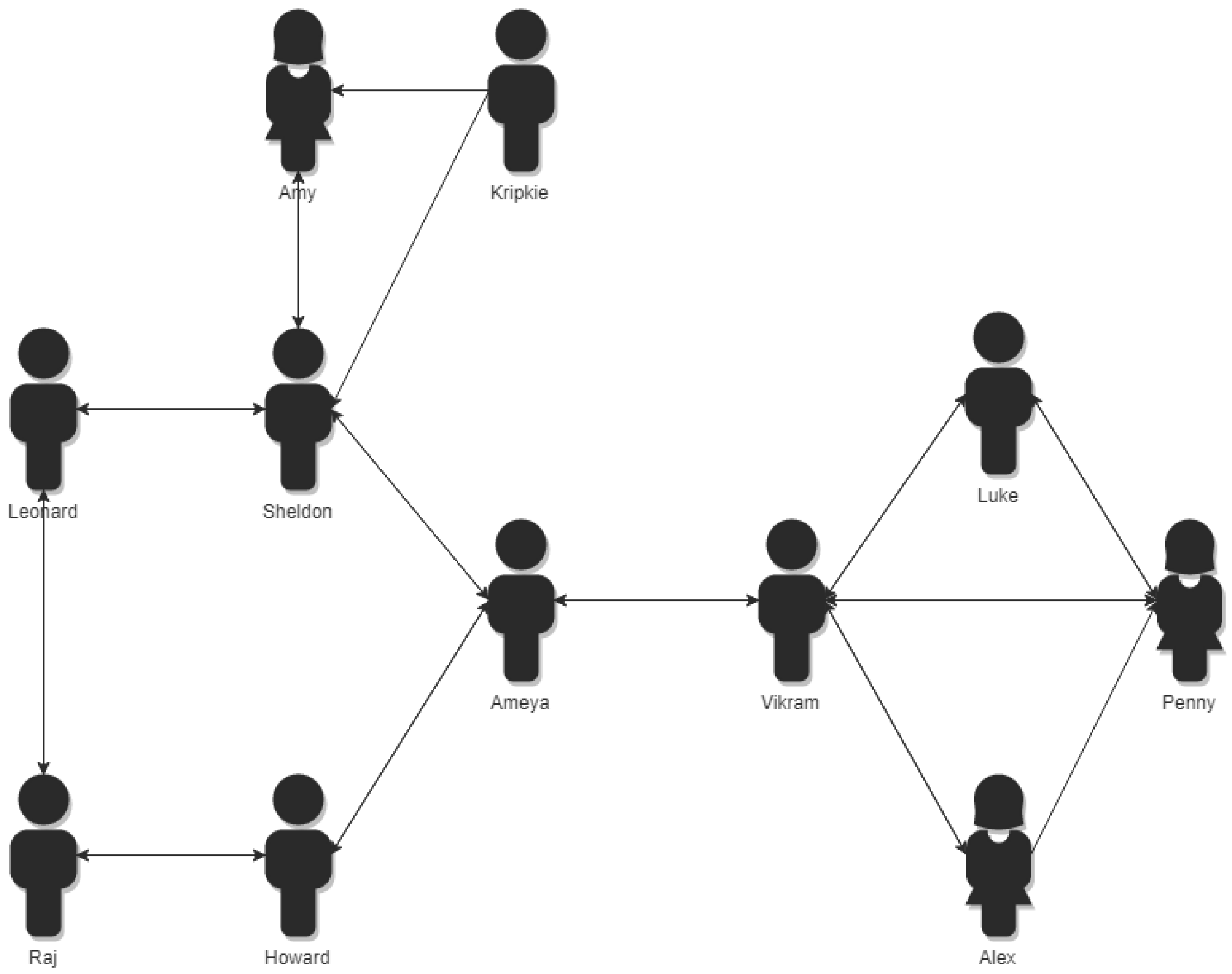
Groups
It helps to keep track of different groups along with the members in each group. It also helps us have different types groups like Organizations, Courses etc.

2.2. Foreign Key Constraints

The following are the foreign keys used in the previously-mentioned relations:

- student_id in Logs references student_id in Student
- location_id in Logs references location_id in Location
- location_id in DiaryEntry references location_id in Location
- privacy_id in DiaryEntry references privacy_id in PrivacySetting
- student_id in DiaryEntry references student_id in Student
- diary_entry_id in Comments references diary_entry_id in DiaryEntry
- student_id in Comments references student_id in Student
- student_id_1 and student_id_2 in Friends reference student_id in Student
- from_student_id in Activity references student_id in Student
- privacy_id in Activity references privacy_id in PrivacySetting
- to_diary_entry_id in Activity references diary_entry_id in DiaryEntry
- student_id in Groups references student_id in Student
- privacy_id in Groups references privacy_id in PrivacySetting

3. Friendship Map



Bidirectional arrows denote friends i.e. the friend request was ‘accepted’. Unidirectional arrows denote that the friend request is ‘pending’ or was ‘declined’.

4. SQL Queries

4.1. DDL Commands

4.1.1 To create a database

```
CREATE DATABASE IF NOT EXISTS university_student_network;
```

4.1.2. Create Student Table

```
CREATE TABLE IF NOT EXISTS Student
(
student_id INT NOT NULL,
student_username VARCHAR(20),
password VARCHAR(20) NOT NULL,
name VARCHAR(20),
age INT,
group_id_list JSON DEFAULT NULL,
friends_list JSON DEFAULT NULL,
gender VARCHAR(20),
city VARCHAR(20),
last_login_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
last_logout_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
profile_pic BLOB,
liked_locations_list JSON,
PRIMARY KEY(student_id)
);
```

4.1.3. Create Location Table

```
CREATE TABLE IF NOT EXISTS Location
(
 location_id INT PRIMARY KEY NOT NULL,
 location_name VARCHAR(1000) NOT NULL,
 latitude VARCHAR(1000) NOT NULL,
 longitude VARCHAR(1000) NOT NULL,
 like_counter INT
);
```

4.1.4. Create Logs Table

```
CREATE TABLE IF NOT EXISTS Logs
(
 student_id INT,
 login_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 logout_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 location_id INT,
 FOREIGN KEY (student_id) REFERENCES Student(student_id),
 FOREIGN KEY (location_id) REFERENCES Location(location_id)
);
```

4.1.5. Create PrivacySetting Table

```
CREATE TABLE IF NOT EXISTS PrivacySetting
(
 privacy_id INT PRIMARY KEY NOT NULL,
 privacy_type VARCHAR(20) NOT NULL
);
```

4.1.6. Create DiaryEntry Table

```
CREATE TABLE IF NOT EXISTS DiaryEntry
(
student_id INT NOT NULL ,
diary_entry_id INT PRIMARY KEY NOT NULL,
title VARCHAR(50),
timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
diary_entry_text TEXT,
diary_entry_multimedia BLOB,
privacy_id INT NOT NULL,
location_id INT,
like_counter INT,
FOREIGN KEY (location_id) REFERENCES Location(location_id),
FOREIGN KEY (privacy_id) REFERENCES PrivacySetting(privacy_id),
FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

4.1.7. Create Comments Table

```
CREATE TABLE IF NOT EXISTS Comments
(
 student_id INT NOT NULL,
 diary_entry_id INT,
 comment_id INT PRIMARY KEY NOT NULL,
 comment_body VARCHAR(1000) NOT NULL,
 timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
 FOREIGN KEY (diary_entry_id) REFERENCES DiaryEntry(diary_entry_id),
 FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

4.1.8. Create Friends Table

```
CREATE TABLE IF NOT EXISTS Friends
(
 student_id_1 INT NOT NULL,
 student_id_2 INT NOT NULL,
 timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
 status VARCHAR(10) NOT NULL,
 PRIMARY KEY (student_id_1, student_id_2),
 FOREIGN KEY (student_id_2) REFERENCES Student(student_id),
 FOREIGN KEY (student_id_1) REFERENCES Student(student_id)
);
```

4.1.9. Create Activity Table

```
CREATE TABLE IF NOT EXISTS Activity
(
 from_student_id INT,
 to_diary_entry_id INT,
 activity_id INT PRIMARY KEY,
 timestamp TIMESTAMP,
 activity_type VARCHAR(20),
 privacy_id INT,
 FOREIGN KEY (from_student_id) REFERENCES Student(student_id),
 FOREIGN KEY (privacy_id) REFERENCES PrivacySetting(privacy_id),
 FOREIGN KEY (to_diary_entry_id) REFERENCES DiaryEntry(diary_entry_id));
```

4.1.10. Create Groups Table

```
CREATE TABLE IF NOT EXISTS Groups
(
 group_id INT PRIMARY KEY NOT NULL,
 group_name VARCHAR(20) NOT NULL,
 student_id INT NOT NULL,
 members_list JSON,
 timestamp TIMESTAMP NOT NULL,
 privacy_id INT NOT NULL,
 FOREIGN KEY (student_id) REFERENCES Student(student_id),
 FOREIGN KEY (privacy_id) REFERENCES PrivacySetting(privacy_id)
);
```

4.2. Stored Procedures/DML Commands

4.2.1 Stored Procedure for Inserting Values into Student during sign-up

```
DELIMITER //
DROP PROCEDURE IF EXISTS signup
CREATE PROCEDURE signup (IN student_id INT, IN student_username VARCHAR(20), IN password
VARCHAR(20), IN name VARCHAR(20), IN age INT, IN group_id_list JSON, IN friends_list JSON, IN gender
VARCHAR(20), IN city VARCHAR(20), IN last_login_timestamp TIMESTAMP, IN last_logout_timestamp
TIMESTAMP, IN profile_pic BLOB, IN liked_locations_list JSON)
BEGIN

SELECT CURRENT_TIMESTAMP INTO last_login_timestamp;
SELECT CURRENT_TIMESTAMP INTO last_logout_timestamp;
INSERT IGNORE INTO Student VALUES (student_id, student_username, password, name, age, group_id_list,
friends_list, gender, city, last_login_timestamp, last_logout_timestamp, profile_pic, liked_locations_list);

END//
DELIMITER ;
```

4.2.2. Stored Procedure to update profile picture

```
DELIMITER //

DROP PROCEDURE IF EXISTS edit_profile_pic
CREATE PROCEDURE edit_profile_pic(IN id INT, IN profile_pic BLOB)
BEGIN
 UPDATE Student SET profile_pic=profile_pic WHERE student_id=id;
END //
DELIMITER ;
```

4.2.3. Stored Procedure for Sending Friend Request

```
DELIMITER //
DROP PROCEDURE IF EXISTS friend_request
CREATE PROCEDURE friend_request
```

```
(IN id_1 INT, IN id_2 INT)
BEGIN
INSERT IGNORE INTO Friends VALUES (id_1, id_2, null, 'pending');
END //
DELIMITER ;
```

4.2.4. Stored Procedure for Accepting Friend Request

```
DELIMITER //
DROP PROCEDURE IF EXISTS friend_request_accept;
CREATE PROCEDURE friend_request_accept
(IN id_1 INT, IN id_2 INT)
BEGIN
UPDATE Student SET friends_list = JSON_ARRAY_APPEND(friends_list, '$', id_1) WHERE student_id = id_2;
UPDATE Student SET friends_list = JSON_ARRAY_APPEND(friends_list, '$', id_2) WHERE student_id = id_1;
UPDATE Friends SET status= 'accepted' WHERE student_id_1=id_1 AND student_id_2=id_2;
END //
DELIMITER ;
```

4.2.5. Stored Procedure for Declining Friend Request

```
DELIMITER //
DROP PROCEDURE IF EXISTS friend_request_decline;

CREATE PROCEDURE friend_request_decline
(IN id_1 INT, IN id_2 INT)
BEGIN
UPDATE Friends SET status= 'declined' WHERE student_id_1=id_1 AND student_id_2=id_2;
END //
DELIMITER ;
```

4.2.6. Stored Procedure for Creating a New Diary Entry

```
DELIMITER //
DROP PROCEDURE IF EXISTS diary_entry
CREATE PROCEDURE diary_entry(IN student_id INT,IN diary_entry_id INT, IN title VARCHAR(20),IN timestamp
TIMESTAMP,IN diary_entry_text TEXT, IN diary_entry_multimedia BLOB,IN privacy_id INT,IN location_id
INT,like_counter INT)
BEGIN
  SELECT CURRENT_TIMESTAMP INTO timestamp;
  INSERT IGNORE INTO DiaryEntry VALUES (student_id, diary_entry_id, title, timestamp, diary_entry_text,
diary_entry_multimedia, privacy_id, location_id, like_counter);
END //
DELIMITER ;
```

4.2.7. Stored Procedure to Edit Diary Entry Title

```
DELIMITER //
DROP PROCEDURE IF EXISTS edit_entry_title
CREATE PROCEDURE edit_entry_title(IN id INT, IN title VARCHAR(20))
BEGIN
  UPDATE DiaryEntry SET title=title WHERE diary_entry_id=id;
```

```
END //
DELIMITER ;
```

4.2.8. Stored Procedure to Display Student Entry

```
DELIMITER //
DROP PROCEDURE IF EXISTS display_student_entries;
CREATE PROCEDURE display_student_entries
(IN id INT)
BEGIN
SELECT title, timestamp, diary_entry_text, diary_entry_multimedia, like_counter
FROM DiaryEntry
WHERE student_id = id
ORDER BY timestamp DESC;
END//
DELIMITER ;
```

4.2.9. Stored Procedure to Display Student Details

```
DELIMITER //
DROP PROCEDURE IF EXISTS display_student_details;
CREATE PROCEDURE display_student_details
(IN id INT)
BEGIN
SELECT profile_pic, name, student_username, JSON_LENGTH(friends_list) AS friend_count, group_id_list
FROM Student
WHERE student_id =id;
END //
DELIMITER ;
```

4.2.10. Stored Procedure for printing names of Friends of a given student

```
DELIMITER //
DROP PROCEDURE IF EXISTS friends;
CREATE PROCEDURE friends(IN id INT)
BEGIN
SET @i = 0;
SELECT JSON_LENGTH(friends_list) INTO @frnd_len FROM Student WHERE student_id=id;
CREATE TABLE temp(ID_1 INT);
WHILE @i < @frnd_len DO
  SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@i,']')) INTO @temp_id FROM Student WHERE
student_id=id;
INSERT INTO temp VALUES(@temp_id);
SELECT @i+1 INTO @i;
END WHILE;

SELECT name
FROM Student
WHERE student_id IN (SELECT ID_1 FROM temp);
DROP TABLE temp;
END //
```

```
DELIMITER ;
```

4.2.11. Stored Procedure for printing names of Friends of Friends of a given Student

```
DELIMITER //
DROP PROCEDURE IF EXISTS friends_of_friends;
CREATE PROCEDURE friends_of_friends(IN id INT)
BEGIN

SET @i = 0;
SELECT JSON_LENGTH(friends_list) INTO @frnd_len FROM Student WHERE student_id=id;

CREATE TABLE temp(ID_1 INT, PRIMARY KEY(ID_1));
WHILE @i < @frnd_len DO
 SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@i,']')) INTO @temp_id FROM Student WHERE student_id=id;
 INSERT IGNORE INTO temp VALUES(@temp_id) ;
 SELECT JSON_LENGTH(friends_list) INTO @fof_len FROM Student WHERE student_id=@temp_id;
 SET @j =0;
 WHILE @j < @fof_len DO
   SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@j,']')) INTO @fof_temp_id FROM Student WHERE
student_id=@temp_id;
   INSERT IGNORE INTO temp VALUES(@fof_temp_id) ;
   SELECT @j+1 INTO @j;
 END WHILE;
 SELECT @i+1 INTO @i;
END WHILE;

DELETE FROM temp WHERE ID_1=id;

SELECT name
FROM Student
WHERE student_id IN (SELECT ID_1 FROM temp);

DROP TABLE temp;

END //
DELIMITER ;
```

4.2.12. Stored Procedure to retrieve posts of Friends of Friends of a Student containing a specified term such as 'ice hockey'

```
DELIMITER //
DROP PROCEDURE IF EXISTS friends_of_friends_search;
CREATE PROCEDURE friends_of_friends_search(IN id INT, IN search_term VARCHAR(20))
BEGIN

SET @i = 0;
SELECT JSON_LENGTH(friends_list) INTO @frnd_len FROM Student WHERE student_id=id;

CREATE TABLE temp(ID_1 INT, PRIMARY KEY(ID_1));
WHILE @i < @frnd_len DO
```

```
 SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@i,']')) INTO @temp_id FROM Student WHERE student_id=id;
 INSERT IGNORE INTO temp VALUES(@temp_id) ;
 SELECT JSON_LENGTH(friends_list) INTO @fof_len FROM Student WHERE student_id=@temp_id;
 SET @j =0;
 WHILE @j < @fof_len DO
   SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@j,']')) INTO @fof_temp_id FROM Student WHERE
student_id=@temp_id;
   INSERT IGNORE INTO temp VALUES(@fof_temp_id) ;
   SELECT @j+1 INTO @j;
 END WHILE;
 SELECT @i+1 INTO @i;
END WHILE;

DELETE FROM temp WHERE ID_1=id;

SELECT DiaryEntry.student_id, DiaryEntry.title, DiaryEntry.diary_entry_text, DiaryEntry.diary_entry_multimedia
FROM DiaryEntry
WHERE DiaryEntry.diary_entry_text LIKE '%'||@search_term||'%' AND DiaryEntry.privacy_id>=1 AND
DiaryEntry.student_id IN (SELECT ID_1 FROM temp);

DROP TABLE temp;

END //
DELIMITER ;
```

4.2.13. Stored Procedure for displaying all diary entries by Friends of a Student, posted during the last week

```
DELIMITER //
DROP PROCEDURE IF EXISTS friends_entries_last_week
CREATE PROCEDURE friends_entries_last_week(IN id INT)
BEGIN
SET @i = 0;
SELECT JSON_LENGTH(friends_list) INTO @frnd_len FROM Student WHERE student_id=id;
CREATE TABLE temp(ID_1 INT);
WHILE @i < @frnd_len DO
  SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@i,']')) INTO @temp_id FROM Student WHERE
student_id=id;
INSERT INTO temp VALUES(@temp_id);
SELECT @i+1 INTO @i;
END WHILE;

SELECT DiaryEntry.student_id, DiaryEntry.title, DiaryEntry.diary_entry_text, DiaryEntry.diary_entry_multimedia,
DiaryEntry.timestamp
FROM DiaryEntry
WHERE DiaryEntry.timestamp BETWEEN SUBDATE(CURDATE(), INTERVAL 1 WEEK) AND NOW() AND
DiaryEntry.student_id IN (SELECT ID_1 FROM temp);
DROP TABLE temp;
END //
DELIMITER ;
```

4.2.14. Stored Procedure for displaying all locations liked by Friends of a Student

```
DELIMITER //
DROP PROCEDURE IF EXISTS friends_liked_locations;
CREATE PROCEDURE friends_liked_locations(IN id INT)
BEGIN
SET @i = 0;
SELECT JSON_LENGTH(friends_list) INTO @frnd_len FROM Student WHERE student_id=id;
CREATE TABLE temp(ID_1 INT);
WHILE @i < @frnd_len DO
  SELECT JSON_EXTRACT(friends_list,CONCAT('$[',@i,']')) INTO @temp_id FROM Student WHERE
student_id=id;
INSERT INTO temp VALUES(@temp_id);
SELECT @i+1 INTO @i;
END WHILE;

SELECT Student.name, Student.liked_locations_list
FROM Student
WHERE Student.student_id IN (SELECT ID_1 FROM temp);
DROP TABLE temp;
END//
DELIMITER ;
```

4.3. Procedure Call to Insert/Edit Data

4.3.1. Calls to Sign Up Students

```
CALL signup(1, 'avs431', 'pwd1', 'Ameya', 22, '[]', '[]', 'Male', 'Mumbai', null, null, 'male_dp.png', '[]');
CALL signup(2, 'vsb259', 'pwd2', 'Vikram', 22, '[]', '[]', 'Male', 'Hyderabad', null, null, 'male_dp.png', '[]');
CALL signup(3, 'luk123', 'pwd3', 'Luke', 17, '[]', '[]', 'Male', 'Pasadena', null, null, 'male_dp.png', '[]');
CALL signup(4, 'alx456', 'pwd4', 'Alex', 19, '[]', '[]', 'Female', 'Los Angeles', null, null, 'female_dp.png', '[]');
CALL signup(5, 'pen678', 'pwd5', 'Penny', 21, '[]', '[]', 'Female', 'San Diego', null, null, 'female_dp.png', '[]');
CALL signup(6, 'hwd123', 'pwd6', 'Howard', 20, '[]', '[]', 'Male', 'Pasadena', null, null, 'male_dp.png', '[]');
CALL signup(7, 'shel567', 'pwd7', 'Sheldon', 19, '[]', '[]', 'Male', 'Pasadena', null, null, 'male_dp.png', '[]');
CALL signup(8, 'leo456', 'pwd8', 'Lenoard', 19, '[]', '[]', 'Male', 'Los Angeles', null, null, 'male_dp.png', '[]');
CALL signup(9, 'raj567', 'pwd9', 'Raj', 21, '[]', '[]', 'Male', 'Pasadena', null, null, 'male_dp.png', '[]');
CALL signup(10, 'amy789', 'pwd10', 'Amy', 18, '[]', '[]', 'Female', 'Los Angeles', null, null, 'female_dp.png', '[]');
CALL signup(11, 'krip123', 'pwd11', 'Kripkie', 18, '[]', '[]', 'Male', 'Los Angeles', null, null, 'male_dp.png', '[]');
```

4.3.2. Calls to update liked locations

```
UPDATE Student SET liked_locations_list=JSON_ARRAY_APPEND(liked_locations_list, '$', 'New York')
WHERE student_id = 2;
UPDATE Student SET liked_locations_list=JSON_ARRAY_APPEND(liked_locations_list, '$', 'Los Angeles')
WHERE student_id = 6;
UPDATE Student SET liked_locations_list=JSON_ARRAY_APPEND(liked_locations_list, '$', 'San Francisco')
WHERE student_id = 7;
```

4.3.2. Calls to send friend request

```
CALL friend_request(1,2);
CALL friend_request(1,6);
CALL friend_request(1,7);
CALL friend_request(9,6);
CALL friend_request(9,8);
CALL friend_request(7,8);
CALL friend_request(7,10);
CALL friend_request(3,2);
CALL friend_request(4,2);
CALL friend_request(5,2);
CALL friend_request(3,5);
CALL friend_request(4,5);
CALL friend_request(11,10);
CALL friend_request(11,7);
```

4.3.3. Calls to accept friend request

```
CALL friend_request_accept(1,2);
CALL friend_request_accept(1,6);
CALL friend_request_accept(1,7);
CALL friend_request_accept(9,6);
CALL friend_request_accept(9,8);
CALL friend_request_accept(7,8);
CALL friend_request_accept(7,10);
CALL friend_request_accept(3,2);
CALL friend_request_accept(4,2);
CALL friend_request_accept(5,2);
CALL friend_request_accept(3,5);
```

4.3.4. Call to create diary entry

```
CALL diary_entry(2, 100,'My First Project',NULL,'My first project is to make a database','database.png',2,1,10);
```

4.3.5. Call to edit entry title

```
CALL edit_entry_title(100,'My Updated Project');
```

4.3.6. Call to display Student entries

```
CALL display_student_entries(2);
```

4.3.7. Call to display Student details

```
CALL display_student_details(1);
```

4.3.8. Call to display friends of a Student

CALL *friends*(1);

4.3.9. Call to display friends of friends of a Student

CALL *friends_of_friends*(1);

4.3.10 . Calls to display entries by Student's friends of friends having keyword 'database'

CALL *friends_of_friends_search*(1, **'database'**);

4.3.11. Calls to display entries by Student's friends last week

CALL *friends_entries_last_week*(1);

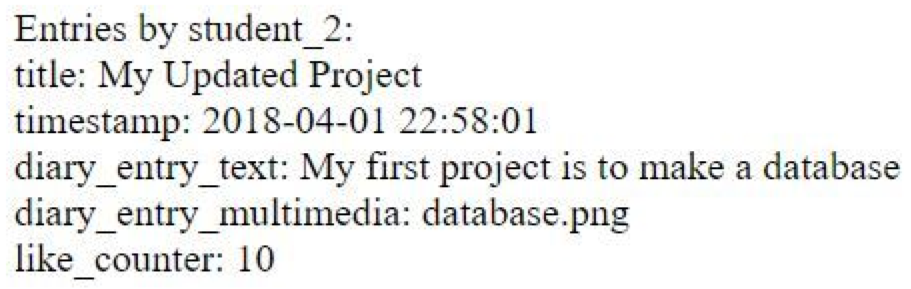
4.3.12. Calls to display locations liked by a Student's friends

CALL *friends_liked_locations*(1)

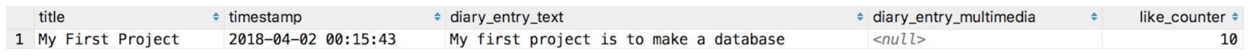
4.4. Selected SQL Outputs

Note: All the outputs are from PHP directly and from DataGrip which we used as SQL IDE. We will be attaching the PHP file along with the document.

4.4.1. Display the Diary Entry by the student_2

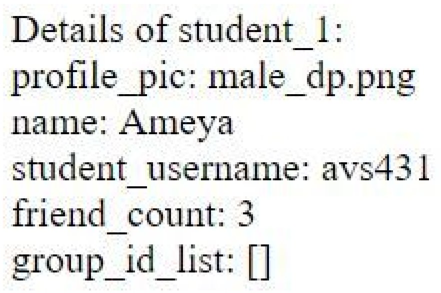


```
Entries by student_2:
title: My Updated Project
timestamp: 2018-04-01 22:58:01
diary_entry_text: My first project is to make a database
diary_entry_multimedia: database.png
like_counter: 10
```

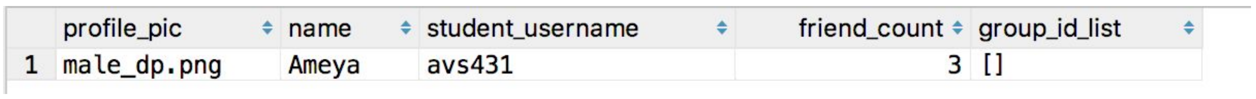


	title	timestamp	diary_entry_text	diary_entry_multimedia	like_counter
1	My First Project	2018-04-02 00:15:43	My first project is to make a database	<null>	10

4.4.2. Display details of student_1

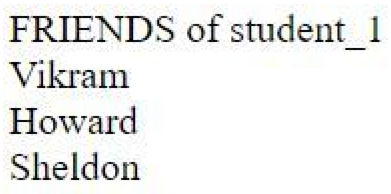


```
Details of student_1:
profile_pic: male_dp.png
name: Ameya
student_username: avs431
friend_count: 3
group_id_list: []
```

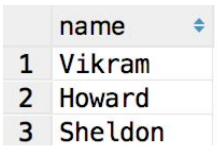


	profile_pic	name	student_username	friend_count	group_id_list
1	male_dp.png	Ameya	avs431	3	[]

4.4.3. Display friends of student_1

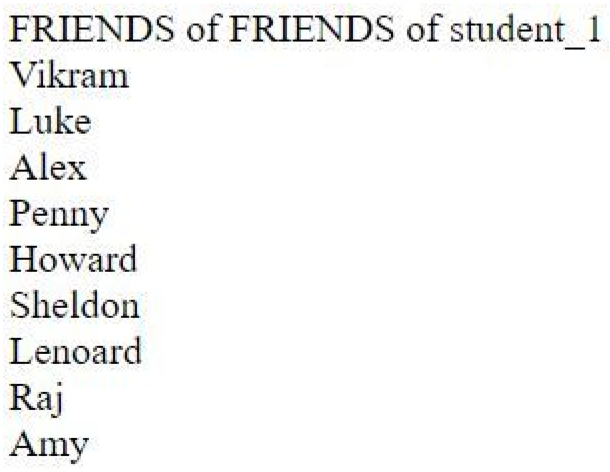


```
FRIENDS of student_1
Vikram
Howard
Sheldon
```

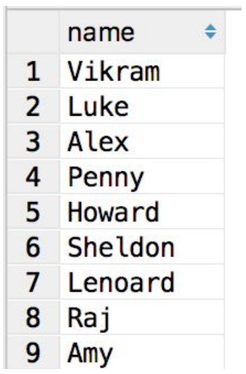


	name
1	Vikram
2	Howard
3	Sheldon

4.4.4. Display friends of friends of student_1

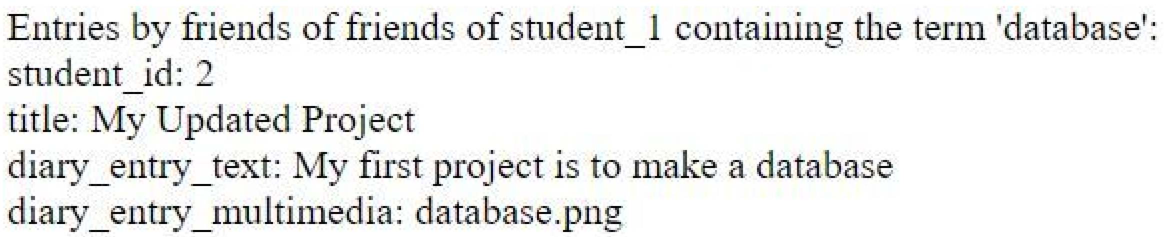


```
FRIENDS of FRIENDS of student_1
Vikram
Luke
Alex
Penny
Howard
Sheldon
Lenoard
Raj
Amy
```

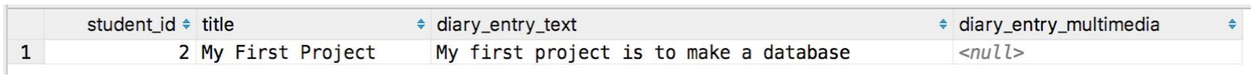


	name
1	Vikram
2	Luke
3	Alex
4	Penny
5	Howard
6	Sheldon
7	Lenoard
8	Raj
9	Amy

4.4.5. Display diary entries by student's friends of friends containing the word 'database'

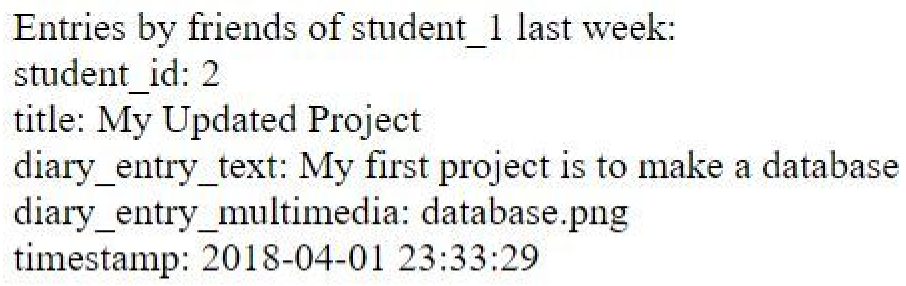


```
Entries by friends of friends of student_1 containing the term 'database':
student_id: 2
title: My Updated Project
diary_entry_text: My first project is to make a database
diary_entry_multimedia: database.png
```

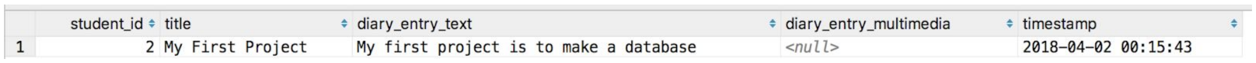


	student_id	title	diary_entry_text	diary_entry_multimedia
1	2	My First Project	My first project is to make a database	<null>

4.4.6. Display diary entries by friends of student during the week

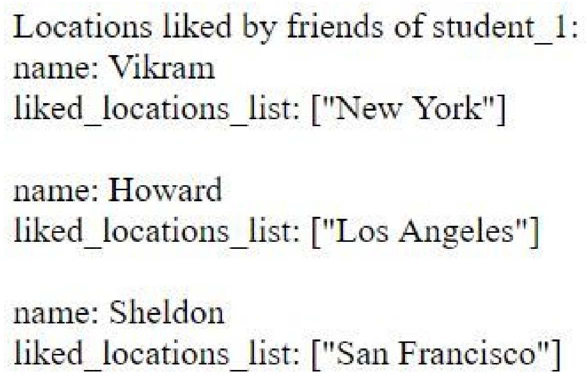


```
Entries by friends of student_1 last week:
student_id: 2
title: My Updated Project
diary_entry_text: My first project is to make a database
diary_entry_multimedia: database.png
timestamp: 2018-04-01 23:33:29
```



	student_id	title	diary_entry_text	diary_entry_multimedia	timestamp
1	2	My First Project	My first project is to make a database	<null>	2018-04-02 00:15:43

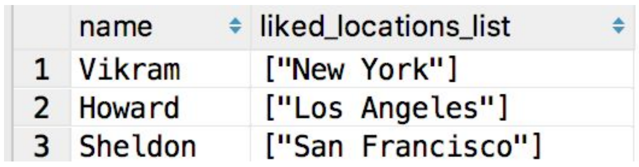
4.4.7 Display locations liked by friends of a student



```
Locations liked by friends of student_1:
name: Vikram
liked_locations_list: ["New York"]

name: Howard
liked_locations_list: ["Los Angeles"]

name: Sheldon
liked_locations_list: ["San Francisco"]
```



	name	liked_locations_list
1	Vikram	["New York"]
2	Howard	["Los Angeles"]
3	Sheldon	["San Francisco"]