University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2
Date: March 1, 2023

Group Number: 197

Name	Student Number	CS Alias (Userid)	Preferred Email Address
Colis Cheng	25934150	U6N0B	colischeng@gmail.com
Kathy Khong	71453039	Q3H2B	kaathykhong@gmail.com
Kaushik Kolla	74167503	H9K3B	kolla.kaushik123@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

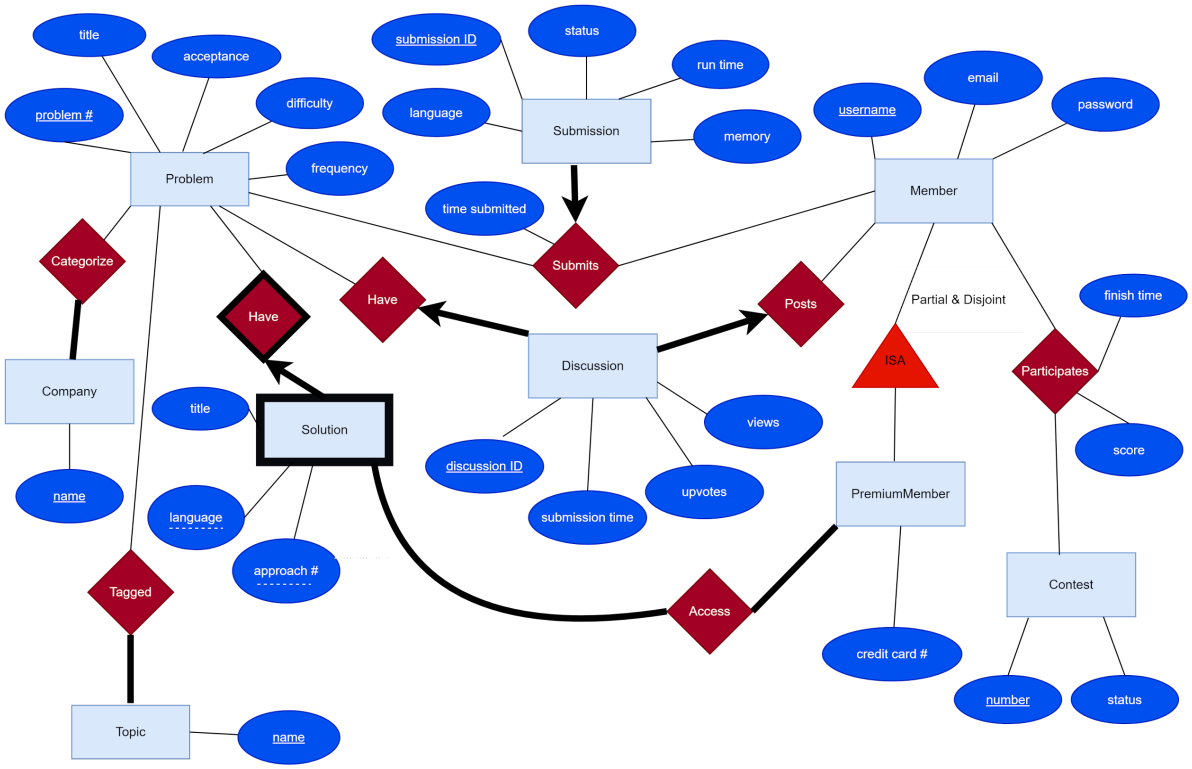
---

2. A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.

Our team is modeling the design of Leetcode, the data structures and algorithms interview preparation website. We model the distinction between paid and non-paid members and access to questions, curated solutions, and community generated solutions.

---

3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why. If you have decided not to implement the suggestions given by your project mentor, please be sure to leave a note stating why. This is not to say that you must do everything that your project mentor says. In many instances, there are trade-offs between design choices and your decision may be influenced by different factors. That being said, your TAs will often leave suggestions that are meant to help massage your project into a form that will fit with the requirements in future project milestones. If you choose not to take their advice, it would be helpful for them to know why in order to better assist the group moving forward.



*Implemented TA's suggestion of adding the Partial and Disjoint specification in the IsA relationship.
*Also, “credit card #” is no longer part of the primary key for PremiumMember, it just has to be not null. This makes it easier to model in a separate table and allows the same credit card to be used by 2 or more different members

---

4. The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.

b. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.

PK are underlined
FK are bolded

Entities

- Member(
  username: string,
  email: string UNIQUE,
  password: string) <- {email} is a CK
- PremiumMember(
  username: string,
  creditCardNum: char(16) NOT NULL)
- Problem(
  problemNum: int,
  title: string UNIQUE,
  acceptance: float,
  difficulty: string,
  frequency: float) <- {title} is a CK
- Solution(
  problemNum: int,
  lang: string,
  approachNum: int,
  title: string)
- Discussion(
  discussionID: int,
  submissionTime: dateTime,
  upvotes: int,
  viewsNum: int, <- {memberUsername, submissionTime} is a CK
  memberUsername: string NOT NULL,
  problemNum: int NOT NULL
  (memberUsername, submissionTime) UNIQUE)
- Submission(
  submissionID: int,
  lang: string,

status: string,
runtime: float,
memory: float,
problemNum: int NOT NULL,
username: string NOT NULL
timeSubmitted: dateTime, <- {username, timeSubmitted} is a CK
(username, timeSubmitted) UNIQUE)

- Contest(
  number: int,
  status: string)
- Company(name: string)
- Topic(name: string)

Relationships

- CompanyCategorizedProblem(
  problemNum: int,
  companyName: string)
- TopicTaggedProblem(
  problemNum: int,
  topicName: string)
- MemberParticipatesInContest(
  username: string,
  contestNumber: int,
  finishTime,
  score)

*All Premium Members having access to all Leetcode Solutions does not need to be explicitly stated (would be a significant waste of memory), but this relationship does exist. Furthermore, it was stated that we would need SQL assertions to represent this (something that has not been covered yet).

---

5. Functional Dependencies (FDs)

a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).

PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

***Meaningful FDs not identified through PK/CK are show in bold**

Entities

Problem:
- problemNum -> title, acceptance, difficulty, frequency
- title -> problemNum, acceptance, difficulty, frequency

Solution
- problemNum, lang, approachNum -> title
- **problemNum, approachNum -> title** (solutions written in different languages have the same title and {problemNum, approachNum} is not unique)

Discussion
- discussionID -> submissionTime, upvotes, viewsNum, memberUsername, problemNum)
- memberUserName, submissionTime -> discussionID, problemNum, upvotes, viewsNum

Submission
- submissionID -> lang, status, runtime, memory, problemNum, username, timeSubmitted
- username, timeSubmitted -> submissionID, lang, status, runtime, memory, problemNum
- **runtime -> status** (could be float value, or “N/A” if not accepted; “N/A” is represented as -1 for domain consistency)
- **memory -> status** (could be float value, or “N/A” if not accepted; “N/A” is represented as -1 for domain consistency)

Member
- username -> email, password
- email -> username, password

PremiumMember
- username -> creditCardNum

Contest
- number -> status

Company
- Only 1 trivial FD

Topic
- Only 1 trivial FD

Relationships

CompanyCategorizedProblem

- Only 1 trivial FD

TopicTaggedProblem

- Only 1 trivial FD

MemberParticipatesInContest

- username, contestNumber -> finishTime, score

Note: In your list of FDs, there must be some kind of valid FD other those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process.

---

6. Normalization

a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization. You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, ...). ALL Tables must be listed, not only the ones post normalization.

PK are underlined
FK are bolded

Entities

- Member(
  username: string,
  email: string UNIQUE,
  password: string)

  Keys: {userName}, {email}

  username -> email, password
  email -> username, password

  For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

- PremiumMember(
  username: string,
  creditCardNum: char(16) NOT NULL)

  Keys: {userName}

  username -> creditCardNum

  For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

- Problem(
  problemNum: int,
  title: string UNIQUE,
  acceptance: float,
  difficulty: string,

frequency: float)

Keys: {problemNum}, {title}

problemNum -> title, acceptance, difficulty, frequency
title -> problemNum, acceptance, difficulty, frequency

For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

- Solution needs to be normalized(
  problemNum: int,
  lang: string,
  approachNum: int,
  title: string)

  Keys: {problemNum, lang, approachNum}

  problemNum, lang, approachNum -> title
  problemNum, approachNum -> title

  {problemNum, approachNum} is not a superkey. Normalization is needed

  Decompose on *problemNum, approachNum -> title:*

  Solution1(**problemNum**, approachNum, title) and

  Solution2(**problemNum**, lang, approachNum)

  No implicit FDs and now all FDs satisfy the definition of BCNF. Normalization is complete.

- Discussion(
  discussionID: int,
  submissionTime: dateTime,
  upvotes: int,
  viewsNum: int,
  memberUsername: string NOT NULL,
  problemNum: int NOT NULL)

  Keys: {discussionID}, {memberUsername, submissionTime}

discussionID -> submissionTime, upvotes, viewsNum, memberUsername, problemNum)
memberUserName, submissionTime -> discussionID, problemNum, upvotes, viewsNum

For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

- Submission needs to be normalized(
  submissionID: int,
  lang: string,
  status: string,
  runtime: float,
  memory: float,
  problemNum: int NOT NULL,
  username: string NOT NULL
  timeSubmitted: dateTime)

Keys: {submissionID}, {username, timeSubmitted}

submissionID -> lang, status, runtime, memory, problemNum, username, timeSubmitted
username, timeSubmitted -> submissionID, lang, status, runtime, memory, problemNum
runtime -> status
memory -> status

{runtime} and {memory} are not superkeys. Normalization is needed.

Decompose on *runtime -> status:*

Submission1(runtime, status) and

Submission2(submissionID, lang, runtime, memory, **problemNum**, **userName**, timeSubmitted)

We cannot decompose memory -> status because out of Submission1 and Submission2, there is no single relation that contains all of {memory, status}. Normalization is complete.

- Contest(
  number: int,
  status: string)

  Keys: {number}

  number -> status

  For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

- Company(name: string)

  Key: {name}

  Only trivial dependencies; no normalization needed.

- Topic(name: string)

  Key: {name}

  Only trivial dependencies; no normalization needed.

Relationships

- CompanyCategorizedProblem(
  problemNum: int,
  companyName: string)

  Key: {problemNum, companyName}

  Only trivial dependencies; no normalization needed.

- TopicTaggedProblem(
  problemNum: int,
  topicName: string)

  Key: {problemNum, topicName}

  Only trivial dependencies; no normalization needed.

- MemberParticipatesInContest(
  username: string,

contestNumber: int,
finishTime,
score)

Key: {userName, contestNumber}

username, contestNumber -> finishTime, score

For all non-trivial functional dependencies X->b, X is a superkey for a relation. No normalization needed.

---

7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc.

```
CREATE TABLE Member
(username CHAR(50),
 email CHAR(100),
 password CHAR(50),
PRIMARY KEY (username),
UNIQUE (email));

CREATE TABLE PremiumMember
(username CHAR(50),
 creditCardNum CHAR(16) NOT NULL,
PRIMARY KEY (username),
FOREIGN KEY (username) REFERENCES Member(username));

CREATE TABLE Problem
(problemNum INTEGER,
 title CHAR(120),
 acceptance FLOAT,
 difficulty CHAR(6),
PRIMARY KEY (problemNum),
UNIQUE (title));

CREATE TABLE Solution1
(problemNum INTEGER,
 approachNum INTEGER,
 title CHAR(120),
PRIMARY KEY (problemNum, approachNum),
FOREIGN KEY (problemNum) REFERENCES Problem(problemNum));

CREATE TABLE Solution2
(problemNum INTEGER,
 lang CHAR(10),
 approachNum INTEGER,
PRIMARY KEY (problemNum, lang, approachNum),
FOREIGN KEY (problemNum) REFERENCES Problem(problemNum));

CREATE TABLE Discussion
(discussionID INTEGER,
 submissionTime DATETIME,
 upvotes INTEGER,
```

```
 viewsNum INTEGER,
 memberUsername CHAR(120) NOT NULL,
 problemNum int NOT NULL,
PRIMARY KEY (discussionID),
UNIQUE (memberUsername, submissionTime),
FOREIGN KEY (memberUsername) REFERENCES Member(username),
FOREIGN KEY (problemNum) REFERENCES Problem(problemNum));

CREATE TABLE Submission1
(runtime FLOAT,
 status CHAR(8),
PRIMARY KEY (runtime));

CREATE TABLE Submission2
(submissionID INTEGER,
 lang CHAR(10),
 status CHAR(8),
 runtime FLOAT,
 memory FLOAT,
 problemNum INTEGER NOT NULL,
 username CHAR(50) NOT NULL,
 timeSubmitted DATETIME,
PRIMARY KEY (submissionID),
UNIQUE (username, timeSubmitted),
FOREIGN KEY (problemNum) REFERENCES Problem(problemNum),
FOREIGN KEY (username) REFERENCES Member(username));

CREATE TABLE Contest
(number INTEGER,
 status CHAR(9),
PRIMARY KEY(number));

CREATE TABLE Company
(name CHAR(50),
PRIMARY KEY(name));

CREATE TABLE Topic
(name CHAR(50),
PRIMARY KEY(name));

CREATE TABLE CompanyCategorizedProblems
(problemNum INTEGER,
```

```
 companyName CHAR(50),
PRIMARY KEY(problemNum, companyName),
FOREIGN KEY(problemNum) REFERENCES Problem(problemNum),
FOREIGN KEY(companyName) REFERENCES Company(name));

CREATE TABLE TopicTaggedProblems
(problemNum INTEGER,
 topicName CHAR(50),
PRIMARY KEY(problemNum, topicName),
FOREIGN KEY(problemNum) REFERENCES Problem(problemNum),
FOREIGN KEY(topicName) REFERENCES Topic(name));

CREATE TABLE MemberParticipatesInContest
(username CHAR(50),
 contestNumber INTEGER,
 finishTime DATETIME,
 score INTEGER,
PRIMARY KEY(username, contestNumber),
FOREIGN KEY(username) REFERENCES Member(username),
FOREIGN KEY(contestNumber) REFERENCES Contest(number));
```

---

8. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later on.

/*Member*/

```
INSERT INTO Member(username, email, password)
VALUES ('tylershaw', 'tylershaw1999@gmail.com', 'Tsshaw10!10@');

INSERT INTO Member(username, email, password)
VALUES ('arianagrande', '7ringsbyari@gmail.com', 'Donutandsweets911!');

INSERT INTO Member(username, email, password)
VALUES ('ririrocks2023', 'rihanna_royal_superbowl@gmail.com', 'shinewithFenty101$$!');

INSERT INTO Member(username, email, password)
VALUES ('swiftyswiftxpxp', 'tswiftlovestory121019@gmail.com', 'Shakeitoff2015!');

INSERT INTO Member(username, email, password)
VALUES ('samsmith67', 'ssmithalldayyee0420@gmail.com', 'st@ywithMe005');

INSERT INTO Member(username, email, password)
VALUES ('eddySheeran', 'edsheerperf1010@gmail.com', 'strawberryLips@@!96');

INSERT INTO Member(username, email, password)
VALUES ('baebeyonce711', 'crazyinlove9999@gmail.com', '2000heartsForYOU!');

INSERT INTO Member(username, email, password)
VALUES ('smileyMiley', 'smileyallday@gmail.com', '2023BouqetFlowers!');

INSERT INTO Member(username, email, password)
VALUES ('lalisablink', 'lalisarocks@gmail.com', 'BlackPinkinyourarea<3!');

INSERT INTO Member(username, email, password)
VALUES ('starboy', 'theweeknd@gmail.com', 'darkKnight2018$');
```

/*Premium Member*/

```
INSERT INTO PremiumMember(username,creditCardNum)
VALUES ('arianagrande', '1111222233334444');

INSERT INTO PremiumMember(username,creditCardNum)
VALUES ('tylershaw', '5555666677778888');
```

```
INSERT INTO PremiumMember(username,creditCardNum)
VALUES ('ririrocks2023', '1000272791918888');

INSERT INTO PremiumMember(username,creditCardNum)
VALUES ('swiftyswiftxpxp', '5151707022228866');

INSERT INTO PremiumMember(username,creditCardNum)
VALUES ('samsmith67', '5151707022228866');
```

/*Problem*/

```
INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (1, 'Two Sum', '49.5', 'Easy');

INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (2, 'Add Two Numbers', '40.2', 'Medium');

INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (3, 'Longest Substring Without Repeating Characters', '33.8', 'Medium');

INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (4, 'Median of Two Sorted Arrays', '36.0', 'Hard');

INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (5, 'Longest Palindromic Substring', '32.4', 'Medium');

INSERT INTO Problem(problemNum, title, acceptance, difficulty)
VALUES (6, 'Zigzag Conversion', '44.7', 'Medium');
```

/*Solution1*/

```
INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (1, 1, 'Two Sum BruteForce');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (1, 2, 'Two Sum Using HashMap');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (2, 1, 'Add Two Numbers Elementary Math');

INSERT INTO Solution1(problemNum, approachNum, title)
```

VALUES (3, 1, 'Longest Substring Without Repeating Characters Brute Force');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (3, 2, 'Longest Substring Without Repeating Characters Sliding Window');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (4, 1, 'Median of Two Sorted Arrays Brute force');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (4, 2, 'Median of Two Sorted Arrays Two Pointers');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (5, 1, 'Longest Palindromic Substring Brute force');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (5, 2, ' Longest Palindromic Substring Dynamic Programming');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (6, 1, 'Zigzag Conversion Simulate Zigzag Movement');

INSERT INTO Solution1(problemNum, approachNum, title)
VALUES (6, 2, 'Zigzag Conversion String Traversal');

/*Solution2*/
INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (1, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (1, 'Java', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (1, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (1, 'Python3', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (2, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (2, 'Java', 2);

```
INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (2, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (2, 'Python3', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (3, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (3, 'Java', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (3, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (3, 'Python3', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (4, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (4, 'Java', 2);


INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (4, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (4, 'Python3', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (5, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (5, 'Java', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (5, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
```

VALUES (5, 'Python3', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (6, 'Java', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (6, 'Java', 2);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (6, 'Python3', 1);

INSERT INTO Solution2(problemNum, lang, approachNum)
VALUES (6, 'Python3', 2);

/*Discussion*/
INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (1, '2020-01-01 12:01:39', 200, 784, 'tylershaw',1);

INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (2, '2020-01-02 12:03:40', 304, 64, 'arianagrande',2);

INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (3, '2020-05-02 08:03:40', 500, 1000, 'arianagrande',3);

INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (4, '2021-08-26 06:08:29', 30, 7, 'ririrocks2023',4);

INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (5, '2022-07-18 09:11:27', 397, 111, 'smileyMiley',5);

INSERT INTO Discussion(discussionID, submissionTime, upvotes, viewsNum,
memberUsername, problemNum)
VALUES (6, '2022-07-19 11:11:41', 600, 1392, 'smileyMiley',6);

/*Submission1*/

```
INSERT INTO Submission1(runtime, status)
VALUES (1.5, 'accepted');

INSERT INTO Submission1(runtime, status)
VALUES (-1, 'rejected');

INSERT INTO Submission1(runtime, status)
VALUES (0.4, 'accepted');

INSERT INTO Submission1(runtime, status)
VALUES (0.2, 'accepted');

INSERT INTO Submission1(runtime, status)
VALUES (0.5, 'accepted');
```

/*Submission2*/

```
INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (1, 'Java', 1.5, 39, 1, 'baebeyonce711',  '2022-01-03 02:37:39');

INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (2, 'Java', -1, -1, 2, 'baebeyonce711',  '2022-01-03 01:37:50');

INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (3, 'Python3', 0.4, 32, 4, 'lalisablink',  '2021-04-22 08:29:58');

INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (4, 'Python3', 0.2, 37, 5, 'eddySheeran',  '2021-08-31 13:33:57');

INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (5, 'Java', 0.5, 49, 6, 'starboy',  '2021-12-31 20:47:08' );

INSERT INTO Submission2(submissionID,  lang, runtime,  memory,  problemNum, username,
timeSubmitted)
VALUES (6, 'Python3', -1, -1, 3, 'swiftyswiftxpxp',  '2021-09-21 21:30:08');
```

/*Contest*/

```
INSERT INTO Contest(number, status)
VALUES (1, 'ended');

INSERT INTO Contest(number, status)
VALUES (2, 'ended');

INSERT INTO Contest(number, status)
VALUES (3, 'ended');

INSERT INTO Contest(number, status)
VALUES (4, 'open');

INSERT INTO Contest(number, status)
VALUES (5, 'open');

INSERT INTO Contest(number, status)
VALUES (6, 'open');
```

/*Company*/

```
INSERT INTO Company(name)
VALUES ('Microsoft');

INSERT INTO Company(name)
VALUES ('Facebook');

INSERT INTO Company(name)
VALUES ('Google');

INSERT INTO Company(name)
VALUES ('Tiktok');

INSERT INTO Company(name)
VALUES ('Bloomberg');

INSERT INTO Company(name)
VALUES ('Apple');
```

/*Topic*/

```
INSERT INTO Topic(name)
VALUES ('Dynamic Programming');

INSERT INTO Topic(name)
VALUES ('Arrays');

INSERT INTO Topic(name)
VALUES ('Linked lists');

INSERT INTO Topic(name)
VALUES ('Strings');

INSERT INTO Topic(name)
VALUES ('Sorting');
```

/*CompanyCategorizedProblems*/

```
INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (1, 'Microsoft');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (1, 'Facebook');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (1, 'Google');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (2, 'Bloomberg');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (2, 'Google');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (3, 'Google');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (3, 'Tiktok');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (4, 'Apple');
```

```
INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (4, 'Facebook');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (5, 'Google');

INSERT INTO CompanyCategorizedProblems(problemNum, companyName )
VALUES (6, 'Tiktok');
```

/*TopicTaggedProblems*/

```
INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (1, 'Dynamic Programming');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (1, 'Arrays');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (2, 'Arrays');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (3, 'Arrays');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (3, 'Strings');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (4, 'Arrays');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (4, 'Sorting');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (5, 'Strings');

INSERT INTO TopicTaggedProblems(problemNum, topicName)
VALUES (6, 'Arrays');
```

/*MemberParticipatesInContest*/

```
INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('ririrocks2023', 2,  '2023-01-31 20:47:08', '79');

INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('smileyMiley', 1,  '2022-12-31 20:49:08', '90');

INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('samsmith67', 3,  '2023-02-14 21:56:01', '68');

INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('starboy', 2,  '2023-01-31 22:33:01', '72');

INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('arianagrande', 1,  '2022-12-31 8:49:01', '85');

INSERT INTO MemberParticipatesInContest(username, contestNumber, finishTime, score)
VALUES ('baebeyonce711', 3,  '2023-02-14 12:04:01', '81');
```