CS585 HW2:SQL Rubrics Fall 2021

Note for Graders:

- **-1** if files are uploaded in .zip format.
- **-1** if the student does not mention the DB software used to write the queries.
- **-1** if the assignment is submitted after the deadline (10/13/21 11:59 PM PDT). [Saty] - mild delay, upto 30 minutes, is ok]
- Maximum score for every student is capped at 6.

Question 1:

Total: 2 marks

Aim:
To build all the tables mentioned in the question and insert (required) values in the correct format.

Required approach:
Build "Employee", "Meeting", "Notification", "Symptom", "Scan", "Test", "Case", "Health_Status", and insert all the required values. Identify the right primary key and foreign key relationships.

Sample Solution - DB software MySQL:

```
CREATE TABLE Employee(
ID          varchar(50),
name        varchar(50),
office_num    varchar(50),
floor_num     int CHECK (floor_num BETWEEN 1 and 10),
phone_num     varchar(50) NOT NULL UNIQUE,
email_add     varchar(50)NOT NULL UNIQUE,
PRIMARY KEY(ID)
);

CREATE TABLE Meeting(
meeting_ID    varchar(50),
employee_ID varchar(50),
room_num      varchar(50) NOT NULL,
floor_num int NOT NULL CHECK (floor_num BETWEEN 1 and 10),
meeting_start_time  int  NOT NULL CHECK (meeting_start_time BETWEEN 8 and 18),
PRIMARY KEY(meeting_id, employee_id),
FOREIGN KEY Meeting (employee_ID) references Employee(ID)
);

CREATE TABLE Notification(
notification_ID         varchar(50),
employee_ID varchar(50),
notification_date date NOT NULL,
notification_type varchar(50) NOT NULL CHECK (notification_type IN
('mandatory','optional')),
PRIMARY KEY (notification_ID),
FOREIGN KEY notification (employee_ID)references Employee(ID),
FOREIGN KEY notification (employee_ID)references Meeting(employee_ID)
);


CREATE TABLE Symptom(
row_ID          varchar(50),
employee_ID varchar(50),
date_reported date NOT NULL,
symptom_ID  int  NOT NULL CHECK (symptom_ID BETWEEN 1 and 5),
PRIMARY KEY(row_ID),
FOREIGN KEY Symptom (employee_ID) references Employee(ID)
);

CREATE TABLE Scan(
scan_ID         varchar(50),
scan_date       date NOT NULL,
scan_time int NOT NULL CHECK (scan_time BETWEEN 0 and 23),
employee_ID varchar(50),
temperature  float NOT NULL,
PRIMARY KEY(scan_ID),
FOREIGN KEY Symptom (employee_ID) references Employee(ID)
);

CREATE TABLE Test(
test_ID         varchar(50),
location        varchar(50) NOT NULL,
test_date       date NOT NULL,
test_time int NOT NULL CHECK (scan_time BETWEEN 0 and 23),
employee_ID varchar(50),
test_result varchar(50) CHECK (test_result IN ('positive','negative')),
PRIMARY KEY(test_ID),
FOREIGN KEY Test (employee_ID) references Symptom(employee_ID),
FOREIGN KEY Test (employee_ID) references Employee(ID),
FOREIGN KEY Test (employee_ID) references Scan(employee_ID)
);
```

```
CREATE TABLE Case_Table(
case_ID        varchar(50),
employee_ID varchar(50),
case_date date NOT NULL,
resolution     varchar(50) NOT NULL CHECK (resolution IN ('back to work','left the company',
'deceased')),
PRIMARY KEY (case_ID),
FOREIGN KEY (employee_ID) references Test(employee_ID)
);

CREATE TABLE Health_Status(
row_ID         varchar(50),
employee_ID varchar(50),
health_status_date date NOT NULL,
status varchar(50) NOT NULL CHECK (status IN ('sick','hospitalized', 'well')),
PRIMARY KEY Health_Status (row_ID),
FOREIGN KEY (employee_ID) references Case_Table(employee_ID)
);

INSERT INTO
Employee(ID, name, office_num, floor_num, phone_num, email_add)
VALUES
(1, 'a',1,1,1,'a@gmail.com'),
(2, 'b',1,2,12,'b@gmail.com'),
(3, 'c',3,3,572,'c@gmail.com'),
(4, 'd',1,4,571,'d@gmail.com'),
(5, 'e',2,5,585,'e@gmail.com'),
(6, 'f',3,1,526,'f@gmail.com'),
(7, 'g',1,7,789,'g@gmail.com'),
(8, 'h',2,8,456,'h@gmail.com'),
(9, 'i',3,9,402,'i@gmail.com'),
(10, 'm',1,10,455,'j@gmail.com');

INSERT INTO
Meeting (meeting_ID, employee_ID, room_num, floor_num, meeting_start_time)
VALUES
(1, 1,1,1,10),
(1, 2,1,1,10),
(2, 2,2,1,10),
(2, 4,2,1,10),
(3, 3,2,1,9),
(4, 4,2,5,13),
(5, 5,2,5,13),
(5, 6,2,5,13),
(6, 6,2,5,13),
(7, 7,1,7,16),
(8, 8,1,7,16),
(9, 9,3,9,14),
(10, 10,3,9,14);

INSERT INTO
Notification (notification_ID, employee_ID, notification_date, notification_type)
VALUES
(1, 1,'2021-10-01','mandatory'),
(2, 2,'2021-09-10','mandatory'),
(3, 6,'2021-07-02','optional'),
(4, 8,'2021-09-15','optional');

INSERT INTO
Scan (scan_ID, scan_date, scan_time, employee_ID, temperature)
VALUES
(1, '2021-07-20',8,1,100.0),
(2, '2021-08-02',8,2,97.1),
(3, '2021-09-07',10,3,97),
(4, '2021-06-12',12,4,97),
(5, '2021-05-22',12,5,96),
(6, '2021-01-01',16,6,97.3),
(7, '2021-03-26',16,7,101),
(8, '2021-04-15',17,8,97),
(9, '2021-07-24',13,9,97),
(10, '2021-08-11',13,10,98.2);

INSERT INTO
Symptom (row_ID, employee_ID, date_reported, symptom_ID)
VALUES
(1, 1,'2021-08-11',1),
(2, 2,'2021-07-16',2),
(3, 3,'2021-05-10',3),
(4, 4,'2021-03-02',4),
(5, 5,'2021-08-11',5),
(6, 6,'2021-05-02',5),
(7, 7,'2021-07-18',4),
(8, 8,'2021-02-08',4),
(9, 9,'2021-06-20',4),
(10, 10,'2021-08-17',3);

INSERT INTO
Test (test_ID, location, test_date, test_time, employee_ID, test_result)
VALUES
(1, 'clinic','2021-05-01',12,1,'positive'),
(2, 'clinic','2021-07-10',12,2,'positive'),
(5, 'hospital','2021-05-18',17,6,'negative'),
(7, 'clinic','2021-06-19',10,8,'negative'),
(6, 'hospital','2021-05-18',17,6,'positive'),
(8, 'hospital','2021-09-19',17,6,'positive');

INSERT INTO
```

Case_Table (case_ID, employee_ID, case_date, resolution)
VALUES
(1,1,'2021-07-01','back to work'),
(2,2,'2021-06-10','deceased'),
(6,6,'2021-09-05','left the company'),
(8,8,'2021-05-22','back to work');

INSERT INTO
Health_Status (row_ID, employee_ID, health_status_date, status)
VALUES
(1,1,'2021-05-01','sick'),
(2,2,'2021-07-24','well'),
(3,2,'2021-07-10','sick'),
(4,1,'2021-06-21','well'),
(5,1,'2021-05-16','hospitalized'),
(6,6,'2021-05-18','sick'),
(7,6,'2021-06-02','well'),
(8,8,'2021-09-19','sick'),
(9,8,'2021-09-30','hospitalized');

Rubrics:

- **No deductions** for any query structure to create the entities and their relationships.
- **-1** for missing any entity.
- **-1** for missing any required attribute from the question.
- **-1** for missing primary key or for creating an incorrect primary key.
- **-1** for missing foreign key or for creating an incorrect foreign key.
- **-0.5** for using an inappropriate format. For example, the date column does not use date type(use varchar maybe). [Saty]: using ints for dates is ok
- **-0.5** for each time a student uses an inappropriate range. For example, the floor is out of range between 1 - 10, or time is out of range between 0 - 23, symptom_id out of range between 1 - 5.

Question 2:

Total: 1 mark

Aim:
To find the symptom id with the highest frequency of occurrence in the symptom table.

Required approach:
Count the number of symptom id occurrences in the symptom table, sort it by decreasing order, and get the first row. (use different database systems may have different approaches).

Answer - DB software - MySQL:

Solution 1: Will return only one of the most self-reported symptoms.

```
SELECT symptom_ID
FROM Symptom
GROUP BY symptom_ID
ORDER BY COUNT(symptom_ID)
DESC LIMIT 1;
```

Solution 2: Will return all the most self-reported symptoms.

```
SELECT symptom_ID
FROM Symptom
GROUP BY symptom_ID
HAVING COUNT(*) =
(SELECT MAX(count_symptoms) FROM
(SELECT symptom_ID, COUNT(*) as count_symptoms
FROM Symptom
GROUP BY symptom_ID) as a);
```

No deductions for any alternative approaches that derive the same result.

Rubrics:

- **-0.5** if you return the symptom id with the frequency (the question does not mention you should return the frequency), but the overall approach is correct.
- **-1** for incorrect answer.

Question 3:

Total: 1 mark

Aim:
To extract the sickest floor.

Required approach:
Join tables Employee and Health_Status and extract all the records that indicate Status as Sick in the Health_Status entity. Group these records by floor_num, and display the floor with the highest count.

Answer - DB software - MySQL:

Solution 1: Will only return one of the sickest floors of all time.

```
SELECT floor_num FROM Health_Status h
INNER JOIN Employee e
ON h.employee_ID = e.ID
WHERE status = "sick"
GROUP BY floor_num
ORDER BY count(*) DESC
LIMIT 1;
```

Solution 2: Will only return all the sickest floors of all time.

```
SELECT floor_num FROM Health_Status h
INNER JOIN Employee e
ON h.employee_ID = e.ID
WHERE status = "sick"
GROUP BY floor_num
HAVING COUNT(*) =
(SELECT MAX(count_sick) FROM
(SELECT COUNT(*) as count_sick FROM Health_Status h
INNER JOIN Employee e
ON h.employee_ID = e.ID
WHERE status = "sick"
GROUP BY floor_num) as a);
```

Rubrics:

- **No deductions** for any alternate approaches with a different query structure. These approaches should use the ‘sick’ value in the ‘status’ attribute in the ‘health_status’ entity and give the same output as the above query. Definition of ‘sick’ if correlated with a date period should be mentioned in README.
- **-1 for incorrect answer.**

Question 4:

Total: 1 mark

Aim:
To provide all the required stats between any assumed start and end date.

Answer - DB software - MySQL:

Number of Scans:
```
SELECT count(*)
FROM Scan
WHERE scan_date BETWEEN '2021-05-01' AND '2021-09-30';
```

Number of tests:
```
SELECT count(*)
FROM Test
WHERE test_date BETWEEN '2021-05-01' AND '2021-09-30';
```

Number of employees who self-reported symptoms:
```
SELECT count(*)
FROM (SELECT DISTINCT (employee_id)
FROM Symptom
WHERE date_reported BETWEEN '2021-05-01' AND '2021-09-30'
GROUP BY employee_ID) AS self_reported_employees;
```

Number of positive cases:
```
SELECT count(*)
FROM Test
WHERE
test_result = 'positive' AND
test_date BETWEEN '2021-05-01' AND '2021-09-30';
```

Rubrics:

- **No deductions** for any alternate query structure that can produce the same results as the above queries.
- **-0.25** for missing the DISTINCT keyword in finding the number of employees who self-reported the symptoms.
- **- 0.25** for every wrong query.

Question 5:

Total: 2 marks

Aim: To create any valid query that involves table division

Rubrics:

- **-2** for incorrect query.
- **-1** if the query makes sense and is correct but not based on table division.