# Name: Gautam Pranjal
# USC ID: 8410482919
# Contact tracing System ER diagram

**Assumptions:**

1. The employee table holds the ground truth for the entire company. Minimum alterations are done to this table (explained later in the tradeoff section).
2. Employee must register with the phone application, one person one phone.
3. If the temperature is measured as high for an employee, then he must get tested within a day.
4. Meeting can be conducted in a room only if it's available.
5. Every floor in the company's building will have at least one room.
6. If an employee gets tested, they must change the status on their phone app as soon as possible.
7. The company has enough space to store the database.
8. If any employee dies, they will be removed from the company's phone app. They will still be present in the employee table for future reference.

**Walkthrough of my design:**

1. We start off with the "Floors" table. In a building there can be multiple floors. This entity will store all the floors. All of them will have a unique "floor_id", and a floor name (usually what happens in a company). Following are the relationship with the Floor entity:
    a. Rooms: The Floors entity is connected to the rooms in a 1:m relationship. This is because the floor will have at least one room and can max out till m rooms.
    b. Facilities: There is a 1:m relationship from Floors to Facilities as a floor can have multiple facilities available.
2. The "Rooms" entity has a primary key of "room_id" and a foreign key of "floor_id". We have attributes like loc_on_floor (left wing, right wing, etc), available, room_type (hangout room, meeting room, personal room, etc). Following are the relationships the Room entity has:
    a. Rooms: 1:1, one room can only be present on one floor.
    b. Meeting Rooms: 0:m, a room cannot be booked at all, hence 0 and the same room can be booked multiple times, hence many.
3. Employees Entity: The main table captures the details of the company's employees. It has the "employee_id" as the primary key to uniquely identify a record. The table also stores, the employee's name, phone number, email id, department, designation, regisApp (if the employee is registered with the company's app or not). It can be argued that the regisApp attribute is kind of not required as it is mandatory for an employee to register with the app, but there can be cases when a new employee just joined the company, and his phone doesn't support the app. So, till the time he gets the required device the value for this cell for that employee will be different and can be used to monitor the employee. Following are the relationships it has with other entities.

a. Scanner: 0:m, it is assumed the scanning for temperature is done at random, so an employee can get scanned or not. Hence 0. Many because that employee can get checked for the temperature more than once in a day.
b. Participants: 0: m, an employee doesn't participate in any of the meetings, hence 0 or he can participate in multiple meetings, hence many.
c. Facilities: 1:m, once reaching the office in the morning, he'll go to his work area and might stay there for the entire day, hence 1 or he can move around various facilities present in the company, hence many.
d. Phone App: 1:1, an employee can only register once and must register with the app, hence 1:1 relationship.

4. Participants Entity: The relation between the Employees and the Meeting rooms is many to many. Employees can attend multiple meetings, and, in a meeting, there can be multiple employees, so I used a bridge table, "Participants". Following are the relationships:
    a. Employees:Participants = 0:m. Employee cannot participate in any meetings hence 0.
    b. Meeting Rooms:Participants = 1:m. A meeting room can have 1 or more employees present in the room.

    I used a bridge table here to reduce the data redundancy, that will arise if I connect the Employees table with the Meeting Rooms table directly.
5. Meeting Rooms Entity: Stores the records of the meetings conducted in which room. The primary key meeting _id is used to capture the meeting record. Following are the relationships with other tables:
    a. Meeting Rooms: Rooms = 1:1. Every room entry in the meeting room will have one record in the rooms table.
    b. Meeting Rooms:Participants = 1:m. A meeting room can have 1 or more employees present in the room
    c. Meeting Rooms:Phone App = 1:m. A meeting room can have 1 or more employees present in the room. For 1 employee, one app and if multiple employees present then respective many apps.
6. Phone App Entity: The employee in the company must register with the phone app. I used the employee_id as the primary key for this since the employee can only register once with the app. Doesn't make any sense if the employee has multiple accounts on the app. I have created attributes which captures the test_id (foreign key), meeting_id (foreign key: used to see the meetings the employee attended), status (health status: well/hospitalized/sick), alert_msg (type of message he'll receive like take a covid test or have caution, etc), symptom_id (foreign key), start_date (the date when an employee got tested positive for COVID), expected_return_date (is calculated as the start_date + 2 weeks). Following are the relationships for the phone app entity with the other entities:
    a. Covid Testing: o:m, a "well" person doesn't want to get tested, hence 0. But once notified by the app, the person can take multiple tests within days (like in NBA before the vaccines rolled out, the players had to take 3-4 tests within a span of 3 days before the game).

b. Phone App: 0:m, if no COVID positive case arise in the company, then no one else will be notified. But if a case happens then multiple employees will get notified in their apps via the tracking system.
c. Employees: 1:1 one employee one app
d. Symptoms: 1:1, one employee will select only one record from the Symptom table (explained later in the Symptom Entity section).
e. Facilities: 1:m, one employee can be present in one facility or can move around and go to various facilities.
f. Meeting Rooms: 0:m, the employee cannot participate/attend in any meetings, or he can attend in many meetings.

7. Symptoms Entity: This table record the combination of the 5 symptoms in a total of $2^5 = 32$ rows. This is done so as reduce the data redundancy from the Phone App. With this design the Phone App will need to store just the symptom id which can tell about the combination of symptoms the employee has. It will have a 1:1 relation with the Phone App.
8. Facilities Entity: The Employees to Floor relation is a many to many relations. Many employees can go to many floors. To remove the data redundancy from the employees table, I have used a bridge table, "Facilities" table. Following are the relationships:
    a. Employees: Facilities: 1:m, an employee can either stay at 1 facility for the entire time and not move, hence 1 or he can move around different facilities, hence m.
    b. Floors: Facilities: 1: m. A floor can have a minimum of 1 facility and a max of many, hence m.
9. Scanner Entity: This entity captures the employees who get scanned for temperature randomly in the company. Also stores the date and time when the scan happened. The primary key is the scan_id used to uniquely identify each scan. Following are the relations with other entities:
    a. Employee: 1:m, the temperature scanner scans at least one employee, hence the range from 1 to m.
    b. Covid Testing: 1:1, if the scanned results is high, the employee will have to get himself tested. For one scan the employee will be getting tested once. Hence 1:1.
10. Covid Testing: This entity records the employee's covid test result uniquely using "test_id" as the primary key. Other than that, it also records, the result of the test, where the test was conducted (test_location: on-site/hospital), date and time of test. Following are the relationships with other entities:
    a. Scanner: 0:1, if the temperature scanned is normal, then the employee won't be getting tested for covid hence 0 or else 1 if the temperature is high.
    b. Phone App: 1:1, the test can only belong to one employee and not many, hence this 1:1.

**The design decisions I took or some tradeoffs between my final design with my previous design?**

1. I tried to reduce a lot of data redundancy by introducing the bridge table between employees table and meeting rooms table. Since the employees table won't be touched/altered often, I

made the bridge by introducing the Participants table. All the function can be implemented on the Participants and Meeting Rooms table, without accidently altering the employee's table.

2. Initially, I had a symptom attribute inside the Phone App table, but in this case, there were a lot of NULL's coming in the table which was wasting a lot of space. So, I moved the symptoms to another table (which has a fixed number of rows) and just used the symptom_id to refer it back in the Phone App.
3. I tried not to alter the employees table much and keep it intact as this table represents the ground truth about the employees of the company. I could have added the floor_id column to the employees table but then again, it would have created a lot of redundant data in multiple rows (employee going to 5 floors -> creating 5 more records inside the employees table). Hence, made another table, Facilities to capture the movement of the employee within the company.