CS2102 Project AY2021/2022 SEM 1

Project Group Number: 48
Members:
CHUA YONG SING (A0205846B)
YAN JING YI (A0204822N)
HENG HONG CHUAN(A0182889E)
MOHAMED RIYAS (A0194608W)

1. Team member contributions

Name	Contribution
CHUA YONG SING	● Arrange and tidy up ER diagram for deliverable 1 ● Write and test functionalities: a. view_manager_report() b. approve_meeting() c. change_capacity() d. non_compliance() ● Helped contributes to some parts in final report ● Help to test final proc.sql
YAN JING YI	● Helped to tidy ER diagram and some parts in first deliverable. ● Write and test functionalities: 1. view_future_meeting() 2. remove_employee() 3. join_meeting() 4. leave_meeting() ● Helped make minor edits to some helper functions that are used repeatedly in some procedures ● Helped contributes to some parts in final report ● Help to test final proc.sql
HENG HONG CHUAN	● Arrange and tidy up ER report for deliverable 1 ● Create data for all the tables ● Write and test functionalities: a. view_booking _report() b. book_room() c. unbook_room() d. search_room() ● Helped contributes to some parts in final report ● Help to test final proc.sql
MOHAMED RIYAS	● Arrange and tidy up ER report for deliverable 1 ● Write schema ● Write and test functionalities: a. add_employee() b. add_department() c. add_room() d. remove_department() e. declare_health() f. contact_tracing() ● Helped contributes to some parts in final report ● Help to test final proc.sql

2. ER Diagram

2.1 Original ER Diagram

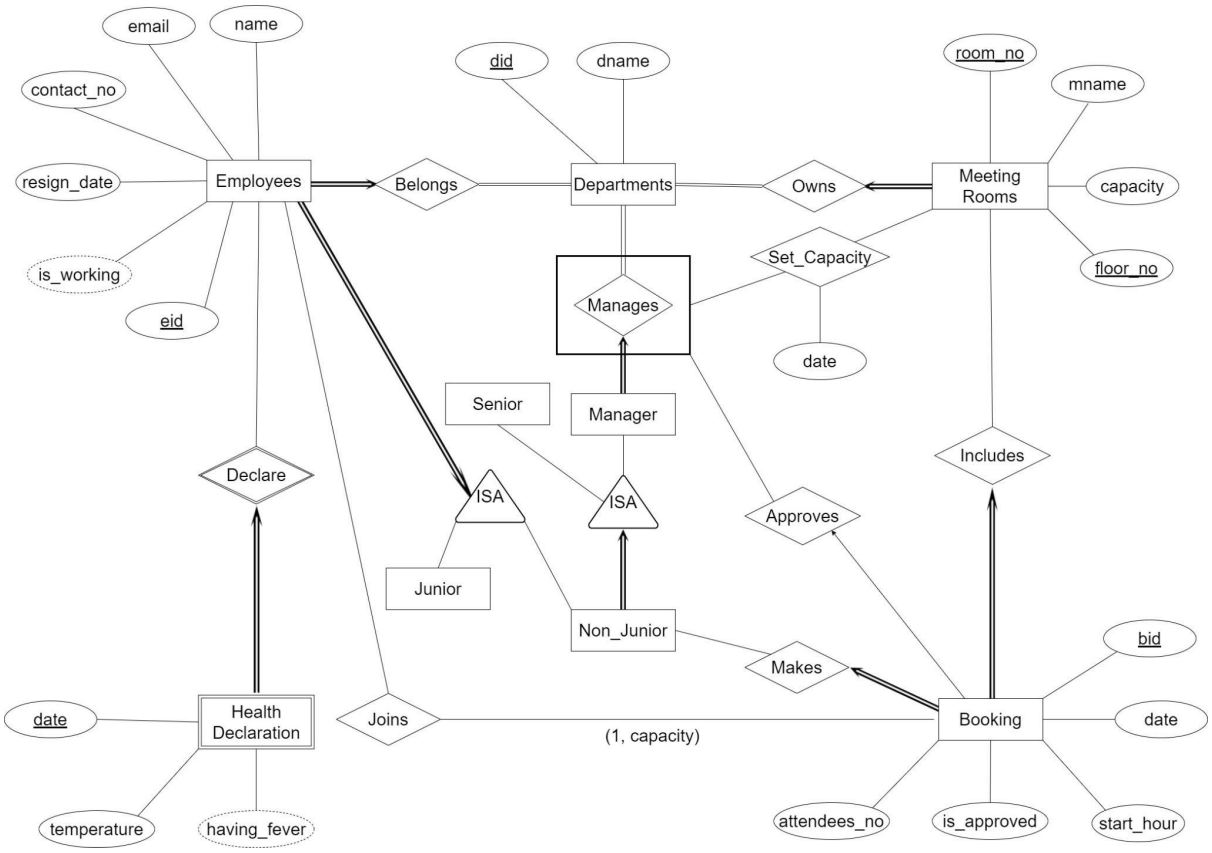

Figure 1. The original ER Diagram.

2.2 Updated ER Diagram

Note that we will be using the proposed ER diagram instead of the original ER diagram (that we submitted on week 6) for our application.

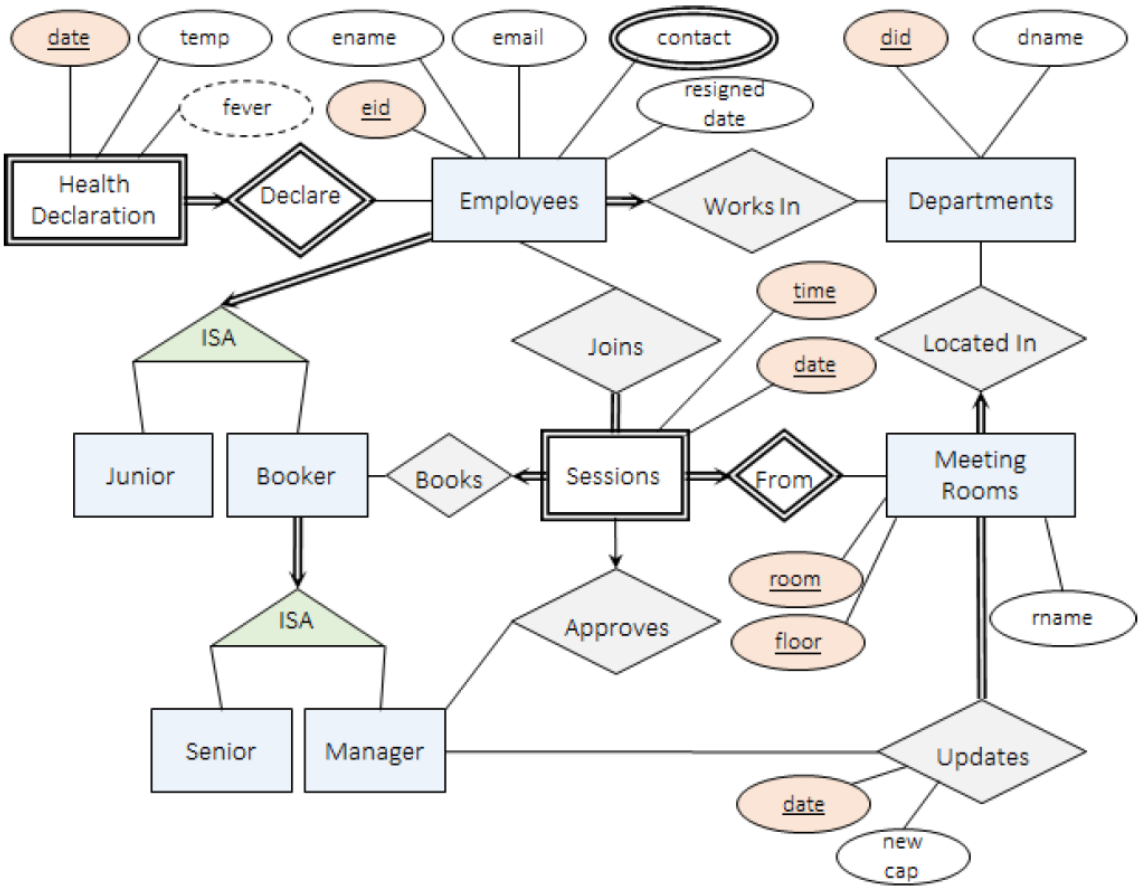

Figure 2. The updated ER Diagram. We will be using this for our application.

2.3 Justification for any non-trivial design decisions made in the ER model

We decided to use the proposed ER diagram instead of the one we submitted in the first deliverable. Following are some significant differences between our original ER and the proposed ER model:

1. Our original ER diagram assumed that the attribute “contact_no” (contact) will only contain 1 number. However, it is not practical as most people have multiple phone numbers. Therefore we adopted the proposed ER which made contact a multi-valued attribute.
2. Our original “Booking” (Sessions) entity was not designed as a weak entity but after looking at the proposed ER, we think that it is more reasonable to design it as a weak entity since it is dependent on the “Meeting Rooms” and will not exist if there is not a meeting room. Also, this helps to ensure that no two sessions can happen at the same meeting room at the same time.

3. We removed the “Manages” entity from our original ER model as the department that a manager manages can be inferred from their department id (did).
4. Instead of recording the capacity as an attribute of the “Meeting Rooms” entity, we feel that the proposed ER’s design to put it as an attribute in the “Updates” relation is a better design. This depicts a clearer picture of the relationship between “Manager” and “Meeting Rooms” where a manager will update the max capacity of a room on specific dates. It will also be easier to query the maximum capacity of a meeting room on the “Updates” relation table to check if an employee can join a meeting on a certain date with the date attribute in the “Updates” table.

2.4 Five of the application’s constraints that are not captured by the proposed ER data model.

1. Each employee will have a unique email address.
   (This will be enforced using the UNIQUE constraint in the schema)
2. The declared temperature can only be between 34 (hypothermia) to 43 Celsius (hyperthermia).
   (This will be enforced using our valid_temp constraint defined in the schema)
3. Participants cannot leave a meeting once it has been approved.
   (Captured using leave_meeting() function)
4. An employee with fever will be removed from all future meetings. The employees, he or she had close contact with, 3 days prior to the date he or she has a fever, are removed from their future meetings in the next 7 days.
   (Captured in function contact_tracing() function)
5. An employee who is having a fever cannot book or join a room.
   (This will be captured using our trigger BookingRoomTrigger and JoinMeetingTrigger)

3. Relational database schema

```
DROP TABLE IF EXISTS
    Departments, Employees, HealthDeclaration, MeetingRooms, Junior, Booker,
    Senior, Manager, BookSessions, Joins, Updates
CASCADE;

-- Departments
CREATE TABLE Departments (
    did INTEGER PRIMARY KEY,
    dname VARCHAR(50) NOT NULL
);

-- Employees (also includes 'Works In' relation)
CREATE TABLE Employees (
     eid INTEGER GENERATED ALWAYS AS IDENTITY (MINVALUE 1 START WITH 1) PRIMARY
KEY,
    ename VARCHAR(50) NOT NULL,
    email VARCHAR(50) UNIQUE,
    home_contact INTEGER,
    mobile_contact INTEGER,
    office_contact INTEGER,
    resigned_date DATE DEFAULT NULL,
    did INTEGER, -- note: we did not use NOT NULL here because when we remove
department, for resigned employees there would be issue
    FOREIGN KEY (did) REFERENCES Departments(did)
);

-- Health Declaration (weak entity)
CREATE TABLE HealthDeclaration (
    hdate DATE,
    temp numeric constraint valid_temp check (temp BETWEEN 34.0 AND 43.0),
    eid INTEGER,
    PRIMARY KEY (eid, hdate),
    FOREIGN KEY (eid) REFERENCES Employees(eid)
        ON UPDATE CASCADE ON DELETE CASCADE
);
```

```sql
-- Meeting Rooms (also includes 'Located In' relation)
CREATE TABLE MeetingRooms (
    room_num INTEGER,
    floor_num INTEGER,
    rname VARCHAR(50) NOT NULL,
    did INTEGER NOT NULL,
    PRIMARY KEY(room_num, floor_num),
    FOREIGN KEY (did) REFERENCES Departments(did)
);

-- ISA Employees to Junior
CREATE TABLE Junior (
    jid INTEGER PRIMARY KEY,
    FOREIGN KEY (jid) REFERENCES Employees(eid)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- ISA Employees to Booker
CREATE TABLE Booker (
    bid INTEGER PRIMARY KEY,
    FOREIGN KEY (bid) REFERENCES Employees(eid)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- ISA Booker to Senior
CREATE TABLE Senior (
    snid INTEGER PRIMARY KEY,
    FOREIGN KEY (snid) REFERENCES Booker(bid)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- ISA Booker to Manager
CREATE TABLE Manager (
    mid INTEGER PRIMARY KEY,
    FOREIGN KEY (mid) REFERENCES Booker(bid)
        ON UPDATE CASCADE ON DELETE CASCADE
);
```

```sql
-- Sessions (weak entity)
-- (also includes 'Books' and 'Approves' relation)
CREATE TABLE BookSessions (
    stime INTEGER constraint valid_hour check (stime BETWEEN -1 AND 24),
    sdate DATE,
    room_num INTEGER,
    floor_num INTEGER,
    bid INTEGER NOT NULL,
    mid INTEGER DEFAULT NULL,
    PRIMARY KEY (stime, sdate, room_num, floor_num),
    FOREIGN KEY (room_num, floor_num) REFERENCES MeetingRooms(room_num, floor_num),
    FOREIGN KEY (bid) REFERENCES Booker(bid),
    FOREIGN KEY (mid) REFERENCES Manager(mid)
);

-- Joins relation (DOES NOT display BookSessions total participation constraint)
CREATE TABLE Joins (
    eid INTEGER,
    stime INTEGER,
    sdate DATE,
    room_num INTEGER,
    floor_num INTEGER,
    PRIMARY KEY (eid, stime, sdate, room_num, floor_num),
    FOREIGN KEY (eid) REFERENCES Employees(eid),
    FOREIGN KEY (stime, sdate, room_num, floor_num) REFERENCES BookSessions(stime,
    sdate, room_num, floor_num)
        ON DELETE CASCADE
);
```

```
-- Updates relation (DOES NOT display MeetingRooms total participation
constraint)
CREATE TABLE Updates (
    udate DATE,
    room_num INTEGER,
    floor_num INTEGER,
    mid INTEGER,
    new_cap INTEGER constraint valid_cap check (new_cap > 0),
    PRIMARY KEY (udate, room_num, floor_num),
    FOREIGN KEY (room_num, floor_num) REFERENCES MeetingRooms(room_num, floor_num),
    FOREIGN KEY (mid) REFERENCES Manager(mid)
);
```

3.1 Justification for any non-trivial design decisions made in the relational database schema

Below are the non-trivial design decisions made in our relational database schema:

1. As captured by the ER diagram, contact is a multi-valued attribute since there is a likelihood that employees have multiple contact numbers. Therefore, in our relational database schema, the Employees table has the attributes home_contact, mobile_contact and office_contact to store the home phone number, mobile phone number and office phone number of the employees respectively.
2. The MeetingRooms table has no attribute for capacity of the meeting room. The current capacity of each meeting room can be obtained from the Updates table where the date and capacity is inserted or updated for every change in capacity of each meeting room. If the MeetingRooms table has a capacity attribute and we do not use the Updates table to obtain the capacity of a meeting room, we will not be able to track the date of update of each meeting room's capacity. Since we will need to remove all booked meetings after the date of update where the number of participants is larger than the updated capacity, we will not be able to do so if we only update the capacity attribute in MeetingRooms. Therefore, the design of not having a capacity attribute and instead obtaining the updated capacity from Updates table will be better.
3. In addition, we have designed the Updates table such that there can only be 1 change to capacity for a particular meeting room for a particular date. That is because if there are multiple entries belonging to the same room for that same date, then it will raise a confusion regarding which entry to use to determine the capacity of the room for that particular date.

4. We combine Sessions weak entity set, and Books and Approves relation into one BookSessions table for our relational database schema. This allows us to keep track of the booker of the meeting session, and whether the meeting session has been approved by a manager. The booker id is included as an attribute for BookSessions as bid and the approval of the meeting session is represented using mid. When the meeting session is approved, the id of the manager that approves the session will be stored in mid, and if it has not been approved, mid will be null.
5. The Employees table includes the Works In relation to record the department each employee belongs to.
6. The MeetingRooms table includes the Located In relation to record which department the meeting room belongs to.
7. The proposed ER diagram reflected a many-to-one participation constraint where an employee must belong (work in) a department. However, when an employee resigns, he or she will not be working in any department, hence in the schema, we have relaxed the constraint as it is not mandatory for an employee to be working in a department.
8. The ‘fever’ attribute under the HealthDeclaration table in the ER diagram is a derived variable. So, instead of having fever as an attribute in the HealthDeclaration, we created a view to derive the fever status from the temperature in HealthDeclaration so as to reduce redundancy in the HealthDeclaration table.
9. The proposed ER diagram reflected a partial participation constraint (Between Updates and Manager) where the capacity of a particular room must be changed by the manager belonging to the same department. In the schema, the manager id is not part of the primary key for the Updates table. That is because, whenever a new room is added, the capacity of that room needs to be included under the Updates table but there is no requirement to have an entry under the manager attribute. Only for subsequent times when the capacity of the room gets changed, an entry under the manager attribute is required. However to prevent multiple update entries from different managers for the same room for the same date, we have created our trigger such that our stated requirements are fulfilled.
10. We have designed the Employee table such that a unique employee ID is auto generated by the schema by using the identity column on the eid attribute.
11. For the Joins table, we have used ‘ON DELETE CASCADE’ on the foreign keys so that when a booking gets deleted, all the attendees of that booking will also get deleted from the Joins table.

3.2 Five of the application's constraints that are not captured by the proposed relational schema. i.e., the constraints that are to be enforced using triggers

1. An employee who is having a fever cannot join a booked meeting.
   (This will be captured using our trigger JoinMeetingTrigger)
2. The employee booking the room immediately joins the booked meeting.
   (Captured using JoiningBookTrigger)
3. A resigned employee cannot join, book or approve a meeting.
   (Captured using JoinMeetingTrigger, BookingRoomTrigger, ApproveSessionTrigger)
4. A manager can only approve a booked meeting in the same department as the manager.
   (Captured using ApproveSessionTrigger)
5. The future meetings of an employee will be removed if an employee resigns.
   (Captured with ResignTrigger)

4. Three most interesting triggers implemented for the application

- Trigger Name: ResignTrigger
  Usage & justification: It is used to remove the employee from future meetings, delete future bookings that they have booked and set their department ID to NULL whenever we call remove_employee() function. This is important because firstly it meets the constraints for resigned employees. Secondly, the trigger sets the department to NULL so that departments with resigned employees and no current employees can be removed in the future, if they need to be removed. Thirdly, doing them in a trigger instead of the function would mean that all the constraints and requirements that we mentioned above will be met and achieved even when the remove_employee() function is not called and someone decides to manually update the resigned_date attribute under the Employees table that indicates that an employee has resigned.

- Trigger name: JoinMeetingTrigger
  Usage & justification: It ensures that employees that are to be added to the Joins table whenever we call join_meeting() function are not resigned employees, or those that have fever. It also ensures that employees cannot join past meetings, meetings that have been approved or meetings that have reached full capacity. This helps to ensure that all the constraints with respect to joining a meeting that cannot be captured by the ER or schema, are checked thoroughly. In addition, doing them in a trigger instead of the function would mean that all the constraints and requirements that we mentioned above will be met and achieved even when the join_meeting() function is not called and someone decides to manually add himself to a meeting, which can happen occasionally.

- Trigger name: EmailTrigger
  Usage & justification: This trigger is called whenever a new employee is added. The objective of this trigger is to create a unique email for the newly added employee and update it under the newly added employee entry under the Employees table. The trigger is important as firstly it ensures that a unique and valid email is generated by taking in the newly added employee's id and name. The employee's id guarantees that the generated email is unique while the regex used guarantees that the generated email is valid. Secondly, it ensures that even if an employee is directly added to the Employees table through insertion (instead of calling the add_employee() function), a valid and unique email is generated. This is crucial as at times, users may decide to manually add an employee through insertion rather than calling the function.

5. Assumptions made for some functionalities & additional note

Functions	Assumptions/ additional note
add_department	- It is possible for a department to not have any employees belonging to it. We feel that this is not a serious issue.
add_room	- We assume for simplicity that a new room created today can only be used for meetings from the day tomorrow.
change_capacity	- The date passed in as the input is assumed to be today. - For a particular meeting room, there can only be 1 entry for that room for a particular date. As such, when this function is called, if there is already an entry for that particular room for that particular date, the existing entry will get overwritten with the new parameters provided in the function, if parameters pass the required constraints.
add_employee	- We assume that every employee who has not resigned can only belong to 1 department. - We have added triggers to ensure that there is overlap between the subgroups namely Junior, Senior & Manager. Similarly, we have added a trigger to ensure no overlap between Junior and Booker tables. - The schema does not ascertain that every employee inserted to the Employee table manually, belongs to 1 department. We check this by using a trigger.
remove_employee	- If an employee resigns, the future sessions which he booked in BookSessions will be removed and the future meetings after his resignation date in the Joins table will be removed as well. However, the future sessions that he only has approved (if he was a manager) are kept. - When an employee resigns, the department is set to NULL so that departments with only resigned employees can be deleted in the future (as per requirement). - The date passed in as the input is assumed to be today or in the past.
search_room	- The date passed in as the input is assumed to be today or future date. - The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour. - The capacity passed in as the input is assumed to be valid. For instance, the capacity cannot be 0 or negative.

book_room	- An employee can book more than 1 meeting on the same day and time as long as different rooms. This can be in the case of team bonding events where the organizer books multiple rooms all at the same date and time. - The date passed in as the input is assumed to be today or future date. - The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour. - An employee who has not declared temperature today cannot book a room and will be treated the same as someone who has fever today. - If a room is either completely or partially unavailable for the duration that a booker wants to book, the whole booking is rejected even for time slots when the room is partially available. For instance, if a booker wants to book room 1 between 12pm and 2pm and room 1 is unavailable between 1pm and 2pm (because of another meeting), the whole booking is rejected. If the booker now wants to book room 1 between 12pm and 1pm instead, he/she needs to call the function with start hour = 12 and end hour = 13. - A booker does not need to be in the same department as the room.
unbook_room	- The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour. - Booker cannot unbook any past approved meeting even if it was booked by him/her. - The meeting room parameters passed in as input are assumed to exist and valid. - If a session that a booker is attempting to unbook was not booked by him/her or does not exist, it gets ignored.
join_meeting	- An employee can join more than 1 meeting on the same day and time as they might stay for a while in each meeting room and move to another but we still need their record for contact tracing. - The date passed in as the input is assumed to be today or future date. - The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour. - An employee who has not declared temperature today cannot join a room and will be treated the same as someone who has fever today. - An employee joining the meeting can be from a different department than the room. - An exception is printed if an employee has resigned or has fever today or did not declare temperature today or intends to

	join a past meeting or has passed in invalid hours or his name is already inside the table for the meeting that he intends to join. For other issues (e.g meeting that employee intends to join has already been approved or has reached capacity or does not exist), that meeting(s) is/are ignored and only the other meeting(s) (if employee had passed in multiple hours) without issues is/are processed successfully.
leave_meeting	- If a booker of a meeting leaves a meeting, the booking will be deleted from BookSession through the "LeaveMeetingTrigger" trigger. So, intuitively, if the attendees number for a meeting drops to 0, the meeting will get deleted as well. - The date passed in as the input is assumed to be today or future date. - The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour. - If the meeting that an employee intends to leave has already been approved or does not exist, simply ignore it. - Note that the constraints to check (e.g whether a meeting has been approved or not) are done in the function instead of the trigger (unlike other functions) so that the contact tracing function can remove attendees from even approved meetings if the attendee has fever or is a close contact. The other methods were complex and we thought it could cause serious issues.
approve_meeting	- Manager that is approving the meeting can be having a fever. - Manager that is approving the meeting may not be in the same department as the booker of the meeting session. That is because we are taking the interpretation that the manager needs to be in the same department as the room. - Manager can approve or disapprove the meeting, depending on his/her opinions or what he/she is feeling at that point in time. - If a manager decides to approve a meeting that has already been approved by another manager, no overwrite should be done and this is enforced here. - If a manager decides to disapprove a meeting, he/she can only either disapprove a meeting which has not been approved or disapprove a meeting that he/she has previously approved, provided the other constraints are met. A manager cannot disapprove a meeting that has been approved by another manager (*the no sabotage rule*). - As per stated requirements, if a manager decides to disapprove a meeting, the meeting is immediately deleted and the attendees who have joined that particular meeting are immediately removed as well. - The date passed in as the input is assumed to be today or future date.

	- The hours provided as inputs are assumed to be valid. For instance, the start hour cannot be lesser than or equal to the end hour.
declare_health	- A resigned employee cannot declare temperature after his resignation date. - We assume that someone will not attempt to redeclare temperature more than once a day. We also assume the date passed will be today. - We will be using a view for the 'fever' derived variable.
contact_tracing	- This function has been tweaked to take in the date as well for simplicity. So, it takes in an employee ID and the date and checks whether the employee has fever on the provided date and does the rest that is required. - If an employee has not declared temperature on the provided date, we cannot ascertain whether the employee has fever or not. So, in that case, we simply raise a notice. - We assume that employees would usually declare their temperature at the start of the day, before they attend any meetings on that day. As such, we have defined close contacts to be all employees in the same *approved* meeting room from the past 3 days as the employee. (*i.e.,* from day D-3 [inclusive] to day D [exclusive]). - So, the current date is actually considered to be part of future meetings (again based on the assumption that employees would declare or be forced to declare their temperature at the start of the day before going for meetings). - As such, future meetings in the next 7 days of close contacts include the current date (today) and the next 7 days from today (*i.e.,* from day D [inclusive] to day D+7 [inclusive]). Close contacts are removed from these meetings and if these meetings were booked by close contacts, the whole meeting is removed.
non_compliance	- For resigned employees, if they have resigned before the provided query start date, they are not included in the final result. - Otherwise, if the employee has resigned, their ID will still show up on the query if they have any non-compliance within the provided query start date till the resigned date/query end date whichever comes earlier.

6. Analysis of normal forms of the relational database schema

Relation	Non-Trivial & Decomposed Functional Dependencies	BCNF	3NF
Employees (eid, ename, email, home_contact, mobile_contact, office_contact, resigned_date, did)	Key: {eid} Candidate key: {email} Prime Attributes: {eid, email} ● {eid → email} ● {eid → ename} ● {eid → home_contact} ● {eid → mobile_contact} ● {eid → resigned_date} ● {eid → did}	True	True All LHS are superkeys
HealthDeclaration (hdate, temp, eid)	Key: {hdate, eid} Prime Attributes: {hdate, eid} ● {hdate, eid → temp}	True	True ALL LHS are superkeys
MeetingRooms (room_num, floor_num, rname, did)	Key: {room_num, floor_num} Prime Attributes: {room_num, floor_num} ● {room_num, floor_num → rname} ● {room_num, floor_num → did}	True	True All LHS are superkeys
BookSessions (stime, date, room_num, floor_num, bid, mid)	Key: {stime, date, room_num, floor_num} Prime Attributes: {stime, date, room_num, floor_num} ● { stime, date, room_num, floor_num → bid} ● {stime, date, room_num, floor_num → mid}	True	True FD are either trivial or LHS are superkey
Updates(udate, room_num, floor_num, mid, new_cap)	Key: {udate, room_num, floor_num} ● {udate, room_num, floor_num → mid} ● {udate, room_num, floor_num → new_cap}	True	True All LHS are superkeys

The rest of the relations in the schema does not have non-trivial and decomposed functional dependencies. These relations are therefore automatically in 3NF.

7. Reflection

Summary of any difficulties encountered and lessons learned from the project.

- There were many ambiguities in some of the rules to allow or not allow the employees to book or join a meeting pertaining to the health declaration which we as a group have to come to a common consensus to decide on the approach to solve the ambiguity.
- We also thought a lot about what is the best way to update the meeting room capacity as at first we thought that we can change the capacity for a room for the future.
- We learnt that there are many uncertainties in the real world which need our assumptions to make the implementation of the database system more realistic.
- We have to take into consideration many factors and checks when we write a function or procedure as there are many different factors that can determine if an action is allowed (e.g. if an employee declared their health, if an employee resigned, if an employee has a fever). This is reflective of how we make decisions and carry out an action in real life.
- We have also learnt that common helper functions are useful for checking for the same conditions/constraints across different functionalities in order to be consistent with how we check for our constraints and for ease of update.