CS2102 Project Report
Team 65

Toh Yi Zhi	A0222554L
Vikas Harlani	A0214360U
Wang Zihan	A0214345M
Zhuang Jianning	A0214561M

Project Responsibilities

Shared
- ER Diagram
- Normal Form Analysis

Yi Zhi
- view_booking_report() in Admin Functionalities
- view_future_meeting() in Admin Functionalities
- view_manager_report() in Admin Functionalities
- Non-Trivial Design Decision (ER Diagram)

Vikas
- join_meeting(), leave_meeting(), approve_meeting() in Core Functionalities
- Triggers related to Basic and Core Functionalities
- Test cases for Basic and Core Functionalities
- Reflection (Report)

Zihan
- ContactTracingLog, Joins Schema Refinement
- Health Functionalities
- non_compliance() in Admin Functionalities
- Triggers related to Health Functionalities
- Test cases for Health Functionalities and non_compliance()
- Non-Trivial Schema Designs (Report)

Jianning
- Schema Design
- Basic Functionalities
- search_room(), book_room(), unbook_room() in Core Functionalities
- Triggers related to Basic and Core Functionalities
- Test cases for Basic and Core Functionalities
- Interesting Triggers (Report)

Constraints

1. Each employee is assigned a unique employee ID (eid). [ER/Schema]
2. Each employee is assigned a unique email address (email). [Schema]
3. Each employee records the following information: name (name), contact numbers (contact) and resignation date (resigned date).[ER/Schema]
4. Each department is identified by their unique department ID (did). [ER/Schema]
5. Each department records the following information: department name (dname). [ER/Schema]
6. Each meeting rooms can be uniquely identified by their floor number (floor) and their room number (room). [ER/Schema]
7. Each meeting room records the following information: room name (rname). [ER/Schema]
8. An employee must belong to exactly one department. [ER/Schema]
9. A department may have zero or more employee. [ER/Schema]
10. A meeting room must be located in exactly one department. [ER/Schema]
11. A department may have zero or more meeting room. [ER/Schema]
12. Each employee must be one and only one of the three kinds of employees: junior, senior or Manager. [ER/Trigger]
13. A junior employee cannot book any meeting room. [ER/Trigger]
14. A senior or a manager can book meeting rooms.[ER/Trigger]
15. A meeting room can only be booked by one group for the given date and time.[ER/Schema]
16. If an employee is having a fever, they cannot book a room. [Trigger]
17. Any employee can join a booked meeting.[ER/Schema FK Join to Session]
18. The employee booking the room immediately joins the booked meeting. [Trigger]
19. If an employee is having a fever, they cannot join a booked meeting. [Trigger]
20. A manager approves a booked meeting [ER/Schema]
21. A manager can only approve a booked meeting in the same department as the manager (i.e., the manager and the meeting room are in the same department). [Trigger]
22. A booked meeting is approved at most once.[ER/Trigger]
23. Once approved, there should be no more changes in the participants and the participants will definitely come to the meeting on the stipulated day. [Trigger]
24. A manager from the same department as the meeting room may change the meeting room Capacity. [Trigger]
25. A booking can only be made for future meetings. [Trigger]
26. An employee can only join future meetings. [Trigger]
27. An approval can only be made on future meetings. [Trigger]
28. Every employee must do a daily health declaration. [Check within function]
29. A health declaration records temperature and the date. [ER/Schema]
30. A health declaration for a given employee can be uniquely identified by the date. [ER/Schema]
31. If the declared temperature is higher than 37.5 Celsius, the employee is having a fever. [Schema/Function]
32. The declared temperature can only be between 34 (hypothermia) to 43 Celsius (hyperthermia). [Schema]
33. When an employee resign, all past records are kept. [Trigger]

34. When an employee resign, they are no longer allowed to book or approve any meetings. [Trigger]
35. Contact tracing constraints.[Trigger]

Non-Captured Constraints

The following constraints are not captured: 2 (can be captured in schema), 16, 18 (partially captured by total participation constraints from Employees to Joins which ensures that at least someone joins immediately but we cannot be sure that it is indeed the employee who books the meeting room), 19, 23,25, 26, 27, 32, 34 as well as any constraints related to contact tracing as summarised in 35. Constraints 25, 26, 27 and 32 are common sense constraints.

ER Data Model

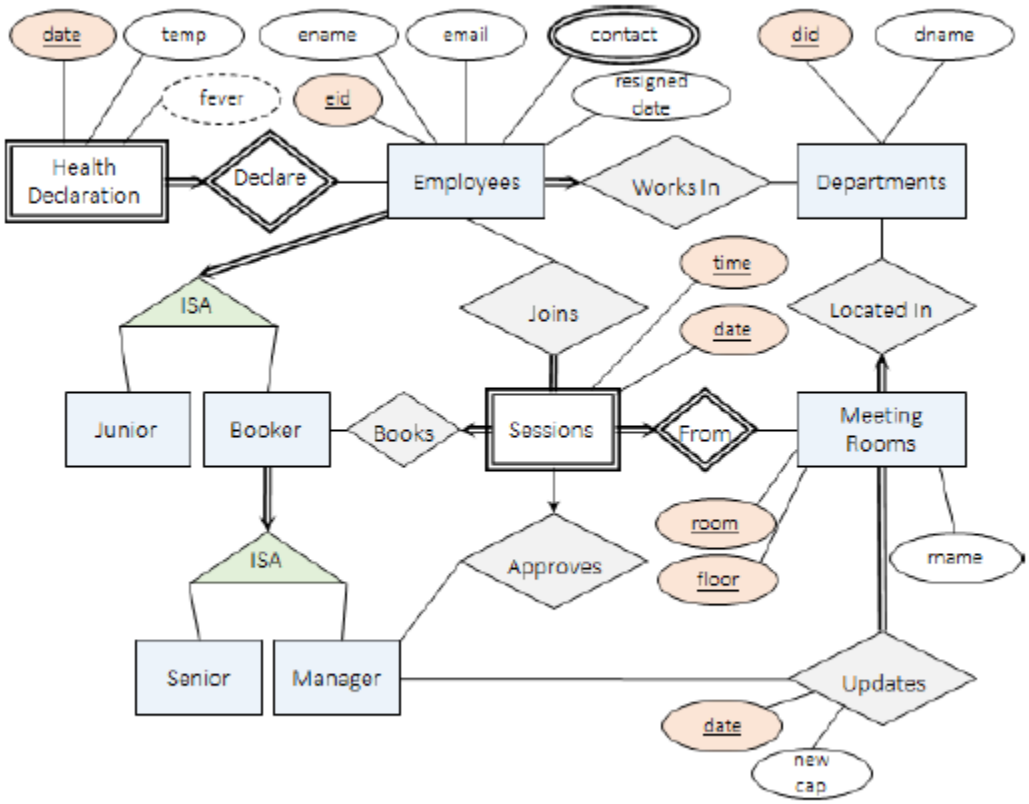

Figure 1: The proposed ER diagram.

Non Trivial Design Decisions

1.Department
There is a many to one relationship between Employees and Departments with the assumption that each employee must belong to only 1 department. Each department can have any number of employees (can also be empty).

2.Health Declaration
There is a many to one relationship between Health Declarations and Employees with the assumption that each health declaration is submitted by only 1 employee. Each employee can submit up to any number of health declarations over a time period (but at most one a day).

Moreover, based on the assumption that health declaration only exist when there is employees (impossible for the presence of health declaration without employees), there is a weak entity relationship between health declaration and employees

3.Meeting Room and Department
There is a many to one relationship between Meeting Room and Department with the assumption that all rooms are being allocated to all Departments and each meeting room belongs to only one department.

4.Sessions
There is a weak entity between Sessions and Meeting rooms based on the assumption that meeting rooms must exist before sessions can exist. In each session, only one meeting room can be booked.

5.All employees are either Juniors or Meeting Bookers (covering and non-overlap). Only Meeting Bookers can book meeting rooms. All Meeting Bookers are either Seniors or Managers (covering and non-overlap). Only Managers can approve a booking which has to be approved.

6.Manager
Each Session can also be approved by one manager, hence there is a many to one relationship between the Sessions and Managers.

Constraints not Captured
2. Each employee is assigned a unique email address (email). [Schema]
16. If an employee is having a fever, they cannot book a room. [Trigger]
18. The employee booking the room immediately joins the booked meeting. [Trigger]
19. If an employee is having a fever, they cannot join a booked meeting. [Trigger]
21. A manager can only approve a booked meeting in the same department as the manager (i.e., the manager and the meeting room are in the same department). [Trigger]
22. A booked meeting is approved at most once.[??/Trigger]
24. A manager from the same department as the meeting room may change the meeting room Capacity. [Trigger]
25. A booking can only be made for future meetings. [Trigger]
26. An employee can only join future meetings. [Trigger]
27. An approval can only be made on future meetings. [Trigger]
28. Every employee must do a daily health declaration. [Check within function]
31. If the declared temperature is higher than 37.5 Celsius, the employee is having a fever. [Schema/Function->Trigger?]
32. The declared temperature can only be between 34 (hypothermia) to 43 Celsius (hyperthermia). [Schema]
33. When an employee resigns, all past records are kept. [Trigger]
34. When an employee resigns, they are no longer allowed to book or approve any meetings. [Trigger]
35. Contact tracing constraints are omitted from this list.[Trigger]

Relational Database Schema

Non Trivial Designs

1. Health Declaration Temperature Checks
   In the HealthDeclarations table, there are two checks:

   a. CHECK (temp BETWEEN 34 AND 43)
      This checks for Constraint 32, ensuring the declared temperature can only be between 34 and 43 degrees Celcius
   b. CHECK (temp <= 37.5 OR fever = TRUE)
      This checks for Constraint 31, ensuring that if the declared temperature is higher than 37.5 Celcius, the employee is recorded to have a fever.

2. Updates and Meeting Room and capacity > 0
   In the Updates table there is a constraint non_negative capacity
   CONSTRAINT non_negative_capacity CHECK (new_capacity > 0)

   This fulfills the total participation of at least one employee in sessions, hence the capacity always has to be more than 0.

3. Booking time as integers between 0 to 23 to simulate 1 hour intervals
   In the Sessions and Joins tables, there is a check:
   CHECK (booking_time BETWEEN 0 and 23)

   Working with the assumption that each session lasts in one hour intervals beginning at the start of each hour, this ensures there are only 24 booking slots each day, and booking times cannot go beyond the 24 hours in a day.

4. Splitting Employee Contacts and Employees Tables
   Instead of storing the multivalued attribute contact directly in one Employees table, the contact_number and corresponding eid of the employee is stored in a separate EmployeeContacts table, with a Foreign Key eid referencing the Employees table.

   This will avoid storing multiple distinct attributes (ie contact1, contact2, contact3) in the Employees table, guarding against normal form violations.

5. Contact Tracing Log
   Although not present in the ER diagram, the ContactTracingLog table is useful for the implementation of the contract_tracing() function and remove_close_contacts() trigger functions. As close contacts are identified, they will be inserted into the ContactTracingLog, where an INSERT ON ContactTracingLog trigger will then remove them from future meetings.

6. Joins table includes On DELETE CASCADE Sessions
   A join for a session cannot exist if that session does not exist. Therefore, Joins REFERENCES Sessions with an ON DELETE CASCADE, which will delete the join if the session is deleted.

7. Sessions table includes On DELETE CASCADE Meeting Rooms
   Each session forms a weak entity set with Meeting Rooms, where a session in that meeting room cannot exist if that meeting room does not exist. Therefore, Sessions REFERENCES MeetingRooms with an ON DELETE CASCADE, which will delete the session if the meeting room is deleted.

Non-Captured Constraints

12. Each employee must be one and only one of the three kinds of employees: junior, senior or Manager. [ER/Trigger]
13. A junior employee cannot book any meeting room. [ER/Trigger]
14. A senior or a manager can book meeting rooms.[ER/Trigger]
16. If an employee is having a fever, they cannot book a room. [Trigger]
18. The employee booking the room immediately joins the booked meeting. [Trigger]
19. If an employee is having a fever, they cannot join a booked meeting. [Trigger]
21. A manager can only approve a booked meeting in the same department as the manager (i.e., the manager and the meeting room are in the same department). [Trigger]
23. Once approved, there should be no more changes in the participants and the participants will definitely come to the meeting on the stipulated day. [Trigger]
24. A manager from the same department as the meeting room may change the meeting room Capacity. [Trigger]
25. A booking can only be made for future meetings. [Trigger]
26. An employee can only join future meetings. [Trigger]
27. An approval can only be made on future meetings. [Trigger]
33. When an employee resigns, all past records are kept. [Trigger]
34. When an employee resigns, they are no longer allowed to book or approve any meetings. [Trigger]
35. Contact tracing constraints.[Trigger]

Interesting Triggers

1. employee_resigned

```
CREATE TRIGGER employee_resigned
AFTER UPDATE ON Employees
FOR EACH ROW WHEN (NEW.resigned_date IS NOT NULL)
EXECUTE FUNCTION remove_future_records();
```

When an employee resigns, we still want to keep all the past records regarding this employee for contact tracing purposes. However, they are no longer allowed to book, join or approve any meetings. Hence, any future records related to those actions are removed.

Since we still want to keep the information of a resigned employee, the schema implementation for Employees has a resigned_date attribute initially set to NULL. When an employee resigns, this attribute is simply set to their resignation date which has to be CURRENT_DATE or earlier so the employee tuple will not have to be deleted. Hence, employee_resigned triggers AFTER UPDATE ON Employees with the condition that NEW.resigned date becomes NOT NULL.

Within the remove_future_records() function, there are 3 cases to check. Firstly, if the employee is the booker of a session of a future meeting, the session, along with all its participants, are removed. This is done by deleting any tuples from Sessions WHERE (booker_eid = NEW.eid AND booking_date > CURRENT_DATE). When the session is removed, all other employees who joined the session are automatically removed due to ON DELETE CASCADE from Sessions to Joins. Secondly, if the employee is a participant of any future meetings, they are removed even if the meeting is approved. This is done by deleting any tuples from Joins WHERE (employee_eid = NEW.eid AND booking_date > CURRENT_DATE). Lastly, if the employee is a manager who has approved meeting sessions, those sessions would revert back to the state where it is not approved. This is done by updating Sessions and setting the manager_eid back to NULL WHERE manager_eid = NEW.eid AND booking_date > CURRENT_DATE.

Addresses non-captured constraints 33 and 34

2. capacity_overflow_after_reduction

```
CREATE TRIGGER capacity_overflow_after_reduction
AFTER UPDATE ON Updates
FOR EACH ROW WHEN (NEW.new_capacity < OLD.new_capacity)
EXECUTE FUNCTION remove_overflow_sessions();
```

When a meeting room has its capacity changed, any room booking after the change date with more participants (including the employee who made the booking) will automatically be removed. This is regardless of whether they are approved or not.

Since we already have a trigger that checks that meeting capacity is not exceeded BEFORE INSERT ON Joins, the only case when capacity of a meeting would be exceeded is when the new capacity is less than the old capacity after we update the capacity. Hence we only need to check WHEN (NEW.new_capacity < OLD.new_capacity).

Within the remove_overflow_sessions() function, we need to check for all the meeting sessions in the future affected by the capacity change. Hence, we need a cursor to iterate through each tuple from Joins WHERE floor_num = NEW.floor_num AND room_num = NEW.room_num AND booking_date > NEW.update_date. To get the number of participants in each session, we can simply GROUP BY (booking_date, booking_time) which uniquely identifies each session given the floor_num and room_num of the affected meeting room. We then SELECT COUNT(*) to count the number of employees who have joined each unique session.

In the LOOP, we fetch the cursor into a record and check if the number of participants in that session is greater than the new capacity. We delete the session if it is. When the session is removed, all other employees who joined the session are automatically removed due to ON DELETE CASCADE from Sessions to Joins.

Addresses meeting room dynamics requirements.

3. close_contacts_remove

```
CREATE TRIGGER close_contacts_remove
AFTER INSERT ON ContactTracingLog
FOR EACH ROW EXECUTE FUNCTION remove_close_contacts();
```

Once an employee is recorded to have a fever, that employee as well as close contacts of that employee will have to be removed from future meetings from the day of the fever D (all future for the employee with fever and D+7 for close contacts).

Employee with fever is added to ContactTracingLog with contact_type ‘primary’ while close contacts of that employee are added with contact_type ‘contact’. There are no further checks on each row as INSERT on the ContactTracingLog table will only happen after it fulfills the requirements of a primary or close contact, which is checked in the contact_tracing() function. Hence, There is no need to block the INSERT, so the remove_close_contacts() function is called AFTER there is an INSERT on the ContactTracingLog table.

Inside remove_close_contacts(), the new rows’ contact_type will be checked. If it is a ‘primary’, then that employee is removed from all future sessions, and all future bookings done by that employee will also be removed regardless of meeting approval status. If it is a ‘contact’, then the employee is only removed from future joined meetings between D and D+7 regardless of meeting approval status. This bypasses the disallow_update_when_approved() trigger function (which enforces the constraint of no updates on a meeting after it is approved) where there are clauses to check if this deletion on the Joins table is due to contact_tracing().

Addresses contact tracing requirements and constraints (Constraint 35).

4. Update_participants_after_approval

```
CREATE TRIGGER update_participants_after_approval
BEFORE INSERT OR DELETE ON Joins
FOR EACH ROW EXECUTE FUNCTION disallow_update_when_approved();
```

Once a booking is approved, there should be no more changes in the participants and they will definitely come to the meeting room on the stipulated day.

However, there were some contradictory requirements whereby participants can be forcefully removed even after the session has been approved. For example, when a capacity change results in an exceed in capacity, the session and all its participants are still removed regardless of whether they are approved or not (Interesting Trigger 2). Contact tracing will also remove participants if they have fever or are close contacts regardless of meeting approval (Interesting Trigger 3) Also, an employee will be removed from all sessions after their resignation date even if the meeting is approved (Interesting Trigger 1).

Hence, we had to do further checking of these special cases within the disallow_update_when_approved() function. For INSERT ON Joins, there are no special cases to consider so we only need to check if any session an employee is joining has manager_eid IS NOT NULL. We can RETURN NULL if the session has been approved. For DELETE ON Joins, we have to first check if the employee_eid has resigned_date IS NOT NULL. We allow employees who have resigned to leave approved meetings. For cases of contact tracing removal, there will be checks if the employee is in the ContactTracingLog, and if the booking_date is within the time since the contact_date where contacts need to be removed (all future meetings for ‘primary’ contacts and D to D+7 meetings for ‘contact’). We can then check if the session is already approved and we disallow employees leaving the meeting in such cases. For cases where the session is removed for example due to an exceed in capacity, the session would be removed first before deletion on Joins. Hence, we can just allow deletion from Joins whenever the session no longer exists.

Addresses Non-captured constraints 23

5. add_booker_as_participant

```
CREATE TRIGGER add_booker_as_participant
AFTER INSERT ON Sessions
FOR EACH ROW EXECUTE FUNCTION add_booker_to_session();
```

When a senior employee or a manager books a room, they are also counted as a participant if they do not have a fever.

Since a successful insertion into Sessions already checks for valid date, hour, eid, resignation status and health declaration through the IF checks within our book_room() function OR through various triggers for BEFORE INSERT ON Sessions, it is safe to assert that the booker also meets the requirements to be added to the meeting.

Hence, within add_booker_to_session(), we simply INSERT INTO Joins VALUES (NEW.booker_eid, NEW.floor_num, NEW.room_num, NEW.booking_date, NEW.booking_time). Since we also asserted that capacity > 0 in our schema, we will always be able to add the booker as the first participant without exceeding the capacity.

Having this trigger allows us to automatically add the booker of the meeting as the first participant.

Addresses Non-captured constraints 18

6. approve_meeting_same_did_as_manager

```
CREATE TRIGGER approve_meeting_same_did_as_manager
BEFORE UPDATE ON Sessions
FOR EACH ROW WHEN (NEW.manager_eid IS NOT NULL)
EXECUTE FUNCTION disallow_approve_for_diff_did();
```

Part of the managerial duty is that every booking must be approved by a manager from the same department.

Since approval of a meeting is done by updating the manager_eid of a Session tuple, we have to check if the manager approving the meeting is from the same department BEFORE UPDATE ON Sessions. We additionally include the constraint that NEW.manager_eid IS NOT NULL since employees resigning might revert sessions back to the state where it is not approved (manager_eid becomes NULL again).

Within the disallow_approve_for_diff_did() function, we can simply determine the manager_did and room_did and check if they are the same. We do not allow the approval otherwise.

Addresses Non-captured constraints 21

Normal Forms Analysis

Departments(did, dname)

No violation, table is in 3NF.
There are only 2 attributes in this table.

Employees(eid, ename, email, resigned_date, did)

PRIMARY KEY: {eid}
Known FDs: {eid} -> {ename, email, resigned_date, did}, {email} -> {eid, ename, resigned_date, did}
Prime Attributes: {eid, email}

Let *Employees(eid, ename, email, resigned_date, did)* be represented by *R(A, B, C, D, E)* where *A, B, C, D, E* represent *eid, ename, email, resigned_date, did* respectively.

The following closures can be derived:

{A}+ = {A, B, C, D, E}
{B}+ = {B}
{C}+ = {A, B, C, D, E}
{D}+ = {D}
{E}+ = {E}
{A,B}+ = {A,B, C, D, E}
{A,C}+ = {A, B, C, D, E}
{A,D}+ = {A,B, C, D, E}
{A,E}+ = {A, B, C, D, E}
{B,C}+ = {A, B,C, D, E}
{B,D}+ = {B,D}

{B,E}+ = {B,E}
{C,D}+ = {A, B, C,D, E}
{C,E}+ = {A, B, C, D, E}
{D,E}+ = {D,E}
{A,B,C}+ = {A,B,C,D,E}
{A,B,D}+ = {A,B, C, D, E}
{A,B,E}+ = {A,B, C, D, E}
{A,C,D}+ = {A,B, C, D, E}
{A,C,E}+ = {A, B, C, D, E}
{A,D,E}+ = {A, B, C, D,E}
{B,C,D}+ = {A, B,C,D, E}

{B,C,E}+ = {A, B, C, D, E}
{B,D,E}+ = {B,D,E}
{C,D,E}+ = {A, B, C,D,E}
{A,B,C,D}+ = {A,B,C,D,E}
{A,B,C,E}+ = {A,B,C,D,E}
{A,B,D,E}+ = {A,B, C, D,E}
{A,C,D,E}+ = {A, B, C,D,E}
{B,C,D,E}+ = {A, B,C,D,E}
{A,B,C,D,E}+ = {A,B,C,D,E}

Any closures that contain *A or C* on the left hand side is a superkey. It can also be seen that any closures without *A or C* on the left hand side are trivial.

Thus there are no non-trivial and decomposed FDs where the LHS is not a superkey. Hence, the Employees table is in 3NF.

EmployeeContacts(eid, contact_number)

No violation, table is in 3NF.
There are only 2 attributes in this table.

HealthDeclarations(eid, declare_date, temp, fever)

PRIMARY KEY: {eid,declare_date}
Known FDs: {eid,declare_date} -> {temp, fever}, {temp} -> {fever}
Prime Attributes: {eid,declare_date}

Let *HealthDeclarations(eid, declare_date, temp, fever)* be represented by *R(A, B, C, D)* where *A, B, C, D* represent *eid, declare_date, temp, fever* respectively.

The following closures can be derived:

{A}+ = {A}	{A,C}+ = {A,C,D}	{A,B,C}+ = {A,B,C,D}
{B}+ = {B}	{A,D}+ = {A,D}	{A,B,D}+ = {A,B,C,D}
{C}+ = {C,D}	{B,C}+ = {B,C,D}	{A,C,D}+ = {A,C,D}
{D}+ = {D}	{B,D}+ = {B,D}	{B,C,D}+ = {B,C,D}
{A,B}+ = {A,B,C,D}	{C,D}+ = {C,D}	{A,B,C,D}+ = {A,B,C,D}

Non-trivial and decomposed FDs on R: {A,B} -> {A,B,C,D}, {C} -> {D}

There is a violation of BCNF where $\{C\}^+ = \{C,D\}$

BCNF decomposition:
$\{temp\}^+$ = {temp, fever}, hence {temp} -> {fever} violates BCNF
Decompose HealthDeclarations to R1(temp, fever) and R2(eid, declare_date, temp)

R1(temp, fever) is in BCNF
R2(eid, declare_date, temp) is in BCNF

F1: {temp} -> {fever}
F2: {eid, declare_date} -> {temp}

$\{eid,declare\_date\}^+$ = {eid,declare_date, temp, fever} using F1 and F2 (transitivity)
{temp} -> {fever} derived directly from F1

R1(temp, fever) R2(eid, declare_date, temp) is a dependency-preserving BCNF decomposition of HealthDeclarations.

Juniors/Bookers/Seniors/Managers(eid)

No violation, table is in 3NF.
There is only 1 attribute in the table.

MeetingRooms(floor_num, room_num, rname, did)

PRIMARY KEY: {floor_num, room_num}
Known FDs: {floor_num, room_num} -> {rname, did}
Prime Attributes: {floor_num, room_num}

Let *MeetingRooms(floor_num, room_num, rname,did)* be represented by *R(A, B, C, D)* where *A, B, C, D* represent *floor_num, room_num, rname, did* respectively.

The following closures can be derived:

{A}+ = {A}	{A,C}+ = {A,C}	{A,B,C}+ = {A,B,C,D}
{B}+ = {B}	{A,D}+ = {A,D}	{A,B,D}+ = {A,B,C,D}
{C}+ = {C}	{B,C}+ = {B,C}	{A,C,D}+ = {A,C,D}
{D}+ = {D}	{B,D}+ = {B,D}	{B,C,D}+ = {B,C,D}
{A,B}+ = {A,B,C,D}	{C,D}+ = {C,D}	{A,B,C,D}+ = {A,B,C,D}

Non-trivial and decomposed FDs on R: {A,B} → {C,D}

Any closures that contain *A, B* on the left hand side is a superkey. Any closures without *A or B* on the left hand side are trivial.

Thus there are no non-trivial and decomposed FDs where the LHS is not a superkey. Hence, the MeetingRooms table is in 3NF.

Sessions(floor_num, room_num, booking_date, booking_time, booker_eid, manager_eid)

PRIMARY KEY: {floor_num, room_num, booking_date, booking_time}
Known FDs: {floor_num, room_num, booking_date, booking_time} → {booker_eid}, {floor_num, room_num, booking_date, booking_time} → {manager_eid}
Prime Attributes: {floor_num, room_num, booking_date, booking_time}

Let *Sessions(floor_num, room_num, booking_date, booking_time, booker_eid, manager_eid)* be represented by *R(A, B, C, D, E, F)* where *A, B, C, D, E, F* represent *floor_num, room_num, booking_date, booking_time, booker_eid, manager_eid* respectively.

*Listing out all closures for 6 attributes is excessive

Non-trivial and decomposed FDs on R:

{A,B,C,D} → {E} {A,B,C,D} → {F}

Any closures that contain *A*, *B*, *C, D* on the left hand side is a superkey. Any closures without *A, B, C or D* on the left hand side are trivial.

Thus there are no non-trivial and decomposed FDs where the LHS is not a superkey. Hence, the Sessions table is in 3NF.

Joins(employee_eid, floor_num, room_num, booking_date, booking_time)

No violation, table is in 3NF.
Since all the attributes are prime attributes present in the primary key, the right hand side of all FDs will always contain prime attributes.

Updates(manager_eid, floor_num, room_num, update_date, new_capacity)

PRIMARY KEY: {manager_eid, floor_num, room_num, update_date}
Known FDs: {manager_eid, floor_num, room_num, update_date} -> {new_capacity}
Prime Attributes: {manager_eid, floor_num, room_num, update_date}

Let *Updates(manager_eid, floor_num, room_num, update_date, new_capacity)* be represented by *R(A, B, C, D, E)* where *A, B, C, D, E* represent *manager_eid, floor_num, room_num, update_date, new_capacity* respectively.

The following closures can be derived:

{A}+ = {A}
{B}+ = {B}
{C}+ = {C}
{D}+ = {D}
{E}+ = {E}
{A,B}+ = {A,B}
{A,C}+ = {A,C}
{A,D}+ = {A,D}
{A,E}+ = {A,E}
{B,C}+ = {B,C}
{B,D}+ = {B,D}

{B,E}+ = {B,E}
{C,D}+ = {C,D}
{C,E}+ = {C,E}
{D,E}+ = {D,E}
{A,B,C}+ = {A,B,C}
{A,B,D}+ = {A,B,D}
{A,B,E}+ = {A,B,E}
{A,C,D}+ = {A,C,D}
{A,C,E}+ = {A,C,E}
{A,D,E}+ = {A,D,E}
{B,C,D}+ = {B,C,D}

{B,C,E}+ = {B,C,E}
{B,D,E}+ = {B,D,E}
{C,D,E}+ = {C,D,E}
{A,B,C,D}+ = {A,B,C,D,E}
{A,B,C,E}+ = {A,B,C,E}
{A,B,D,E}+ = {A,B,D,E}
{A,C,D,E}+ = {A,C,D,E}
{B,C,D,E}+ = {B,C,D,E}
{A,B,C,D,E}+ ={A,B,C,D,E}

Non-trivial and decomposed FDs on R: {A,B,C,D} → {E}

Any closures that contain *A*, *B*, *C, D* on the left hand side is a superkey. Any closures without *A*, *B*, *C or D* on the left hand side are trivial.

Thus there are no non-trivial and decomposed FDs where the LHS is not a superkey. Hence, the Updates table is in 3NF.

ContactTracingLog(employee_eid, contact_date, contact_type, log_time)

No violation, table is in 3NF.
Since all attributes are prime attributes present in the primary key {employee_eid, contact_date, contact_type, log_time}, the right hand side of all FDs will always contain prime attributes.

Reflection

One of the difficulties we encountered during this project was combining and compiling our individual codes together. In order to complete the project within a relatively short time frame, we had split the functionalities between members of our group. However, while compiling, sometimes one member's change would cause another member's code to break. For example, if one person had to change the contents of a particular table in the schema, another person's code that inserted values into that table resulted in an error. Another instance was when one person's new trigger caused another trigger to not function as intended. Reflecting on this, in the future, we will keep in mind how our changes will affect the rest of the codebase, and try our best to limit any breaks that occur.

One specific difficulty we encountered was while coding the function disallow_update_when_approved() as part of the update_participants_after_approval trigger. This trigger triggered before insert or delete on Joins and the function was supposed to check if the corresponding session had already been approved. If already approved, the function would raise an exception, denying the update. After coding out the trigger and function and then running the code on our test cases, we noticed that there were certain cases where the expected output wasn't given. This was because there are special cases where the session is already approved and the participants are required to be updated. These cases include when the employee leaves a meeting due to resigning or due to being a close contact. Another case was when the session had to be removed due to capacity overload. Once we added checks for these special cases, the trigger and function worked as intended and our test cases gave the expected output.

One of the lessons we learned was to test each functionality for **all** the edge cases before moving on to the next. For some of the functionalities, we would write the code and then test to see a few working cases and the more obvious error cases. However, sometimes, we forgot to check for the more obscure edge cases. For example, for the change_capacity function, we added a check to ensure the employee updating the capacity is a manager, but forgot to ensure that this manager is from the same department as the room. We found this out later, during the testing phase, and then had to update the function to add this additional check. Although we managed to catch this miss during the testing phase, we feel that we could've easily missed it and we learned we should thoroughly check all the edge cases once during the coding phase.

Another lesson learned was to include notices and exceptions in our code. In the beginning, we were wondering if we needed to include notices and exceptions in our code, as it wasn't required and seemed like additional work. However, we decided to

include them, and we are glad we did. Even though it isn't part of the requirements, we felt that it really helped to speed up our testing. Rather than having to follow the terminal output and compare it with the data.sql file line by line, the notices and exceptions allowed us to quickly verify the negative cases. Moving on, we feel it is always a good idea to include notices and exceptions, if not for the grader, for our own reference.

In addition, we also learned the power of plpgsql. Without the various control structures such as IF and LOOPs, it would've been impossible to carry out the required functionalities of the project. While adding IF blocks and various checks to our functions, we also realised that with plpgsql, it removed the need for triggers. Of course, this is if only function calls are used and no manual inserts/deletes of data.

Overall, we feel that this project was a great opportunity to practice our sql skills, especially triggers and plpgsql. We also feel that the project helped us review concepts and allowed us to learn new ones.