CS2102
ER Data Model Submission
Team 14

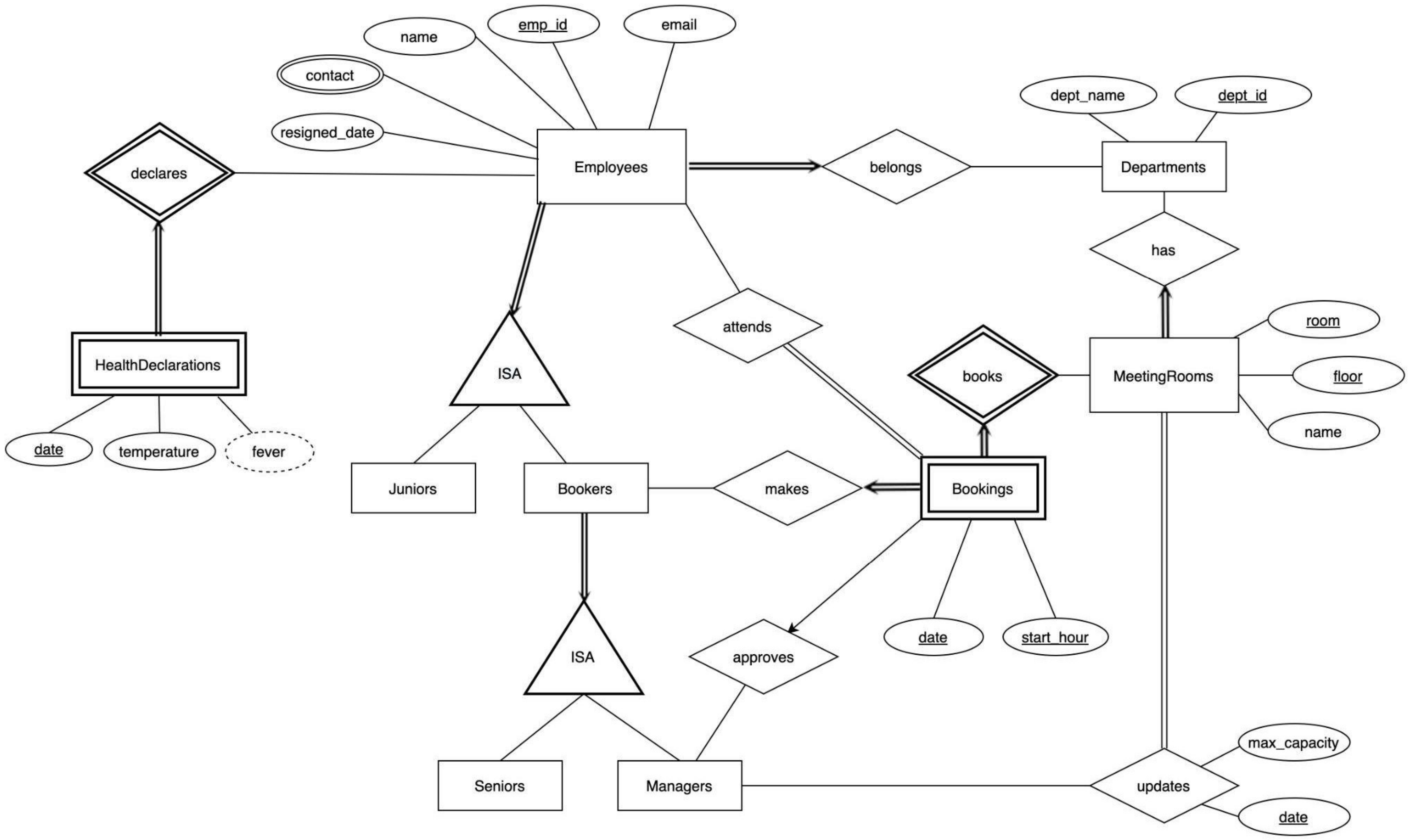
Tan Rui Yang (A0219814B)
Alvin Tan Guo Hao (A0217220X)
Tan Hui En (A0221841N)
Tan Le Yi (A0225644E)

1. Project Responsibilities

Team member	Project responsibilities
Tan Rui Yang	- Came up with ER Model and the constraints not captured by the model. - Designed relational database schema. - Functions/Procedures - add_department - add_employee - unbook_room - declare_health - view_future_meeting - Triggers & Trigger Functions (not shown) - auto_generate_emp_id_and_email - a_prevent_direct_insert_into_employees - employee_kind_specified - check_update_on_employees - add_or_change_junior - delete_junior - add_or_change_bookers - add_or_change_manager - delete_manager - add_or_change_senior - delete_senior - fever_checking - emp_resigned_cannot_declare_health - cannot_unbook_booking_in_the_past - Normal form analysis and report writing
Alvin Tan Guo Hao	- Came up with ER Model and the constraints not captured by the model. - Designed and wrote the relational database schema. - Functions/Procedures - remove_department - remove_employee - join_meeting - contact_tracing - niew_manager_report - Triggers & Trigger Functions (not shown) - remove_future_meetings - check_resign_date - prevent_delete_on_employees - prevent_attends_during_fever - prevent_update_if_approved - prevent_insert_and_update_if_conflict - prevent_insert_and_update_if_resigned - prevent_exceed_of_max_capacity - perform_contact_tracing - Normal form analysis and report writing
Tan Hui En	- Came up with ER Model and the constraints not captured by the model.

	- Designed relational database schema. - Functions/Procedures - add_room - search_room - leave_meeting - non_compliance - Triggers & Trigger Functions (not shown) - check_leave - check_if_room_exists_in_updates - Normal form analysis and report writing
Tan Le Yi	- Came up with ER Model and the constraints not captured by the model. - Created ER Diagram - Designed relational database schema. - Functions/Procedures - change_capacity - book_room - approve_meeting - View_booking_report - Triggers & Trigger Functions (not shown) - prevent_approval_by_incorrect_manager - a_ensure_manager_changing_capacity_not_resigned - prevent_capacity_change_by_incorrect_manager - remove_future_bookings_exceeding_capacity - prevent_updates_table_delete - add_booker_to_attends - c_ensure_booker_not_fever - a_ensure_booker_not_resigned - b_ensure_booking_date_is_not_in_past - a_ensure_approver_not_resigned - b_ensure_meeting_not_already_approved - Normal form analysis and report writing

2. ER Data Model



Note 1: Chosen interpretation of "*A manager from the same department approves the booking.*" from step 3 of booking procedure:
(1) The manager must be in the same department as the **room**.

Note 2: For contact tracing, we defined "close contacts" as the participants present when the sick employee is present in parts of the approved booking.
E.g: Employee 1 was in the approved meeting from 12pm - 2pm, Employee 2 was in the same meeting from 12pm - 1pm, Employee 3 was in the same meeting from 1pm - 2pm. If Employee 2 has a fever, the close contact is only Employee 1.

2.1 Justifications for non-trivial design decisions made

1. Each meeting room must belong to a certain department so that we will know who can approve the booking (the manager in the same department as the room).

2. Model *Employees* as a superclass and *Juniors, Bookers* as a subclass of *Employees*, followed by *Senior* and *Manager* as subclasses of *Bookers* so that we can show the following constraints:
    i. Only *Seniors* and *Managers*, who are collectively called *Bookers*, can make bookings
    ii. Only *Managers* can approve bookings

3. Made *Bookings* a weak entity which uses date, start_hour, floor and room as a primary key so that we can implement the booking related functions easily since these are the inputs to the routines.

4. Did not enforce total participation constraint between Employees and Health since it is possible for employees to not declare their health at least once, as implied by the "non_compliance" routine.

5. Employees can have more than one contact number to accommodate multiple contact numbers such as home and office numbers.

6. *MeetingRooms* has no *capacity* attribute. The update of the maximum capacity is done in another entity, *Updates,* which allows us to handle capacity changes for future dates.

2.2 Constraints not captured

1. Does not enforce the maximum number of employees in a meeting. For example, a meeting could have 10 participants even if the room's capacity is only 5.

2. Does not enforce that employees having fever cannot attend meetings.

3. Does not enforce that participants of the meeting cannot be changed once the booking is approved.

4. Does not enforce that the manager that approves the booking must be from the same department.

5. Does not enforce the case when an employee resigns, future records of meetings are removed, and the employee is no longer available to create bookings.

3. Relational Database Schema

3.1 Justifications for non-trivial design decisions made

1. If the meeting room is referencing a particular department id, the department cannot be removed until all the meeting rooms under that department are transferred to another department. This is to prevent the case where a meeting room does not belong to any department after the department it originally belonged to has been deleted.

2. We used *SERIAL* for *emp_id* in the *Employees* table to automatically generate a unique employee id every time a record is inserted.

3. We used a *CHECK CONSTRAINT* for *temperature* in the *HealthDeclarations* table to ensure that the value must be between 34 and 43 degrees Celsius inclusive.

4. We used a *CHECK CONSTRAINT* for *fever* in the *HealthDeclarations* table to ensure that there will not be a case where *fever* = *FALSE* and *temperature* > 37.5 degrees Celsius in a single record.
    - Even though *fever* is a derived attribute and if our procedures are implemented correctly, this condition may not be satisfied, we chose to implement this constraint in the schema itself as an additional layer of insurance for data validity.

5. We used a *CHECK CONSTRAINT* for start_hour in Bookings to ensure that the hour is between 0 to 23.

6. We decided to create a table for each seniority, Junior, Senior and Manager, instead of a single table with a seniority attribute. This is so that we can accommodate for future updates to the system where there might be additional attributes for a particular seniority.

7. We decided to create DEFER CONSTRAINTS for the foreign key constraint in Seniors and Managers that is INITIALLY DEFERRED, so that it allows one to insert into Seniors or Managers table first before inserting into Bookers. This not only prevents users from directly inserting into Bookers using INSERT or UPDATE without inserting into Seniors or Bookers, but also ensures that the newly inserted employee is either a senior or manager before we grant him/her the authority to make booking.

3.2 Constraints not captured:

1. Does not capture the total participation constraint between *MeetingRooms* and *Updates*.
    - E.g. A record can exist in the *MeetingRooms* table with *(floor, num)* = (1, 2) without a corresponding record with *(floor, num)* = (1, 2) inside the *Updates* table.

2. Does not capture that the employee booking a room is also counted as a participant of the meeting.
    - E.g. A record can exist in the *Bookings* table with *booker_id* = 123 without a corresponding record with *emp_id* = 123 inside the *Attends* table for this particular booking.

3. Does not capture that an employee that is having a fever is unable to join a meeting.
    - E.g. If there exists an employee with *fever* = *TRUE*, he or she still can join the meeting according to the relational schema.

4. Does not capture that an employee must be either a Junior or Booker, and that a Booker must be either a Senior or Manager.
    - E.g. A record can be inserted into the *Employees* table without inserting into either *Juniors*, *Bookers*, *Seniors* or *Managers*, which means that the corresponding employee is neither a junior, senior nor manager.

5. Does not capture that the details and participants of the meeting cannot be changed once the booking is approved.
    - E.g. According to our schema, the attributes in the *Bookings* table can still be changed even when the manager_id is *NOT NULL (implying already approved)*. New participant records corresponding to an approved meeting can also be inserted into the *Attends* table.

4. Interesting Triggers

3 most interesting:

1. A trigger *prevent_insert_and_update_if_conflict* was implemented before insert or update on Attends table to ensure that an employee cannot be attending more than 1 meeting at the same time slot. This is also to help make contact tracing easier as we know definitely which meeting the employee attended.

2. A trigger *auto_generate_emp_id_and_email* that will automatically generate the employee id and the email was implemented before inserting on Employees table. The employee id is generated using SERIAL (automatically increased by 1) while the email address is generated using string concatenation in the format “name” + “emp_id” + “@company.com”. This is to make sure that the emp_id and email are

auto-generated by computer and the email address is always unique before we insert into the Employees table regardless of what email and id we provided.

3. A trigger, *check_if_room_exists_in_updates*, will check that after insertion on table MeetingRooms, there should be a corresponding row in table Updates which specifies the room's capacity. This trigger is deferrable and is initially deferred so that it only checks after a new meeting room is added and the capacity of the room is updated as well. This trigger ensures that there is total participation between MeetingRooms and Updates. It also prevents users from directly inserting into MeetingRooms table without using the procedure add_room.

5. Application Functionalities

For any criterion, process, design issues that is not explicitly stated, you are free to decide on how to address that issue in a reasonable way and justify your decisions in your report

1. We allow employees to change their seniority by inserting into any of the tables junior, senior and manager.
    a. On Juniors/Bookers/Managers/Seniors, we used quite a number of triggers to ensure that each employee is able to change seniority and that they must be in exactly one seniority.
    b. However, we only allow Junior to promote to Senior or Manager and Senior to Manager. We do not allow Manager or Senior to demote to Junior and Manager to demote to Senior since there are tables referencing emp_id in Bookers and Managers.

2. We allow managers to schedule room capacity updates for future dates.
    a. When adding participants to a meeting room in a booking for a future date, only the latest room capacity update will be taken into account.
        - E.g.
            I. On Day 0 (D0), Manager schedules max capacity for room A to be 5 on D5 and 8 on D7.
            II. If someone books a room for D8, the max capacity is taken to be 8.
    b. When a new room capacity update is made, we will delete all bookings that exceed the new max capacity that occur on dates after this, unless there is another room capacity update scheduled further in the future that makes that booking valid.
        - E.g.
            I. On Day 0 (D0), Manager first schedules max capacity for room A to be 8 on D7.
            II. From D8 onwards, any scheduled bookings for room A that has more than 8 participants will be deleted.
            III. On D0, Manager then schedules max capacity for room A to be 5 on D5.

    IV. From D6 - D7, any scheduled bookings for room A that has more than 5 participants will be deleted. Those from D8 onwards will remain since the max capacity for D8 has already been changed to 8.

3. When employees are deleted, instead of deleting them, we set their resigned_date.

4. We ensure that the application schema is not violated when users directly INSERT, DELETE or UPDATE any table.
    a. For all tables in the schema, we tried to cater for the case where users execute INSERT/UPDATE/DELETE on the tables directly without the functions/procedures. We needed to prevent any violation of the project requirements.
    b. For example, we create a trigger that checks AFTER INSERT ON Employees to ensure that the newly added employee id is in the Juniors or Bookers table. If this is not the case, then we will RAISE EXCEPTION and cancel the transaction. Similarly, when one tries to directly UPDATE Employees table, we have another trigger to check if the user is attempting to update emp_id or email, if this is the case, we will not update the emp_id and email attributes but we will still allow the update on other attributes.

5. Employees who did not declare their temperatures for the day cannot be added to any meetings to ensure that only employees that are confirmed not to have a fever can be added to a meeting.

6. If a booker was in close contact with an employee having fever, his bookings will also be cancelled so that is it aligned to the requirements that an employee having a fever will have his booking cancelled as well.

7. When contract_tracing is performed, all meetings that were approved with employees in close contact will have the approval status removed. This is to allow managers to confirm again who are the remaining employees in the booking before approving it again.

8. All the assumptions when using the application:
    a. All meeting rooms under a department must be moved to other department before a department can be deleted.
    b. All employees under a department must be moved to other department before a department can be deleted.
    c. An employee can only be demoted if he does not make or approve any booking or update the max_capacity of any meeting room.

6. Analysis of Normal Forms of Relational Database Schema

Departments(<u>dept_id</u>, dept_name)

- Key: {dept_id}, Prime attribute: {dept_id}
- Non-trivial and decomposed functional dependencies:
    - {dept_id} → {dept_name}
- Since the table only contains two attributes, it is in BCNF and thus 3NF.

Employees(<u>emp_id</u>, email, name, phone, office, home, resigned_date, dept_id)

- Key: {emp_id}, {email}, Prime attribute: {emp_id, email}
- Non-trivial functional dependencies:
    - {emp_id} → {email, name, phone, office, home, resigned_date, dept_id}
    - {email} → {emp_id, name, phone, office, home, resigned_date, dept_id}
- Employees Table is in BCNF as every non-trivial and decomposed functional dependencies (such as {emp_id} → {name}, {emp_id} → {email}, {email} → {emp_id}) has a superkey on its LHS.
- However, since emp_id and email are both unique, they are functionally equivalent. Thus, we could generate the email based on the emp_id and name. As such, attribute email could be removed.

Juniors(<u>emp_id</u>)

- Key: {emp_id}, Prime attribute: {emp_id}
- Since the table only has one attribute, there is no redundancy in the table. Hence, it is in BCNF and thus 3NF.

Bookers(<u>emp_id</u>)

- Key: {emp_id}, Prime attribute: {emp_id}
- Since the table only has one attribute, there is no redundancy in the table. Hence, it is in BCNF and thus 3NF.

Seniors(<u>emp_id</u>)

- Key: {emp_id}, Prime attribute: {emp_id}
- Since the table only has one attribute, there is no redundancy in the table. Hence, it is in BCNF and thus 3NF.

Managers(<u>emp_id</u>)

- Key: {emp_id}, Prime attribute: {emp_id}
- Since the table only has one attribute, there is no redundancy in the table. Hence, it is in BCNF and thus 3NF.

HealthDeclarations(<u>emp_id</u>, <u>date</u>, temperature, fever)

- Key: {emp_id, date}, Prime attribute: {emp_id, date}
- Non-trivial and decomposed functional dependencies:
    - {emp_id, date} → {temperature} (LHS is superkey)

    - {temperature} → {fever} (LHS is not superkey, this FD violates BCNF property)
- Hence the table is not in BCNF. Furthermore, since the RHS (fever) is not in prime attributes, normalization is required.
- Decompose the table into R1(<u>emp_id</u>, <u>date</u>, temperature) and R2(<u>temperature</u>, fever), then the two tables are in BCNF. This is also dependency-preserving as {emp_id, date} → {temperature} can be derived from R1, and {temperature} → {fever} can be derived from R2. When combining the functional dependency in R1 and R2, we get F = {{emp_id, date} → {temperature}, {temperature} → {fever}}, which is equivalent to the functional dependencies in the original table.

MeetingRooms(<u>floor</u>, <u>room</u>, name, dept_id)
- Key: {floor, room}, Prime attribute: {floor, room}
- Non-trivial and decomposed functional dependencies:
    - {floor, room} → {name}
    - {floor, room} → {dept_id}
- For all the non-trivial and decomposed functional dependencies as shown above, the LHS is in the superkey, hence the table is in BCNF, and thus 3NF.

Bookings(booker_id, <u>date</u>, <u>start_hour</u>, <u>floor</u>, <u>room</u>, manager_id)
- Key: {date, start_hour, floor, room}, Prime attribute: {date, start_hour, floor, room}
- Non-trivial and decomposed functional dependencies:
    - {date, start_hour, floor, room} → {booker_id}
    - {date, start_hour, floor, room} → {manager_id}
- For all the non-trivial and decomposed functional dependencies as shown above, the LHS is in the superkey, hence the table is in BCNF, and thus 3NF.

Attends(<u>emp_id</u>, <u>date</u>, <u>start_hour</u>, <u>floor</u>, <u>room</u>)
- Key: {emp_id, date, start_hour, floor, room}, Prime attribute: {emp_id, date, start_hour, floor, room}
- Since all the attributes in the table are inside the prime attributes, hence the table is in 3NF. Furthermore, since all the functional dependencies in this table have superkey on their LHS, hence the table is in BCNF.

Updates(<u>date</u>, <u>floor</u>, <u>room</u>, max_capacity, manager_id)
- Key: {date, floor, room}, Prime attribute: {date, floor, room}
- Non-trivial and decomposed functional dependencies:
    - {date, floor, room} → {max_capacity}
    - {date, floor, room} → {manager_id}
- For all the non-trivial and decomposed functional dependencies as shown above, the LHS is in the superkey, hence the table is in BCNF, and thus 3NF.

7. Reflection

There was some ambiguity in the requirements and we had to account for it or make some assumptions. Some of the assumptions made were mentioned in section 5 (Application Functionalities).

We realize that a lot of triggers are required to prevent users from directly calling INSERT, UPDATE or DELETE to all the tables. Hence, we spent a lot of time writing and changing triggers to handle this problem.

We were conflicted on whether to represent employees with a single table that has an attribute to indicate seniority or have 4 different tables (employees, junior, senior and manager). We wanted to allow users to change the seniority of each employee. However, according to the ER diagram we created, we use ISA for Employees to separate Juniors and Bookers (Seniors and Managers). Hence, we will have foreign key constraint in which Seniors and Managers emp_id is referencing Bookers emp_id. If we insert into Seniors or Managers first without inserting the emp_id into Bookers, then the foreign key constraint is violated. However, if we allow users to insert into Bookers first without insert into Seniors or Managers, one can directly call INSERT INTO Bookers without calling INSERT INTO Seniors or Manager. This will result in violation of the constraint that each employee must be either junior, senior or manager. After a long discussion whether to add an attribute "seniority" in the Employees and remove the Juniors, Seniors, Managers and Bookers tables or follow the original schema, we finally decide to use DEFER constraint that defers the foreign key constraint in Seniors and Managers so that one can insert into Seniors and Managers first before insert into Bookers and if an emp_id is in Bookers then it must be in either Seniors or Managers.

Another thing that we learnt after one hour of debugging is the importance of using brackets to highlight order of precedence. Some of our triggers made use of both AND and OR clauses and without the brackets, it made the triggers execute incorrectly.

We split the functionalities equally among the four of us, so we implemented the routines and triggers independently. Afterwards, when we combined our code, we realised there were some conflicting triggers and had to spend extra time resolving the issues. One of the conflicts was that we prevented employees from leaving an approved meeting room. However, there was another functionality that was supposed to remove employees that have a fever from meeting rooms, whether approved or not.

Technical skills learnt:

- Learn and apply DEFER CONSTRAINT and DEFERRABLE triggers.
- Learn how to use SERIAL to auto increment id.
- Learn how to get the current date using keyword CURRENT_DATE, and current time using keyword EXTRACT.
- Generate series for dates to cross join employees so that we can compute the number of days an employee did not declare his temperature.

Working on a real-life example has also challenged us to think about constraints beyond what was given and allowed us to experience first-hand how complicated and intertwined constraints can be in a real database schema.

Overall, it has been a very fruitful and meaningful learning experience, as we were able to apply everything we have learnt in the semester to potential real life situations.