NATIONAL UNIVERSITY OF SINGAPORE,
SCHOOL OF COMPUTING
CS2102
AY 2021/2022, SEMESTER 1

CS2102 PROJECT REPORT

Noel Mathew Isaac A0202072Y
Vanshiqa Agrawal A0201228W
Kevin Mingtarhja A0219748N
Wong Khia Xeng A0200902Y

6 November 2021

Disclaimer: Run the files in order of schema.sql -> proc.sql -> data.sql

1. List of Project Responsibilities

Name	Responsibilities
Noel Mathew Isaac	● Created dummy data file ● Worked on triggers and functions related to Departments, Meeting Rooms, Joins
Vanshiqa Agrawal	● Worked on triggers and functions related to Updates and HealthDeclaration
Wong Khia Xeng	● Worked on triggers and functions related to Sessions
Kevin Mingtarja	● Worked on triggers and functions related to Employees, Senior, Manager, Booker, Junior

The project responsibilities were mostly evenly distributed. We worked on the first part of the project by creating the ER model and schema together. After working on some of the triggers and functions separately (divided based on workload for each table), we came together to explain and justify our functions and provide feedback to each other. Next, each of us tested on the other person's functions to ensure that everyone's functions would be reviewed at least once before combining them into proc.sql.

2. ER Model

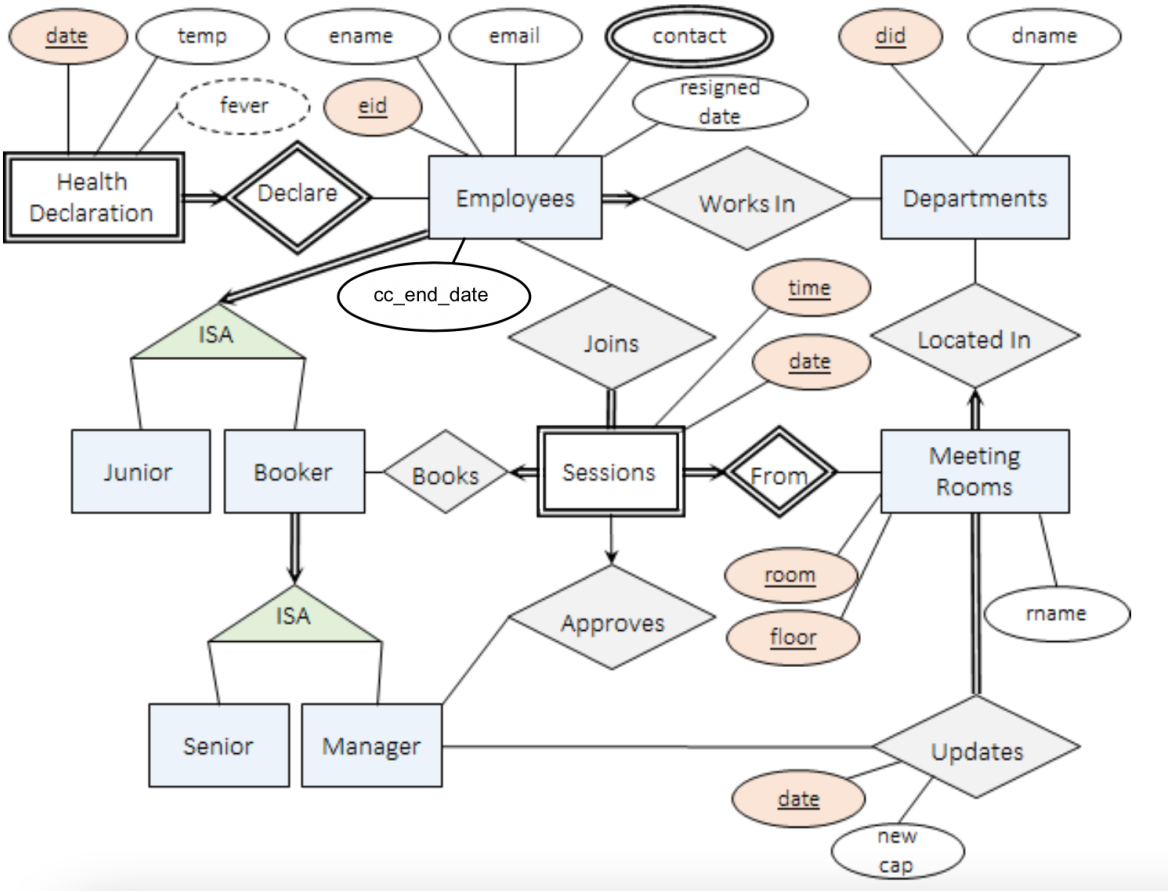

Fig 1 - Final ER Model

2.1 Justification of Non-Trivial Design Decisions

Non-trivial design decisions for entities:

1. **Employees**
   An employee is either a junior or a booker. A booker is then either a senior or a manager. This allows for the higher admin rights of a senior and manager to be easily distinguished from that of a junior in the database system. The double-lined arrows in the 2 ISA relationships denote the covering constraint where the employee has to be either of the two subclasses but cannot be more

than 1 subclass, and that the booker has to be either of the two subclasses but not more than 1 subclass. The contact is now a multi-valued attribute because each employee can have 3 contact numbers. The *cc_end_date* stands for the end date of the close contact period for that employee.

2. **HealthDeclaration**
   The Employees table has a many-to-one constraint with HealthDeclaration. This is because each employee can have multiple health declarations while a health declaration can only belong to one employee. The health declaration is a weak entity because it cannot exist with an employee and has to be uniquely identified considering the employee id. The date along with the employee id forms the primary key for health declaration to uniquely identify and track the daily health declaration of each employee. 'Fever' is a derived attribute that will indicate the true or false based on the 'temp' attribute.
3. **Session**
   The session entity is a weak entity because it cannot exist without a meeting room. It can only be uniquely identified by considering the primary key of the meeting room and if the respective meeting room is deleted, the session entity will no longer exist. Since the time and date are also primary keys, this implements the constraint of not being able to book the same meeting room twice at the same time. The time is taken as the start time of a session and each session is only an hour long. So if an employee wants a 2-hour session, it will be recorded as two Session rows in the Sessions table.

Non-trivial design decisions for Non-trivial Relationships:

1. **Works In**
   An employee has to belong to one and only department, hence the total participation constraint.

2. **Located In**
   A meeting room has to belong to one and only department, hence the total participation constraint.
3. **Joins**
   There is a participation constraint of Sessions in the Joins relationship because once an employee books a session, that employee automatically joins that session.
4. **Updates**
   There is a participation constraint of MeetingRooms in Updates because the very first capacity of each meeting room; when the meeting room is first inserted into the database, is recorded in the Updates table itself.

2.2 (5) Application Constraints Not Captured In ER Diagram

1. If the employee who is booking a room has a fever, then he is not allowed to book the meeting.
2. An employee who has a fever cannot join a session.
3. A manager who is approving a session has to be from the same department as the employee who booked the session.
4. When an employee declares a fever, he/she is removed from all future sessions.
5. When an employee resigns, he/she is removed from future meeting sessions.

These constraints are not captured by the ER diagram and are enforced separately in the schema and using triggers.

3. Relational Database Schema

The relational database schema is as attached in the report, “schema.sql”.

3.1 Justification of Non-Trivial Design Decisions

Employees

- The multivalued *contact* attribute was split into 3 attributes - *mobile_number, home_number* and *office_number*. This was chosen over the method of creating an additional table for the contacts so that searching for an employee’s details would be faster. We also did not combine the 3 phone numbers into a single string attribute so that it is still possible to search employees by his/her phone numbers. The phone numbers are not marked as unique because it is possible for two employees to share phone numbers.
- The *resigned_date* is initially null and then filled with the resignation date so to check if an employee has resigned, the attribute can just be checked for a null value.
- The *cc_end_date* is initially null. When an employee is identified as a close contact till Day D, *cc_end_date* is updated to Day D. If the current *cc_end_date* is more than Day D, the *latest cc_end_date* only is stored.

Health Declaration

- The *fever* is a derived attribute that depends on the *temp* attribute's value and automatically generates True if *temp* > 37.5.
- We placed a constraint to check that the input *temp* is within the acceptable body temperature range.
- Since HealthDeclaration is a weak entity of Employees, we put an ON DELETE ON UPDATE CASCADE constraint on the *eid* attribute.

Sessions

- Sessions table also comprises the Approves relationship so it has a foreign key *approver_eid* to account for that.
- Since it is a weak entity of MeetingRooms, there is an ON DELETE ON UPDATE CASCADE on the (*room, floor)* set that references the MeetingRooms
- The *booker_eid* has a not null constraint to enforce the fact that a booking cannot be made without a booker making the booking.
- There is also a ON DELETE ON UPDATE CASCADE constraint on the *booker_eid* foreign key because if the employee booking the session is deleted or updated in the database, the session's records need no longer exist.

Works In

- There is an attribute called *did* in the Employees table which is a Foreign Key referencing the id of the Departments table to indicate which department an employee belongs to and implement the Works In relationship.
- The *did* has constraint NOT NULL in the schema to ensure all employees have a department.
- The team decided not to create a table called Works In as we felt that implementing the Schema using the Foreign Key in Employees improves readability and maintainability.

Located In

- There is an attribute called *did* in the MeetingRooms table which is a Foreign Key referencing the id of the Departments table to indicate which department each meeting room belongs to and implement the Located In relationship.
- The *did* has constraint NOT NULL in the schema to ensure all rooms belong to a department.
- The team decided not to create a table called Located In as we felt that implementing the Schema using the Foreign Key in MeetingRooms improves readability and maintainability.

Books

- There is an attribute called *booker_eid* in the Sessions table which is a Foreign Key referencing the id of the Booker table to keep track of which employee booked the session and implement the Books relationship.
- Since every session needs to have a booker, the *booker_eid* has constraint NOT NULL.
- The team decided not to create a table called Books as we felt that implementing the Schema using the Foreign Key in Sessions improves readability and maintainability.

Approves

- There is an attribute called *approver_eid* in the Sessions table which is a Foreign Key referencing the id of the Manager table to keep track of whether a session has been approved and implement the Approves relationship.
- If a session has not been approved, then the *approver_eid* will be NULL.
- The team decided not to create a table called Approves as we felt that implementing the Schema using the Foreign Key in Sessions improves readability and maintainability.

3.2 5 Application Constraints Not Captured In RDBS

1. If the employee who is booking a session has a fever, then he is not allowed to book the meeting.
2. An employee who has a fever cannot join a session.
3. A manager who is approving a session has to be from the same department as the employee who booked the session.
4. When an employee declares a fever, he/she is removed from all future meeting sessions.
5. When an employee resigns, he/she is removed from future meeting sessions.

4. Functions

4.1 Justification of Non-Trivial Function Implementation Decisions

search_room

- We have interpreted "*A manager from the same department approves the booking.*" from step 3 of the booking procedure as **The manager must be in the same department as the room.**
- So the return id of the department is the id of the department to which the meeting room belongs.

add_room

- Since each room is associated with a department, we have added an did (department_id) input parameter to the function.
- Additionally, since adding a room requires the capacity of the room to be initialised in the Updates table, we add parameter capacity and manager_eid to the function.
- Once a room is inserted into the MeetingRooms table, a corresponding entry is inserted into the Updates table with the room capacity specified when executing the function. As only a manager can update capacity, the manager_eid is used during insertion into the Updates table

1-hour meeting sessions

- As all the meetings are made up of 1 or more 1-hour slots, we have decided to round down the start_time and round_up the end_time to the nearest hour for all functions taking in these parameters.
- This allows function calls by the user where (end_hour-start_hour) < 1 to work consistently to book a minimum 1 hour slot and also satisfies the assumption that all meetings start on the hour.
- For example if a user tries to book a meeting from 9:01:00 to 9:57:00, then the booking will take place from 9:00:00 to 10:00:00.

- The rounding has been implemented for the following functions: **book_room**, **unbook_room, join_meeting, leave_meeting, approve_meeting and search_room.**

contact_tracing

- When an employee is recorded to have a fever for day D, only meetings from D-1, D-2 and D-3 is considered (excluding Day D) because it is assumed that employees declare temperature at the start of the day and have not actually attended the meetings for the day yet.

declare_health

- An employee is able to declare health more than once a day because there is a trigger attached to INSERT ON HealthDeclaration that checks the employee has already declared *temp* for the *date*. If yes, the trigger will run an update function instead. If not, the trigger allows the current insert operation to continue. This is to accommodate for human errors made by the employee when entering temperatures.

5. (3) Most Interesting Triggers

1. **Table**: Joins

   Trigger name: check_join_constraints

   Design: Trigger called BEFORE INSERT on Joins table

   Usage: This trigger is used to check that the employee joining a meeting satisfies all the required constraints. This is done by calling the can_join_meeting function. As there are several constraints to be checked before an employee can join, the function contains multiple IF-ELSE statements for each constraint. If any of the IF-ELSE statements find a constraint violation, then an EXCEPTION is raised with the appropriate message. If no violation is found then the function returns NEW and the entry is successfully inserted into the Joins table.

2. **Table:** Sessions

   Trigger name: check_booking_constraints

   Design: Trigger called BEFORE INSERT on Sessions table

   Usage: This trigger is used before any insertions on the Sessions table to check if the arguments passed into book_room() meet the constraints of booking a room. In particular, the trigger checks that the booker does not have a fever, the booker is a senior/manager, the time of the booking is of a future date, the booker must not have resigned and the booker must not be an active close contact. These constraints are not enforced in the ER diagram or the schema declaration and thus are encompassed under this trigger instead. One notable line of code in this trigger is the use of SELECT COALESCE(fever, false) that allows the booker to book a room without submitting a health declaration yet for the day.

3. **Table:** Employees

   Trigger name: employees_manual_insert_check

Design: Deferred trigger called AFTER INSERT OR UPDATE on Employees table. The trigger timing is AFTER since deferred triggers do not work with the BEFORE timing.

Usage: This deferred trigger is used to enforce the constraint that each employee must be one of the three kinds of employees: junior, senior or manager. And this trigger enforces it by disallowing manual inserts to the Employees table without a corresponding insertion to Junior, Senior, or Manager in the same transaction.

The trigger function checks whether or not there exists an entry in either one of Junior, Senior, or Manager with the same eid as NEW.eid. If so, then it's a valid operation or else it will delete the new entry in the Employees table and raise an exception, since this trigger is called after insert or update.

The trigger needs to be deferred because if not, it is guaranteed to violate the constraint that is set in the trigger function after the initial insertion to Employees, since at that point in time there will not be any entries with the corresponding eid yet in Junior, Senior, or Manager. Additionally, adding to any of those three tables beforehand also cannot be done due to the foreign key constraints.

Hence the solution is to defer the trigger. So since this trigger will run at the end of a transaction, by the time it checks and assuming the transaction is valid, there will already be a corresponding entry with the same eid as NEW.eid in either one of Junior, Senior, or Manager, which will not violate the constraint given.

6. Analysis of Normal Forms of RDBS

Only the HealthDeclaration table is not in 3NF. This is mainly because *fever* is a derived attribute that creates redundancy, but makes checking processes in the functions easier.

Violating closure = $\{temp\}^+ = \{temp, fever\}$
Dependency-preserving BCNF: R1(*temp, fever)*, R2(*temp, eid, date)*

7. Reflection

It was very interesting taking a very real-life situation and seeing how to turn it into a functional database system. We learnt that firstly, there are many options when it comes to creating an ER model and the difficult part is choosing the best model. For example, in our first submission, the MeetingRoom table had an attribute called *maxCapacity* and *dateUpdated* whilst the final ER model has a separated Updated relationship. The former worked as well but practically, we realised that managers might want to have a record of previous meeting room capacities so a whole table would be better.

Coming up with the ER model also made creating the RDBS and constraints much easier because since the ER model was finalised, we did not have to constantly change the RDBS and propagate the changes to the constraints and triggers written out.

In terms of the triggers, we did a lot of discussion on how to enforce each trigger and function and carried out several test cases that highlighted new problems and would have to fix the functions based on that. For this, we did have a bit of difficulty creating a dummy_data because we realised that online generators of data will not be able to enforce the Foreign Key constraints from the RDBMS. So for this, we had to manually ensure the dummy data did not violate any constraints and we also wrote a Python script to generate data for the Sessions and Joins tables.

In terms of working as a group, the communications between members were quite prompt and thorough and since we managed to split the project quite evenly. We split the functions based on each table and assigned a table to each person so that the person in charge of his/her table has a thorough understanding of that table and there will be no overlap in changes to the RDBMS or overlap in triggers when working remotely. Then, we tested each other's functions and triggers for more thorough testing. It was also very helpful that we had weekly meetings as this encouraged us to finish our assigned work on time for the next meeting.

Overall, we feel that this project has given us a good understanding of the fundamental concepts of Database design and implementation and has imparted valuable practical skills which can be applied during internships and personal projects.