ER Data Model Report

Team 45

Lim Chek Jun [A0183389M] [E0310184]

Timothy Yu Zhiwen [A018610M] [E0310405]

Wong Hon Wee, Howard [A0220867B] [E0556059]

Zhong Shuhao [A0201600H] [E0415409]

2. Responsibilities

The following section is related to the Responsibilities of each student in the project.

Names	Responsibilities
Lim Chek Jun	**Documentation Checking** **Documentation - 3NF for Employees** **SQL Functions:** Basic **SQL Trigger:** Basic
Timothy Yu Zhiwen	**Documentation - Skeleton** **Documentation -3NF for MeetingRooms and Updates** **Schema Creation** **Data population** **SQL Functions:** Admin
Wong Hon Wee, Howard	**Documentation - Triggers** **Documentation - 3NF for Sessions and Joins** **SQL Functions:** Core **SQL Trigger:** Core -
Zhong Shuhao	**Documentation - Checking** **Documentation - 3NF for HealthDeclaration** **SQL Functions:** Health **SQL Trigger:** Contact Tracing

3. Entity Relationship Model

The following section is related to the Entity Relationship Model of the project.

Note: The Entity Relationship Model is obtained from the Proposed ER Diagram for CS2102 Project file. Most related information found in this section may also be found in the stated document. Nonetheless, we will still paraphrase the information to suit our project.

3.1. ER diagram

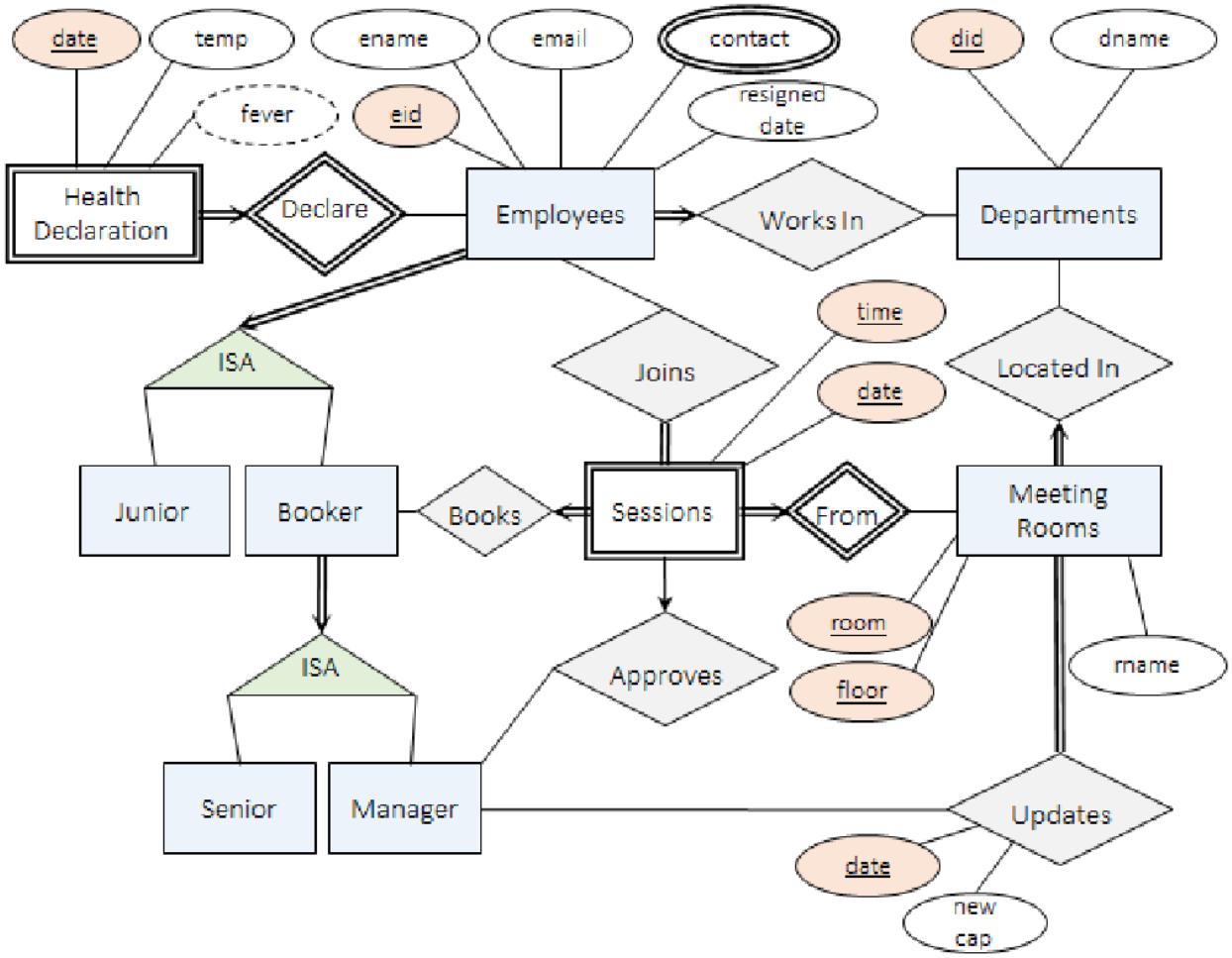

Figure 1: Proposed ER Diagram

3.2. Design Decisions for ER diagram

The following subsections is the Design Decisions of the ER diagram.

3.1.1. Employees

Let us assume an Employee is based on our modern world. If that is the case, there might be multiple ways one could contact such an Employee. In this case, we accept two different types of contact number, one being the primary (e.g. mobile phone number) and the other being secondary, (e.g. emergency contact number).

3.1.2. Health Declaration

According to the ER diagram, Health Declaration is a weak entity set that is dependent on Employee. In the event an Employee doesn't declare for a particular date, this will not be recorded into the Health Declaration, which is useful for checking those that did or did not perform Health Declaration on any particular day.

Furthermore, we deemed it unnecessary to insert time within Health Declaration as it is not required and will not impact any functions heavily. If an employee needs to retake their temperature on the same date, the old record is simply deleted and replaced with the new record.

To add on, we let fever be a derived attribute from the temperature attribute, as it can be derived when inserting the Health Declaration data.

3.1.2. Meeting Rooms and Updates

The Meeting Rooms contains information regarding its floor number, room number, name and the department it is located under. Initially, it was more intuitive for a Meeting Room to contain the capacity of the room. However, all past information regarding the previous capacities of a Meeting Room will not be retained. While it was possible to add a date regarding the capacity change in the Meeting Rooms table, this will not retain the uniqueness of each Meeting Room. (i.e. Each tuple in the Meeting Room might have the same floor and room number, but have different dates).

To add on, even if capacity was a derived attribute from Meeting Rooms, it will complicate the process of removal of booking sessions (assuming that the new capacity is smaller). For example:

1. Assume there is a Meeting Room R with capacity C0 with a Session S of C2 people attending.
    a. This happens on date D + 7 (7 days from now).
2. Suppose we change the capacity of R to C1 on date D + 10, where C1 < C2 < C0

    a. As a result, S will not be removed since it happens on a date that is earlier than D + 10
    b. However, the derived attribute capacity will be changed to C1.
3. Afterwards, suppose we change the capacity of R back to C0 on date D + 5. In this case, there are 2 scenarios
    a. Scenario 1: If we were to check for removal first prior to the change of capacity, this will cause S to be checked against the old capacity C1, where it will be eventually removed although it shouldn't be
    b. Scenario 2: If we were to check for removal after the change of capacity, this would require us to query for the actual capacity between D + 5 and D + 10.
        i. While this is possible to be done, this makes the derived attribute unnecessary as we still are required to query for the capacity during capacity change.

To prevent complications, we decided that having an Updates as a table is necessary to indicate the actual capacity of any Meeting Room. Furthermore, by minimal requirements, a Meeting Room is required to have a capacity when added to the database. To ensure that a Meeting Room always has a capacity, we included in total participant constraints between Meeting Rooms and Updates.

3.1.3. Junior, Senior, Manager and Booker

Our initial proposal for Employees, Juniors, Senior and Manager was to simply have an Employee to be either a Junior, Senior or Manager. This is shown as according to the below figure:

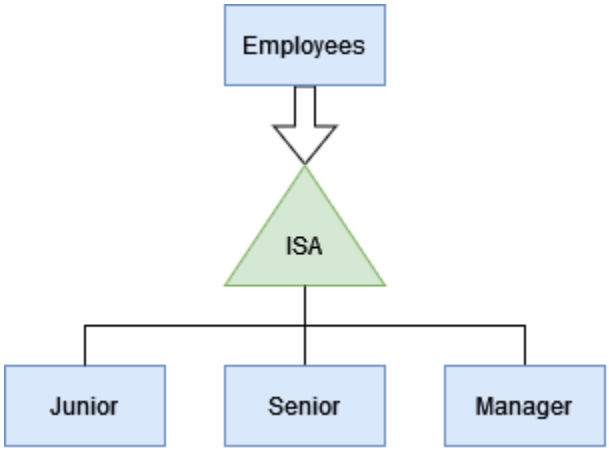

Figure 2: Initial ISA proposal

The reason behind the Figure 2 proposal was that it was much more intuitive as opposed to our current ER diagram especially since it was a requirement for an Employee to be a Junior, Senior or Manager. Furthermore, it also seemed to reduce the number of entities. This resulted in 3 scenario from here on:

1. Create an aggregate relationship between Senior and Manager.
    a. However there was no logic in having a relationship between Senior and Manager.
2. Create a separate Books relation for both Senior and Manager.
    a. This seemed redundant as both Senior and Manager books a Session the same way as each other.
3. Create a ternary relation between Senior, Manager and Session.
    a. However, similar to Scenario 1, this requires Senior and Manager to have logic between each other.

Furthermore, this design also may seem like Senior and Manager are both required to perform a booking on a Session, making it counter-intuitive. This resulted in us creating another level of ISA relationship, where a Booker is either a Senior or Manager. This allows Booker to have a singular books relationship with Sessions.

3.1.4. Sessions and Meeting Rooms

In our ER diagram, Sessions is a weak entity set that depends on Meeting Rooms. This ensures that each Meeting Room can only have up to one Session per particular date and time. Note that for simplicity, we assume that each session lasts for only an hour.

3.1.5. Sessions, Books, Joins and Approves

For our interpretation of the term *"A manager from the same department approves the booking"*. We decided that the interpret it as:

"The manager must be in the same department as the **room**."

Sessions was an entity that has multiple different relationships to it. An Employee joins a Session, a Booker books a Session and a Manager approves a Session. As a result, there are several decisions that ultimately result in our current design.

For example, here are some constraints and reasoning:

- Booker needs to participate in the Session he or she booked.
    - This requires Sessions to have a total participants constraint with Joins to ensure that a Session must have an Employee (i.e. the booker).
- A Session must be booked by one booker.
    - This requires Sessions to have a total participants constraint with Books to ensure that a Session must be booked by a booker.
    - This also requires Sessions to be in a one-to-many relationship since a booker could book multiple Sessions.
- A Manger could approve a Session or not, but must be from the same department as the **Meeting Room**.

- This requires Sessions to have a one-to-many relationship with the Manager entity. However, to ensure the constraint of the same department as the Meeting Room, it would cause the ER diagram to be too convoluted with the design thus far. Hence, we left this as one of the uncaptured constraints in our ER diagram (as stated below).

3.2. Uncaptured Constraints for ER diagram

For the 5 uncaptured constraints, we selected the ones we deemed to be the most critical to the database. The following constraints are not captured within the ER Diagram itself:

1. Contact tracing implementation - Employees in close contact of those who have fever will be removed from all Sessions.
2. When an employee resigns, they are no longer allowed to book or approve any meetings.
3. Once approved, there should be no more changes in the participants and the participants definitely will come to the meeting on the stated date.
4. A booking must be approved by a manager from the same department.
5. If an employee is having a fever, they cannot join a booked meeting.

4. Relational Database Schema

The following section is related to the Relational Database Schema of the project.

4.1. Design Decisions for Relational Schema

The following subsections is the Design Decisions of the Relational Schema.

4.1.1. Employees

In the Employees table, we ensure that the *email* is unique by providing the UNIQUE constraint on the *email*, covering what the constraint in which the ER diagram does not capture.

Furthermore, by our ER diagram, an Employee has a total participant constraint with Departments in a one-to-many relationship. To capture these constraints in our schema, we let Employees reference the Department table by holding the *did* (Department ID) attribute that IS NOT NULL. We chose this over having a separate Works In table for simplicity.

4.1.2. Meeting Rooms

In the Meeting Rooms table, notice that in the ER diagram, Meeting Rooms have a one-to-many relationship with Departments with a total participants constraint. This is similar to the constraint as stated in the Employees table. Hence, we also capture these constraints by having each Meeting Room reference Department table by holding the *did* (Department ID) attribute that IS NOT NULL.

4.1.3. Junior, Booker, Senior and Manager

As shown in our ER diagram, it is stated that an Employee is either a Junior or Booker, and a booker is either a Senior or Manager. To capture the IS A relation, we let Junior and Booker reference Employee's *eid* (Employee ID) attribute. Similarly, Senior and Manager references Booker's eid (Employee ID) attribute as well to indicate that a Booker has to be a Senior or Manager. Furthermore, to capture the IS A relationship, we set all of them to be ON DELETE CASCADE, such that when we delete an Employee, their roles in the respective tables are deleted as well.

4.1.4. Health Declaration

Since Health Declaration is a weak entity set that is dependent on Employee, we ensure this constraint by having Health Declaration table reference Employee's *eid* attribute. Furthermore, we have to ensure that it is part of a PRIMARY KEY together with the date so that we can uniquely define whether an Employee performed their Health Declaration within that particular date.

The Relational Schema itself, however, is unable to capture the constraint where an Employee could possibly perform a Health Declaration twice within the same day. This constraint can only be captured via our functionality and using a trigger.

Furthermore, when performing Health Declaration, our functionality will provide an automatic check of the temperature of the Employee to determine whether they have a fever or not (37.5 degree celsius or above).

4.1.5. Updates

From the ER diagram, we know that Updates is a many-to-many relationship between Meeting Room and Manager. To ensure the many-to-many relationship, we created a separate Updates table, such that it references both Meeting Room's PRIMARY KEY (floor and room) and Manager's PRIMARY KEY (eid, or Manager's ID).

4.1.6. Sessions

Let us consider the design of Sessions. Sessions is a weak entity set that is dependent on the PRIMARY KEY of the Meeting Rooms, hence, an instance of Sessions holds a room and a floor of the Meeting Rooms.

For our interpretation of the term *"A manager from the same department approves the booking"*. We decided that the interpret it as:

"The manager must be in the same department as the **room**."

4.1.7. Books

Since we know that a Booker books a Session, having a one-to-many relationship, we can merge the Books entity into the Sessions as according to the ER diagram for simplicity. As a result, together with total participation constraint, a Sessions will reference an instance of a Booker's *eid* (Employee ID) and the reference cannot be NULL.

4.1.8. Joins

It is known that an Employee is able to Join a Session, and since they can join as many Sessions as possible, they are in a many-to-many relation. Each tuple in the Joins table is required to reference the Sessions PRIMARY KEY (time, date, Meeting Room's room and floor). Similarly, each tuple also references the Employee's PRIMARY KEY (*eid*) for the Employee that is joining.

4.1.9. Approves

For Approves, it is a one-to-many relationship between Sessions and Manager. We decided to merge the Approves entity with the Sessions entity in one table for simplicity since only a Session can only be approved by one Manager. Hence we have the Sessions table reference

from the Manager's PRIMARY KEY (*eid)*. To add on, we also know that a Session cannot be approved (via the lack in total participants constraint), we do not set the *eid* in Sessions to be NOT NULL.

4.2. Uncaptured Constraints for Relational Schema

The following constraints are not captured by the Relational Schema, but could be captured with our Triggers.

1. When an employee resigns, they are no longer allowed to book or approve any meetings.
2. If an employee is having a fever, they cannot book a room.
3. The employee booking the room immediately joins the booked meeting.
4. An approval can only be made on future meetings. i.e. a Manager could approve a meeting before their retirement or before they get fever.
5. Contact tracing implementation - Employees in close contact of those who have fever will be removed from all Sessions.

5. Three Most Interesting Triggers

This section consists of the three most interesting triggers that our database have.

5.1. check_join

The trigger check_join is triggered before any insertion of rows is done on the joins table in order to verify the validity of the meeting and the employees joining it. When triggered, the trigger calls the function check_joining which checks if the meeting the employee is trying to join has been booked and approved already before checking whether the employee also does not have any fever or has been in close contact in order to maintain safe distancing measures. The trigger also checks to ensure that the meeting has not been approved by any managers yet and that the meeting date has already passed before allowing the employee to join the meeting. This trigger was implemented this way in order to ensure that all employees trying to join the meetings meet the requirements even if the function join_meeting is not called.

5.2. check_book

The trigger check_book is triggered on insertion of any rows to make sure that the employee is allowed to book a meeting. When triggered, the trigger calls the function check_booking that checks if the employee has had any fever or has been in close contact recently in order to maintain safe distancing measures and is also not a junior employee who is not allowed to book any meetings. Lastly, the trigger checks to make sure that the date of the meeting has not has not passed yet. The trigger was implemented in order to make sure that all attempts to book meetings be it through the function or manually is checked before it is allowed through.

5.3. check_retired_Health_Declaration

The trigger check_retired_Health_Declaration is triggered before any insertion of rows on the Health Declaration table in order to verify the validity of the employee's health declaration. When triggered, the trigger calls the function is_employee_retired that checks if the employee's resigned_date occurs before the date in the inserted row. If that is the case, the insertion is not allowed as retired employees cannot perform health declarations. By extension since they cannot perform health declarations, they also cannot participate in any meetings. There are actually 3 other similar triggers that call the same function for the tables Update, Sessions and Joins just in case the user tries to insert rows for a retired employee.

6. Analysis of Database Schema's Normal Forms

This section comprises the normal forms analysis of our database schema. Those tables that are not included are as follows:

- *Departments*
- *Junior*
- *Booker*
- *Senior*
- *Manager*

This is because they are trivially BCNF as they have either 1 or 2 attributes in the table. By the lemma: *"Tables that satisfy BCNF will also satisfy 3NF"*, the above tables are indeed in 3NF.

6.1. Employees

Functional Dependencies:
{eid, ename} -> {email}
By reflexivity {eid} -> {email}

{eid} -> {ename}
{eid} -> {email}
{eid} -> {primary_contact}
{eid} -> {secondary_contact}
{eid} -> {resigned_date}
{eid} -> {did}

{email} -> {eid}
{email} -> {ename}
{email} -> {primary_contact}
{email} -> {secondary_contact}|
{email} -> {resigned_date}
{email} -> {did}

Keys: {eid}, {email}

Prime Attributes: {eid, email}

Non-trivial and decomposed FDs:

For all the FDs whose LHS is {eid}, {eid} the primary key of the table and is trivially a superkey.

For all the FDs whose LHS is {email}, the closure is:

{email} += {eid, ename, primary_contact, secondary_contact, resigned_date, did, email}

Since the RHS has every attribute in the table, it means that {email} is a candidate key, and thus a superkey.

Hence, it is in 3NF.

6.2. HealthDeclaration

Functional Dependencies of HealthDeclaration:

{eid, date} -> {temp}, {eid, date} -> {fever}, {temp} -> {fever}

Key: {eid, date}

Prime Attribute: {eid, date}

Non-trivial and decomposed FDs:

For {temp} -> {fever}, the LHS does not compromise of a superkey:

{temp} += {temp, fever}.

Similarly, for {temp} -> {fever}, the RHS also does not compromise a prime attribute.

Hence, it is not in 3NF.

BCNF decomposition:

R1 = {<u>eid, date</u>, temp} with foreign key constraints R1.temp references R2.temp

R2 = {<u>temp</u>, fever}

For R1, {eid, date} is the primary key for HealthDeclaration and is trivially a superkey. **Hence R1 is in BCNF**. For R2, since there are only 2 attributes, **R2 is trivially in BCNF**.

6.3. MeetingRooms

Functional Dependencies of MeetingRooms: {room, floor} -> {rname} and {room, floor} -> {did}

Key: {room, floor}

Prime Attribute: {room, floor}

Non-trivial and decomposed FDs:

For {room, floor} -> {rname} and {room, floor} -> {did}

{room, floor} is the primary key of the table MeetingRooms and is trivially a superkey.

Hence, it is in 3NF.

6.4. Updates

Functional Dependencies of Updates: {room, floor, eid, date} -> {new_cap}

Key: {room, floor, eid, date}

Prime Attribute: {room, floor, eid, date}

Non-trivial and decomposed FDs:

For {room, floor, eid, date} -> {new_cap}

{room, floor, eid, date} is the primary key of the table Updates and is trivially a superkey.

Hence, it is in 3NF.

6.5. Sessions

Functional Dependencies of Sessions:

{room, floor, time, date} -> {meid}, {room, floor, time, date} -> {beid}

Key: {room, floor, eid, date}

Prime Attribute: {room, floor, time, date}

Non-trivial and decomposed FDs:

For {room, floor, time, date} -> {meid} and {room, floor, time, date} -> {beid}

{room, floor, time, date} is the primary key of the table Sessions and is trivially a superkey.

Hence, it satisfies 3NF since every non-trivial and decomposed FD has a superkey as its left hand side.

6.6. Joins

Functional Dependencies of Joins: None

Key: {eid, room, floor, time, date}

Since the primary key of the table contains all the attributes in the table, the table is indeed **in 3NF.**

7. Reflection

Throughout the project, we as a team went through several hardships, both team-communication and database system wise. An example of team-communication difficulties would be that we do not have a readily available time for us to meet up virtually and communicate about the project. As a result, it was difficult to socially bond with one another to improve our communication flow during the project. As for an example of project difficulty, we had a hard time deciding what we should initially do for the project. The project details were complicated, making it difficult to understand what we should be doing first to start the project.

However, such hardships provided us with valuable experience that help us develop and hone our skills for database systems. One example would be our meetings. While most of our meetings were awkwardly timed, we ensured that each meeting we had was properly scheduled and had timely lengths, having almost every moment in the meeting be about the project itself. Furthermore, while we had difficulty starting off, once we finished the schema, we tasked ourselves to do a particular section, having those with shorter sections work on the document or help populate the data.