CS2102
Database Systems
Project Report

Team 70

Name	Matriculation number
Tan Yee Jet	A0200699Y
Hoang Trong Tan	A0219767M
Nguyen Minh Hieu	A0200814W
Vo Quang Hung	A0200697A

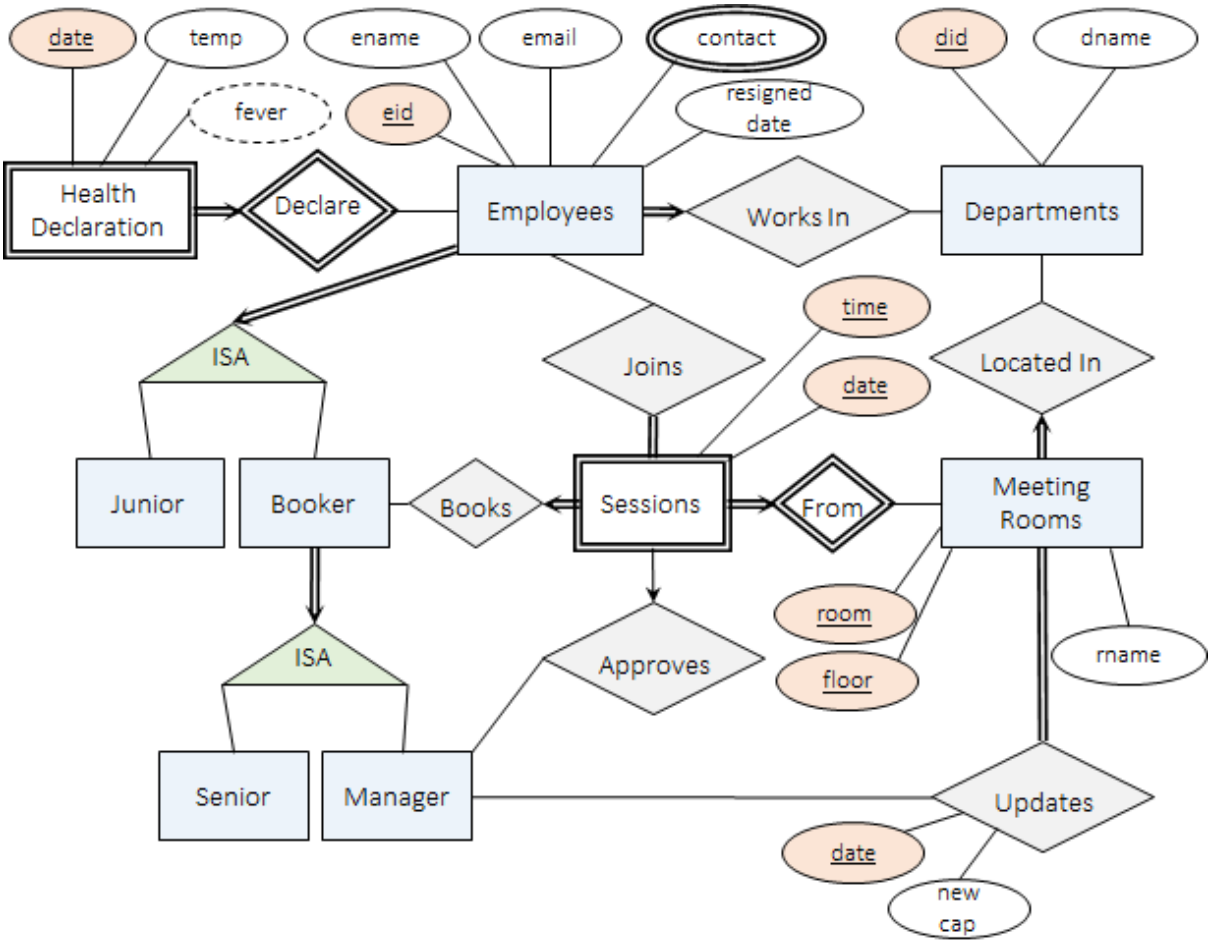
Table of content

Group 70

1. Project responsibilities

Name	Responsibilities
Everyone	● ER diagram ● Write report
Tan Yee Jet	● Generate data ● Write Health functions ● Test Admin functions
Hoang Trong Tan	● Write Core functions ● Write Utilities functions for checking ● Test Health functions
Nguyen Minh Hieu	● Generate data ● Write Basic functions ● Test Core functions
Vo Quang Hung	● Write Admin functions ● Test Basic functions ● Test Core functions

2. ER diagram



We decided to adopt the proposed ER diagram provided by the teaching team for the development process.

a. Non-trivial design decisions of the ER diagram

i. We decided to use a two-level layout for the Employees ISA hierarchy instead of the one-level layout (Junior, Senior and Manager) or the three-level layout (based on capabilities), with regards to the proposed ER.

1. The one-level layout is inconvenient when querying for booking eligibility because we will need to check 2 tables -- Manager and Senior instead of just the Booker table.
2. The three-level layout is redundant as the uppermost layer with no booking or approving capabilities has no actual usability, which gives the same result as querying directly from the Employees table.

ii. Not putting capacity inside Meeting Rooms:

1. A good argument is made in the proposed ER document which considers the case of future capacity update. In this case, if we put the capacity inside the

meeting room, it would also affect the queries that rely on the current capacity. However, as the `change_capacity` function specifies the assumption that the update only happens on the current day, the future update can be considered out of scope.

2. Another reason not to put the capacity in Meeting Rooms is that it would be error-prone to update the value every time we change room capacity. While querying for the latest room capacity from Updates is costly, we consider it acceptable in the small scope of the project.

b. 5 uncaptured constraints of the ER diagram

i. Before the removal of a department, all Meeting Rooms and Employees related to that department must also be removed or reallocated to another department. The relations connected to Departments include Employees, MeetingRooms, Sessions, Joins, Updates. Users have to manually call `unbook_room` to remove all Sessions related to the department to be deleted. Additionally, users should completely remove or relocate Employees or Meeting Rooms to another department by using `DELETE` or `UPDATE` statements on Employees and Meeting Rooms relations. Otherwise, the `remove_department` function cannot be performed and will raise an error.

ii. Employees must make daily health declarations. In the ER diagram, no cardinality constraint can be used to indicate that each employee must declare their health daily.

iii. The employee booking the room immediately joins the booked meeting. Instead, this constraint is captured in the `book_room` procedure.

iv. There should be no more changes in the participants and the participants will definitely come to the meeting on the stipulated day, if the meeting is approved. The ER diagram does not record the change in the status of the meeting. This constraint is enforced in the `book_room` procedure.

v. If the declared temperature exceeds 37.5 on a day, that employee will have a fever. Because the fever status can be derived from temperature, the ER diagram does not capture this constraint. Instead, this constraint is enforced when the employee declares his temperature, in the `declare_health` procedure.

3. Relational database schema

a. Non-trivial design decision of the schema

i. Employees

1. In the ER diagram, contact is a multivalued attribute. Following the proposed ER diagram, we split the contact attribute to 3 attributes: mobile number, office number, home number. Because employees can have the same home number but their mobile number and the office number are unique to them, we decided to set the office number and mobile number to be `UNIQUE`.
2. To implement the resignation function, we have a column named `resigned_date` with default value `NULL`. When we want to remove an employee, we will update the `resigned_date` with a date. When we need to check whether an employee resigned, we will check if the value of `resigned_date` is `NULL`.
3. We specify the behavior of deleting foreign key department ID to `NO ACTION` to prevent deletion of existing employees when we remove department.

ii. Departments

1. No non-trivial design.

iii. HealthDeclaration

1. The table contains eid from Employees because it is a weak entity set. We set the primary key to `(eid, date)`.
2. We have 2 constraints -- `valid_temp` and `fever_check` in this table. The first constraint is used to check the validity of temperature, which needs to be in the range from 34 to 43. The second constraint is used to check the validity of fever status and temperature.

iv. MeetingRooms

1. We specify the behavior of deleting foreign key Department ID (did) to `NO ACTION` to prevent deletion of existing meeting rooms when we remove department.

v. Sessions

1. A session is a weak entity set dependent on the meeting room to ensure that each meeting room can only have exactly one session with a session having a partial key of date and time. Because of that, we set the primary key to `(date, time, room, floor)`.
2. To implement the assumption that time is the start time with each session lasting only one hour, we set the type of time to `INTEGER` and have a constraint to check if time is in the range from 0 to 23.

vi. Booker / Manager

1. To implement the two-level ISA hierarchy, we have decided to create only two tables -- Booker and Manager for simplicity and easy maintenance. The junior employees will be in the Employees table but not in the Booker table. The

senior employees will be in the Booker table but not the Manager table. The managers will be recorded in the Manager table.

vii. Joins

1. We record meeting participations by putting employee ID and their meeting sessions in this table. We set the primary key to `(eid, time, date, room, floor)` where `(time, date, room, floor)` is used to specify the session.
2. We specify the behavior of an update or deletion of the foreign key for Sessions to `CASCADE` so that we only need to focus on the Sessions when we want to make an edit.

viii. Updates

1. This table records all the updates on meeting room capacity.
2. The datetime column has type `TIMESTAMP` to allow updates on the same day but at different times.
3. `eid` is included to record the manager ID making the update. When we first create a meeting room, `eid` can be set to `NULL`.

b. 5 uncaptured constraints of the schema

In our implementation, only 3 uncaptured constraints are captured by triggers whereas the others are captured using checks inside of functions. Hence, constraints listed are not necessarily enforced using triggers.

i. When a manager updates a meeting room capacity, any future meeting in which the number of attendees exceeds the updated meeting room capacity should be cancelled. This is captured by the trigger `remove_bookings_over_capacity`.

ii. When a fever is detected, the employee should be removed from future meetings. If the employee is the one booking a meeting, the meeting is cancelled. In addition, the employee's close contacts should also be removed from future meetings and have their bookings removed, but only for the future 7 days. This is captured by the trigger `fever_detected`.

iii. When an employee resigns, the employee should be removed from future meetings. If the employee is the one booking a meeting, the meeting is cancelled. If the employee is the one approving a meeting, the meeting is reverted back to booked and not-yet-approved (the meeting is not cancelled). This is captured by the trigger `resignation_sop`.

iv. An employee can only approve, book, join, unbook or leave future meetings. This is captured in the functions `book_room`, `unbook_room`, `join_meeting`, `leave_meeting` and `approve_meeting`. In these functions, we check that the date input is larger than the current date before allowing for any meeting joins, meeting bookings or meeting approvals. They could also be done using triggers.

v. A manager can only approve meetings in meeting rooms or update capacity of meeting rooms under his or her department. This is captured in the functions `approve_meeting` and `change_capacity`. In these functions, we check that the manager and the meeting room have the same department ID before allowing for any meeting approvals or meeting room capacity updates. They could also be done using triggers.

4. Three most interesting triggers

a. The `fever_detected` trigger.
   i. It is used to trigger the series of events when a fever is detected upon a temperature declaration of more than 37.5 of any employee. We first check if the employee is a booker. If the employee is a booker, all future meetings the employee booked will be deleted, approved or not, otherwise the employee will just be removed from all future meetings, approved or not. After that, we query the close contacts of said employee. Similarly for close contacts, they are removed from future meetings and have their bookings removed, but only for the future 7 days.
   ii. The trigger is designed as such because we are catering for 2 categories of employees when a fever is detected in a health declaration. The first category is the employee having a fever and the second is the employee's close contacts. For the employee having a fever, the procedure that follows applies to all future bookings or meetings. However, for close contacts, the procedure only applies to bookings or meetings in the future 7 days.

b. The `resignation_sop` trigger.
   i. It is used to trigger the series of events when there is an update on the resignation date of any employee. We first get all future meetings which are approved by the employee and set them to unapproved. We then delete all future meetings which are booked by the employee and remove the employee from all future meetings.
   ii. The trigger is designed as such because we need to keep past records of the employee when he resigns, rather than just deleting the employee from the Employees table. Therefore, this trigger specifically removes future records (approval, booking and joins) of the resigned employee only and retains the past records.

c. The `remove_bookings_over_capacity` trigger.
   i. It is used to trigger the series of events when there is a new update on the capacity of any meeting room. It will get all future meetings for that meeting room in which the number of attendees exceeds the updated meeting room capacity and delete all of those meetings. Consequently, employees initially joining these meetings will automatically exit the meeting. This is implemented using `ON DELETE CASCADE` on the Joins table.
   ii. One interesting case that this trigger also considers is that if the room capacity is changed in the future (e.g. on day D + 3 and day D + 10 where D is current day) multiple times, all meeting sessions that do not satisfy the capacity constraint between day D + 3 and day D + 10 will also be deleted. However, since our application assumes that one can only update the capacity on the current day, this case is never reached in actual usage.

5. Analysis of normal forms

FD means functional dependency, LHS and RHS is left/right hand side (of the equations)

a. Employees (eid, ename, email, home_number, mobile_number, office_number, resigned_date, did)
   i. FD:
      1. {eid} → {ename, home_number, resigned_date, did}
      2. {email} → {ename, home_number, resigned_date, did}
      3. {mobile_number} → {ename, home_number, resigned_date, did}
      4. {office_number} → {ename, home_number, resigned_date, did}
   ii. Superkey: {eid, email, mobile_number, office_number}
   iii. Non-trivial and decomposed FD: we observe that
      1. Every FD without {eid}, {email}, {mobile_number}, {office_number} on LHS will have trivial FDs
      2. This leaves only non-trivial FDs with {eid}, {email}, {mobile_number}, {office_number} on LHS
      3. After decomposing, we will have non-trivial and decomposed FD with only superkeys on LHS
   iv. Therefore, Employees relation is in BCNF

b. Departments (did, dname)
   i. Table only has 2 columns
   ii. Table is in BCNF

c. HealthDeclaration (eid, date, temp, fever)
   i. FD:
      1. {eid, date} → {temp, fever}
      2. {temp} → {fever}
   ii. Superkey: {eid, date}
   iii. Non-trivial and decomposed FD:
      1. {eid, date} → {temp}
      2. {eid, date} → {fever}
      3. {eid, temp} → {fever}
      4. {date, temp} → {fever}
      5. {eid, date, temp} → {fever}
      6. {eid, date, fever} → {temp}
   iv. LHS of (3), (4) are not superkeys
   v. RHS of (3), (4) are not prime attributes
   vi. Table not in 3NF
   vii. BCNF decomposition:
      1. R1 (temp, fever)
         a. Table only has 2 columns
         b. Table is in BCNF
      2. R2 (temp, eid, date)
         a. Non-trivial and decomposed FD:
            i. {eid, date} → {temp}
         b. All LHS are superkeys

c. Table is in BCNF

d. Sessions (date, time, room, floor, eid_booker, eid_manager)
   i. FD:
      1. {date, time, room, floor} → {eid_booker, eid_manager}
   ii. Super key: {date, time, room, floor}
   iii. Non-trivial and decomposed FD:
      1. {date, time, room, floor} → {eid_booker}
      2. {date, time, room, floor} → {eid_manager}
   iv. Table is in BCNF

e. MeetingRooms (room, floor, did, rname)
   i. FD:
      1. {room, floor} → {did, rname}
   ii. Super key: {room, floor}
   iii. Non-trivial and decomposed FD:
      1. {room, floor} → {did}
      2. {room, floor} → {rname}
   iv. Every non-trivial and decomposed FD has a superkey on LHS
   v. Table is in BCNF

f. Booker (eid)
   i. Table only has 1 column
   ii. Table is in 1NF

g. Manager (eid)
   i. Table only has 1 column
   ii. Table is in 1NF

h. Joins (date, time, room, floor, eid)
   i. FD: None
   ii. Non-trivial and decomposed FD: None
   iii. Superkey: {date, time, room, floor, eid}
   iv. Table is in BCNF

i. Updates (room, floor, datetime, eid, new_cap)
   i. FD:
      1. {room, floor, datetime} → {eid, new_cap}
   ii. Super key: {room, floor, datetime}
   iii. Non-trivial and decomposed FD:
      1. {room, floor, datetime} → {eid}
      2. {room, floor, datetime} → {new_cap}
   iv. Every non-trivial and decomposed FD has a superkey on LHS
   v. Table is in BCNF

6. Extra assumptions made during the development process

a. Managers having fever still can approve meetings.
b. For the `book_meeting` function, a meeting is only booked if all hours between starting and ending time are available.
c. For `unbook_meeting`, `approve_meeting`, `join_meeting` and `leave_meeting`, execute the function for all meetings satisfying the constraints. For the ones not satisfying constraints, ignore and raise a notice.
d. The `change_capacity` function will be called for the current day by default, but the date can be set manually by the user.
e. If a manager resigns, his changes on the capacity of the meeting rooms will still be valid.
f. Future meetings are assumed to be after the current date only.
g. The default date format is “yyyy-mm-dd”, e.g. 2021-10-16.
h. The employees are gentlemanly, i.e you can not override an unapproved booking by booking the same slot. If person A books a room but is unapproved, and person B wants the same room for the same time, person A will unbook the room so person B can book it.
i. Employees have to have at least 1 health declaration before they can approve or book or join any meeting. This is to ensure that they do not have fever.
j. We do not allow the booker to leave the meeting. To leave the meeting, the booker has to unbook the meeting first.
k. A room’s capacity can be changed multiple times a day, and we will only take the most recent one

7. Reflection after the project

One of the difficulties that we faced was data generation. Initially, we used Mockaroo for data generation. However, we soon found that a lot of the data violates the constraints in the schema. We ended up writing our own script which generates data consistent with these constraints. In the case that we missed out any aspects that should be part of the test case, we manually add those in afterwards. While Mockaroo generates more realistic data, we learnt that it is still better to write our own script as it is more consistent with our schema. If possible, we could use a mix of both methods, just to get more realistic values for certain columns.

Besides, due to lacking experiences in designing and implementing database systems, our project is like an iterative process. Sometimes, we need to go back and start from the beginning. In our case, in the beginning, we interpreted some points in the project description wrongly, hence we had to edit the schema even though the functions are already half done. Moreover, halfway through implementing functions, we found that some requirements are not so specific. This is difficult as everyone in the group might have a different interpretation. We learnt that we should think about the requirements of the project carefully and figure out these difficulties in the first step so that we do not need to loop back every so often. With that, we learnt that communication is especially important. Therefore, we had 1 – 2 meetings every week, and each session was about 1 – 1.5 hours. This type of regular and short meetings proves to be very useful in keeping us on the same page.

The third difficulty was to make reasonable assumptions in the parts where the requirements are vague. We understand that this is to simulate the discrepancy between the customer requirements and the technical requirements for the developers during the development process. Therefore, we put in our best to make reasonable assumptions that would fit the scenarios in real life. In the case of facing dilemmas, we try to find answers in the Luminus project forum. We learnt that, in real projects, communications between the team and the stakeholders are very important.

Lastly, we feel that coming up with an ER diagram from the project requirements was the hardest part. We made many errors at this stage of the project, resulting in major redesign of the ER diagram. After some reflection, we think it was because we have not summarized all the constraints from the project requirements which led to these mistakes. In the future, if we have a chance to work on a database project in practice, the first thing we would do is to summarize and clarify the requirements from the beginning by seeking clarifications from the users in order to avoid such misunderstanding situations.

We appreciate the lessons learnt throughout the project. From ER modelling to logical schema design to implementing functions and triggers to debugging and testing, we grew so much in every part of the process. We also appreciate the friendships as well as the hardships that have brought us to this stage. It was not easy and was well worth it.