AY20/21 SEMESTER 2
CS2102 TEAM PROJECT

TEAM 41

Team Members:

Firzan Armani Bin Fajar Masazman	A0184164A
Hemanshu Gandhi	A0180329E
Loong Jian Hung, Joel	A0182725B
Zhou Tianyu	A0187444W

Team Responsibilities

Firzan:

- Creation of PostgreSQL relational database schema.
- Creation of functions and procedures (1, 2, 10, 11, 15, 25, 30).
- Creation of triggers.
- Populating tables with data.
- Writing report.

Hemanshu:

- Creation of PostgreSQL relational database schema.
- Creation of functions and procedures (6, 7, 8, 9, 16, 17, 21).
- Creation of triggers.
- Populating tables with data.
- Writing report.

Joel:

- Setting up GitHub repo.
- Creation of PostgreSQL relational database schema.
- Creation of functions and procedures (3, 4, 5, 13, 14, 18, 19, 20, 22).
- Creation of triggers.
- Populating tables with data.
- Writing report.

Tianyu:

- Creation of PostgreSQL relational database schema.
- Creation of functions and procedures (12, 23, 24, 25, 26, 27, 29).
- Creation of triggers.
- Populating tables with data.
- Writing report.

Difficulties Encountered and Lessons Learned

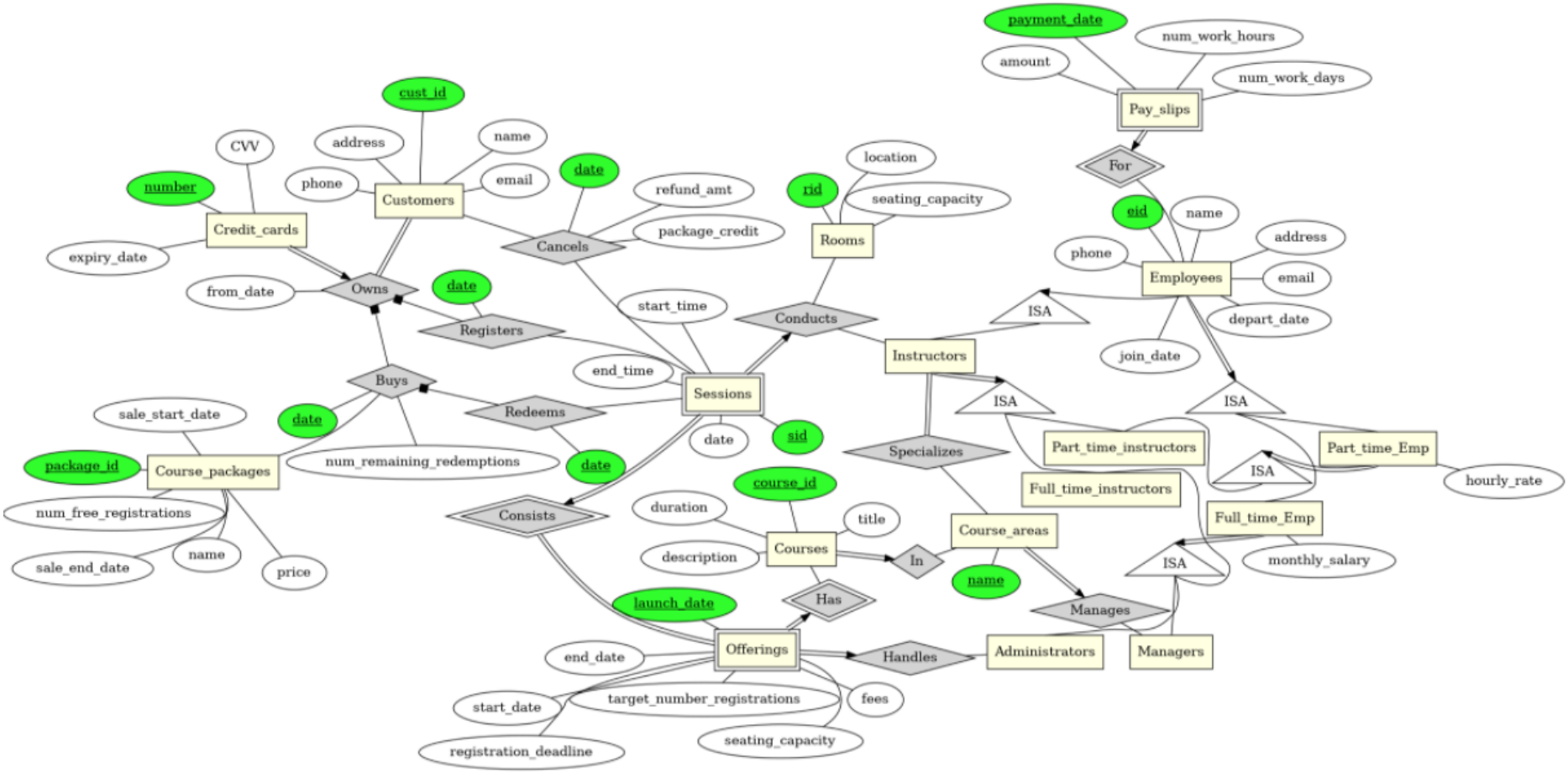
Firstly, understanding the application domain was a challenge, as a lot of information was given to us over several paragraphs in English. Also, when certain aspects of the requirements and expected behaviours were not clear-cut, we would make use of the course forum in order to look at the respective clarifications given by our professors.

Secondly, deciding how to collaborate as a team was a challenge, as it was hard to separate concerns for each member. For instance, we needed to collaborate together to implement the schema.

Thirdly, some routines require knowledge that was not taught in the module such as handling arrays and returning JSON objects. Hence, we needed to search online for resources to build more complex PLPGSQL queries.

One lesson learned is that when a team is tasked to design a database, it is first important to engage the business stakeholders to gain a clear understanding of the application domain; and to consistently engage with them to clarify expected behaviours. If this is not done, then the team might possibly spend a lot of time developing the solution with certain design considerations in mind, which might later require major modifications as their understanding of the requirements changes.

Another lesson learned is that good communication is important for group work in databases, as it is not always easy to separate concerns and work independently.


payment_date
num_work_hours
amount
num_work_days
Pay_slips
For
cust_id
address
name
CVV
phone
email
date
refund_amt
rid
location
seating_capacity
number
Customers
Credit_cards
Cancels
package_credit
Rooms
eid
name
address
phone
Employees
email
expiry_date
Owns
from_date
date
Registers
start_time
Conducts
ISA
depart_date
join_date
Instructors
end_time
Sessions
Buys
Redeems
ISA
ISA
sale_start_date
date
date
sid
Part_time_instructors
Part_time_Emp
package_id
Course_packages
num_remaining_redemptions
date
Specializes
ISA
hourly_rate
Full_time_instructors
num_free_registrations
Consists
course_id
duration
title
Course_areas
Full_time_Emp
sale_end_date
name
price
description
Courses
In
name
ISA
monthly_salary
Has
launch_date
Manages
Offerings
Handles
Administrators
Managers
end_date
start_date
target_number_registrations
fees
registration_deadline
seating_capacity

5 Application Constraints Not Captured By ER Diagram

1. Part-time instructors cannot teach more than 30 hours each month.

2. Registration deadline for course offering is at least 10 days before the start date.

3. Constraints on the earliest and latest timings that sessions can be conducted, as well as certain timings when they cannot occur (ie. 12pm-2pm).

4. No two sessions for the same course offering can be conducted on the same day and at the same time.

5. Each instructor must not be assigned to teach two consecutive sessions.

Non-Trivial Design Decisions in Schema

1. Our schema has a separate table for Specializes, rather than combining it with the Instructors table. This is because it is possible that an Instructor can specialize in 0 or more course areas. The approach we chose ensures that the instructor table only has one record per Instructor (where the instructor id is the only attribute that acts as the primary key), while the Specializes table can have 0 or more records per Instructor (the instructor id and course area is the composite primary key). The alternative decision involving a combined table (the instructor id and course area is the composite primary key of the Instructors table) would result in multiple records per instructor, possibly increasing the risk of duplicates when this table is queried by the user but the result is incorrectly handled.

2. Total participation constraint of Credit cards is enforced using triggers instead of being an attribute of Customers. Instead of only keeping track of the latest credit card that the customer owns, we keep tracking of every credit card he/she owns. Also, this is critical since the owner might want to retrieve information that requires past credit cards such as all packages a customer has purchased. Hence, the Credit cards and Customers are better separated and the constraint better enforced using multiple triggers.

3. Almost all identifiers have type SERIAL as the identifiers are automatically incremented for every new entry into the table. However, it is not possible for Sessions as the identifier is sequential for each course offering instead of sequential for all sessions. Thus, a raw integer is used for the session identifier.

Constraints that are Not Enforced by Relational Schema

1. Total Participation Constraint between Credit_cards and Owns.

2. Key Participation Constraint between Credit_cards and Owns.

3. A customer can have at most one active or partially active package.

4. Maximum capacity of a session is limited by the room capacity.

5. ISA Constraint between Employees, Part_time_Emp, Full_time_Emp, Instructors, Part_time_Instructors, Full_time_Instructors, Administrators and Managers.

Non-Trivial Design Decisions (Miscellaneous)

1. For the functions where certain course sessions need to be identified (e.g. function 21), an extra parameter for launch date is added because the course identifier and session number alone are insufficient to identify a particular session. For example, if the same Course has two offerings running concurrently, where each offering has a session with the label 1, the launch date will be needed to uniquely identify the session that is of interest.

2. There are triggers to prevent update or delete on multiple tables such as Owns. For Owns, we decided to restrict updates and deletes to prevent unintended consequences such as loss or mismatch of critical information. Hence, if there are changes to the credit card a customer owns, a new entry to be inserted into the Owns table.

3 Most Interesting Triggers

1. **employee_ftpt_constraint_trigger**. This trigger is used to enforce the covering constraint (must be satisfied) where a full-time employee must be either a manager, administrator or full-time instructor, as well as ensure that an employee only belongs to exactly one of these subclasses (overlap constraint must not be satisfied).

   The design of employee_ftpt_constraint_trigger is such that we first check whether the covering constraint is true after insertions or updates to the Employees table, by deferring the constraint checking to right before the transaction commit. If it is not (the employee is not in any of the three tables Administrators, Managers, Full_time_instructors after the insertion or update), an exception is raised and the transaction fails. Then, the overlap constraint is checked by checking whether combinations of 2 of the 3 subclass tables contain a record for the employee, in which case the transaction fails.

2. **registered_redeemed_before_cancel_trigger**. This trigger serves multiple purposes. Firstly, it checks if the session had been registered or redeemed by the customer. If it had not been registered or redeemed, the trigger would not allow the new row to be added into the Cancels table. This would prevent illegal cancellation and unintended refund of fees or package's session slots. Secondly, it would check if the customer is cancelling a session paid by credit card or redeemed using his/her package. If the session was paid by credit card, the customer would only be refunded 90% of the course fees if it is at least 7 days before the session date. If the session was redeemed, the customer would only be refunded one package's session slot if it is at least 7 days before the session date.

   The trigger will always be activated before every insert on Cancels. Instead of trusting the data input, the trigger ensures that the new row will always be correct as per the design requirements. In addition, as Cancels deals with critical parts of the system, such as the refunds, it is prudent to correctly verify every input into the table.

3. **sequential_sid_trigger**. This trigger, while simple, is an important one as it enforces the consecutive numbering of sessions for each course offering. It works by retrieving the current maximum session number from each course offering, and numbering the new session to be inserted into the table the current maximum session number of its corresponding course offering incremented by 1. This was used in place of the Serial data type, which was our initial design to enforce this consecutive numbering and which we used as the data type for most of our IDs; however, the Serial data type could not enforce the consecutive numbering of sessions *starting from 1* for every course offering as it would just add on to the numbering regardless of whether the newly added session is the first from its course offering.