CS2102 REPORT (PART I)

Team 50

Team Member:

He Tianyu A0177829J
Liu Yiqiu A0194516X
Song Qifeng A0194563W
Zhang Yuhao A0194564U

Project Responsibilities

1. He Tianyu:
    a. Schema: customers, credit_cards, course_packages, cancels
    b. Routines: 3. add_customer, 4. update_credit_card,
       11. add_course_package, 14. get_my_course_package, 17. register_session,
       21. update_instructor, 22. update_room, 26. promote_courses
    c. Debug/Modify: owns, buys, redeems, registers, get_available_course_package

2. Liu Yiqiu:
    a. Routines: 12. get_available_course_package,
       13. buy_course_package, 23. remove_session
    b. Debug/Modify: course_packages

3. Song Qifeng:
    a. Schema: employees, part_time_emp, full_time_emp, instructors,
       full_time_instructors, part_time_instructors, managers, administrators,
       course_areas, specializes, pay_slips, owns, buys, redeems, registers
    b. Routines: 1. add_employees, 2. remove_employee, 6. find_instructors,
       7. get_available_instructors, 16. get_available_course_sessions,
       19. update_course_session, 20. cancel_registration, 25. pay_salary,
       30. view_manager_report
    c. Debug/Modify: buy_course_package

4. Zhang Yuhao:
    a. Schema: courses, offerings, course_areas, sessions, rooms, specializes
    b. Routines: 5. add_course, 8. find_rooms, 9. get_available_rooms,
       10. add_course_offering,
       15. get_available_course_offerings, 18. get_my_registrations,
       24. add_session, 27. top_packages, 28. popular_courses,
       29. view_summary_report
    c. Debug/Modify: courses, offerings, sessions, registers, buys, redeems

ER Model

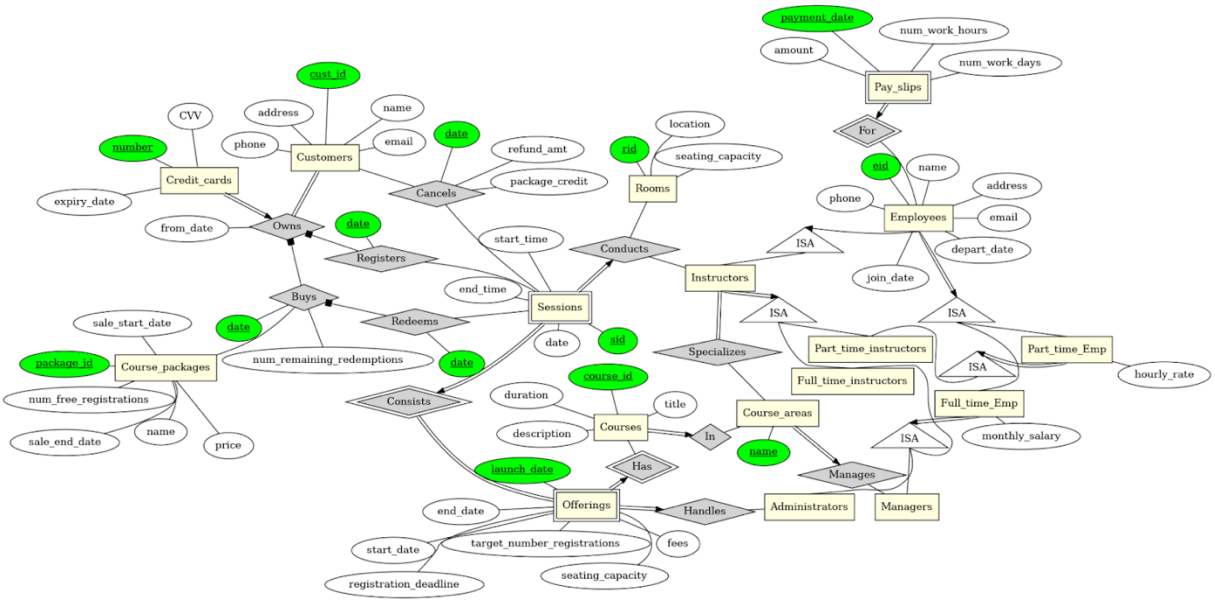

Figure 1 Schema

We used the suggested ER diagram.

5 of the application's constraints that are not captured by your ER model

1. No two sessions for the same course offering can be conducted on the same day and at the same time.
2. The instructor assigned to teach a course session must be specialized in that course area.
3. Each instructor can teach at most one course session at any hour.
4. Each instructor must not be assigned to teach two consecutive course sessions.
5. The seating capacity of a course session is equal to the seating capacity of the room where the session is conducted.

Relational Database Schema

- For all character attributes in the schema that we can estimate their length roughly, we try to use VARCHAR instead of TEXT to improve the searching efficiency.
- In *employees*, we enforce a constraint that the join_date of an employee must be earlier than the depart_date to avoid invalid input.
- *part_time_emp* & *full_time_emp* are subclasses of *employees*, thus they inherit eid from *employees*, and propagate any delete/ update to *employees*. Two positive value constraints are enforced to ensure that employees' salaries are some positive numbers: hourly_rate > 0 & monthly_salary > 0.
- *instructors* is a subclass of *employees* as well, so its eid references employees, and propagate any delete/ update to *employees.*
- Similiarly, *part_time_instructors* is a subclass of *part_time_emp*; *full_time_instructors, administrators & managers* are subclasses of *full_time_emp.*
- *part_time_emp* are all *part_time_instructors*, the non-overlap constraint is enforced by its own, since *part_time_emp* has only one subclass that is *part_time_instructors.*
- In *course_areas*, we need to ensure that each course area is managed by at least and at most one manager. Thus, Manager_eid references *managers* and should be not null. This is to satisfy the total participation constraint. The key constraint is enforced naturally by the primary key here.
- Since each instructor specializes in one or more course areas, the primary key is (eid, name) in *specializes.* Also, we note that *instructors* must have total participation in *course_areas*, and this constraint is not really enforced by the primary key of *specializes*.
- *pay_slips* is a weak entity set, whose corresponding owner entity is *employees*, so the primary key is (eid, payment_date), and the total participation constraint (of the weak entity set) is enforced by the primary key it own. amount, num_work_hours & num_work_days are all set to be not null and non-negative; moreover, num_work_days should be within the range [0, 31], since there are at most 31 days in a month.
- In the table *courses*, course_id is expressed as an integer (i.e. the first course is numbered as 1, and the second is 2). The duration is counted in hours and is expressed as an integer. Each *courses* is in exactly one *course_areas*, which implies two constraints: key and total participation. The key constraint is enforced by the primary key. And the total participation constraint is satisfied since area is not null.
- The *offerings* is a weak entity set, whose corresponding owner entity is *courses*. This is implemented by setting (course_id, launch_date) as the primary key. And the total participation constraint implied by the weak entity relationship is enforced by the primary key it own. The deadline check constraint is to check if the registration deadline of this course offering is at least 10 days before its start date. Each course offering is handled by an administrator, which implies two constraints: key and total participation. The key constraint is enforced by the primary key. And the total participation constraint is satisfied since administrator is an not-null integer.
- In *rooms*, location attribute is several number characters, expressed in floor and room numbers.
- *sessions* is a weak entity set, whose corresponding owner entity is *offerings*. This is implemented by setting (sid, course_id, launch_date) as the primary key. And the total

participation constraint implied by the weak entity relationship is enforced by the primary key it own. Based on the description, we assume each session starts and ends on the hour, so start_time and end_time are both set to be integers. Furthermore, the starting hour and ending hour must be within the range [9,12] or [14,18], and this is enforced by a constraint. Each session is conducted in exactly one room, and has exactly one instructor. These two total participation constraints are enforced by not-null rid and not-null instructor. On the other hand, two key constraints are enforced by the primary key.

- In *customers*, similar to other tables, its primary key *cid* is declared as a unique integer defined on insertion starting from 1. Other attributes are declared as type text because of uncertain length of the attributes. There is no attribute for credit card number. Instead, credit card information is maintained in another table *credit_cards* and relationship *owns*. When adding a new customer with a card number, the procedure *add_customer* inserts into two tables (*customers* and *credit_cards*).
- *owns* maintains the relationship between *customers* and *credit_cards*. The *add_customer* procedure does not insert into *owns*. Instead, such operation is done via trigger *owns_log_trigger* that inserts into *owns* on each insertion of *credit_cards*..
- In *credit_cards*, its primary key *card_number* is not of type integer because the maximum value of integers is 2,147,483,647, which is not long enough to hold a card number. Instead, we use the type text. Attribute *cvv* of the table is declared to be an integer because it is a three-digit integer. The attribute *from_date* is not a fixed value, it is updated to the current date each time the credit card information is updated via procedure *update_credit_card*.
- In table *course_packages*, for its two date attributes, *sale_start_date* must be strictly smaller than *sale_end_date*, which is enforced by the schema. Float attribute *price* must be non-negative, which is enforced similarly. For integer *num_free_registrations*, no constraint is enforced in the schema.
- The relationship *buys* does not make references to *owns* but directly makes references to *customers* and *credit_cards* separately through their primary keys *cust_id* and *card_number*. On successful purchase of a course package, the attribute *num_remaining_redemptions* of the inserted *buys* record is copied from *num_free_registrations* from the purchased course package and the attribute is constrained to be non-negative, which is enforced in the schema. Attribute *date* is assigned on insertion to the date that purchase happens on. Each purchase is identified through the customer, credit card, the package bought and the date and therefore, the primary key is the tuple *(cust_id, card_number, package.id, date)*.
- Unlike *buys*, *registers* makes reference to *owns* instead of *customers* and *credit_cards* separately. *Registers* is a relationship between *owns* and *sessions*. Other than the attributes from those tables, it also stores *date* of the registration, which is also part of its primary key.
- *redeems* is a relationship between *sessions* and *buys*. Other than the attributes to identify referenced *sessions* and *buys* records, it also keeps *date* as the date of the redemption, which is also part of its primary key. All attributes inside *registers* are constrained to be *not null.*
- *cancels* is a relationship between *customers* and *sessions*, and is identified by the customer, session and its own *date*. Other than these attributes, it also keeps necessary attributes for the cancel operation: float number *refund_amt* and integer *package_credit*.

5 of The Application's Constraints that Are Not Enforced by Relational Schema

- Each instructor specializes in one or more course areas, this is not enforced be the data schema, which will be enforced by the procedure add_employees().
- Each *employees* is either a *part_time_emp* or *full_time_emp*, the covering constraint and the non-overlap constraint are not enforced by the schema. The non-overlap constraint is enforced by the trigger "if_full_then_not_part_trigger" & "if_part_then_not_full_trigger". The covering constraint will be enforced by the procedure add_employees().
- *part_time_emp* are all *part_time_instructors*, the covering constraint is not enforced by the schema. The covering constraint will be enforced by the procedure add_employees().
- *full_time_emp* is one of the followings: *full_time_instructors, administrators & managers.* The covering constraint and the non-overlap constraint are not enforced by the schema. The non-overlap constraint is enforced by the trigger "is_only_instructor_trigger", "is_only_admin_trigger" & "is_only_manager_trigger". The covering constraint will be enforced by the procedure add_employees().
- Each *instructors* is either full-time or part-time, which is not enforced by the schema. The non-overlap constraint is enforced by the trigger "if_full_then_not_part_trigger" & "if_part_then_not_full_trigger". The covering constraint will be enforced by the procedure add_employees().

Three Most Interesting Triggers

Name of The Triggers:

1. pay_slips_trigger
2. if_full_then_not_part_trigger
3. is_only_instructor_trigger

Usage of The Triggers:

- Pay_slips_trigger checks whether the amount of salary is too little (negative) or too much (>100000), because both kinds of amounts are unlikely to be correct. The trigger will refuse such amount to be inserted into pay_slips table.

- If_full_not_part_trigger checks when an employee is being inserted into part_time_emp table, the employee is not in the full_time_emp table concurrently, if so, the trigger will refuse this insertion.

- Is_only_instructor checks that an employee is not also an administrator or a manager before the employee is inserted into the instructor table.

Justification:

- For an education company, the salary of an employee definitely cannot be negative, and it is also unlikely to be more than $100,000 be it a full-time or part-time employee.

- Full-time employees and part-time employees are mutually exclusive, an employee should not be both full-time and part-time. Similarly, an employee should not have multiple roles among instructor, administrator, and manager.

Summary of Difficulties & Lessons Learnt

- We found that only the owner of a table can alter the table. When implementing procedures, we sometimes need to alter/drop a table, and this appears to be quite inconvenient if the developer of the procedure is not the owner of the table.

- It was not easy to generate data for all tables because of the constraints, triggers, and references all over the schema.

- The system language of my computer is Chinese and hence the error messages are in Chinese, making it difficult to search for the solution on the internet.

- I used DBeaver to develop procedures at first, and DBeaver keeps throwing syntax errors (e.g. 'not terminated dollar quote') even when the code ran successfully on the psql server.

- Due to some unknown reasons (perhaps due to the difference between SQL and PostgreSQL or the different versions of PostgreSQL), the solutions I found on the internet cannot run on the psql server.

- Testing the procedures/functions is difficult because we need to check through all involved tables to make sure the expected result is achieved.

- Different team members were in charge of different parts of schema and procedures, therefore when one involves tables/procedures written by another member, one needs to be aware of how the other member implemented the code. Especially for procedures and functions, all members must be interpreting the descriptions in the same way.

- The variable names declared inside a function, the input names, and the column names should be different, otherwise, it would take extra work to specify the exact object I want.
- Sometimes, more than one function is involved in one single function. It makes the debugging process more challenging. It takes a long time to figure out which function the problem is on.

- Some knowledge is not covered in class(how to create arrays, some type conversion issues, …). But I think it is a good way to learn.

- It is worth noticing that the 'tab' key which we were using quite often to represent space when we code can not be detected in Postgres, which actively demonstrates the fact that avoiding the use of tab and replacing it with proper spaces are essential and necessary for coding in Postgres if a clear and clean layout is needed.

- We have divided the creation of tables and functions/procedures into different sections and assign them to individuals. This brings the problem that we do not have the permission to directly modify other team members' tables or functions/procedures. When I intend to debug other team members' work, I have to contact them to drop the table

first. An alternative way is to upload the modified version to shared Github space and wait for them to change as follows.

- Postgres is punctuation syntax sensitive. For example, the smart quotes ‘ and ‘ are not suitable for Postgres whereas straight quotes are only accepted.

- Once the schema tables are created with specified columns and subsequent procedures and functions which depend on such tables are developed, it is extremely hard to change the tables due to their wide and huge impact.

END OF FILE