Project Report

Team 57

Amelia Regina Sutikna (A0200711A)

Fidella Widjojo (A0200673N)

Hubertus Adhy Pratama Setiawan (A0200816R)

Jusuf Nathanael (A0200810A)

1. **Roles and Responsibility**

   Amelia Regina Sutikna:

   1. Write routine 1 - 10
   2. Write trigger:
      a. offering_deadline
   3. Write report

   Fidella Widjojo:

   1. Write routine 16 - 25
   2. Write triggers:
      a. can_redeems, can_registers, can_cancels
      b. update_redeems, update_registers
      c. cant_update_redeems, cant_update_registers
      d. update_instructor, update_room
      e. insert_session, delete_session
      f. insert_payslip
      g. cant_update_delete_cancels, cant_update_delete_payslip
      h. update_offering_old, update_offering_new
   3. Write report

   Hubertus Adhy Pratama Setiawan:

   1. Translate ER model to relational database schema:
      a. Customers, Credit_Cards
      b. Course_packags, Buys
      c. Registers, Cancels, Redeems
      d. Courses, Course_offerings, Sessions
   2. Create synthetic application data
   3. Write routine 26 - 30
   4. Write triggers:
      a. update_delete_employees
      b. check_active_packages

5. Write report

Jusuf Nathanael:

1. Translate ER model to relational database schema:
    a. Employees, Part_time_employees, Full_time_employees
    b. Administrator, Managers, Instructors, Part_time_instructors, Full_time_instructors
    c. Rooms
    d. Pay_slips
    e. Specializes
2. Write routine 11 - 15
3. Write triggers:
    a. check_manager_role_trigger, check_administrator_role_trigger
    b. check_ft_instructor_role_trigger, check_pt_instructor_role_trigger
    c. check_employees_insertion_trigger, check_instructors_insertion_trigger
    d. check_ft_employees_insertion_trigger, check_pt_employees_insertion_trigger

2. ER Data Model

Our team decided to use the suggested ER data model for this project:

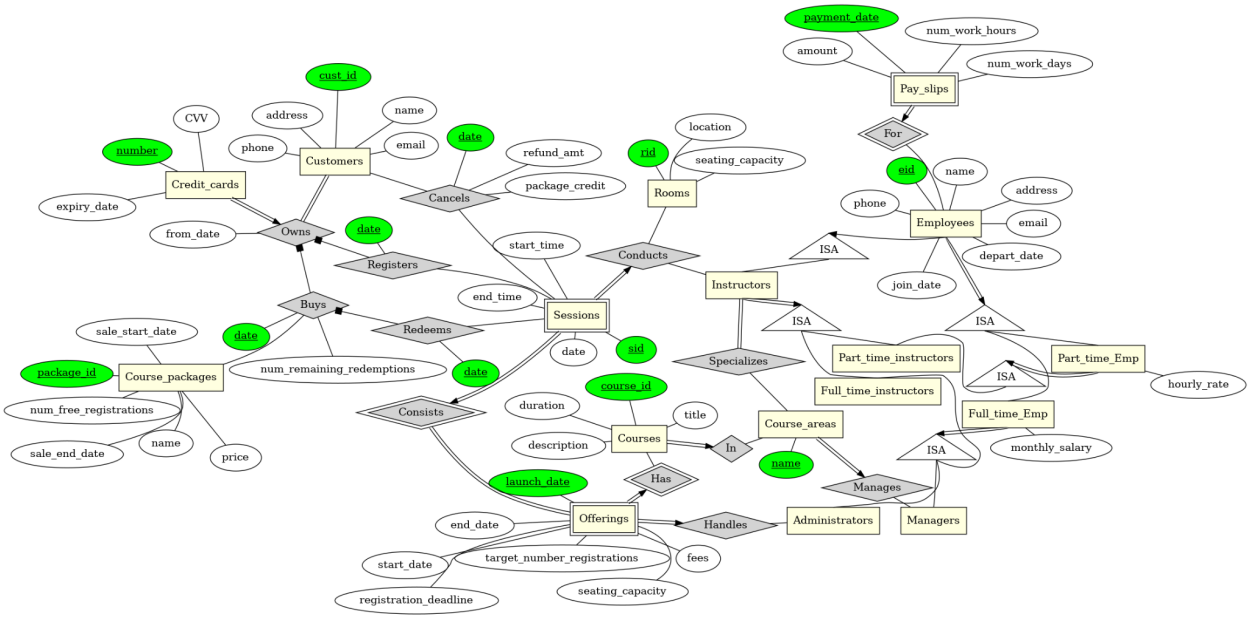

Figure 1: ER data model

a. Justification for non-trivial design

Not applicable as the suggested ER data model was used for this project.

b. Five application constraints that are not enforced by the ER model

i. The earliest session can start at 9 am and the latest session (for each day) must end by 6 pm, and no sessions are conducted between 12 pm to 2 pm.

ii. Each instructor must not be assigned to teach two consecutive course sessions; i.e., there must be at least one hour of break between any two course sessions that the instructor is teaching.

iii. Each part-time instructor must not teach more than 30 hours for each month.

iv. An instructor who is assigned to teach a course session must be specialized in that course area

v. The number of workdays for the month is given by (last workday - first workday + 1). The first workday is equal to the day of the employee's

joined date if the employee's joined date is within the month of payment; otherwise, it is equal to 1. The last workday is equal to the day of the employee's departed date if the employee's departed date is within the month of payment; otherwise, it is equal to the number of days in the month.

3. Relational Database Schema

a. Justification for non-trivial design

i. In the Owns relationship (Credit_cards table in the schema), (card_number) was chosen to be the primary key instead of (cust_id, card_number) on the assumption that each customer can only have one credit card at any point in time. Therefore, the card_number is sufficient to uniquely identify the customer.

ii. For the full-time and the part-time instructors, the multiple inheritances of a full-time/part-time instructor being both a full-time/part-time employee and an instructor was modeled and enforced on the schema level. Initially, our team attempted to model this using the answer from Tutorial 3 no 2b:

```
CREATE TABLE Full_Time_Instructors (
   eid integer,
   FOREIGN KEY (eid) REFERENCES Full_Time_Employees
         REFERENCES Instructors
         ON DELETE CASCADE
);
```

But, PostgreSQL marked this as syntactically incorrect. This problem was fixed by changing the schema to the following:

```
CREATE TABLE Full_Time_Instructors (
   eid integer,
   eid_emp integer,
   FOREIGN KEY (eid_emp) REFERENCES Full_Time_Employees (eid)
         ON DELETE CASCADE,
   FOREIGN KEY (eid) REFERENCES Instructors ON DELETE CASCADE,
   PRIMARY KEY (eid),
   CHECK (eid = eid_emp)
);
```

b. Five application constraints that are not enforced by the schema

i. The seating capacity of a course offering is equal to the sum of the seating capacities of its sessions.

ii. Session numbers: sessions can be deleted resulting in not consecutive session numbers.

iii. Each employee in the company is either a manager, an administrator, or an instructor. An employee can't be both part-time and full-time.

iv. Each course offering has a start date and an end date that is determined by the dates of its earliest and latest sessions.

v. For a credit card payment, the company's cancellation policy will refund 90% of the paid fees for a registered course if the cancellation is made at least 7 days before the day of the registered session; otherwise, there will be no refund for a late cancellation.

4. **Triggers (three most interesting)**
    a. **Name:** insert_payslip

    Usage: insert_payslip → before insertions on Pay_slips

    Justification: This trigger is interesting because it is complex and has to firstly check whether the employee is full-time or part-time. If the employee is full-time, then it needs to determine again the number of days worked based on join date and departure date to determine their pay from the monthly salary. If the employee is part-time, then it needs to determine the total hours worked to determine their pay from the hourly rate.

    b. **Name:** can_registers/can_redeem

    Usage: can_registers → before insertions on the Registers table

    can_redeems → before insertions on the Redeems table

    Justification: These two triggers function in a similar manner, just on two different tables and using two different payment methods (redeem uses course package, while register uses credit card). They both first check whether there are empty spots available in the session that the customer wants to register for. Then they also ensure that the customer has not registered for any other session from this same course before being able to register them in the course.

    c. **Name:** update_offering_old/update_offering_new

    Usage: update_offering_new → after insertions or updates on the Sessions table

    update_offering_old → after deletes or updates on the Sessions table

    Justification: The start date, end date, and seating capacity columns of the Offerings table depends on the information of the sessions it currently has. After an insertion/update/delete on the Session table, the seating capacity of the offering may increase or decrease and the start/end date of the offering may change.

5. Summary

a. Difficulties encountered

i. It was very hard to ensure that all application constraints were satisfied. As users are assumed to be able to do ad-hoc SQL queries without using the routines, numerous triggers and checks must be implemented to ensure the validity of the data. Some of these proved to be not trivial to implement.

ii. Generating mock data for testing the application turned out to be non-trivial. The vast application constraints forced us to manually generate the data as it is very hard for our team to come up with the script to automate this task given the limited time and knowledge.

iii. Our team decided to first create all the schema, create the data, and then start working on the routines and triggers. Having mock data was believed to ease the process of testing and debugging. After all the routines and triggers are put in place, all the INSERT statements in data.sql become unusable and new data must be generated from scratch. For example, one of the triggers forces the user to use add_employees when inserting an employee to the table and will throw an exception that a type (manager/administrator/instructor) must be specified when they try to use a regular INSERT statement.

iv. After the mock data and the triggers were finished, some data which are crucial for testing cannot be inserted into the database due to the application constraints. For example, routine no. 29 (view_summary_report) requires the transaction data from the last N months. But, a session cannot be inserted when the course offering's registration deadline has passed.

v. Unfamiliarity with SQL/PLpgSQL was a major obstacle for our team to be able to implement the routines and the triggers quickly and efficiently. Some of the routines took our team more than 10 hours to write, test, and debug.

vi. Our team had difficulties in managing time and splitting the tasks. We underestimated the amount of work we had to do and started working on this project very late.

b. Lessons learned

This project allowed us to be able to understand and appreciate the complexity of a real-world database application. Having done the process ourselves, we now understand how difficult it is to translate the application requirements from written text to ER, translate ER to SQL schema, implement and test the APIs, and create the mock data without violating any application constraints. Furthermore, it allowed our team to learn more about the SQL language; the syntax, the functions, and the beauty of it. Lastly, this project allowed us to learn how to better manage our time and how to work together in a team.