AY20/21 Semester 2 CS2102 Database Systems

Project Report

Lecturer: Dr. Chan Chee Yong & Dr. Xiao Xiaokui

Project Team Number: 44

Group Members:

Ben-Hanan Choong See Kaay (A0206153R)

Hans Sebastian Tirtaputra (A0199399X)

Isaac Tin Kah Ong (A0206151W)

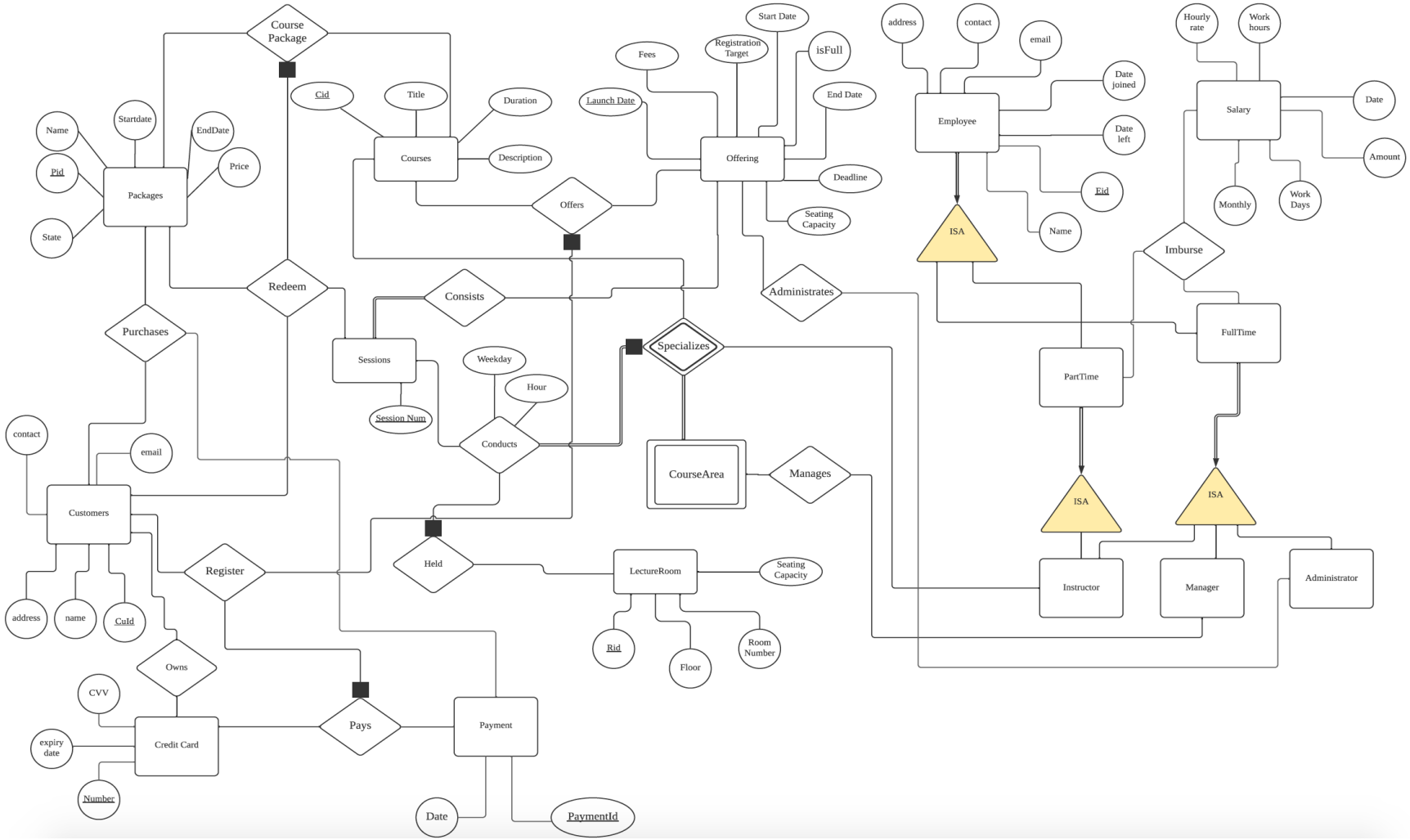
Royce Ho Shou Yee (A0199477A)

1. Project Responsibilities

Name	ER Model	Schema Creation	Triggers	Application Functionalities	Project Report
Ben-Hanan	✔	✔	✔	✔	✔
Hans	✔	✔	✔	✔	✔
Isaac	✔	✔	✔	✔	✔
Royce	✔	✔	✔	✔	✔

2. Entity-Relationship (ER) Data Model

Our team originally proposed the following ER data model for the database application.



2.1 Justifications for Non-Trivial Design Decisions in Entity-Relationship Model

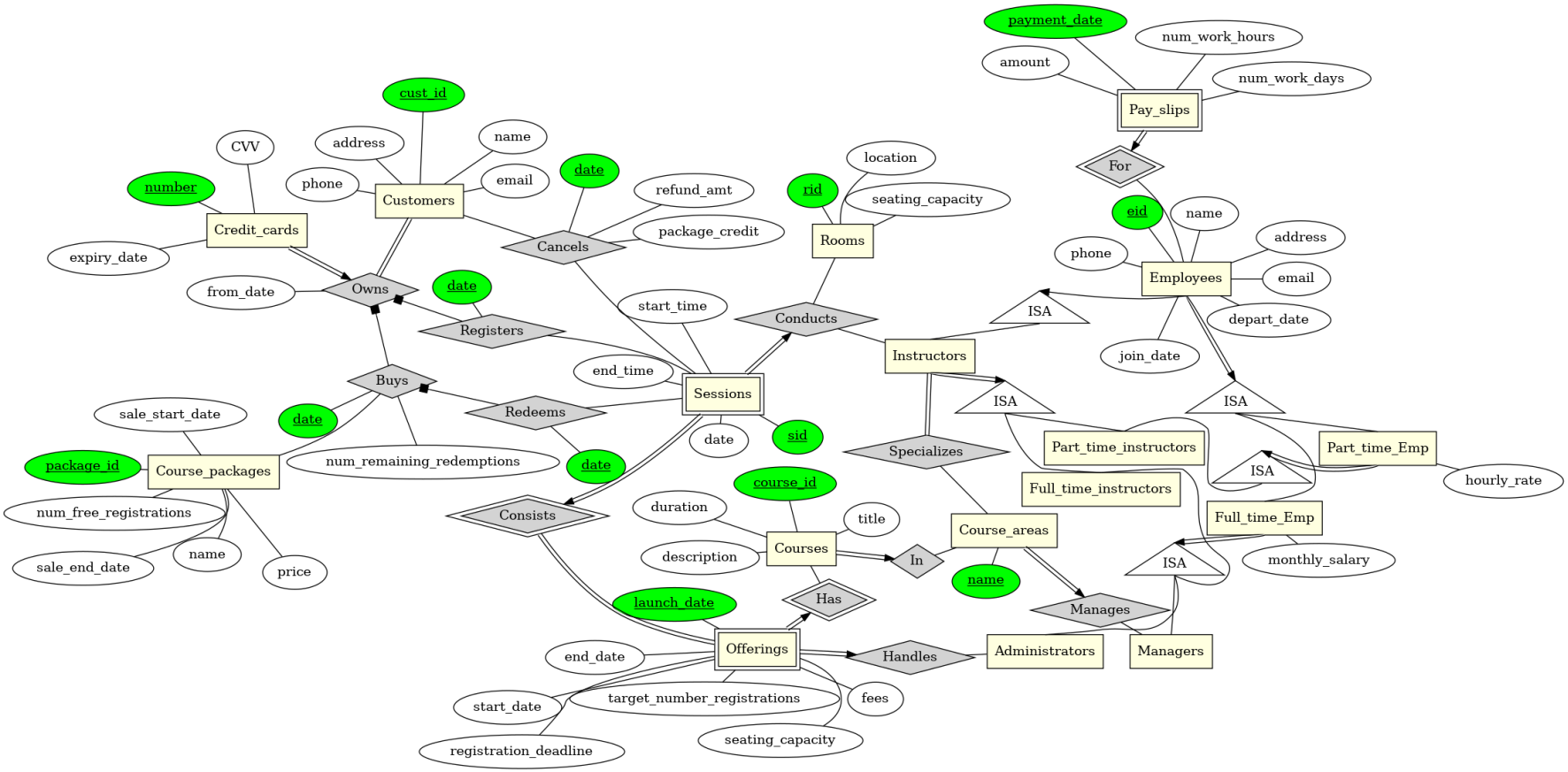
- Packages related to Courses through Course Package as a package needs to know which kind of course it belongs to.
    - Since the customers can redeem free sessions from packages, the relationship set redeem aggregates Course Packages to get packageid (Pid) and courseid (CID) and redeem is related to sessions, hence containing information of which package and which course the session is from.
- Register relation aggregates offers relation to get courseId and course offering Id, and aggregates pays relation to get credit card number and paymentid and lastly is related to customers.
    - Customers register for course, and fees are retrieved from course offering, and then payment is made via credit card number and that payment will be stored with a unique identifier paymentid in the case of refunds or cancellations in the future.
- Employee is parent class, with part-time and full-time as child classes. Child class of part time is instructor, while child classes of full-time are manager and Administrator
    - Employees can be either part-time or full-time employees, and hence ISA used as employees are either part-time or full-time.
    - Since only instructors can be part-time, ISA used here as instructors is part-time employee.
    - Instructor, manager and Administrator can all be full-time, and hence ISA used here as they can be full-time employees

2.2 Application Constraints not Captured by Entity-Relationship Model

1. For a redeemed course session, the company's cancellation policy will credit an extra course session to the customer's course package 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.

2. The registration deadline for a course offering must be at least 10 days before its start date.
3. Each part-time instructor must not teach more than 30 hours for each month.
4. The earliest session can start at 9am and the latest session (for each day) must end by 6pm, and no sessions are conducted between 12pm to 2pm.
5. 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.
6. Each full-time employee has a fixed monthly salary, while each part-time instructor has a fixed hourly rate and his/her monthly salary is computed based on the total number of hours worked for that month.

However, upon the realization of the various design limitations in our ER model, we decided to use the recommended ER Data Model provided by the Module. Hence the ER model we have decided to use is as below in the next page.


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

2.3 Justifications for Non-Trivial Design Decisions in new Entity-Relationship Model

This is copy pasted from given ER diagram (Referenced from provided ER diagram)

- **Credit card information** This design models `Credit_cards` as an entity (rather than as an attribute of `Customers`) using the `Owns` relationship to relate to `Customers`. The reason is that the customer's credit card information may change over time (by the routine `update_credit_card`). The current active credit card is determined by the `from_date` attribute of Owns. A registration paid with a credit card is modeled using the `Registers` relationship which is a binary relationship between `Sessions` & `Owns` which keeps track of the credit card used for the payment.

- **Derived attributes** Some of the data are derived attribute values (e.g., the start & end date of `Offerings`). Such data could either be explicitly stored (i.e., captured in the ER model) or computed at runtime (not captured in the ER model)

- **`Registers/Redeems/Cancels` Relationships** The `Registers`, `Redeems`, and `Cancels` relationships each has a date attribute to record the date of the transaction. By having this attribute as part of the key of the relationship, it supports multiple instances of the relationship to exist for the same set of relationship participants.

2.4 Application Constraints not Captured by new Entity-Relationship Model

Similar to the above application constraints explained in section 2.2, below are the application constraints not captured by the new entity-relationship model.

1. For a redeemed course session, the company's cancellation policy will credit an extra course session to the customer's course package 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.

2. The registration deadline for a course offering must be at least 10 days before its start date.
3. Each part-time instructor must not teach more than 30 hours for each month.
4. The earliest session can start at 9am and the latest session (for each day) must end by 6pm, and no sessions are conducted between 12pm to 2pm.
5. 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.
6. Each full-time employee has a fixed monthly salary, while each part-time instructor has a fixed hourly rate and his/her monthly salary is computed based on the total number of hours worked for that month.

3. Relational Database Schema

3.1 Justifications for Non-Trivial Design Decisions in Schema

1. Some tables such as `Course_packages`, `Rooms`, `Employees`, `Courses` and `Customers` used constraint `GENERATED AS IDENTITY` that allows us to automatically assign a unique number to a column. It is the SQL standard-conforming variant of the old `SERIAL` column. This was to ensure that the identity of each entry in the tables could be assigned a unique key that could identify it accurately without having to manually code to search for the unique identification number to assign to new entry, e.g: counting the total number of entries in the table, taking that count and adding 1 to it and then inserting it into the table as a new entry with the new calculated eid.

2. Representing `Owns` table in `Credit_cards` and `Customers` instead of having a standalone `Owns` table
    a. This design decision was made so that we can enforce the total participation and key constraint of `Credit_cards` with respect to `Owns`, as well as enforce the total participation constraint of `Customer` with respect to `Owns`. If we do not

combine the Owns table within the Credit_cards table, we would not be able to enforce the constraint given in the entity-relationship diagram.

3. Representing Conducts in CourseOfferingSessions instead of having a standalone Conducts table
    a. The design decision was made so that we can enforce the total participation and key constraint of Sessions (CourseOfferingSessions in our schema) with respect to Conducts. Without doing so, it would be impossible for us to guarantee that each session only has one instructor conducting it, and is conducted in only one room.

4. Not all the subclasses of Employees have foreign keys referenced to Employees. For e.g: Part_time_emp, Full_time_emp and Instructors eid references Employees as all employees that are part time or full time are employees. Instructors table was added because we wanted to keep track of all part time and full time instructors in a table. Managers, Administrators references Full_time_emp as they are both definitely full time, and Part_time_instructors references both Instructors and Part_time_emp and Full_time_instructors references both Instructors and Full_time_emp because they are both instructors and either full or part time employees. So this was made this way to make it easier and more accurate to identify employees on what type of employees they are.

5. Representation of instructor specialization. This was done only storing instructor eid in the Instructors table, and storing the course_area_name that the instructors specialized in within a separate Specializes table. This is so that we can ensure that each instructor has at least one or more specialization (total participation constraint). Taking into consideration that we are referencing an instructor's eid from CourseOfferingSessions, we need to uniquely identify the instructor. Thus

`course_area_name` is included in `Specializes` table so that each instructor can have more than one course area specialization, and yet maintain its unique eid.

3.2 Application Constraints not Enforced by Schema

1. Each customer can have at most one active or partially active package.
2. A room can only be used to conduct at most one course at any time.
3. Part-time instructors must not teach more than 30hours for each month.
4. Full-time employee cannot be a part time instructor.
5. Instructors can only teach at most one course session at any hour.

4. Three Most Interesting Triggers

1. `update_course_offering_seating_capacity`
    a. **Usage:** Since seating capacity of course offerings are dependent on seating capacity of all the sessions under course offering, after every insert or update into course offering sessions, this trigger will update the seating capacity of the course offering that session that was just added/updated is related to.
    b. **Design Justification:** There are constraints where the total sum of registrations for all course sessions under a course offering cannot exceed the seating capacity of the course offering. Hence, there is a need to ensure accuracy and reliability of course offering seating capacity at all times. The trigger was thus created to ensure two things - that the seating capacity of course offerings is updated whenever a new course session under the course offering is added, and that the seating capacity of course offerings is updated whenever a session's room is changed/updated. This is a very useful trigger as now, whenever course sessions are added or updated, there is no need to worry about updating the course

offering's seating capacity in the functions whenever course sessions are updated.

```
CREATE OR REPLACE FUNCTION update_course_offering_seating_capacity()
RETURNS TRIGGER AS $$
DECLARE
        newSum INT;
BEGIN
        SELECT SUM(R.seating_capacity)
        FROM Rooms R, CourseOfferingSessions C
        WHERE R.rid = C.rid GROUP BY (launch_date, course_id)
        HAVING NEW.launch_date = launch_date AND NEW.course_id = course_id
        INTO newSum;
        UPDATE CourseOfferings
        SET seating_capacity = newSum
        WHERE launch_date = NEW.launch_date AND course_id = NEW.course_id;
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_course_offering_seating_capacity
AFTER INSERT OR UPDATE ON CourseOfferingSessions
FOR EACH ROW
EXECUTE FUNCTION update_course_offering_seating_capacity();
```

2. `check_registers` & `check_redeems`
    a. **Usage:** The two triggers are combined as they work together to fulfill one purpose - to ensure that for each course offered by the company, a customer can register for at most one of its sessions before its registration deadline as we count redeems and registers both as registration.
    b. **Design Justification:** There are many functions that change the `Registers` and `Redeems` tables. As such, every time a function that has something to do with registration of a customer is run, there is a need to ensure that the constraint that a customer can register for at most one session of each course offered is fulfilled. Thus, these triggers were created to ensure that whenever there is an update or insertion into the `Registers` or `Redeems` tables, the sum of registers

and redeems for sessions that the customer has made for the particular course offering does not exceed 1.

```
CREATE OR REPLACE FUNCTION check_register()
RETURNS TRIGGER AS $$
DECLARE
   registrationDeadline DATE;
BEGIN
    SELECT registration_deadline FROM CourseOfferings WHERE launch_date = NEW.launch_date AND course_id = NEW.course_id INTO registrationDeadline;
  IF EXISTS (
        SELECT 1 FROM Registers
        WHERE NEW.launch_date = launch_date
        AND NEW.course_id = course_id
        AND NEW.cust_id = cust_id
    ) THEN
        RAISE EXCEPTION 'Cannot register for more than one session of same course offering';
        RETURN NULL;
    ELSIF (NEW.registers_date >= registrationDeadline) THEN
        RAISE EXCEPTION 'Must register before registration deadline';
        RETURN NULL;
    ELSIF EXISTS (
        SELECT 1 FROM Redeems
        WHERE NEW.cust_id = cust_id
        AND NEW.course_id = course_id
        AND NEW.launch_date = launch_date
    ) THEN
         RAISE EXCEPTION 'This course session of this course offering has already been redeemed by customer';
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_register
BEFORE INSERT ON Registers
FOR EACH ROW
EXECUTE FUNCTION check_register();

  /*check redeems*/

CREATE OR REPLACE FUNCTION check_redeems()
```

```
pp. RETURNS TRIGGER AS $$
qq. DECLARE
rr.     registrationDeadline DATE;
ss. BEGIN
tt.     SELECT registration_deadline FROM CourseOfferings WHERE launch_date =
    NEW.launch_date AND course_id = NEW.course_id INTO registrationDeadline;
uu.     IF EXISTS (
vv.         SELECT 1 FROM Registers
ww.             WHERE NEW.launch_date = launch_date
xx.         AND NEW.course_id = course_id
yy.         AND NEW.cust_id = cust_id
zz.     ) THEN
aaa.             RAISE EXCEPTION 'Cannot register for more than one session of
    same course offering';
bbb.             RETURN NULL;
ccc.       ELSIF (NEW.redeems_date >= registrationDeadline) THEN
ddd.             RAISE EXCEPTION 'Must register before registration deadline';
eee.             RETURN NULL;
fff.     ELSIF EXISTS (
ggg.             SELECT 1 FROM Redeems
hhh.             WHERE NEW.cust_id = cust_id
iii.         AND NEW.course_id = course_id
jjj.         AND NEW.launch_date = launch_date
kkk.       ) THEN
lll.          RAISE EXCEPTION 'This course session of this course offering has
    already been redeemed by customer';
mmm.             RETURN NULL;
nnn.       ELSE
ooo.             RETURN NEW;
ppp.       END IF;
qqq.    END;
rrr. $$ LANGUAGE plpgsql;
sss.
ttt. CREATE TRIGGER check_redeems
uuu.    BEFORE INSERT ON Redeems
vvv.    FOR EACH ROW
www.    EXECUTE FUNCTION check_redeems();
```

3. check_customer_active_packages

   a. **Usage:** Ensure that each customer can have at most one active or partially active package

b. **Design Justification:** Course packages are modified by many of our functions. In order to ensure the constraint that there can only be at most one active or partially active package, we either have to perform a check in every function that modifies the course packages, or we would have to add a trigger. Hence we decided to add `check_customer_active_packages` as a trigger to ensure that the constraint is kept. This is an interesting trigger as the trigger involved checks with many other tables such as the `Buys`, `Redeems`, and `CourseOfferingSessions` table. This is to appropriately determine the number of partially active packages for a customer. With many considerations to think about, it was definitely an interesting trigger to create.

```
CREATE OR REPLACE FUNCTION check_customer_active_packages() RETURNS
TRIGGER AS $$
DECLARE numActivePackages INT;
numPartiallyActivePackages INT;
BEGIN numActivePackages := (
    SELECT COUNT(*)
    FROM Buys
    WHERE NEW.cust_id = cust_id
          AND num_remaining_redemptions >= 1 -- At least one unused
session in the package
);
numPartiallyActivePackages := (
    SELECT COUNT(*)
    FROM Buys B1
    WHERE NEW.cust_id = B1.cust_id
          AND B1.num_remaining_redemptions = 0 -- All sessions in
package have been redeemed
        AND EXISTS(
            SELECT 1 -- 7 days before day of registration
            FROM (
                    Buys B2
                    JOIN Redeems R ON (
                        B2.buys_date = R.buys_date
                        AND B2.cust_id = R.cust_id
                        AND B2.number = R.number
```

```
                        AND B2.package_id = R.package_id
                    )
                )
                JOIN CourseOfferingSessions CS ON (
                    R.sid = CS.sid
                    AND R.course_id = CS.course_id
                    AND R.launch_date = CS.launch_date
                )
            WHERE B2.cust_id = B1.cust_id
                AND B2.package_id = B1.package_id
                AND B2.num_remaining_redemptions = 0
                  AND CS.session_date - CURRENT_DATE >= 7 -- At least
7 days to get refund
        )
);
RAISE NOTICE 'Active Packages: %',
numActivePackages;
RAISE NOTICE 'Partially Active Packages: %',
numPartiallyActivePackages;
/** At most one active or at most one partially active package */
IF (
    numActivePackages = 0
    AND numPartiallyActivePackages = 0
) THEN RETURN NEW;
ELSE RAISE EXCEPTION 'A customer can have at most one active or
partially active package.';
END IF;
END;
$$ LANGUAGE plpgsql;
```

5. BCNF or 3NF Decomposition of Tables

BCNF and 3NF decomposition of tables is not included in this submission of the report.

6. Summary of difficulties faced and Lessons Learnt

6.1 Problems faced

1) ER diagram design
    a) During the early phase of the project, when designing the ER diagram, we decided to assign different parts of the prompt to different people in a bid to divide and conquer. We spent the majority of our time working on the ER diagram and repeatedly making changes to it based on our individual interpretations of the question prompt. As a result, we had to spend time to change the many inconsistencies for our finalised ER diagram submission. After the consultation and review of our ER diagram, we decided to use the suggested design as we felt that there would be too many emergent issues and complications later into the project if we continued with our own design. All these problems were probably due to our inexperience in designing database systems.
2) Schema translation
    a) We had difficulties translating all of the constraints of the ER diagram to SQL simply using table constraints. We had difficulties ensuring total participation and key constraints. This was in part due to the lack of knowledge of triggers and we had to spend time figuring out whether we could either combine a table or place some constraints in the schema to achieve what we had to implement.
3) Collaboration on NUS SOC Postgres server
    a) It was difficult to do testing on the postgres server during the process of building our schema due to permission issues. For example, a table can only be removed by the user that created it. As such, we did a bulk of testing and building on our own local Pgadmin servers.

6.2 Lessons learnt

Besides the technical skills we picked up from working on this project, we also managed to practice soft skills like teamwork and communication. We learnt how to work together as a team to design a database system. In response to problem 1), We feel that our idea of ‘divide and conquer’ was good but we should have first established a standardised interpretation of the prompt so that all the members would be on the same page. We should have also regularly met to ensure standardisation instead of meeting towards the end when we had already diverged. We also learnt to not dwell so much on schema translation in the early stages of the project as many things that we thought were problems could actually be solved later on through other methods. Overall, the development of this application has given us a valuable and insightful first-hand experience in designing database systems and writing functions to query meaningful data and perform functions on the data.