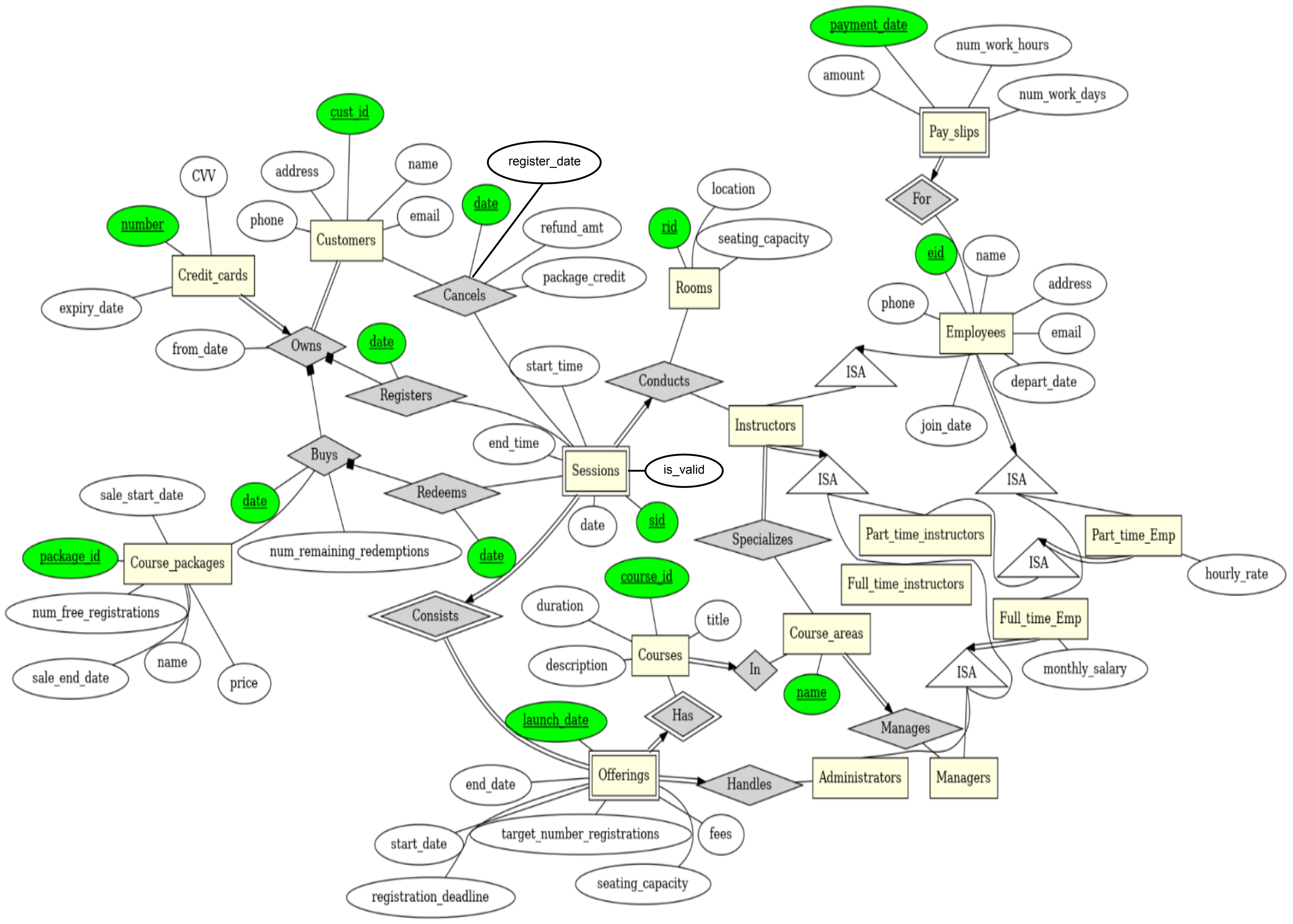
CS2102
Datebase Systems
Project Report

Team 99 Members：

Team Members	Matriculation Number
HU JIAJUN	A0209261L
XIE YAOREN	A0200116E
YAO YUMING	A0204722R
ZHANG ZHIYAO	A0209261L

Angeda


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

1. List of the project responsibilities

Hu Jiajun: SQL query, schema, testing data, triggers, PL/pgSQL routines, report editing

Xie Yaoren: SQL query, schema, testing data, triggers, PL/pgSQLroutines, report editing

Yao Yuming: SQL query, schema, testing data, triggers, PL/pgSQL routines, report editing

Zhang Zhiyao: SQL query, schema, testing data, triggers, PL/pgSQL routines, report editing

2. ER data model for our application and justification

A. ER model picture attached as page 2

B. Justifications for any non-trivial design decisions in the ER model.

i. Add cancellations table to record all the refund records. The registration records in Redeem and Register will be deleted after the customer requests a refund. Having a Cancel table allows us to record down the registration record, and to answer queries regarding refund value efficiently.

ii. For the Register, Redeem, and Cancels table, the date attribute is marked as part of the primary key because we want to allow each customer to register for the same session several times. Similarly, for the Buys table, the date attribute is marked as part of the primary key because we want to allow each customer to buy the same package several times.

iii. We model Conduct as a ternary relationship among Sessions, Rooms and Instructors. This is done so that a (instructor, room, session) pair would mean that that instructor conducts that session in that room. Note that when the ternary relationship is replaced by a collection of binary relationships, then the fact that instructor I teaches session S, and session S is in room R, and room R is used by instructor I does not necessarily mean that instructor I teaches session S in room R. Hence, here the ternary relationship design meets our needs.

iv. We add an is_valid attribute to Session because, to support the remove_session request, we do not delete the session record if there is at least one registration for the session. However, in the case where there is no registration or redeem record in the Register and Redeem table, if the session is registered then cancelled, we still prefer to keep track of the cancellation record. In that case, removing the session would lead to a violation of foreign key constraint in the Cancel table. Hence, we decided to add in an is_valid attribute and mark the session as invalid instead of deleting the record when we perform any valid remove_session request.

v. We maintain a register_date attribute in Cancels to keep track of the registration date of the session that is cancelled. This allows us to check whether a customer has registered for (possibly registered for then cancelled) some course offering in the last six months so that we can tell whether the customer is classified as an active customer. The attribute needs to be added to support this operation because for the current implementation, we would remove the record from Register and Redeem when the customer chooses to cancel that registration. Without the register_date attribute in Cancels, after removing the record from Register and Redeem we would no longer be able to tell when the customer register for a certain course offering and whether that registration is done within the last six months.

vi. The seating capacity for a course offering is a derived attribute that can be calculated from the sum of the seating capacity of all the course sessions under that course offering.

vii. We model Pay_slips as a weak entity for employees because a payslip should not exist by itself without belonging to some employee.

viii. There is a total and key participation constraint on Credit_cards with respect to Owns because each credit card should be owned by exactly one person. There is a total participation constraint on Customers with respect to Owns because the application requires the Customers to have at least one credit card.

ix. There is an ISA relationship between Employee and (Part_time_Emp, Full_time_Emp) that satisfies the covering constraint and does not satisfy the overlap constraint because each employee is either a part time employee or a full time employee. It can not be composed of more than one type, such as both part time employee and full time employee or be none of the type.

x. There is an ISA relationship between Full_time_Emp and (Administrators, Managers) that satisfies the covering constraint and does not satisfy the overlap constraint because each full time employee is either an administrator or a manager. It can not be composed of more than one type, such as both an administrator and a manager or be none of the type (cannot be instructors since all instructors are part time employees).

xi. The duration in the course package is a derived attribute as it can be calculated from start_date and end_date. Similarly, the salary amount in pay is also a derived attribute.

xii. There is a many to one relationship from session (weak entity) to course_offerings (owner entity), since session can only be uniquely identified considering the primary key of course_offerings. There is a total participation constraint on course_offerings with respect to the Consists relationship because each course offering consists of one or more sessions.

xiii. There is a many to one relationship from course_offerings (weak entity) to course (owner entity), since course_offerings can only be uniquely identified considering the primary key of course.

C. List down 5 of the application's constraints that are not captured by your ER model.

i. 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.

ii. The registration deadline for a course offering must be at least 10 days before its start date.

iii. The sessions for a course offering are numbered consecutively starting from 1.

iv. Each instructor can teach at most one course session at any hour. 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.

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

3. Relational database schema:

A. Non-trivial design decisions in the relational database schema

i. We combine “session” entity and “conduct” relationship and “consists” relationship into table. This is because each session is conducted in one room and has one instructor. However, each room can be conducted by many sessions and each instructor can conduct many sessions. Thus,there is a one-to-many relationship between “session” and “room” as well as “session” and “instructors”. Thus, “conduct” will take the primary of “session” as its primary key and “room” primary key and “instructor” primary key as a foreign key. Then we further combine “session” and “conduct” into one single table and continue to use “session” primary key and has a copy of “room” primary key and “instructor” primary key as the foreign key. Similarly for the “Consists” relationship.

ii. We combine “Offerings” and “has” relationships and the “handles” relationship into one single table. This is because each course has many offerings and each offering belongs to one single course. Thus, there is a one-to-many relationship between Offerings and Courses. Thus, “has” relationship will have “offerings” primary key as its primary key and “course” primary as its foreign key. Thus, we can combine “offerings” and “has” into one single table and the primary key of the resultant table is still the primary key of “offerings”. Similarly for the “handles” relationship.

iii. We combined “Pay slip ” and “for ” relationships into one single table. This is because each employee has many payslip and each payslip belongs to one single employee. Thus, there is a one-to-many relationship between “Payslip” and “Employee”. Thus, “for” relationship will have “Payslip” primary key as its primary key and “employee” primary as its foreign key. Thus, we can combine “payslip” and “for” into one single table and the primary key of the resultant table is still the primary key of “payslip”.

iv. We have a check constraint for payment_date in Pay_slips_for table so as to ensure that the payment_date is a valid payment date. This can ensure data accuracy.

v. We have a check constraints for date in Sessions_Conducts_Consists table to ensure that the session date is a valid date as the session can only be conducted on weekdays and only from 09:00 to 12:00 and 14:00 to 18:00.

vi. We have a check constraints for date in Offerings_Has_Handles table to ensure that the registration deadline for a course offering must be at least 10 days before its start date.

vii. We have a check constraints for date in Offerings_Has_Handles table to ensure that the seating capacity of the course offering must be at least equal to the course offering's target number of registrations.

viii. We have a check constraints for date in Cancels table to ensure that either the refund amount is not null or package_credit is not null. This is to prevent the invalid case that both are null or both are not null.

ix. We have a check constraints for num_work_hours and num_work_days in Pay_slips_For to ensure that either the values for the number of workdays for the month (for a part-time employee) or the values for number of work hours for the month (for a full-time employees) should be null.

x. We add the serial (integer auto increment) data type in the Employees table and Customers table and Course_packages table and Courses_In table to fulfill the requirement of identifiers generated by the system.

B. List down up to 5 of the application's constraints that are not enforced by your relational schema

i. "Each employee in the company is either a manager, an administrator, or an instructor", **it is impossible to set any constraints across** the **Employees Relational Schema, Managers Relational Schema, Administrators Relational Schema and Instructors Relational Schema** to enforce the manager tuple **can only** be inserted the **Managers Relational Schema,** the administrator tuple **can only** be inserted the **Administrators Relational Schema,** the instructor tuple **can only** be inserted the **Instructors Relational Schema.**

ii. "One customer can only register for at most one of the course offering sessions before its registration deadline", **it is impossible to set any constraints across** the **Register Relational Schema** and **Redeem Relational Schema** to enforce that one registered session tuple can only be inserted into either the **Register Relational Schema** or the **Redeem Relational Schema.**

iii. "The seating capacity of a course session is equal to the seating capacity of the room where the session is conducted, and the seating capacity of a course offering is equal to the sum of the seating capacities of its sessions", **it is impossible to set any constraints in the Offerings_Has_Handles Relational Schema** to enforce seating capacity of a course offering is equal to the sum of the seating capacities of its sessions since the seating capacity is dynamically changed based on the session tuples in the **Sessions_Conducts_Consists Relational Schema**.

iv. "Each instructor must not be assigned to teach two consecutive course sessions" **it is impossible to set any constraints in** the **Sessions_Conducts_Consists Relational Schema** to enforce that an instructor is not teaching two consecutive course sessions.

v. "An instructor who is assigned to teach a course session must be specialized in that course area" **it is impossible to set any constraints across** the **Sessions_Conducts_Consists Relational Schema** and the **Specializes Relational Scheme** to enforce that for every instructor with an eid teaching a session under certain course area in the **Sessions_Conducts_Consists Relational Schema**, there exists a corresponding record in the **Specializes Relational Schema**.

4. Three most interesting triggers implemented for the application.

A. Add a trigger in add_employee function

```
create or replace function add_part_time_instructor_on_addition_of_part_time_emp() returns trigger as $$
begin
    insert into Part_time_instructors values (NEW.eid);
    return null;
END;
$$ language plpgsql;

create trigger add_part_time_instructor_on_addition_of_part_time_emp_trigger
after insert on Part_time_Emp
for each row execute function add_part_time_instructor_on_addition_of_part_time_emp();
```

Name: add_part_time_instructor_on_addition_of_part_time_emp_trigger

Usage: This trigger allows us to achieve the functionality:
whenever there is a new tuple inserted into Part_time_Emp, we insert a tuple into Part_time_instructors to record the employee id.

Justification: This design allows us to ensure there is a consistency in the table. This can ensure data integrity among the tables.

B. Add a trigger in register_session

```
create or replace function decrement_num_remaining_redemptions() returns trigger as $$
begin
    update Buys B set num_remaining_redemptions = B.num_remaining_redemptions - 1
    where (B.date = NEW.buy_date and NEW.number = B.number and NEW.package_id = B.package_id);
    return null;
END;
$$ language plpgsql;

create trigger decrement_num_remaining_redemptions_trigger
after insert on Redeem
for each row execute function decrement_num_remaining_redemptions();
create or replace function register_session (in _cust_id int, in _course_id int, in _launch_date date,
                                               in _sid int, in _payment_method int)
```

Name: decrement_num_remaining_redemptions_trigger

Usage: This trigger allows us to achieve the functionality:
whenever a tuple is deleted in the Redeem, we will update the number of remaining redemptions corresponding to the respective package.

Justification: This design allows us to maintain consistency of the tables and ensure that the number of remaining redemptions are recorded correctly.

C. Add a trigger in add_session

```
create or replace function update_start_end_date_and_capacity_of_course_offering() returns trigger as $$
declare
    curs cursor for (select * from Sessions_Conducts_Consists as SCC inner join Rooms as R on R.rid = SCC.rid
                     where SCC.launch_date = NEW.launch_date and SCC.course_id = NEW.course_id and SCC.is_valid = 0);
    r record;
    max_date date;
    min_date date;
    total_seating_capacity int;
begin
    -- each course offering has a start date and an end date that is determined by the dates of its earliest and latest sessions
    select min(date), max(date) from Sessions_Conducts_Consists
            where launch_date = NEW.launch_date and course_id = NEW.course_id and is_valid = 0 into min_date, max_date;
    if min_date is null then
        min_date := NEW.date;
    end if;
    if max_date is null then
        max_date := NEW.date;
    end if;
    -- the seating capacity of a course session is equal to the seating capacity of the room where the session is conducted
    -- the seating capacity of a course offering is equal to the sum of the seating capacities of its sessions
    total_seating_capacity := 0;
    open curs;
    loop
        fetch curs into r;
        exit when not found;
        total_seating_capacity := total_seating_capacity + r.seating_capacity;
    end loop;
    close curs;

    update Offerings_Has_Handles set start_date = min_date, end_date = max_date, seating_capacity = total_seating_capacity
            where launch_date = NEW.launch_date and course_id = NEW.course_id;
    return null;
END;
$$ language plpgsql;

create trigger update_start_end_date_and_capacity_of_course_offering_trigger
after insert on Sessions_Conducts_Consists
for each row execute function update_start_end_date_and_capacity_of_course_offering();
```

Name: update_start_end_date_and_capacity_of_course_offering_trigger

Usage: This trigger allows us to achieve the functionality: whenever

Justification: This design allows us to maintain consistency of the tables and ensure that the start date, end date as well as the seating capacity of each course offering are recorded correctly in the Offering_Has_Handles table. The trigger allows us to meet the design requirement that states each course offering has a start date and an end date that is determined by the dates of its earliest and latest sessions.

5. Summary

a. Time management difficulties:

- Our estimation of the time required to complete the tasks of this project is not accurate enough and it makes us have to keep rushing it till 2 am or 3 am in the last few days.

b. Communication efficiency difficulties:

- Due to the limited testing conditions, we had to choose to do code review with each other during our meeting and face the following two difficulties:
    1. How to quickly explain our ideas and codes to others.
    2. How to quickly explain our own difficulties and solve them together efficiently.

c. Tasks Assignment difficulties:

- We assign each group member to do several functions. Due to the nesting of functions, for example, yaoren implements function 7 find_instructors(), but Jiajun needs to call function 7 when implementing function 21 update_instructor() and function 24 add_session(). This caused development difficulties. Jiajun has to wait until yaoren is completed before testing and modifying two of Jiajun's own functions.

d. Functional difficulties:

- Some functions require in-depth understanding and application of sub-query and loops. For example, question 7 and question 9 requires a combination of

loop and sub-query to achieve the required outcome. This seems foreign to us in the first place as we do not know how to integrate the both methodology together. We spent quite some time figuring out how to properly link the different methods together to achieve the result.
- Although we design the Relational Schemas very carefully to cover every situation that we can think of, we still find some omissions or defects in the previous design and we have to re-design the Relational Schemas. it makes us have to spend more time in re-modifying and re-testing the previously completed Function.

e. Testing difficulties:

- We encountered some difficulties when testing our queries as we do not know how to efficiently test our function as we have limited entries for the table and we cannot fully ensure our table fulfils the criteria.
- For example, for the function view_summary_report(), many Relational Schemas set Check Constraint to check date must be equal or later than current date, which makes it difficult for us to insert test data from previous months or years. The only way is to modify the system time or remove Check Constraints of other Relational Schemas, but we believe that this is not the best testing method.

f. Lesson Learnt

- Deeper understanding of ER model and logical database design
- Improve the time management ability and the efficiency of the meeting, we should prepare our own questions before the meeting.
- Through the project, we have a deeper understanding on how to properly design and modify the ER model to suit the need of the design requirements
- Moreover, we also have a deeper understanding on how to translate the ER model to logical data schema, such as how to combine relationships and entities and how to ensure the constraints are fulfilled.

g. Greater appreciation on writing SQL function:

- Previously we have only superficial understanding of the function. However, after the project, we truly understand how to use function and triggers to fulfil more complex requirements.

h. Learn more test methods to improve test efficiency and coverage. We spend the most time on function testing.