CS2102

Database Systems

Project Report

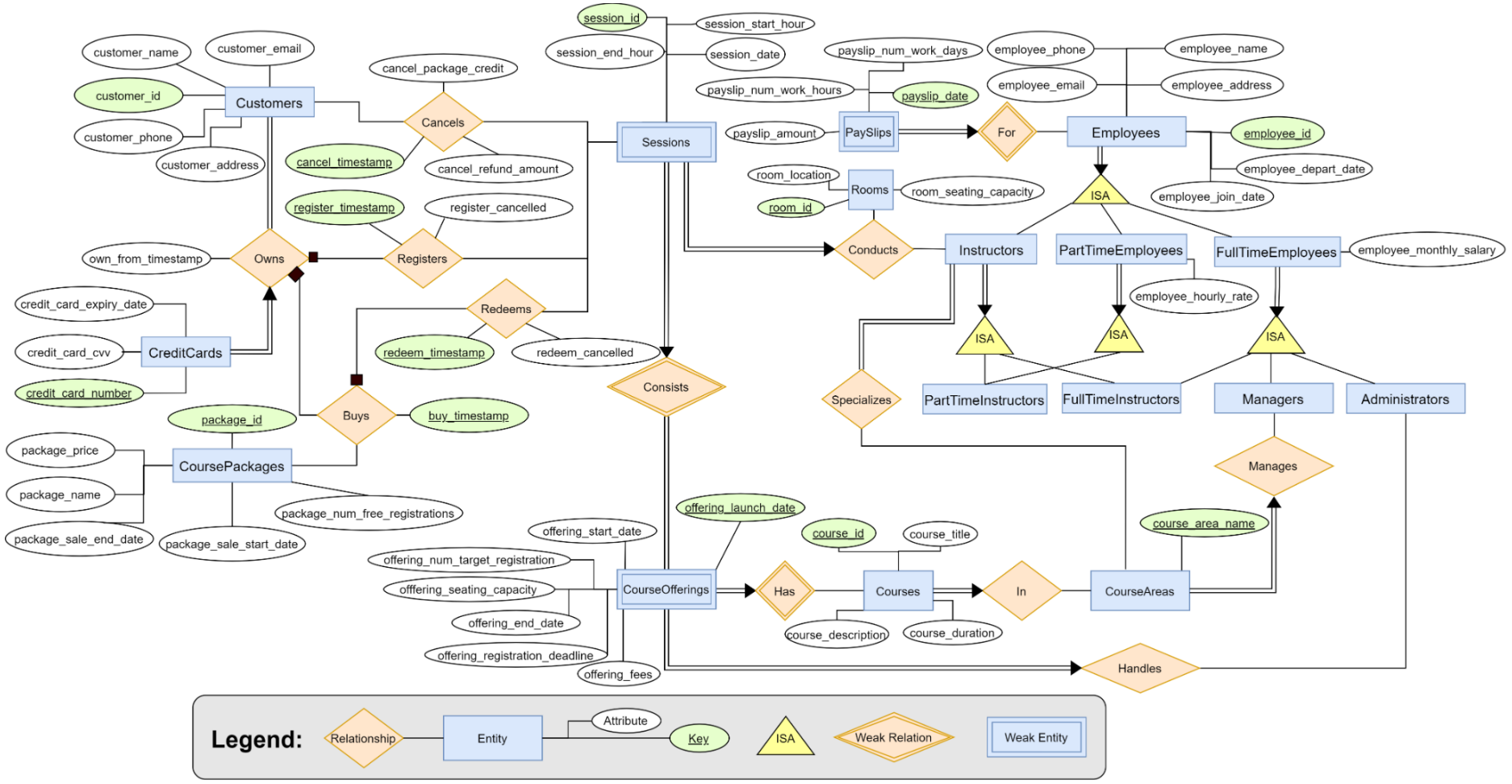
Team: 80

Name	Matriculation Number	NUSNET ID
Daniel Lim Wee Soong	A0152225R	E0021725
Low Qing Ning	A0202215B	E0417622
Ngo Wei Lin	A0185374W	E0318665
Wen Junhua	A0196683L	E0389169

Responsibilities

Name	Contributions
Daniel Lim Wee Soong	Make tests for the functions. Debug and correct errors from the schema/functions/triggers. Implemented functions / triggers and schemas
Low Qing Ning	Make tests for the functions. Debug and correct errors from the schema/functions/triggers. Implemented functions / triggers and schemas
Ngo Wei Lin	Make tests for the functions. Debug and correct errors from the schema/functions/triggers. Implemented functions / triggers and schemas
Wen Junhua	Make tests for the functions. Debug and correct errors from the schema/functions/triggers. Implemented functions / triggers and schemas

ER Data Model



ER Model: Non-trivial Design Decisions

Decision	Rationale
`Redeems`, `Registers`: Usage of `cancelled` flag, and do not delete row from table after a cancellation.	To know whether a redeemed or registered session has been cancelled. It is also possible for a customer to register, then cancel, then register, then cancel, for an unlimited amount of times. We chose to retain these information in the database for future reference, instead of deleting the relevant entities once they are cancelled. The `Cancels` table is also unable to track exactly which registration/redemption was cancelled.
`CourseOfferings`: Weak entity of `Courses` with launch date as key	Since `CourseOfferings` is a weak entity of `Courses`, it will have the course identifier in its key. To uniquely identify each course offering, we chose to use the launch date as part of the key, without an extra integer value as the identifier. The launch date is sufficient to uniquely identify each course offering for a course since offerings for the same course have different launch dates..
`Buys`, `Registers`, `Redeems`, `Cancels`: Only use the transaction time as key	We used the transaction timestamp instead of date for identifying records in Buys/Registers/Redeems/Cancels so that there can be more than one record associated with a customer on the same day. As concurrency is not within the scope of this project, a timestamp is suitable to be used as part of the primary key instead of using a dedicated integer identifier for the above entities/relations.
`Redeems`, `Registers`: Separate registrations by credit card from registration by course package	The logic and information stored for registration by credit card and by course package are quite different. So, it is better to split them into 2 different entities. If they were to be combined into the same table, there would be a lot of redundant data. Relationships with other entities would also be more complicated.

ER Model: Application Constraints Not Captured

1. The number of enrollment for a course session cannot be more than the seating capacity of the lecture room used.
2. 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 the instructor is teaching).
3. Each instructor can teach at most one course session at any time.
4. Part-time instructors must not teach for more than 30 hours each month.
5. For each course offered by the company, a customer can register for at most one of its sessions before its registration deadline.

Requirements: Assumptions/Interpretations

1. `add_employee()`:
    a. **Managers:** Course area is created when inserting managers using this function.
    b. **Instructors:** Course area of an instructor must already exist.
    c. Has an additional attribute to determine if the person is part-time or full-time
        i. Will throw an error if the admin or manager is a part-time employee
        ii. Will also throw an error if the admin has a course area
    d. Returns the employee id inserted for future reference.

2. `add_customer()`
    a. Returns the customer id inserted for future reference.

3. `update_credit_card()`
    a. We assume that the customers do not share credit cards. An error will be returned if a customer attempts to own a credit card that is already previously/currently registered by another customer.

4. `get_available_rooms()`
    a. We assume that the start date and the end date are inclusive.

5. `add_course()`
    a. Returns the `course_id` inserted for future reference.

6. `add_course_offering()`
    a. Returns course offering details upon successful addition of course offering and its sessions
    b. The assignment of instructors attempts to find first-fit (first available instructor found to be able to conduct the session), so the fitting of the instructors might not be optimal.
        i. Optimal scheduling of instructors is not in the scope for CS2102.
        ii. In real-life scenarios, implementation of such scheduling algorithms are much more complex, and are generally implemented using the host language rather than in SQL.

7. `add_course_package()`
    a. Returns the `package_id` inserted for future reference

8. `update_course_session()`
    a. `add_course_offering()` ensures that there is enough seat capacity, but this function does not. This is because the requirements for `remove_session()` already allows for the resultant seating capacity to fall below the target number of registrations.
    b. Only allow changing within the same course offering and not to a session of a different course

c. The redemption/registration timestamp is preserved as we do not want to allow a person to change to a session that is more than 7 days away and cancelling it to get a refund.

9. `top_packages()`
    a. We exclude all course packages with no sales from being presented in the results. This is because it does not make sense to consider a course package with no sales as a ‘top’ course package.

10. `popular_courses()`
    a. Exclude courses with 0 registration as they will be of low statistical use for those who are looking for the information.

11. `buy_course_offerings()`
    a. The customer’s latest credit card will be selected for payment.
    b. If the expiry date of the latest credit card of the customer has already passed, then the purchase will be rejected.

Schema: Non-trivial Design Decisions

Decision	Rationale
`Buys, Owns, Redeems, Registers:` Use `timestamp` instead of `date` as the primary key.	Use a more precise data type, so that the users are not limited to only one operation a day. Also, it is not an application requirement to support concurrency, hence our decision to use time as the key.
`Sessions:` Session numbers are only consecutive upon insertion through `add_course_offering()`	When sessions are added, they are assigned consecutive session numbers as a part of the identifier. This is facilitated by `add_course_offering()`. We omitted triggers or functions to reassign the session numbers when sessions are removed, as this is impractical in terms of speed. It may cause concurrency issues as well, since many rows would need to be updated in the same transaction.
`PaySlips:` Store part-time and full-time payslips in the same table.	Each row stores the information for a part-time and full-time employee. There is redundancy since each employee can only be either part-time or full-time, and never both. Although either payslip_num_work_days or payslip_num_work_hours for each row will always be NULL, we decided to combine all the information into a single table, to minimize the number of joins required to obtain the data.

Schema: Interesting Triggers

1. `check_offering_dates`

   Usage: Sets the start and the end date for `CourseOfferings`

   Why this trigger: Each course offering has a **start date** and an **end date** that is determined by the dates of its earliest and latest sessions, respectively.

   Due to this, for every update of its sessions, there might be a change in the start date and the end date for the particular course offering.

   In our opinion, triggers are the best ways to enforce this as we can hook it to a function after the execution of insertion or update of the date of the session. This will allow us to add sessions without worrying about the update of the start date and the end date.

   Another alternative that we considered is using views. However, a table for the course offerings will still have to be created anyways and the view will inherit most of the attributes of the table. There will be a waste of resources.

2. `customers_total_participation_check`

   Usage: Checks that `customer_id` should exist in at least one record of `Owns` relation. In other words, each customer must own a credit card. A similar concept was used on the other tables to enforce the total participation relationship for `CreditCards`.

   Why this trigger: It is very hard to enforce the total participation constraint of the `Owns` table, as the user can use different functions to insert the values into the table.

   It is error-prone to rely on every related function to enforce this constraint. By using a trigger, it restricts what the user can do and prevents them from removing the credit card information entirely, and this burden of enforcing is not placed on every function that modifies the `Owns` table.

3. `session_collision_check`

   Usage: Check if the newly inserted session collides with any other sessions which are available in the course

   Why this trigger: This will allow us to abstract some of the checks to the trigger instead of checking for this constraint inside each one of the functions. This also blocks anyone who wants to insert manually into the table any sessions which collide with another session should one of the users choose to do so manually.

Schema: Application Constraints Not Enforced

1. Each part-time employee cannot work for more than 30 hours
    a. Checks if part-time employees cannot work for more than 30 hours
2. Check if the customer still has redemption credits left to redeem the different sessions.
3. Check if different sessions will collide with each other during insertion of the session.
4. Check if the employees can only be either admin/instructors/manager.
5. There must be a 1 hour break between sessions for each of the instructors.

These constraints were then enforced by triggers.

Reflection: Difficulties Encountered

1. Design decisions to be made regarding how to delegate the different aspects of the design to different components.
    a. Some of the design can be left to be implemented either as functions, procedures or to be used/enforced in triggers.
    b. Some of the trade-offs between performance and clear/readable code

2. Test and come up with edge cases for the data.
    a. There are numerous edge cases that were hard to detect
    b. Parallel aspects of databases when there are multiple queries inserted at once

3. Time-consuming test manually
    a. Postgres has a command-line interface that is not very friendly to drop and insert the datasets along with routines repeatedly
    b. PGadmin has an interface but importing the test data requires manual clicking the tables one by one
    c. This was resolved with the help of automated testing using psycopg2 library and the creation of unit tests.

4. Spotting logical errors
    a. It was very tedious to spot logical errors within the .sql files as there were many lines of code. Some of the logical errors can be subtle.

Reflection: Lessons Learnt

1. Some checks that are required by many functions can be implemented in the form of triggers in order to save on the lines of code that are required.
2. Unit tests for the functions are very important as it helps to catch errors which we might not spot just by scanning through the code. However, that being said, it goes hand in hand with logical checking as sometimes the unit tests may not be comprehensive enough to detect all edge cases are being properly handled.
3. By implementing more tables that are in BCNF, while there is less redundancy, the joining of different tables might be required to get the information that we need. In some cases where the information is retrieved often, it might be better to just leave the tables together. The trade-off must be considered in order to decide which schema is the best way to implement it.
4. Automated testing helps to identify errors a lot quicker compared to manual testing, as it can help to prevent regressions when updating the application.
5. Some algorithms are not easy to implement in SQL
    a. Example: Pathfinding or scheduling algorithms
    b. It is better to implement them in a host language