COMP 421: Database Systems
Project 1: Database Design and Data Modelling

Hair Albeiro Parra Barrera
Hehuiming Cheng
Zijin Nie
Jifeng Wang
Group 88

1 Requirement Analysis

1.1 Introduction

1.1.1 Purpose

The purpose of this application is to introduce a dating system different to any others: a date rental service. Instead of putting time and effort in looking for a date, in which some cases people are not willing or feel unable to put in, we want to offer them the possibility of renting a suitable dating partner with certain desired characteristics. Consider for instance, an event in which going alone would be rather awkward; rent a date of your liking instead! Sometimes people feel lonely and would simply like some companies to do their preferred activities: walking around a park, going to a restaurant or a cinema, talk at a café, etc. We propose an interactive platform in which people can do this easily.

1.1.2 Scope and special requirements

We limit ourselves to the region of Greater Montreal. Due to safety and legal purposes, we limit the kind of activities allowed on our platform to typical activities in public places, by common transportation and within the applicable area. We also limited ourselves to the general browsing and booking system rather than the security and supervision system, which required some external resources.

1.2 Database Description

1.2.1 Entities and attributes

Our database is constituted of the following entities with their respective attributes:

1. **User**: A user is any individual involved with the service. It is-a Manager, is-a Mate or is-a Customer. It contains basic attributes such as first and last name, date of birth and email, sex (male/female), city of residence, a phone number, city of residence (Montreal, Longueil, Laval, etc.), a unique username which is its primary key and a password.

2. **Manager**: A manager is a sub-entity of the Person entity and serves as an administrator of our platform. It does not contain any additional attribute other than the one inherited from Person.

3. **Mate**: A Mate is a sub-entity of the Person entity , who is the dating partner that customers can choose to go on a date with. It comprises additional attributes which include a nickname, a general description, the Mate's height, weight, description, languages he/she speaks, preferences in terms of dating activities, and an hourly rate which is the price customers pay to rent the Mate for a date.

4. **Application**: In order for a Product to make part of the dating service system, it needs to submit an application. Application is a weak entity which depends on a Mate entity. The application id, *appid* attribute is used for identify applications of each mate. The database keeps track of the application time and whether it is approved or not by a Manager.

5. **Customer**: A Customer is a sub-entity of the Person entity, who is looking for a date with a Product on our platform. The database keeps track of the activity preferences of a Customer.

6. **Order**: A booking is produced whenever a Request is approved by a Product. It is uniquely identified by an artificial *oId*. The database keeps track of the start time, end time, and order status of each Order. An Order can have the following statuses: active, if the request was approved and the respective invoice has been paid; pending, if the order has been placed but the invoice has not yet been paid, completed if the order has been completed (i.e. the Customer and the Product finished the date successfully); and cancelled, if the customer decided to change his/her mind and not go the date anymore.

7. **Activity**: Activity enumerates all the possible activities that could possibly happen in a date. An activity can be of different types: movie at the cinema, strolling at the park, meal at restaurant, assistance to an event, etc. Activity is a weak identity associated with order, it has a weak attribute,*aid* for identification. It also contains a description of the activity.

8. **Invoice**: An Invoice consists of a description of the payment (i.e. the service), the charged amount and the due date. It is uniquely identified by an invoice id (artificial primary key).

9. **Request**: A Request is uniquely identified by an artificial id called **rid**. It also contains an attribute called **rinfo** to state the specific information about a request.

The database also keeps track of the status of each request, which can following: unfinished, waiting for the Mate to decided if he/she is going to approve the request, and finished, the Mate has already reacted with the Request.

1.2.2 Relationships

1. **Reviews**: A *manager* **reviews** an *application* send by a user for becoming a product. It is a one-to-many relationship, because a manager can review many applications. A key constraint is added to the application side, since an application can only be reviewed by one manager. Based on the information provided in the application, that manager shall decide whether to accept or reject the application. The database keeps track of the review time, decision made as well as a comment to the review.

2. **Applies**: A *mate* **applies** an *application*. It is a one-to-many relationship with participation constraints. A product must send at least one application when he/she registers, and he/she is allowed to multiple applications if previous ones are rejected. An application must relate to only one mate.

3. **Rates**: A *Customer* **rates** a *Order*. A customer can only rate a finished order. This relationship is a one-to-many relationship. It has a key participation constraint on order since one order can only be rated once (since a customer rates an specific date). A customer can rate many orders he has made before. The database keeps track of rating date, rating number (1-5) and a comment.

4. **Pays**: A *Customer* **pays** an *invoice*; The relation is a one-to-many relation. It has a key participation constraints on invoice. The customer can pay many invoices, and one invoice can be related to only one customer. He pays with a certain amount of money, with a certain paying method (for simplification, mastercard, paypal, or visa).

5. **Contains**: A *order* may **contain** an *invoice*. It is an one-to-one relationship with key and participation constraint. An invoice must belong to one and only one booking. A booking can have most one invoice for a confirmed booking, or not have an invoice for pending, rejected and cancelled bookings.

6. **Decides**: Once a request is made, a *Mate* **decides** the *Request* (accepts or rejects it). The relationship is a one-to-many relationship. It has a key participation on the request. A mate can decide on many requests, and one request can only be confirmed or rejected by the mate it relates to. The decision time will be stored.

7. **Start**: In the reservation system, a booking starts by a **Customer** starts a **Request** to **Mate**. This is a ternary relationship. It has a key and participation constraints on the Request, since the request can only related to only one pair of mate and customer. The start time of the whole booking will be recorded.

8. **Generate**: After the *Request* is confirmed, the *Request* **Generate** the *Order*. This is a one-to-one relationship with key and participation constraint. A request can generate only one order and an order must correspond to only one request.

9. **Modify**: A *Manager* can **Modify** an *Order* once after it is been generated and before its start time. It is a many-to-many relationship. The time of modify will be recorded.

10. **Overview**: A *Manager* can **Overview** an *Activity*. This is a one-to-many relationship with key and participation constraint. A manager can overview many activities and an activity must be overviewed by only one manager.

11. **Schedule**: An *Order* must **Schedule** an *Activity*. This is a one-to-many relationship with key and participation constraint. An activity must scheduled by only one order and an order must schedule at least one activity.

1.3 Comments

- A *Customer* can only leave a rating (**rates**) of a *Product* if the *Booking Status* is complete. This should be captured at the software level.

- It is possible that many *Orders* for the same *Mate* may be placed at the same time. This should be captured at the software level.

1.4 Application Description

1.4.1 Overview

The web-based application will all entities involved in the relational model to perform all described relations through an interactive interface. In particular, it will allow **customers** to book **Mates** by first seeing their pictures and descriptions of the products, and perhaps even filtering them, and then clicking a "book me!" button. From there, the booking process takes place.

2 ER Translation

- `User(username, email, sex, city, phoneNum, password, dateOfBirth, firstName, lastName)`

- `Mate(username, nickName, description, language, height, weight, hourlyRate, preferences)`
  "username foreign key references User"

- `Customer(username,preferences)`
  "username foreign key references User"

- `Manager(username)`
  "username foreign key references User"

- `Application(appid,username,aTime,isApproved,mngName,rTime,comment,decision)`
  "username foreign key references Mate"
  "mngName foreign key references Manager"

- `Request(rid,rinfo,rstatus,mateName,decTime)`
  "mateName foreign key references Mate"

- `Order(oid,endTime,startTime,ordStatus,rid,inid,custName,ratingDate,comment,rating)`
  "rid foreign key references Request"
  "inid foreign key references Invoice"
  "custName foreign key references Customer"

- `Invoice(inid,description,dueDate,amount,custName,pamount, paytime,method,status,oid`
  "oid foreign key references Order" "custName foreign key references Customer"

- `Activity(aid,oid,description,mngName)`
  "oid foreign key references Order" "mngName foreign key references Manager"

- `Start(rid,mateName,custName,startTime)`
  "mateName foreign key references Mate"
  "custName foreign key references Customer"
  "rid" foreign key references Request"

- `Modify(mngName,oid,modTime)`
  "mngName foreign key references Manager"
  "oid foreign key references Order"

- `Generate(rid,oid)`
  "rid foreign key references Request"
  "oid foreign key references Order"

Remark.

1. In the above translation, it is possible to combine the **Order** and **Invoice** entity since they are a one-to-one relation. However, in out design, we keep them separated for clarity, and to allow flexibility in case of possible further changes or adaptations.

References

[1] Joseph "COMP 421: Database Systems" McGill University Winter 2020