Εθνικό Μετσόβιο Πολυτεχνείο

Βάσεις Δεδομένων

Αναφορά Εξαμηνιαίο Project 2013-2014

Ηλίας Φωτόπουλος
03109106
Θανάσης Βράτιμος
03110769

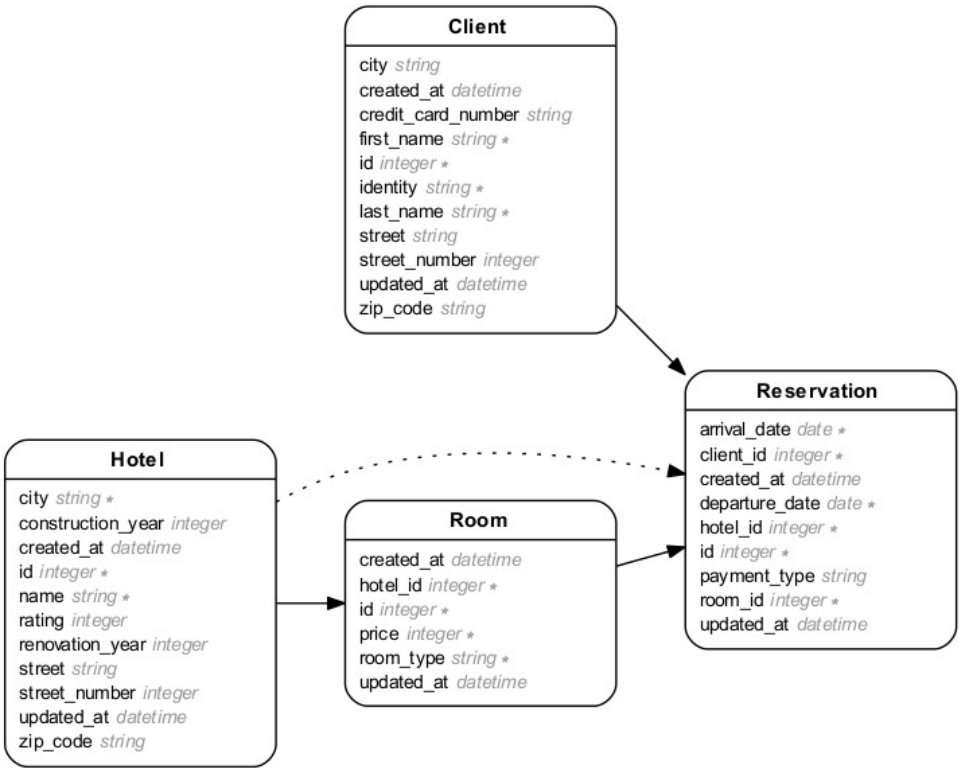
1 Λεπτομέρειες Υλοποίησης

Για την κατασκευή του project έγινε χρήση των παρακάτω τεχνολογιών:

- Ruby on Rails (Framework)
- HTML Sass (Syntactically Awesome Style Sheets)
- Bootstrap (Sass Framework)
- CoffeeScript
- Git (Version Control)

Κατά την φάση του σχεδιασμού αποφασίστηκε η χρήση του RoR framework λόγω της αρχιτεκτονικής MVC που ακολουθεί αλλά και λόγω της φιλοσοφίας του Convention over Configuration (CoC) και Don't Repeat Yourself (DRY).

2 Relational model



3 Constraints

Τα constraints και η business logic της εφαρμογής μας γίνεται εξολοκλήρου μέσα στο framework της RoR (εκτός από τα 2 triggers, βλ.Section Triggers). Σε κάθε φόρμα για update ή edit μιας σχέσης γίνονται κάποια validations. Έτσι ώστε να εξασφαλιστεί data integrity στην βάση δεδομένων μας.

3.1 Hotel Validations

```
1  validates :name, :city, presence: true
2  validates :street_number, numericality: true
3  validates :rating, numericality: true,
4  inclusion: {in: 0..5, message: "rating should be 0-5"}
5
6  validates :construction_year, :renovation_year,
7  inclusion: { in: 1800..Date.today.year, message: "Year ↩
       should be over 1800"},
8    allow_nil: true,
9    format:
10   {
11     with: /(18|19|20)\d{2}/i,
12     message: "should be a four-digit year"
13   }
```

3.2 Room Validations

```
1  validates :price, :room_type, presence: true
2  validates :price, numericality: {greater_than: 0}
```

3.3 Client Validations

```
1  validates :identity, :first_name, :last_name, presence: ↩
       true
2  validates :identity, uniqueness: {message: "This identity↩
       already exists. Identity must be unique!"}
3  validates :street_number, numericality: true
```

3.4 Reservation Validations

Ειδικά για τις Reservation φτιάξαμε μια Service με όνομα ReservationService, η οποία ουσιαστικά ελέγχει να δει αν ένα δωμάτιο είναι ελεύθερο κάποιες ημερομηνίες που της δίνουμε. Η βασική της λογική φαίνεται στην παρακάτω συνάρτηση:

```
1  def room_available?
2   reservations = UpcomingReservationUpdatable.where("↩
       room_id = ?", @room.id)
3   arrival_date = Date.parse @arrival
4   departure_date = Date.parse @departure
5
6   reservations.each do |r|
7     before = (arrival_date < r.arrival_date) && ( ↩
         departure_date < r.departure_date)
8     after = (arrival_date > r.arrival_date) && ( ↩
         departure_date > r.departure_date)
9     if before || after
10      next
11    else
12      return false
13    end
14  end
15  return true
16 end
```

Και τα γενικότερα validation του model Reservation:

```
1  belongs_to :client
2  validates :client_id, presence: true
3
4  belongs_to :room
5  validates :room_id, presence: true
6
7  has_one :hotel, through: :room
8  validates :hotel_id, presence: true
9
10 validates :arrival_date, :departure_date, presence: true
11
12 validate :arrival_before_departure, :↩
     arrival_date_cannot_be_in_the_past, :↩
     departure_date_cannot_be_in_the_past
13
```

```
14 def arrival_before_departure
15 errors.add(:arrival_date, "must be before departure date"↩
       ) if arrival_date >= departure_date
16   end
17
18   def arrival_date_cannot_be_in_the_past
19 errors.add(:arrival_date, "can't be in the past") if ↩
       arrival_date < Date.today
20   end
21
22   def departure_date_cannot_be_in_the_past
23 errors.add(:departure_date, "can't be in the past") if ↩
       departure_date < Date.today
24   end
```

4 SQL Queries

Σε όλες τις σχέσεις δημιουργήθηκαν queries για προβολή, εισαγωγή, ενημέρωση και διαγραφή εγγραφών, τα οποία δεν περιλαμβάνονται την παρούσα αναφορά αφού δεν χρειάζονται κάποια εξήγηση.

4.1 Join queries

Γίνεται συχνή χρήση ερωτημάτων Join στην εφαρμογή μας όπως για παράδειγμα στο Reservations index:

```
1 SELECT reservations.id, reservations.arrival_date, ↩
      reservations.departure_date, reservations.created_at, ↩
      reservations.updated_at, clients.first_name, clients.↩
      last_name, hotels.name, rooms.room_type
2 FROM reservations
3 INNER JOIN clients
4 ON reservations.client_id = clients.id
5 INNER JOIN hotels
6 ON reservations.hotel_id = hotels.id
7 INNER JOIN rooms
8 ON reservations.room_id = rooms.id
9 ORDER BY arrival_date
```

Άλλο ένα παράδειγμα χρήσης Join είναι στο show client, όπου εμφανίζονται και οι reservations του κάθε πελάτη:

```
SELECT reservations.id, reservations.arrival_date, ↩
   reservations.departure_date, reservations.created_at, ↩
   reservations.updated_at, hotels.name, rooms.room_type
FROM reservations
INNER JOIN hotels
ON reservations.hotel_id = hotels.id
INNER JOIN rooms
ON reservations.room_id = rooms.id
ORDER BY arrival_date
```

Ενώ μπορείτε να δείτε και άλλο παράδειγμα χρήσης join στα Views που δημιουργήσαμε για την εφαρμογή.

4.2 Aggregate queries

Στην εφαρμογή μας κάναμε κυρίως χρήση του count για να μετράμε τον αριθμό των reservations ενός πελάτη ή τον αριθμό των δωματίων ενός ξενοδοχείου.

Εύρεση αριθμού δωματίων ενός ξενοδοχείου:

```
SELECT COUNT(*)
FROM rooms
WHERE rooms.hotel_id = hotel_id
```

Εύρεση αριθμού κρατήσεων ενός πελάτη:

```
SELECT COUNT(*)
FROM reservations
WHERE reservations.client_id = client_id
```

4.3 Group By

Στην ενότητα Reports της εφαρμογής γίνεται χρήση της εντολής Group By σε συνδυασμό με την aggregate εντολή count.

Το παρακάτω ερώτημα βρίσκει τον αριθμό των κρατήσεων κάθε ξενοδοχειακής μονάδας.

```
SELECT COUNT(*) as count, hotels.name as name
FROM reservations
INNER JOIN hotels
```

```
4 ON reservations.hotel_id = hotels.id
5 GROUP BY hotels.name
```

Ενώ το παρακάτω βρίσκει τον αριθμό των κρατήσεων ανά τύπο δωματίου.

```
1 SELECT COUNT(*) as count, rooms.room_type as room_type
2 FROM reservations
3 INNER JOIN rooms
4 ON reservations.room_id = rooms.id
5 GROUP BY rooms.room_type
```

4.4 Order By

Έχει γίνει χρήση Order By στα περισσότερα ερωτήματα της εφαρμογής μας. Μερικά παραδείγματα υπάρχουν στα views που εμφανίζονται στο παρακάτω section της παρούσας αναφοράς.

4.5 Group By - Having

Το παρακάτω ερώτημα επιστρέφει τα ξενοδοχεία που έχουν αριθμό δωματίων μεγαλύτερου του 5.

```
1 SELECT COUNT(*) as count, hotels.name as name
2 FROM rooms
3 INNER JOIN hotels
4 ON rooms.hotel_id = hotels.id
5 GROUP BY hotels.id
6 HAVING count > 5
```

5 Views

Δημιουργήσαμε τα παρακάτω δύο views. Το view upcoming reservations περιέχει όλες τις κρατήσεις για τις οποίες η μέρα άφιξης είναι μεταγενέστερη της σημερινής μέρας. Επίσης εκτελεί inner join, με τα tables: clients,hotels,rooms έτσι ώστε να εμφανίσει όνομα πελάτη, ξενοδοχείου αλλά και τον τύπο του δωματίου. Η χρησιμότητα του view είναι η παρουσίαση όλων των κρατήσεων για τις οποίες ο πελάτης δεν έχει φτάσει ακόμη.

```
1 CREATE OR REPLACE VIEW upcoming_reservations AS
```

```
2  SELECT reservations.id, reservations.arrival_date, ↩
       reservations.departure_date, reservations.created_at, ↩
       reservations.updated_at, clients.first_name, clients.↩
       last_name, hotels.name, rooms.room_type
3  FROM reservations
4  INNER JOIN clients
5  ON reservations.client_id = clients.id
6  INNER JOIN hotels
7  ON reservations.hotel_id = hotels.id
8  INNER JOIN rooms
9  ON reservations.room_id = rooms.id
10 WHERE arrival_date > CURDATE()
11 ORDER BY arrival_date
```

Το view upcoming reservations updatable επιτελεί τις ίδιες λειτουργίες με το παραπάνω με δύο βασικές διαφορές:

- Δεν χρησιμοποιεί inner join για να πάρει τα names από τα άλλα tables.
- Περιέχει κρατήσεις όπου η μέρα αναχώρησης (όχι άφιξης) είναι μεταγενέστερη της σημερινής

Το εν λόγω view παίζει σημαντικό ρόλο στην εσωτερική αρχιτεκτονική και λογική της εφαρμογής μας. Χρησιμοποιείται από το ReservationService το οποίο ελέγχει αν ένα δωμάτιο είναι διαθέσιμο για κράτηση.

```
1  CREATE OR REPLACE VIEW upcoming_reservations_updatable AS
2  SELECT *
3  FROM reservations
4  WHERE departure_date > CURDATE()
5  ORDER BY arrival_date
```

6 Triggers

Δημιουργήσαμε τα παρακάτω δύο triggers.
Το trigger delete hotel rooms, το οποίο πριν διαγράψει ένα ξενοδοχείο διαγράφει και όλα τα δωμάτια του. Τα δωμάτια ανήκουν στο αδύναμο σύνολο οντοτήτων "Δωμάτιο" και εξαρτιούνται πλήρως από το ξενοδοχείο στο οποίο ανήκουν, οπότε και δεν έχει νόημα να υφίστανται αν διαγραφεί το ξενοδοχείο τους.

```
1  CREATE TRIGGER delete_hotel_rooms
```

```
2 BEFORE DELETE ON hotels
3 FOR EACH ROW
4 BEGIN
5   DELETE FROM rooms WHERE OLD.id = hotel_id;
6 END
```

Παρομοίως με παραπάνω δημιουργήσαμε ένα trigger που διαγράφει όλες τις κρατήσεις ενός πελάτη πριν διαγραφεί ο πελάτης.

```
1 CREATE TRIGGER delete_client_reservations
2 BEFORE DELETE ON clients
3 FOR EACH ROW
4 BEGIN
5   DELETE FROM reservations WHERE OLD.id = client_id;
6 END
```

Στο σημείο αυτό αξίζει να αναφερθεί ότι το παραπάνω θα μπορούσε να υλοποιηθεί εύκολα και μέσα από το περιβάλλον του RoR Framework (χρήση Rails ActiveRecord callbacks). Αρκεί να προστεθούν τα παρακάτω στα μοντέλα hotel και client αντίστοιχα.

```
1 has_many :rooms, dependent: :destroy
```

```
1 has_many :reservations, dependent: :destroy
```

6.1 Callback vs Trigger

Πλεονεκτήματα Callback:

- Όλη η Business Logic του μοντέλου βρίσκεται στην Rails κάτι το οποίο κάνει πιο εύκολη την συντήρηση και ανάπτυξη του.
- Εύκολο Debug
- Ο κώδικας Ruby γράφετε και διαβάζεται πιο εύκολα
- Η εφαρμογή μας δεν εξαρτάται από την υλοποίηση (και συντακτικό) της βάσης δεδομένων.

Πλεονεκτήματα Trigger:

- Είναι πιο γρήγορο από ένα callback. (Στο callback χρειάζεται να συνδεθείς στην βάση δεδομένων, ενώ στο trigger είσαι ήδη στο layer της βάσης δεδομένων)

Βάση των παραπάνω και δεδομένου το μέγεθος της εφαρμογής το ιδανικό θα ήταν να γίνει χρήση Rails ActiveRecord callbacks.