

Tema d'esame di Applicazioni Web
Parzialmente risolto

21 novembre 2006
Docente Matteo Vaccari

0. (6pt) Scrivere un template `.rhtml` per una tabella di utenti. La variabile `@users` contiene un array di utenti; ogni utente ha gli attributi `"first_name"` e `"last_name"`. Inoltre, vogliamo fare in modo che le righe della tabella siano di colore alternato. La scelta dei colori deve essere fatta separatamente mediante CSS (che dovete scrivere.)

Non credo ci sia bisogno di fornire una soluzione. Chiunque abbia svolto l'elaborato dovrebbe essere in grado di risolvere questo semplice esercizio, peraltro dimostrato anche sul testo.

1. (6pt) Che cosa si intende per sql injection? Come è possibile difendersi, in particolare in un'applicazione Rails?

Vedi testo di Rails, capitolo 21, Securing your Rails Application

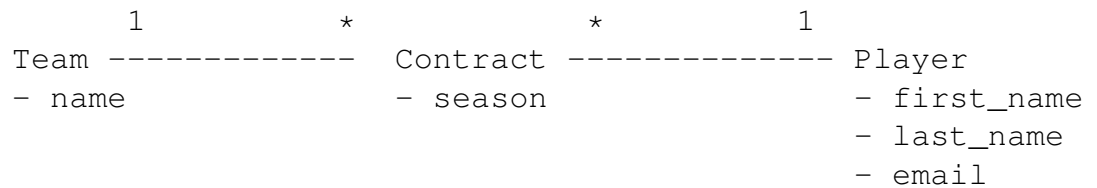
2. (6pt) Che cos'è lo header Content-Type? A che cosa serve? Fare un paio di esempi dei valori che può assumere.

Lo header Content-Type specifica il tipo MIME del documento che viene restituito al browser. In assenza di questo header, il browser non può sapere come interpretare il contenuto della risposta. Il valore più comune è "text/html." Altri esempi: text/plain, image/gif, image/jpeg, application/pdf.

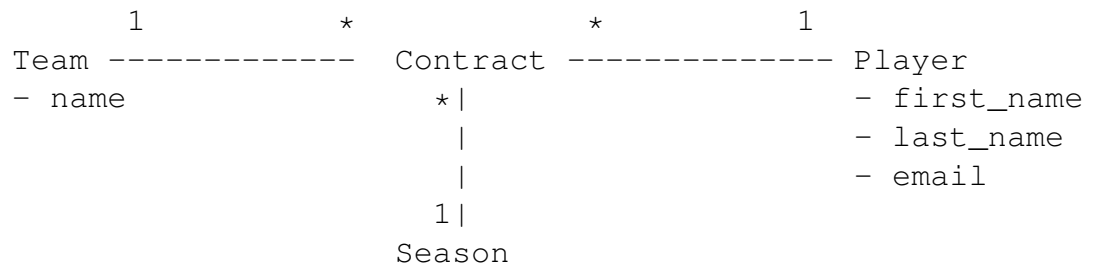
3. (6pt) In un'applicazione web vogliamo tenere traccia di un insieme di squadre (teams) e dei giocatori (players) che giocano in ciascuna squadra. L'ingaggio (contract) di un giocatore però non è permanente, ma cambia ogni stagione (season.)

Disegnare il diagramma entità-relazioni, e il codice sql necessario per implementarlo. (In alternativa a sql scrivere il codice delle migrations.)

La relazione fra giocatori e squadre è multi-a-molti, ma è anche collegata alla stagione dell'ingaggio. Quindi non è sufficiente una semplice tabella di supporto. Occorre espandere la relazione multi-a-molti in due relazioni uno-a-molti; che diventano tre se contiamo anche le stagioni come entità.



Oppure



Le migrazioni per il primo schema sono

```

create_table :team do |t|
  t.column :name, :string
end

create_table :player do |t|
  t.column :first_name, :string
  t.column :last_name, :string
  t.column :email, :string
end

create_table :contract do |t|
  t.column :season, :string
  t.column :team_id, :integer
  t.column :player_id, :integer
end

```

4. (6pt) Nell'applicazione del punto precedente: scrivere le classi ActiveRecord per tutti i modelli che ritenete necessari, facendo in modo che

- sia possibile da un oggetto Player recuperare facilmente il suo Team, data la stagione (es. `player.find_team(season_id)`)
- non sia possibile salvare nel database un Player se non sono valorizzati gli attributi `first_name`, `last_name`, oppure se l'email è già presente nella tabella.

Queste che seguono sono le classi strettamente necessarie, con le dichiarazioni di relazione strettamente necessarie per realizzare `Player#find_team`. (Riesci a vedere perché ognuna delle classi e relazioni è necessaria? Se non riesci, la cosa più semplice è costruire questo problema con Rails per toccare con mano perché ogni pezzetto è necessario.)

Ovviamente non guasta dichiarare qualche relazione in più oltre a quelle strettamente richieste, purché riflettano correttamente lo schema dei dati come risulta dal diagramma E-R e dal codice SQL/migrazioni. Ci sono variazioni possibili; posso costruire il metodo `find_team` anche senza dichiarare le relazioni, anche se preferisco vedere che sei capace di usare correttamente le dichiarazioni `has_many` e `belongs_to`. Altra variazione possibile: nel caso il contratto non venga trovato, questa versione lancia un'eccezione (riesci a capire perché?) Apprezzerei maggiormente una soluzione che restituisca `nil` invece.

```
class Player < ActiveRecord::Base
  validates_presence_of :first_name, :last_name
  validates_uniqueness_of :email

  has_many :contracts

  def find_team(season)
    contracts.find_by_season(season).team
  end
end

class Team < ActiveRecord::Base
end

class Contract < ActiveRecord::Base
  belongs_to :team
end
```