МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**
**(«МПУ»)**
КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

# КУРСОВАЯ РАБОТА

по дисциплине «Веб-технологии»
на тему «Разработка информационной системы библиотеки»

Группа 141131

Студент А.С. Сазонов

Руководитель работы
к.т.н., доцент В.Ю. Радыгин

Москва 2017

# Аннотация

Курсовая работа направлена на практику разработки интерфейсов информационной системы на примере библиотеки. В данном проекте будут продемонстрированны основные методы использования инструмента разработки веб-приложений Ruby on Rails.

# Содержание

# 1. Введение

Данная курсовая работа посвящена разработке интерфейсов информационной системы библиотечных залов, позволяющий вести учет ее элементов и осуществлять надлежащий контроль связей между ними. Также будет реализован поиск книг и осуществлена адаптивность для страниц веб-приложения.

Основным инструментом в разработке проекта является фреймворк Ruby on Rails, написанный на языке программирования Ruby. Данный фреймворк реализует архитектурный шаблон MVC (Model-View-Controller), а также обеспечивает поддержку самых востребованных СУБД на сегодняшний день. В качестве сервера базы данных нами будет использоваться PostgreSQL.

В качестве дополнительных гемов к фреймворку будем использовать гем haml для более удобной шаблонизации веб-страниц и их отдельных частей. А так же гем cocoon который поможет нам управлять вложенными формами.

Для подготовки пояснительной записки применялся набор компьютерной верстки LaTeX.

## Постановка задачи

1. Разработайть и добавить в базовое приложение модели необходимые из предметной области. Описать требуемые ограничения целлостности.
2. Организовать адаптивные интерфейсы редактирования. А так же настроить систему авторизации и навигации для данных интерфейсов.
3. Реализовать поиск книг по всем ее полям, а также всем полям доступных ей объектов.

# 2. Схема базы данных и валидации

Схема базы данных — это фундамент любого проекта на Ruby on Rails. Уделив ее проектированию достаточно внимания можно избежать многих трудностей в будующем.

Чтобы организовать правильное хранение данных на физическом уровне нужно определиться с тем, какие данные мы будем хранить, в каком виде, а также продумать ограничения целлостности и допустимые значения этих данных.

Для написания миграций нам необходимо изучить предметную область нашей будущей информационной системы. Чтобы решить, какие модели должно содержать наше приложении, нам необходимо выделить сущности, имеющие отношение к библиотеке методом семантического моделирования. Этот метод заключается определении структурных компонентов и характеристик данных, что позволяет правильно интерпретировать их разработчику.

Минимальный набор сущностей, позволяющий эффективно интегрировать вебприложение в реальную библиотеку будет таким:

- `Книга` имеющая название, том/часть, номер ISBN и количество экземпляров;
- `Автор` определенный фамилией, именем, отчеством, авторским указателем;
- `Зал` содержащий краткое название и полное название помещения;
- `Стеллаж` с единственным атрибутом – индекс стеллажа;
- `Расположение` представляет собой связующей звено между `Книгой` и `Стеллажом`. Помимо технической информации содержит номера полок;

Вполне возможно, что это не весь список сущностей необходимых для библиотеки, но, как и рекомендуется использовать Ruby on Rails, мы позаботимся о фундаментальной части приложения. И лишь в случае необходимости, потом добавим те, которых будет недоставать.

Помимо набора полей у каждой из моделей, нам понадобится определить как будут взаимодействовать модели друг с другом. Это принципиально важный момент, так как один набор сущностей может иметь сколь угодно много вариантов ассоциаций в моделях. Выбор определяет точные требования поставленные перед ИС.

В рамках данной работы мы не будем акцентрировать внимание особенностях проектирования базы данных. Примем следующую ER-диаграмму, показывающий отношения между этими сущностями, за абсолютно точно определяющей задачи инфологической модели для нашего проекта.

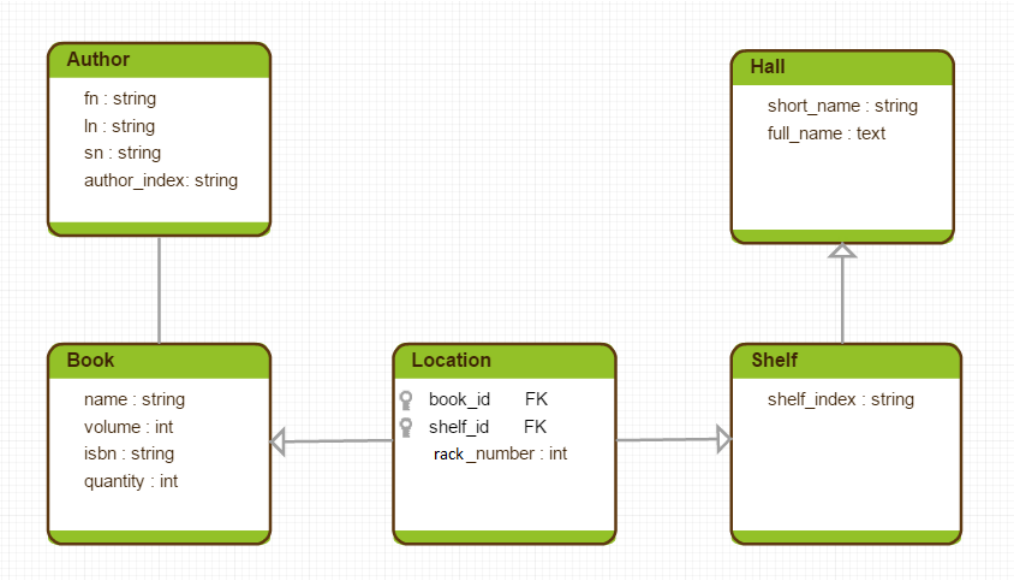


*Рис. 1.* ER-диаграмма

Что бы сократить действия, которые понадобятся разработчику для интеграции сущностей в веб-приложение, мы будем использовать scaffolding. По своей сути это утилита командной строки, доступная при работе с Ruby on Rails, которая генерирует ряд файлов и заполняет их содержимое.

Покажем на примере создания сущности `Книги` образец подобной генерации:

```
rails g scaffold Book name:string volume:integer isbn:string quantity:integer
```

Благодаря scaffold-у, помимо базового файла с миграцией в приложении сгенерировался REST-контроллер для данной модели, шаблон views. Уже на данном этапе наше приложение будет уметь просматривать, создавать, удалять объекты книг. Но это не весь функционал, который должна обеспечивать для данной модели ИС.

После того, как мы создали необходимые сущности перейдем к добовлению ограничений целлостности и добавления индексов в БД, которые требует концептуальная модель.

Продемострируем пример применения ограничения целлостности на физическом уровне изменив миграцию создания таблицы стеллажей следующим образом:

```
create_table :authors do |t|
  t.string :fn, null: false
  t.string :ln, null: false
  t.string :sn
  t.string :author_index, null: false
  t.timestamps null: false
end
```

При необходимости добавить ограничение отсутствующее в Ruby on Rails, нужно вручную прописать его для таблицы в соответствующем файле миграции. Добавим такое ограничение для сущности `Стеллаж` проверив длину авторского указателя (таким же образом в RoR должны быть добавлены внешние и первичные ключи которые нельзя было указать при создании миграции).

```
def change
  reversible do |dir|
    dir.up do
      execute "ALTER TABLE authors ADD CONSTRAINT chk_author_ind_length
        CHECK (length(author_index) >= 3)"
    end
  end
end
```

Закончив с описанием ограничений для физического уровня необходимо их продублировать на уровне классов. Для этого в Rails существует механизм валидаций.

Встроенные доступные валидации объектного уровня представляют собой более гибкий инструмент ограничений нежели ограничения для миграций. Но, как и в случае миграций, всегда можно реализовать свои ограничений целлостности, если это будет необходмо для приложения.

Перенесем описанные нами ранее ограничения для атрибутов стеллажей в их объектное представление:

```
class Author < ActiveRecord::Base
  validates :author_index, :fn, :ln, presence: true
  validates :author_index, length: { minimum: 2 }, allow_blank: true
  validates :author_index, uniqueness:
    {scope: :ln, message: "уже есть у одного из авторов-софамильцев"}
end
```

После написания всех валидаций нам осталось прописать лишь ассоциации представленные на диаграмме в начале раздела.

Связи между сущностями в Ruby on Rails реализуются элементарно: достаточно просто укзатель наличие связи необходимой размерности в соответствующих моделях. Исследовав предметную область и имея некоторый опыт в использовании фреймворка, не будет лишним добавить связь между `Книгой` и `Стеллажом` напрямую. Отметим, что исходные транзитивные связи не удаляются, а лишь добавляются новые.

Благодаря наличию связей между объектами нам стал доступен поиск связанных объектов, а так же их изменения и удаления. Для последующей реализации интер-

фейсов редактирования сразу несколько сущностей нашего приложения нам понадобятся вложенные формы. Включить их можно при помощи опции accepts nested attributes for.

Привидем пример связей для модели `Стеллаж`:

```
class Shelf < ActiveRecord::Base
  belongs_to :hall
  accepts_nested_attributes_for :hall
  has_many :locations, dependent: :destroy
  accepts_nested_attributes_for :locations
  has_many :books, through: :locations
end
```

Мы спроектировали систему сущностей для данной предметной области и реализовали все необходимые ограничения целлостности и ассоциации.

# 3. Заключение

Поставленная задача решена полностью в соответствии с поставленными требованиями. В результате получена полноценная ИС, позволяющая вести учет элементов библиотеки.

Реализованы основные интерфейсы, при помощи которых можно просто и эффективно управлять содержимым приложения и добавлена функция поиска книг.

Элементы интерфейсов адаптивны к просмотру на экранах любого размера.

Разделены обязанности у ролей и выполнено задание по внешнему оформлению проекта.

В рамках разработки ИС созданы концептуальная и даталогическая модели предметной области.

# Список литературы и интернет-ресурсов


1) Флэнаган Д., Мацумото Ю. *Язык программирования Ruby.* — СПб.: Питер, 2011.

2) Снитко Р. *Мир Rails. Правильное обучение разработке веб-приложений на Ruby On Rails.* — 2013.

3) `http://api.rubyonrails.org/` — Official API of the framework, 11.07.2017

4) `http://guides.rubyonrails.org/` — Руководства по отдельным составляющим фреймворка Ruby on Rails., 26.07.2017

5) `http://getbootstrap.com/` — Официальная страница фреймворка Bootstrap с описанием спецификации и руководставами., 26.07.2017

6) `http://rusrails.ru/` — Ruby on Rails руководства, учебники, статьи на русском языке., 12.07.2017

7) Т. Кормен. *Алгоритмы: построение и анализ, 2-е изд.* — Пер. с англ. — М., Издательский дом «Вильямс», 2005.

8) С.М. Львовский. *Набор и вёрстка в системе LaTeX, 3-е изд., испр. и доп.* — М., МЦНМО, 2003. Доступны исходные тексты этой книги.