

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ  
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ  
КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ»

Руководитель работы:  
Доцент, кандидат технических наук  
*Радыгин Виктор Юрьевич*

Студент:  
Войнов Максим Александрович

**Модернизация эталонного проекта "Информационная система  
аэропорт": Реализовать модель авиабилет**

**Курсовая работа по дисциплине  
«Базы данных и СУБД»  
3-й курс, 1-й семестр**

**Москва 2010**

## АННОТАЦИЯ

Курсовая работа посвящена модернизации эталонного проекта "Аэропорт" написанного средствами PostgreSQL. Предназначена для совершенствования навыков владения PostgreSQL.

## СОДЕРЖАНИЕ

Аннотация	2
1. Введение	3
2. Архитектура MVC	4
3. Структура базового проекта	5
4. Общее задание	7
4.1. Постановка задачи	7
4.2. Реализация	7
5. Индивидуальное задание	14
5.1. Примеры работы	14
6. Примеры работы	18
Список литературы	19

## 1. ВВЕДЕНИЕ

Прежде чем приступить к рассмотрению, немаловажно понять с чем мы вообще будем иметь дело. Ни один web-сайт не обходится без программных модулей. Web программирование в настоящее время - это то, без чего невозможно создание даже самого простого сайта. Чтобы создать функционально удобный и современный web-сайт, используют различные технические средства, например, HTML, Flash, JavaScript, различные СУБД. В данном пособии речь пойдет о взаимодействии HTML, СУБД PostgreSQL и языка программирования Ruby. СУБД позволяют хранить большое количество данных в специальных базах. Прайсы, отзывы посетителей, описание товаров, фотоальбомы, статистические данные - некоторые из тех возможностей, которые дает web-программирование. HTML покажет всё эторядовому пользователю сети Интернет. Ruby же используется для написания функций, помогающих работать с этой базой данных.

Небольшой экскурс в историю : PostgreSQL ведет свою "родословную" от коммерческой СУБД Postgres, разработанной, как и многие open-source проэкты, в Калифорнийский университет в Беркли. К разработке Postgres, начавшейся в 1986-м году, имел непосредственное отношение Майкл Стоунбрейкер, руководитель более раннего проэкта Ingres, на тот момент уже приобретенного компанией Computer Associates. Само название "Postgrs" расшифровывалось как "Post in Gres соответственно, при создании Postgres были применены многие уже ранее сделанные наработки. Стоунбрейкер и его студенты разрабатывали новую СУБД в течение восьми лет, с 1986 по 1994 год. За этот период в синтаксис были введены процедуры, правила, пользовательские типы и многие другие компоненты. Работа не прошла даром - в 1995 году разработка снова разделилась: Стоунбрейкер использовал полученный опыт в создании коммерческой СУБД Illustra, продвигаемой его мобственной одноименной компанией(приобретенной впоследствии компанией Informix), а его студенты разработали новую версию Postgres - Postgres95, в которой язык запросов POSTQUEL - наследие Ingres - был заменен на SQL. В тот момент разработка Postgres95 была выведена за пределы университета и передана команде энтузиастов. С этого момента эта СУБД получила имя, под которым она известна и развивается в текущий момент - PostgreSQL.

Рассмотрим взаимодействие PostgreSQL и Ruby и некоторые предоставляемые ими возможности на примере базы данных мини-аэропорта. Эта база позволяет нам хранить полеты, авиакомпании, а также исчерпывающую информацию о них. В наши задачи входит модификация эталонного проэкта так, чтобы пользователь мог беспрепятственно добавлять, редактировать, просматривать и искать информацию. В частности, необходимо реализовать добавление, редактирование и просмотр следующих составляющих:

- 1) Посадочных терминалов;
- 2) Стоек регистрации;
- 3) Авиакомпаний;
- 4) Состояний рейса(отложен, отменен, взлетел, посадка т.п.).

Для реализации этих задач нам понадобится помощь PostgreSQL, Ruby и HTML.

## 2. АРХИТЕКТУРА MVC

Структура рассматриваемого проекта подчиняется так называемому принципу *модель-представление-контроллер*. Model - view-controller (MVC) - это архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента, так что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.

Шаблон MVC позволяет разделить данные, представление и обработку действий

пользователя на три компонента: 1) Модель (Model). Модель представляет данные (обычно для View), а также реагирует на запросы (обычно от контроллера), изменяя своё состояние.

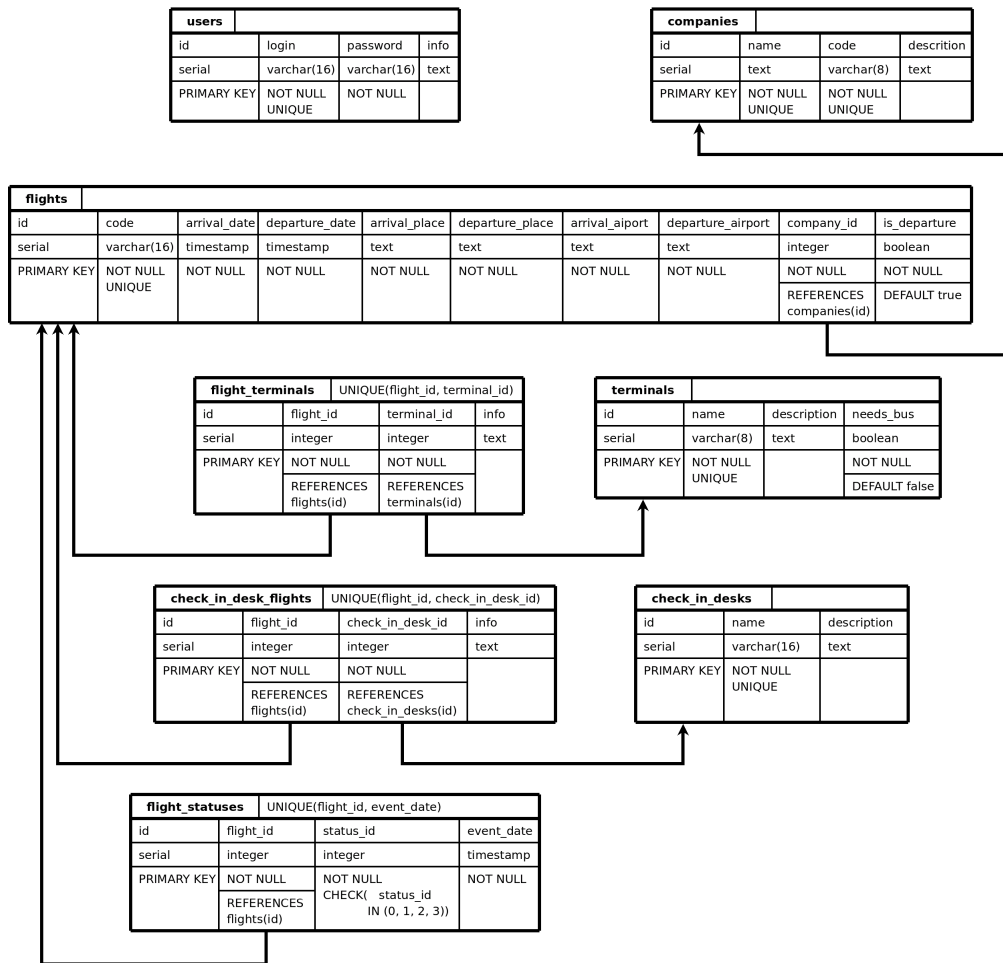
2) Представление (View). Отвечает за отображение информации (пользовательский интерфейс).

3) Поведение (Controller). Интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.



Важно отметить, что как представление, так и поведение зависят от модели. Однако модель не зависит ни от поведения. Это одно из ключевых достоинств подобного разделения. Оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

### 3. СТРУКТУРА БАЗОВОГО ПРОЕКТА



Основной таблицей является отношение `flights`. В нем содержатся следующие атрибуты: `id` полета, код, время отлета рейса, время прибытия рейса, место отлета, место назначения, аэропорта отлета, аэропорт назначения, `id` компании и статус рейса. Данная таблица находится в следующих отношениях:

- М : 1 с таблицей `companies`;
- 1 : М с таблицей `flight_terminals`;
- 1 : М с таблицей `check_in_desk_flights`;
- 1 : М с таблицей `flight_statuses`.

В таблице `companies` находится информация об авиакомпаниях, а точнее: `id` компании, имя, код и описание. Отношение `flight_terminals` описывает посадочные терминалы (`id` посадочного терминала, `id` полета, `id` терминала, дополнительная информация). Оно находится в отношении М : 1 с таблицами `terminals`, которая содержит необходимую информацию об терминалах: `id` терминала, имя, описание и состояние необходимости автобуса для него, и `flights`, описанной выше.

В отношении `check_in_desk_flights` находятся данные об регистрационных стойках. Ее структура очень похожа на структуру `flight_terminals` за исключением того, что вместо `id` терминала тут расположен `id` регистрационной стойки и соответственно она находится в отношении  $M : 1$  с таблицей `check_in_desks` (`id` регистрационной стойки, имя, описание).

Таблица `flights_statuses` содержит информацию об статусе рейса: `id`, `id` полета, статус и дату.

Отдельно можно выделить таблицу `users`, к которой хранятся данные об зарегистрированных пользователях: `id`, имя, пароль, дополнительная информация.

## 4. ОБЩЕЕ ЗАДАНИЕ

**4.1. Постановка задачи.** Реализовать интерфейс для просмотра, редактирования, добавления и удаления следующих составляющих проекта: регистрационных стоек (check - in - desk), посадочных терминалов (terminal), авиакомпаний (company). Также расширить интерфейс по работе с рейсами, добавив туда возможность назначать любому рейсу (удалять назначение, редактировать назначение, просматривать назначение) соответствующие ему регистрационные стойки (check-in-desk-glight), посадочные терминалы (flight-terminal) и события, происходящие с рейсом (flight-status). Под событиями подразумевается следующая информация: рейс отложен, рейс отменен, рейс на посадочной полосе, рейс влетел, рейс сел, идет посадка на рейс на и т.д.

**4.2. Реализация.** 1. *Модель.* С помощью методов модели, в соответствии с концепцией MVC, мы можем модифицировать данные в нашей базе данных. Для этой операции нам необходим объект-посредник между моделью и контроллером, исчерпывающим образом содержащий информацию о рассматриваемом субъекте (стойке, терминале, и т.д.). Этот объект, собственно, и является информационной моделью. Вполне естественно, что этим объектом служит экземпляр модели. Введем конструктор класса. Пусть экземпляр класса тоже включает в себя ассоциативный массив. Этим ассоциативным массивом служит хеш @attributes, определенный для каждого экземпляра класса (если множество аргументов конструктора пусто, все его значения по умолчанию инициализируются nil'ами). Каждый его ключ - символическая константа, поскольку имена атрибутов на протяжении выполнения программы не изменяются. Один такой хеш - модель одной записи из таблицы. С этим хешем сможет работать и контроллер, и представления.

```
def initialize(attributes = {})
  @attributes = {
    :id => nil,
    :name => nil,
    :description => nil
  }
  attributes.each do |k, v|
    @attributes[k] = v
  end
end
```

Иными словами, контроллер и представления работают с экземплярами модели. С помощью вызова из контроллера методов модели с требуемыми аргументами мы можем построить нужные обращения к базе данных.

Сравним организацию ассоциативного массива с реляционной организацией таблицы. Создание таблицы осуществляется классовым методом create\_table():

```
def CheckInDesk.create_table(connection)
  begin
    connection.do("
      CREATE TABLE check_in_desks(
        id serial PRIMARY KEY,
        name varchar(16) UNIQUE NOT NULL,
```

```

        decription text
      ) WITH OIDS
    ")
    return true
  rescue DBI::ProgrammingError => e
    return false
  end
end

```

Отношение создается с внутренними идентификаторами OIDs, являющимися первичным ключом и недоступным извне. SQL-запрос реализуется через интерфейс DBI в секции обработки исключений. В случае генерации одного (таблица уже существует, ошибка в SQL-синтаксисе, неверные параметры и т.д.) метод возвращает false.

Теперь, когда мы организовали связь между моделью и контроллером, следует задуматься о планируемой функциональности. В пользовательском интерфейсе должны быть предусмотрены следующие возможности:

1. Формирование списка всех существующих стоек
2. Добавление, редактирование и удаление объекта
3. Формирование списка всех стоек для рейса
4. Редактирование списка стоек для рейса

Часть этих методов уже реализована в `model.rb`, в частности поиск записи по идентификатору `find_first(connection, id)` и выборка всех кортежей `find_all(connection)`. Этого достаточно для независимой работы со стойками.

Поскольку в моделях пока нет средств обращения к БД с учетом этих связей, реализуем метод `check-in-desk()` в модели `Flights`. На его примере мы также покажем, как сообщаются контролеры модели и производятся запросы к базе.

```

def check_in_desk(connection)
  res = []
  query = ["SELECT cd.*, cdf.id AS cdf_id
            FROM check_in_desks cd JOIN check_in_desk_flights cdf
            ON (cdf.check_in_desk_id = cd.id)
            WHERE cdf.flight_id = ?", self[:id]]
  connection.select_all(*query) do |r|
    f = CheckInDesk.new
    r.column_names.each do |c|
      f[c.to_sym] = r[c]
    end
  end
  res << f
end
return res
end

```

- на входе у функции - идентификатор сессии;
- `res` - массив экземпляров класса `CheckInDesk`;



- `query` - массив, первым элементом которого служит строка SQL-запроса. Передаваемые в неё аргументы заменяются символом "?" после чего последовательно перечисляются в качестве следующих элементов `res`.
- Метод `Action::Base.connection.select_all` генерирует запрос к таблице в соответствии со своим аргументом - массивом `query`, "собранным" в SQL-строку. Заметим, что операция `cid[key]` для экземпляра `CheckInDesk` означает обращение к его хешу `cid.attributes[key.to_sym]` (Данный метод определен в суперклассе `Model` и справедлив для всех его дочерних классов). Т.к. ключи возвращаемого хеша - строковые константы, мы приводим их к символическому виду функцией `to_sym`.

В классе `Model` также определены два метода `table_name()` (для класса и для экземпляра), возвращающие название таблицы в БД, которой соответствует объект или класс.

```
def table_name()
  self.class.table_name()
end
def Model.table_name()
  self.to_s.gsub(/(\W)/, '_\1').downcase + 's'
end
```

Перед промежуточными заглавными буквами имени класс-модели вставляется нижнее подчеркивание, после чего буквы приводятся к строчному виду и дописывается окончание 's' (`CheckInDesk` становится `check_in_desk`). Этот принцип рекомендуется соблюдать при именовании таблиц, в противном случае требуется переопределить метод класса в подклассе.

Если некоторое поле в таблице может принимать ограниченный набор текстовых значений, удобно завести в соответствующем классе хешевую константу, сопоставляющую каждому текстовому значению некоторый идентификатор. С помощью него можно ссылаться на это значение из таблицы.

По этому принципу в модели `FlightStatus` реализованы статусы:

```
STATUSES = {0 => 'отложен',
  1 => 'взлетел'
  2 => 'сел'
  3 => 'отменен'}
...
status_id integer NOT NULL CONSTRAINT status\_id\_ck CHECK (status_id IN (0,1,2,3,4
...

```

Т.о. мы гарантируем, что значение статуса не может быть отлично от всех значений хеша.

Если дан экземпляр класса `FlightStatus`, текстовое представление его статуса можно просмотреть с помощью следующей конструкции:

```
FlightStatus::STATUSES[s[:status_id]]
```

В модуле `Helper` будем реализовывать вспомогательные методы. Например, если пользователь хочет посмотреть список стоек для некоторого рейса, модель выбирает записи из связующей таблицы, где стойки, как мы видели, представлены только их идентификаторами, пользователя не интересующимися.

Метод, возвращающий имя стойки по ее `id`:

```
def convert_to_cid_name(id)
  ans = CheckInDesk.find_first(@db, id)
  return ans[:name]
end
```

При работе с этой моделью мы будем использовать особый тип данных `timestamp`. Этот тип предполагает формат ввода `YYYY.MM.DD HH:MI` (ересь, а приятно...). Мы можем присвоить атрибуту экземпляра значение текущего времени при помощи функции `Ruby Time.now`. *Контроллер*. Форма `HTML` представляет собой документ, созданный с использованием `HTML` элементов. Назначением данной формы является сбор информации от пользователей. После того как пользователь заполнит форму и запускает процесс её обработки, информация из неё попадает в программу, работающую на сервере (скрипту).

Инициализация механизма контроллеров в соответствии с запросом пользователя реализована в `aero.rb`, которая и является нашим скриптом. Содержимое данного куска кода, помогает лучше понять как данные передаются в контроллер.

```
DEFAULT_CONTROLLER = 'Flights'
DEFAULT_ACTION = 'departure_list'
def render()
  cgi = CGI.new('html4')
  begin
    unless cgi.params.include?('controller') or
      valid_controllers.include?(cgi.params['controller'][0])
      cgi.params['controller'] = [DEFAULT_CONTROLLER]
      cgi.params['action'] = [DEFAULT_ACTION]
    end
    c = eval(cgi.params['controller'][0] + 'Controller').new(cgi)
    cgi.out({
      "type" => "text/html; charset=utf-8",
      "language" => "ru"
    }) { c.response() }
    rescue Exception => e
      cgi.out({
        "type" => "text/html; charset=utf-8",
        "language" => "ru"
      }) { display_errors(e) }
  end
end
```

Алгоритм данного скрипта заключается в следующем:

- Создается экземпляр `CGI` для `HTML 4.0`; это ассоциативный массив, через который передаются параметры запроса в виде пар: `key_value_set`, где `value_set` - массив из одного или более значений.

- Метод `has_key?(key)` применяется к хешу и возвращает логическое значение `true` or `false`, в зависимости от того, содержит ли хеш ключ `key`.
- Дописывая к названию контроллера слово `Controller`, получаем название класса-контроллера и создаем его экземпляр, добавив `cgi` в конструктор.
- Метод `cgi.out` пишет `http`-заголовок и тело `s.response()` в вывод. `response()` добавляет к выходному `html`-документу `header` и `footer` из папки `template/layouts`.

Абстрагируясь от реализации конкретных методов, можно сказать, что в целом выполнение метода содержит следующие этапы:

1. Функция `render()` передает значения параметров в `@cgi.params`;
2. Построением методов (экземпляров) класса `Model` или его подклассов производится обращение к соответствующей таблице;
3. Представление обновляется с помощью функции `render_template()`.

Допустим, пользователь хочет посмотреть информацию о некотором рейсе, идентификатор которого передается в контроллер. Ключи `@cgi.params` - строковые константы, а значения `@cgi.params[key]` - массивы (в нашем случае - одноэлементные).

```
def show()
  @item = nil
  if @cgi.params.has_key?('id') and @cgi.params['id'][0] != ''
    @item = Flight.find_first(@db, @cgi.params['id'][0])
  end
  render_template(@item ? 'show' : 'not_found')
end
```

1. Если методу был передан непустой `id`, при помощи метода класса `Flight` находим нужную запись и присваиваем возвращенный ассоциативный массив переменной `@item` (методу передается идентификатор сессии и рейса);
2. Генерируем представление в зависимости от того, был ли найден запрашиваемый рейс.

`@item` - экземпляр модели и переменная экземпляра контроллера, будем использовать в представлении. В базовом классе `Controller` реализован метод `filter_for_params()`, который выбирает из хеша переданных параметров только параметры с названием вида `item[attribute_key]` и удаляет из них названия `item[]`, оставляя только содержимое в `[]`, и возвращает отфильтрованный хеш.

Рассмотрим метод добавления регистрационной стойки для рейса: В хеше параметров передаются идентификаторы рейса и стойки. Этими данными инициализируются значения переменной `@item` (экземпляр `CheckInDeskFlight`), после чего они добавляются в связующую таблицу при помощи вызова `item.save(connection)`.

```
def attach()
  params = filter_for_params()
  @item = CheckInDeskFlight.new
  params.each do |k, v|
    @item[k] = v[0] if k != 'id' and v != ''
  end
end
```

```
@item.save(@db)
render_template(@action)
end
```

- Производим выборку параметров хеш params;
- Создаем экземпляр связующего класса CheckInDeskFlight и заполняем его хеш @attributes переданными значениями. При добавлении в таблицу идентификатор выдается самой СУБД и используется на практике только при извлечении информации (find\_first()), а потом в нашем методе он также игнорируется;
- Запись сохраняется в БД, после чего контроллер обновляет представление.

3. *Представление* Файлы представлений (templates) с точки в Ruby, содержат единственный объект строку. Эта строка - прототип html-кода и потенциальный аргумент функции render\_template(), с помощью которой контроллер обновляет представление после произведенных операций.

```
def render_template(name, mode = :rb)
  if mode == :rb
    f = File.new("templates
/#{Convertors.class_name_to_controller_dir(@controller)}/#{
{name}.rb")
    html = eval(f.read)
    f.close
    return html
  else
    f = File.new("templates/#{@controller}/#{name}")
    html = f.read
    f.close
    return html
  end
end
def Convertors.class_name_to_controller_dir(class_name)
  class_name.to_s.gsub(/(\W)/, '_\1').downcase
end
```

1. Если функция вызывается с одним аргументом, предполагается, что этот аргумент - имя файла-представления без расширения. Полагается что расширением является .rb а путь до файла из каталога templates определяется функцией class\_name\_to\_controller\_dir() (эта функция аналогична уже рассмотренной функции table\_name());

2. Из соответствующего файла считываются данные, после чего интерпретируются и подаются на исполнение. Списки в представлениях формируются следующим способом: Мы можем передать в представление массив экземпляра нашего класса. Для каждого из них будем формировать отдельный html-блок и "пристыковывать" его к уже имеющемуся участку.

```
@items.map do |i|
```

```

count += 1
"
<tr class = 'list#{count%2}'>
<td>#{i[:name]}</td>
<td>#{i[:description]}</td>
<td>
#{action_links(@controller, i[:id], @user)}
</td>
15
</tr>
"
end.join("\n")

```

При создании форм мы можем столкнуться с необходимостью ограничить пользователя в его выборе. Множество стоек тоже ограничено, но постоянно меняется: при "привязке" стойки к рейсу мы должны предложить пользователю только набор существующих на данный момент стоек. Для таких случаев в модуле Helper реализуем функции, формирующие набор нужных опций внутри тега `<select>`. Эти функции возвращают строку с нужным html-кодом. Принцип их работы схож с формированием списков и в подробном объяснении не нуждается.

```

def check_in_desk_select(name, selected, is_nil)
  "<select name = '#{name}'>" +
  CheckInDesk.find_all(@db).map do |c|
    if c[:id].to_i == selected.to_i && selected!=0
      "<option value = '#{c[:id]}' selected>#{c[:name]}</option>"
    else
      "<option value = '#{c[:id]}'>#{c[:name]}</option>"
    end
  end.join("\n") + (is_nil ?
  (selected==0 ?
  "<option value = '' selected> </option>":"<option value=''></option>")
  : "") + "</select>"
end

```

1. На входе у функции - имя параметра, значение которого мы определяем (name), id объекта-значения по умолчанию (selected) и параметр, определяющий, может ли name принять пустое значение (is\_nil);
2. Для каждого объекта из таблицы заносим его имя в список возможных значений; в cgi.params же будем передавать идентификатор;
3. Если selected == 0 и is\_nil == true, значением по умолчанию является пустая строка.

## 5. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

*Постановка задачи.* Полностью реализовать модель "Авиабилет". Данная модель должна содержать следующие пункты:

- Реализовать контроллер для работы с данной моделью.
- Реализовать регистрацию на рейс - при регистрации по номеру паспорта и рейсу автоматически формируется посадочный талон. (Создать модель посадочного талона и необходимый контроллер, талон привязан к рейсу и авиабилету и содержит номер ряда и номер места в ряду. Считаем, что во всех самолетах 20 рядов по 6 мест в каждом ряду (a,b,c,d,e,f) места выдаются по порядку. Когда места кончаются, зарегистрироваться больше нельзя).
- Реализовать удобный просмотр занятых мест для рейса.

**5.1. Примеры работы.** Добавляем новое отношение ticket, со следующими атрибутами (id, flight\_id, passport). Создание таблицы будет осуществляться следующим методом:

```
def Ticket.create_table(connection)
  begin
    connection.do("
CREATE TABLE tickets(
  id serial PRIMARY KEY,
  flight_id integer REFERENCES flights(id) NOT NULL,
  passport varchar(20) NOT NULL
) WITH OIDS
")
```

Определим конструктор и 2 метода, возвращающие рейс на который приобретен билет и талон данного билета:

```
def flight(connection)
  Flight.find_first(connection, @attributes[:flight_id])
end

def coupon(connection)
  Coupon.find_by(connection, 'ticket_id', @attributes[:id])
end
```

добавим представление для данного отношения:

```
<caption>Приобретение авиабилета на рейс #{@flight[:code]}:</caption>
<thead>
<tbody>
<tr>
<th>
  Вылет<br>
  <div style = 'margin-left: 10px;'>город:</div>
  <div style = 'margin-left: 10px;'>время:</div>
  <div style = 'margin-left: 10px;'>аэропорт:</div>
</th>
<td>
```

```

        &nbsp;   <br>
        #{@flight[:departure_place]}<br>
        #{@flight[:departure_date]}<br>
        #{@flight[:departure_airport]}
    </td>
</tr>
<tr>
    <th>
        Посадка<br>
        <div style = 'margin-left: 10px;'>город:</div>
        <div style = 'margin-left: 10px;'>время:</div>
        <div style = 'margin-left: 10px;'>аэропорт:</div>
    </th>
    <td>
        &nbsp;   <br>
        #{@flight[:arrival_place]}<br>
        #{@flight[:arrival_date]}<br>
        #{@flight[:arrival_airport]}
    </td>
</tr>
<tr>
    <th>Авиакомпания:</th>
    <td>#{@flight.company_name()}</td>
</tr>
<tr>
    <th>Рейс является:</th>
    <td>#{@flight[:is_departure] ? 'отлетающим' : 'прибывающим'}</td>
</tr>
<tr>
    <th>Ряд:</th>
    <td>#{@row}</td>
</tr>
<tr>
    <th>Место:</th>
    <td>#{@seat.upcase}</td>
</tr>
<tr>
    <th>Номер паспорта:</th>
    <td><input type = 'text' name = 'item[passport]' size = '30'></td>
</tr>
</tbody>
<tfoot>
    <tr>
        <th colspan = '2'>
            <input type = 'submit' value = 'Приобрести авиабилет'>
            <input type = 'button' value = 'Назад, к списку мест'
                onclick = 'javascript:document.location=\"aero.rb?controller=Tickets'
        </th>
    </tr>
</tfoot>

```

</table>

В данном представлении описывается добавление билета.

6. ПРИМЕРЫ РАБОТЫ

Выход

Регистрационные стойки

Авиакомпании

Вылетающие рейсы

Прилетающие рейсы

Все рейсы

Посадочные терминалы

Аэропорт

Создать новую запись

Номер рейса	Вылет	Посадка	Авиакомпания	Действия
<a href="#">UT005</a>	Баку, Гейдара Алиева: 2010-01-13T08:10:00+00:00	Москва, Шереметьево - 1: 2010-01-13T15:15:00+00:00	ЮТэйр	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">TA007</a>	Бобруйск, Бобруйск: 2010-03-07T23:55:00+00:00	Махачкала, Уйташ: 2010-03-08T07:30:00+00:00	Трансаэро	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">FE008</a>	Дакка, Зиа: 2010-05-11T18:10:00+00:00	Куала-Лумпур, Куала Лумпур Интернешнл: 2010-05-12T00:05:00+00:00	Fly Emirates	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">FE006</a>	Куала-Лумпур, Куала Лумпур Интернешнл: 2010-11-25T06:20:00+00:00	Доха, Доха интернешнл: 2010-11-25T09:10:00+00:00	Fly Emirates	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">AF003</a>	Москва, Шереметьево - 2: 2010-06-17T11:20:00+00:00	Исламабад, Беназир Бхутто: 2010-06-17T20:45:00+00:00	Аэрофлот	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">AF001</a>	Москва, Быково: 2010-10-21T21:35:00+00:00	Смоленск, Смоленск-Южный: 2010-10-22T01:15:00+00:00	Аэрофлот	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">SB002</a>	Москва, Домодедово: 2010-12-31T23:55:00+00:00	Ухта, Ухтинский аэропорт: 2011-01-01T07:45:00+00:00	Сибирь	<a href="#">Редактировать</a> <a href="#">Удалить</a>
<a href="#">TA004</a>	Самара, Курумочь: 2010-08-14T13:50:00+00:00	Москва, Домодедово: 2010-08-14T18:10:00+00:00	Трансаэро	<a href="#">Редактировать</a> <a href="#">Удалить</a>

Готово

vmal3public\_htmlproj\_Aerobodymain.pdf\*sect3.tex (~/publi...Аэропорт - Mozilla ...

Общий список рейсов



Приложения Переход Система

Аэропорт - Mozilla Firefox

Файл Правка Вид Журнал Закладки Инструменты Справка

http://asrv9-ctx-7.msiu.ru/cgi-bin/vma13/aero/aero.rb?controller=Tickets&action=index&flight\_id=7

Аэропорт

**Выход**

- Регистрационные стойки
- Авиакомпании
- Вылетающие рейсы
- Прилетающие рейсы
- Все рейсы
- Посадочные терминалы

**Аэропорт**


**Рейс TA007:**

<a href="#">Место 1A</a>	<a href="#">Бронь</a>	<a href="#">Место 1C</a>	<a href="#">Место 1D</a>	<a href="#">Место 1E</a>	<a href="#">Место 1F</a>
<a href="#">Место 2A</a>	<a href="#">Место 2B</a>	<a href="#">Место 2C</a>	<a href="#">Место 2D</a>	<a href="#">Место 2E</a>	<a href="#">Место 2F</a>
<a href="#">Место 3A</a>	<a href="#">Место 3B</a>	<a href="#">Место 3C</a>	<a href="#">Место 3D</a>	<a href="#">Место 3E</a>	<a href="#">Место 3F</a>
<a href="#">Место 4A</a>	<a href="#">Место 4B</a>	<a href="#">Место 4C</a>	<a href="#">Место 4D</a>	<a href="#">Место 4E</a>	<a href="#">Место 4F</a>
<a href="#">Место 5A</a>	<a href="#">Бронь</a>	<a href="#">Место 5C</a>	<a href="#">Место 5D</a>	<a href="#">Место 5E</a>	<a href="#">Место 5F</a>
<a href="#">Место 6A</a>	<a href="#">Место 6B</a>	<a href="#">Место 6C</a>	<a href="#">Место 6D</a>	<a href="#">Место 6E</a>	<a href="#">Место 6F</a>
<a href="#">Место 7A</a>	<a href="#">Место 7B</a>	<a href="#">Бронь</a>	<a href="#">Место 7D</a>	<a href="#">Место 7E</a>	<a href="#">Место 7F</a>
<a href="#">Место 8A</a>	<a href="#">Место 8B</a>	<a href="#">Место 8C</a>	<a href="#">Место 8D</a>	<a href="#">Место 8E</a>	<a href="#">Место 8F</a>
<a href="#">Место 9A</a>	<a href="#">Место 9B</a>	<a href="#">Место 9C</a>	<a href="#">Место 9D</a>	<a href="#">Место 9E</a>	<a href="#">Место 9F</a>
<a href="#">Место 10A</a>	<a href="#">Место 10B</a>	<a href="#">Место 10C</a>	<a href="#">Место 10D</a>	<a href="#">Место 10E</a>	<a href="#">Место 10F</a>
<a href="#">Место 11A</a>	<a href="#">Место 11B</a>	<a href="#">Место 11C</a>	<a href="#">Место 11D</a>	<a href="#">Место 11E</a>	<a href="#">Место 11F</a>

Готово




vma13 public\_html proj\_Aero body main.pdf \*sect3.tex (~/publi... Аэропорт - Mozilla ...

Список билетов

Приложения Переход Система  16:29

Аэропорт - Mozilla Firefox


Файл Правка Вид Журнал Закладки Инструменты Справка

 <http://asrv9-ctx-7.msiu.ru/cgi-bin/vma13/aero/aero.rb?controller=Coupons&action=show&id=4>  Google 

Аэропорт

Выход	Аэропорт
<a href="#">Регистрационные стойки</a>	Рейс: FE008
<a href="#">Авиакомпании</a>	Вылет: 2010-05-11T18:10:00+00:00 Дакка (Зиа)
<a href="#">Вылетающие рейсы</a>	Посадка: 2010-05-12T00:05:00+00:00 Куала-Лумпур (Куала Лумпур Интернешнл)
<a href="#">Прилетающие рейсы</a>	Номер паспорта: 4806321784
<a href="#">Все рейсы</a>	Место: 5B
<a href="#">Посадочные терминалы</a>	<a href="#">Снять бронь</a>
	<a href="#">Назад, к списку мест</a>

Готово

 vma13 public\_html proj\_Aero body main.pdf \*sect3.tex (~/.publi... Аэропорт - Mozilla ...

Забронированный билет

Приложения Переход Система

Аэропорт - Mozilla Firefox

Файл Правка Вид Журнал Закладки Инструменты Справка

http://asrv9-ctx-7.msiu.ru/cgi-bin/vma13/aero/aero.rb?controller=Tickets&action=new&flight\_id=7&row=3&seat=c

Аэропорт

Выход	Аэропорт																												
<p>Регистрационные стойки</p> <p>Авиакомпания</p> <p>Вылетающие рейсы</p> <p>Прилетающие рейсы</p> <p>Все рейсы</p> <p>Посадочные терминалы</p>	<p>Приобретение авиабилета на рейс TA007:</p> <table border="1"> <tbody> <tr> <td><b>Вылет</b></td> <td>Бобруйск</td> </tr> <tr> <td><b>город:</b></td> <td>2010-03-07T23:55:00+00:00</td> </tr> <tr> <td><b>время:</b></td> <td>Бобруйск</td> </tr> <tr> <td><b>аэропорт:</b></td> <td></td> </tr> <tr> <td><b>Посадка</b></td> <td>Махачкала</td> </tr> <tr> <td><b>город:</b></td> <td>2010-03-08T07:30:00+00:00</td> </tr> <tr> <td><b>время:</b></td> <td>Уйташ</td> </tr> <tr> <td><b>аэропорт:</b></td> <td></td> </tr> <tr> <td><b>Авиакомпания:</b></td> <td>Трансаэро</td> </tr> <tr> <td><b>Рейс является:</b></td> <td>прибывающим</td> </tr> <tr> <td><b>Ряд:</b></td> <td>3</td> </tr> <tr> <td><b>Место:</b></td> <td>D</td> </tr> <tr> <td><b>Номер паспорта:</b></td> <td><input type="text"/></td> </tr> <tr> <td colspan="2"> <p>Приобрести авиабилет    Назад, к списку мест</p> </td> </tr> </tbody> </table>	<b>Вылет</b>	Бобруйск	<b>город:</b>	2010-03-07T23:55:00+00:00	<b>время:</b>	Бобруйск	<b>аэропорт:</b>		<b>Посадка</b>	Махачкала	<b>город:</b>	2010-03-08T07:30:00+00:00	<b>время:</b>	Уйташ	<b>аэропорт:</b>		<b>Авиакомпания:</b>	Трансаэро	<b>Рейс является:</b>	прибывающим	<b>Ряд:</b>	3	<b>Место:</b>	D	<b>Номер паспорта:</b>	<input type="text"/>	<p>Приобрести авиабилет    Назад, к списку мест</p>	
<b>Вылет</b>	Бобруйск																												
<b>город:</b>	2010-03-07T23:55:00+00:00																												
<b>время:</b>	Бобруйск																												
<b>аэропорт:</b>																													
<b>Посадка</b>	Махачкала																												
<b>город:</b>	2010-03-08T07:30:00+00:00																												
<b>время:</b>	Уйташ																												
<b>аэропорт:</b>																													
<b>Авиакомпания:</b>	Трансаэро																												
<b>Рейс является:</b>	прибывающим																												
<b>Ряд:</b>	3																												
<b>Место:</b>	D																												
<b>Номер паспорта:</b>	<input type="text"/>																												
<p>Приобрести авиабилет    Назад, к списку мест</p>																													

Готово

vma13   public\_html   proj\_Aero   body   main.pdf   \*sect3.tex (~/publi...   Аэропорт - Mozilla ...

Шаблон бронирования билета

## СПИСОК ЛИТЕРАТУРЫ

- [1] "Системы баз данных полный курс Г.Гарсиа-Молина, Дж. Ульман, Дж. Уидом, Вильямс, 2003.
- [2] "Введение в системе баз данных К.Дж. Дейт, Вильямс, 2001
- [3] "PostgreSQL для профессионалов Дж. Уорсли, Дж. Дрейк, Питер, 2003
- [4] "Основы PostgreSQL Р. Стоунз, Н. Мэттью, Спб., 2002