CSE 135

Rapid Application Development: Object-Relational Mapping & a lesson on the whys and why nots

Main Problem: Multiple languages and multiple computation servers

- Two different computation servers with two different languages
  - Database server with SQL
  - Application server with Java
  - It will get worse when browser with Javascript enters
- Good: Fine control at all levels
- Bad: too many lines of code spent integrating across languages and computation servers
- A paradox: fundamentally simple computations require too much code

Reducing frictions at the boundaries
How to do db programming with no explicit SQL

Active Record Class Declarations

```ruby
class Customer < ActiveRecord::Base
  set_table_name "customers"
  has_many :orders, :foreign_key => "cust_key"
  belongs_to :nation, :foreign_key => "nation_key"
end

class Order < ActiveRecord::Base
  set_table_name "orders"
  belongs_to :customer, :foreign_key => "cust_key"
  set_primary_key "order_key"
end

class Nation < ActiveRecord::Base
  set_table_name "nation"
  has_many :customers, :foreign_key => "nation_key"
  set_primary_key "nation_key"
end
```

Great savings in number of lines

We implemented subset of the actual graduate admissions application of UCSD in
- Java/JSP/Struts ~2700
- Ruby-on-Rails ~900 – this is a typical gain over Java
Instead of SQL: Navigations across active records

```ruby
<table>
  <% Nation.all.each do |n| %>
    <tr>
      <td><%= n.name %></td>
      <td>
        <table>
          <% n.customers.each do |c| %>
            <tr>
              <td><%= c.name %></td>
              <td><%= c.address %></td>
              <td><%= c.orders.count() %></td>
            </tr>
          <% end %>
        </table>
      </td>
    </tr>
  <% end %>
</table>
```

The SQL queries issued by RoR: Could be many and inefficient!

- **Load Nations**:
  ```sql```
  SELECT * FROM nation
  ```
  One query for every nation key $n$.

- **Load Customers**:
  ```sql```
  SELECT * FROM customers WHERE customers.nation_key = $n$
  ```
  One query for every nation key $n$ and customer key $c$.

- **Load Orders**:
  ```sql```
  SELECT COUNT(*) FROM orders WHERE orders.cust_key = $c$
  ```
  One query for every order key $i$.
Influence/optimize how RoR issues SQL queries by adding hints

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;table&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;% Nation.includes(customers).all.each do</td>
</tr>
<tr>
<td>3</td>
<td>&lt;tr&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;td&gt;&lt;%= n.name %&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;td&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;table&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;% m.customers.each do</td>
</tr>
<tr>
<td>8</td>
<td>&lt;tr&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;td&gt;&lt;%= c.address %&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;td&gt;&lt;%= c.orders.count %&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;/tr&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;/table&gt;</td>
</tr>
<tr>
<td>13</td>
<td>&lt;/table&gt;</td>
</tr>
<tr>
<td>14</td>
<td>&lt;% end %&gt;</td>
</tr>
<tr>
<td>15</td>
<td>&lt;/tr&gt;</td>
</tr>
<tr>
<td>16</td>
<td>&lt;/%&gt;</td>
</tr>
<tr>
<td>17</td>
<td>&lt;/table&gt;</td>
</tr>
</tbody>
</table>

The queries issued upon inclusion of the “includes”

- **Load Nations**
  
  SELECT * FROM nation

- **Load Customers**
  
  SELECT customers.*
  FROM customers
  WHERE (((customers.nation_key = 1) OR (customers.nation_key = 2)) OR (customers.nation_key = 3))

One query for every customer key n.

- **Load Orders**
  
  SELECT COUNT(*) FROM orders WHERE orders.cust_key = c.

But you cannot think of all queries simply as navigations

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;table&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;% Nation.order('name').each do</td>
</tr>
<tr>
<td>3</td>
<td>&lt;tr&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;td&gt;&lt;%= n.name %&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;td&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;% aggregates = Order</td>
</tr>
<tr>
<td>7</td>
<td>.select('order_priority, sum(total_price) as sum_price')</td>
</tr>
<tr>
<td>8</td>
<td>.joins('customer')</td>
</tr>
<tr>
<td>9</td>
<td>.where('nation_key = ?', n.nation_key)</td>
</tr>
<tr>
<td>10</td>
<td>.group('order_priority')</td>
</tr>
<tr>
<td>11</td>
<td>.order('order_priority')</td>
</tr>
<tr>
<td>12</td>
<td>%&gt;</td>
</tr>
<tr>
<td>13</td>
<td>&lt;/tr&gt;</td>
</tr>
<tr>
<td>14</td>
<td>&lt;!--[-- Template for column chart JavaScript component --&gt;]</td>
</tr>
<tr>
<td>15</td>
<td>&lt;/tr&gt;</td>
</tr>
<tr>
<td>16</td>
<td>&lt;/%&gt;</td>
</tr>
<tr>
<td>17</td>
<td>&lt;/table&gt;</td>
</tr>
<tr>
<td>18</td>
<td>&lt;/table&gt;</td>
</tr>
</tbody>
</table>
### Queries issued

**Load Nations**

```sql
SELECT * FROM nation
```

One query for every nation key `n_i`.

**Load Orders**

```sql
SELECT order_priority, SUM(total_price) AS sum_price
FROM orders AS o
JOIN customers AS c ON o.cust_key = c.cust_key
WHERE c.nation_key = <key of n_i>
GROUP BY order_priority
```

### More efficient queries that an SQL guru would have written manually

**Semi Join**

```sql
SELECT n.name, order_priority, SUM(total_price) AS sum_price
FROM orders AS o
JOIN customers AS c ON o.cust_key = c.cust_key
WHERE c.nation_key IN (1,2,3,4,5,6,...,24)
GROUP BY n.name, order_priority
ORDER BY n.name, order_priority
```

**Right Join**

```sql
SELECT n.name, order_priority, SUM(total_price) AS sum_price
FROM orders AS o
JOIN customers AS c ON o.cust_key = c.cust_key
RIGHT JOIN nations AS n ON c.nation_key = n.nation_key
GROUP BY n.name, order_priority
ORDER BY n.name, order_priority
```