

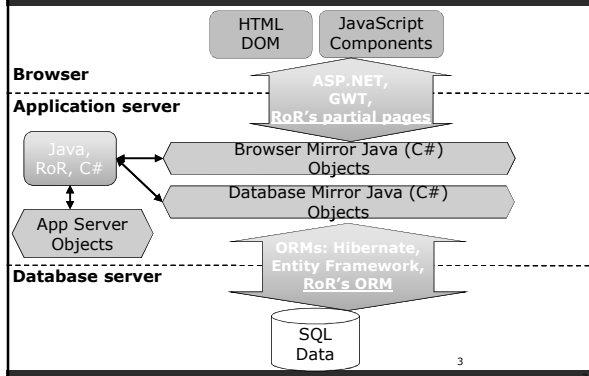
## CSE 135

### 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

```
Nation
1 class Nation < ActiveRecord::Base
2   set_table_name "nation"
3   has_many :customers, :foreign_key => "nation_key"
4   set_primary_key "nation_key"
5 end
```

```
Customers
1 class Customer < ActiveRecord::Base
2   set_table_name "customers"
3   set_primary_key "cust_key"
4   has_many :orders, :foreign_key => "cust_key"
5   belongs_to :nation, :foreign_key => "nation_key"
6 end
```

```
Orders
1 class Order < ActiveRecord::Base
2   set_table_name "orders"
3   belongs_to :customer, :foreign_key => "cust_key"
4   set_primary_key "order_key"
5 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

```
1 <table>
2   <% Nation.all.each.do |n| %>
3   <tr>
4     <td><%= n.name %></td>
5     <td>
6       <table>
7         <% n.customers.each.do |c| %>
8         <tr>
9           <td><%= c.name %></td>
10          <td><%= c.address %></td>
11          <td><%= c.orders.count() %></td>
12        </tr>
13      <% end %>
14    </table>
15  </td>
16 </tr>
17 <% end %>
18</table>
```

---

---

---

---

---

---

---

---

|             |                  |                     |     |
|-------------|------------------|---------------------|-----|
| Argentina   | Diego Maradona   | 43 Via Mallorca     | 256 |
|             | Lionel Messi     | 501 Gilman          | 4   |
| Netherlands | Marco van Basten | 26 La Jolla Village | 0   |
|             | Arjen Roben      | 78 Torrey Pines     | 4   |

---

---

---

---

---

---

---

---

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

(Load Nations) SELECT \* FROM nation

One query for every nation key  $n_i$

(Load Customers) SELECT \* FROM customers WHERE customers.nation\_key =  $n_i$

One query for every nation key  $n_i$  and customer key  $c_i$

(Load Orders) SELECT COUNT(\*) FROM orders WHERE orders.cust\_key =  $c_i$

---

---

---

---

---

---

---

---

## Influence/optimize how RoR issues SQL queries by adding hints

```

1 <table>
2   <% Nation.includes(:customers).all.each.do |n| %>
3   <tr>
4     <td><%= n.name %></td>
5     <td>
6       <table>
7         <% n.customers.each.do |c| %>
8         <tr>
9           <td><%= c.address %></td>
10          <td><%= c.orders.count() %></td>
11        </tr>
12        <% end %>
13      </table>
14    </td>
15  </tr>
16  <% end %>
17</table>

```

---

---

---

---

---

---

---

---

---

---

## The queries issued upon inclusion of the “includes”

(Load Nations) SELECT \* FROM nation

(Load Customers) SELECT customers.\*  
FROM customers  
WHERE (((customers.nation\_key = 0) OR  
(customers.nation\_key = 1) OR  
(customers.nation\_key = 2) OR  
... OR  
(customers.nation\_key = 24)))

Reasonable  
underlying  
assumption  
that outer  
loop does not  
have too  
many entries

One query for every customer key c<sub>i</sub>

(Load Orders) SELECT COUNT(\*) FROM orders WHERE orders.cust\_key = c<sub>i</sub>

---

---

---

---

---

---

---

---

---

---

## But you cannot think of all queries simply as navigations

```

1 <table>
2   <% Nation.order('name').each.do |n| %>
3   <tr>
4     <td><%= n.name %></td>
5     <td>
6       <%
7         aggregates = Order
8           .select('order_priority, sum(total_price) as sum_price')
9           .joins('customer')
10          .where('nation_key = ?', n.nation_key)
11          .group('order_priority')
12          .order('order_priority')
13        %>
14      <!-- Template for column chart JavaScript component -->
15    </td>
16  </tr>
17  <% end %>
18</table>

```

Show for each nation the sum of  
order prices for each order priority,  
ordered by the order priority

---

---

---

---

---

---

---

---

---

---

## Queries issued

```
(Load Nations) SELECT * FROM nation
```

One query for every nation key  $n_i$

```
(Load Orders) 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) 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) 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
```

---

---

---

---

---

---

---

---