So the main thing I want you to take away from this talk is...
Please don’t do it!
Questions?
The point of this talk is to show you that there are ways to work with legacy systems, but that you really, should try to work around them.

Just kidding.
Who’s here?

How many people here are new to Rails?
How many people have to work with legacy databases?
How many people want to learn how to work with legacy?
This is a topic that’s close to me because I work days at a university that uses Oracle and SQL Server databases. In order to use Rails, there were a lot of things I had to figure out.
A legacy database could be anything that existed before Rails, like a wordpress blog or a custom project management app.
Anything that doesn’t follow ActiveRecord conventions

It’s important to note that these aren’t Rails limitations, but rather ActiveRecord limitations.
This is going to be boring, isn’t it?

So, we’re going to talk about legacy schemas and Rails. Things like bad table names, non-incrementing primary keys, stored procedures, compound keys
I bet Obie’s business panel is awesome.

We’ll look at some simple fixes first, like overriding table names and primary keys...
OR THAT MUSICAL PATTERNS ONE...

and then look at some other ways we can make our code easier to read and write while still leveraging the power of Rails
SORRY.
Simple(er) solutions with ActiveRecord

ActiveRecord provides mechanisms to handle some simple cases for us.
Simple problems

Person

<table>
<thead>
<tr>
<th>person_id</th>
<th>firstname</th>
<th>lastname</th>
<th>middlename</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
</table>

Singular table name and prefixed ID
Simple solutions

Person

class Person < ActiveRecord::Base
  set_table_name "person"
  set_primary_key "person_id"
end
Bad Column Names

Person

person_id
person_name
person_email
person_sex

Old-School Prefixes
Rails’ alias_attribute method is a macro for redefining the method at runtime. However, it only works with fieldnames that don’t violate Rails’ conventions.
Sometimes the stuff that ActiveRecord gives you just isn’t enough.
Really Bad Column Names

Prefixes!

Yikes!!

Person

Person id
Person Name
Person e-mail
Person sex

Some databases (SQL SERVER) let you have column names like this. Soaces, dashes, mixed case.
Use a View!

Views to the rescue. MySQL, Microsoft SQL Server, and Oracle support inserting into views too, so your Rails database adapter will never know.
Can anyone give me examples of non incrementing keys? Some good examples?
Try to add them.

Not always possible if there’s a certain level of beaurocracy in your institution. Some people also just think meaningful keys are the most bestest idea.
This example is BAD BAD BAD. If you don’t know why, then please close your laptop and go sit in the corner.
Let’s take a look at using Rails with an existing wordpress database.
This is incredibly common. One of our databases joins 5 columns for course records.
Dr. Nic to the rescue.

http://compositekeys.rubyforge.org/
class EnrollmentRecord

    set_primary_keys :course_id, :section_id, :term_id

end

EnrollmentRecord.find(@course.id, @section.id, @term.id)
Establish a connection

database.yml

```yaml
wordpress:
  adapter: mysql
  database: wordpress
  username: app
  password: your_password
  host: localhost
```
Establish a connection

environment.rb

WordpressPost.establish_connection "wordpress"
Wordpress and Rails can work well together.

class WordpressPost < ActiveRecord::Base

  set_table_name "wp_posts"
  set_primary_key "ID"

  has_many :comments,
            :class_name => "WordpressComment",
            :foreign_key => "comment_post_ID"

  def self.find_by_permalink(year, month, day, title)
    find(:first,
         :conditions => ["YEAR(post_date) = ?
                         AND MONTH(post_date) = ?
                         AND DAYOFMONTH(post_date) = ?
                         AND post_name = ?",
                         year.to_i, month.to_i, day.to_i, title])
  end
end
Foreign Keys are easy

```ruby
has_many :comments,
  :class_name => "WordpressComment",
  :foreign_key => "comment_post_ID"
```
Working with Legacy Data
Some public directory information from a university. It’s fed from a Unisys mainframe on a nightly basis using flatfiles.
Getting even this was a challenge!
THIS IS NOT ALL THE DATA.

The feed also includes social security numbers. We have to protect those
Data security

- Secure internal network between the database server and the mainframe
- Replication to the application database server using Microsoft DTS packages
- Access restricted at the table level
- Access granted to primary developers via views
- Access granted to third-party developers by stored procedures.
Never forget that views are a security tool. Aside from the benefit they give us with regard to changing how data looks, we can use them to abstract availability of data. I *never* see these used in Rails.
Views are often a performance gain!

In some databases you can use views to cut down complex join logic and still treat them like tables. MySQL doesn’t allow subselects, but MS SQL sure does.
Sometimes logic is encapsulated within the database.
There's no direct support for stored procs in ActiveRecord...
...BUT THE CONNECTION CLASS WILL WORK!
Call a procedure

connection.execute("add_user 'Homer', 'Simpson', 'homer@simpsons.com'")

This throws the results away, but it's fine if you just want to call something that has no return value. Execute is good for calling any arbitrary SQL, but
Get a return value

connection.select_value("get_open_tickets 10")

This will get you a single return value from a procedure.
Results come back as an array of hashes, so you access them a little differently, but it’s still pretty nice. A little meta-magic and you can map these to properties of a new instance.
There is no escape!
This declares properties and methods. This has lots of potential.
Biggest problem with Rails and Legacy is “automagical code”

If you can declare your columns like you can with DataMapper or even with a view, you’ve solved the major problems.
DataMapper
Drawbacks?
Doesn’t work well on Windows yet

Cygwin and other methods, but it’s known to be slow.
No Microsoft SQL Server support.

A call for help. I know virtually nothing about DataMapper, but a lot about SQL Server. There’s been interest, but no volunteers yet. I’m tempted but I need help.
What are some strategies you can use to move from legacy systems?
Where are you coming from and where are you going?

If you’re going to move from one platform to another, you need to know what the differences between the platforms.
Each database manufacturer has different types.
NUMBER datatype is used for integers and floats.
DATE is used for Time AND Date.
NUMBER and CHAR can both be used for :BOOLEAN!
Raimonds is a pretty swell guy, and he’s done some great work on this driver which makes working with Oracle very nice.
SQL Server

SQL Server 2000 and 2005 are very different.
TEXT fields with AR are really troublesome.

They’re deprecated anyway. If you have to use a database that uses this type, you have to watch out for statements that use equality.
This solves a lot of stuff. Different datatypes, date fixes, and excellent stored procedure support.
Finally, watch out for triggers!

Legacy databases tend to have a lot of these. You need to write code to handle exceptions that your ORM classes might throw if a trigger causes constraints to be violated or if triggers start doing strange things.
Dealing with Constraints

• Avoid using fixtures for tests. They’re bad news anyway

• Use .new() and stubs

• Duplicate validation / constraints in your code

• Use exception handling

• Use Stubble - [http://github.com/dchelimsky/stubble](http://github.com/dchelimsky/stubble)
MIGRATING TO A NEW DBMS

- rake db:schema:dump, and then paste that in as a new base migration

- sometimes columns aren’t usable this way. Dump schema as SQL and use Regex instead to convert to Ruby.
MS SQL Databases

• Tables within different databases on the same server can be joined as if they are in the same database.