

Active Record: Working with Databases in Rails

Discussion: Working with databases

How do you work with databases today?

- How do you create a new table?
- How do you insert data into your table?
- How do you read data from your table?
- Does each developer have his/her own database?
- etc.

The Rails Way: Working with Databases

- Manage the schema through Migrations
- Use ActiveRecord

Overview: Working with Databases

- Compare the Rails Way with some Java examples
- Create Migrations & Models
- Create Data
- Associations
 - has_many
 - belongs_to
- Retrospective

Does this SQL code look familiar?

```
CREATE TABLE `people` (  
  `personId` int(11) DEFAULT NULL auto_increment PRIMARY KEY,  
  `firstName` varchar(255) DEFAULT NULL NULL,  
  `lastName` varchar(255) DEFAULT NULL NULL,  
  `updatedAt` datetime DEFAULT NULL NULL  
) ENGINE=InnoDB
```

Does this Java code look familiar?

```
private Client getPerson(long personId, Connection conn) throws SQLException {
    ResultSet rs = null;
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement("SELECT * FROM people WHERE personId = :personId");
        stmt.setLong(1, personId);
        rs = stmt.executeQuery();
        if (rs.next())
            return new Person(
                rs.getLong("personId");
                rs.getString("firstName");
                rs.getString("lastName");
                rs.getDate("updatedAt");
            );
        else
            return null;
    } catch (SQLException e) {
        log.error(e);
        throw e;
    } finally {
        if (rs != null)
            rs.close();
        if (stmt != null)
            stmt.close();
    }
}
```

Does this Java code look familiar?

```
private void createPerson(long personId, Person person, Connection conn)
throws SQLException {
    Date updatedDate = new Date(new java.util.Date().getTime());
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement("INSERT INTO people VALUES(:id, :firstName, :lastName, :updatedDate)");
        stmt.setString(1, person.getId());
        stmt.setString(2, person.getFirstName());
        stmt.setString(3, person.getLastName());
        stmt.setDate(4, updatedDate);
        stmt.executeUpdate();
    } catch (SQLException e) {
        log.error(e);
        throw e;
    } finally {
        if (stmt != null)
            stmt.close();
    }
}
```

Do you use Hibernate?

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" -->

<hibernate-mapping default-lazy="false">
<class name="com.company.referencelists.data.Education" table="PIK_SCHOOL_V">
<id name="schoolID" column="SCHOOL_ID" type="java.lang.Long"/>
<property name="formalSchoolName" column="FORMAL_SCHOOL_NAME" type="java.lang.String" />
<property name="formalCollegeName" column="FORMAL_COLLEGE_NAME" type="java.lang.String" />
<property name="displayValue" type="java.lang.String"
formula="FORMAL_SCHOOL_NAME ||
decode(FORMAL_COLLEGE_NAME, NULL, NULL, ' - ' ||
FORMAL_COLLEGE_NAME)"
/>
<property name="countryCode" column="COUNTRY_CODE" type="java.lang.String" />
</class>
</hibernate-mapping>
```

How does Rails think about databases

- Database Schema
 - A part of your application
 - Checked in with the rest of your code
 - Run as part of an application deployment
 - Written in Ruby!!
 - Called **Rails Migrations**
- Much of inserting, updating, selecting is ceremony
 - If we follow conventions we can stay DRY
 - What columns are on a table?
 - Get all records in a table
 - Get one record in a table
 - Called **ActiveRecord**

Migrations

- Created by generators
- `ruby script/generate migration AddPerson`
- Let's look at the file it created

db/migrate/20100414092219_create_people.rb

```
class CreatePeople < ActiveRecord::Migration
  def self.up
    create_table :people do |t|
      t.string :first_name
      t.string :last_name

      t.timestamps
    end
  end

  def self.down
    drop_table :people
  end
end
```

Note: Use local railsbrain to view other methods in migrations

Running a migration

- `rake db:migrate`
- Take a look at your database
- What do you see?

Migrations

- The database knows what migrations have been run
 - `schema_migrations` table
 - Keeps versions of the code and versions of the schema synchronized
- Migrations can be applied or rolled back
 - `rake db:migrate`
 - `rake db:rollback`

app/models/person.rb

```
class Person < ActiveRecord::Base  
end
```

app/models/person.rb

- That's it?!?!?
- It can't do much!
- Let's see...
- `ruby script/console`

Script/Console (irb configured for Rails)

In DOS type

```
ruby script/console
```

Type each line below and see what happens.

```
Person
```

```
Person.create
```

```
Person.create(:first_name => 'Greg')
```

```
Person.all
```

```
Person.first
```

```
Person.find(1)
```

```
Person.find_by_first_name('Greg')
```

Exercise

How do you create a person record for yourself?

Create your record in the database and find it by your last name

How does ActiveRecord do it?

- Convention
 - model 'Person' will talk to the table 'people'
 - getters and setters for each column
 - `find_by_XXX` for each column
- By staying DRY
 - Don't need to keep repeating column names
- Try these in script/console

```
Person.columns
Person.columns_hash['first_name']
Person.columns_hash['first_name'].sql_type
Person.columns_hash['first_name'].type
```

Exercise: Adding methods to Models

- Create a new migration and add an 'age' column to your model.
- Create a new record in your database with your age

Associating Two Models to Each Other

- We got a new requirement "A Person should have many hobbies"
- We already have the Person model let's create the Hobby

- `ruby script/generate migration AddHobby`

- Exercise: Create the table with a name column that is a string and a `person_id` column that is an integer

Has_many associations

To implement the requirement "An Person should have many hobbies"

We add the bold line to app/models/person.rb

```
class Person < ActiveRecord::Base
  has_many :hobbies
end
```

script/console - looking at hobbies

- run rake db:migrate
- Let's go back into our script/console session
- Type each line below and see what happens.

```
person = Person.first  
person.hobbies
```

```
beer = Hobby.create!(:name=>'beer')  
person.hobbies << beer  
Person.hobbies
```

Exercise

Create a hobby for your record in the database

Belongs_to associations

We add the bold line to app/models/hobby.rb

```
class Hobby < ActiveRecord::Base
  belongs_to :person
end
```

Console session

- Let's go back into our script/console session
- Type each line below and see what happens.

```
beer = Hobby.first  
beer.person  
beer.person.hobbies
```

Retrospective: The Rails Way of Working with Databases

- What useful ideas did you get out of this session?
- What would you like to understand better?
- What would you suggest adding to the Rails Way of Working with Databases?

Additional Resources

- <http://guides.rubyonrails.org/migrations.html>