Auditing plugin

by Mihai Târnovan and Gabriel Târnovan

Mihai Târnovan is a partner at Cubus Arts, *http://cubus.ro/*. He found Ruby On Rails 5 years ago, and he liked it so much that he converted all his co-workers from the worlds of Java and PHP. He is currently interested in developing his company's SaaS products for the romanian market, as well as finding new ideas to implement and develop. Mihai is interested in everything related to tech entrepreneurship and spends too much time reading Hacker News.

Contact at mihai.tarnovan[at]cubus.ro or *http://cubus.ro/*

Gabriel Târnovan is a partner at *Cubus Arts*, a web development shop based in Sibiu, Romania where he currently focuses on becoming a Ruby on Rails ninja and giving something back to the local programmer community. At *Cubus Arts* he works to deliver quality products, especially SaaS solutions for the local market.

Contact at gabriel.tarnovan[at]cubus.ro or *http://cubus.ro/*

Auditing is a broad term, but when talking about web applications it usually refers to keeping track of who did what, and when. Auditing changes in models is a common requirement for web applications. For Rails there are several solutions which provide this functionality. We'll be looking at two of the best solutions available - acts_as_audit and paper_trail - and introduce simple_audit, a straightforward approach focusing on human readable audit trails.

Implementing Auditing with Rails

Implementing an auditing solution for ActiveRecord models is pretty straightforward due to the many interception points ActiveRecord provides in the lifecycle of a record with callbacks, observers and cache sweepers. Every change to a record can thus easily be tracked and stored in the database.

```
class AuditObserver < ActiveRecord::Observer
  observe :person, :account

  def after_update(record)
    AuditTrail.new(record, "UPDATED")
  end
end
```

As you can see, the problem with using observers is that the user who performed the update is not accessible. Sweepers don't have this problem, as they can access the session through the controller, but they will only track changes originating in controller actions. To work around this problem when employing observers/callbacks, many solutions use `Thread.local` to store the currently logged user, making the data globally available.

There are two main approaches for storing the audited records' changes:

- one additional audit table for each audited model; audit tables will have the same structure as the audited tables
- a single audit table for all audited models; tracked changes are stored in a serialized form

Using the first approach allows one to issue database queries directly on the audited attributes of a model; this can not be done efficiently with a serialized form used by the second approach.

The second approach is used more widely due its flexibility:

- any number of models can be audited
- changes to the attributes of the audited models don't require any updates to the audits table or to the already audited changes
- database backups / exports can be managed easier
- smaller footprint - only the changed attributes can be stored, not the entire record

Versioning solutions are very similar to auditing solutions; core functionality is basically the same. This is why versioning systems are sometimes also employed for the auditing system of an application or vice versa. See vestal_versions (*http://github.com/laserlemon/vestal_versions*) for a solid versioning solution.

When relying on ActiveRecord's hooks in the lifecycle of a record to perform auditing, one should keep in mind that some operations will not trigger callbacks and will have to be audited separately. Such operations include direct queries to the database, methods that don't instantiate the records like ActiveRecord#delete_all, #update_all, updating counter caches on associations, etc.

Other Auditing Solutions for Rails

acts_as_audit (*http://github.com/collectiveidea/acts_as_audited*) is an established solution for auditing in Rails, having been around since Rails 1.2. It stores each change to a model's attributes in an audit table. The initial value is also stored, making each audit entry self-contained. One can specify which attributes should be audited; the *current_user* method of the controller is used to get the acting user. A list of versions is available for each audited records.

paper_trail (*http://github.com/airblade/paper_trail*) is a good solution for both auditing and versioning your models. Besides the functionality provided by acts_as_audited, it allows you to store arbitrary controller-level information (e.g. remote IP, user agent, etc) and also arbitrary model-level metadata with each version. Metadata fields must have their respective columns in the audits table; this allows audits to be quickly filtered using SQL rather then deserializing all data and then filtering it.

Keeping it simple: simple_audit

simple_audit (*http://github.com/gtarnovan/simple_audit*) is an auditing solution for Rails primarily intended to be used when the human readable representation of the changes is the main reason for implementing the audit. It handles everything from storing record changes to displaying a human readable audit trail.

Many applications need auditing just to show to a privileged user details about actions performed on the database records. Versioning, storing all changed attributes - especially database internal data like record ids - are not as important as displaying a human readable activity feed in a form that is easy to interpret.

simple_audit encourages the developer to generate a readable representation of the relevant attributes of a model. This is especially helpful when tracking changes on the parent of a complex aggregation or on a composite record (see aggregation vs composition - *http://en.wikipedia.org/wiki/Object_composition#Aggregation*). Tracking changes on all records in the database is required in certain situations, but it gets complicated if one has to rebuild an aggregation of several audited records just to display a readable audit log.

Let's look at an example:

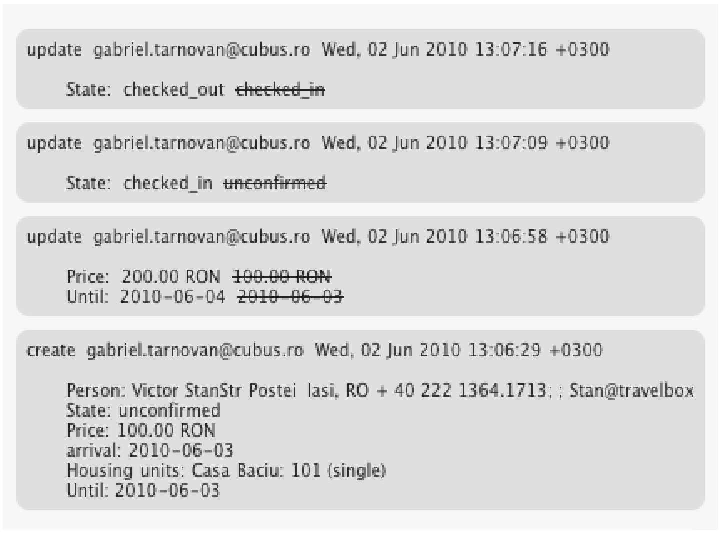
```
class Person < ActiveRecord::Base
  belongs_to :booking
  has_one :address
end

class Room < ActiveRecord::Base
  has_many :room_bookings
  has_many :bookings, :through => :room_bookings
end

class RoomBooking < ActiveRecord::Base
  belongs_to :room
  belongs_to :booking
end

class Booking < ActiveRecord::Base
  has_one :person
  has_many :room_bookings
  has_many :rooms, :through => :room_bookings
  simple_audit do |booking|
    # this is a human readable representation of
    # the relevant attributes
    {
      :person => "#{booking.person.full_name}
#{booking.address.to_s}",
      :housing_units =>
        booking.rooms.collect(&:name).join('; '),
      :arrival => booking.arrival, ...
        # other relevant data
    }
  end
end
```

The audit trail generated by the supplied view helper will look like this:



update gabriel.tarnovan@cubus.ro Wed, 02 Jun 2010 13:07:16 +0300

State: checked_out ~~checked_in~~

update gabriel.tarnovan@cubus.ro Wed, 02 Jun 2010 13:07:09 +0300

State: checked_in ~~unconfirmed~~

update gabriel.tarnovan@cubus.ro Wed, 02 Jun 2010 13:06:58 +0300

Price: 200.00 RON ~~100.00 RON~~
Until: 2010-06-04 ~~2010-06-03~~

create gabriel.tarnovan@cubus.ro Wed, 02 Jun 2010 13:06:29 +0300

Person: Victor StanStr Postei Iasi, RO + 40 222 1364.1713; ; Stan@travelbox
State: unconfirmed
Price: 100.00 RON
arrival: 2010-06-03
Housing units: Casa Baciu: 101 (single)
Until: 2010-06-03

In this context, changes made to booking records are central to the application domain. RoomBooking and Person records are relevant only in the context of a booking, therefore auditing them separately would make little sense and would require recomposing data from several audit entries just to show a single change in a booking.

simple_audit works on Rails 2.3.x and Rails 3.

Future developments of simple_audit

Support for other ORMs is planned to be implemented until end of november. This will include support for DataMapper and MongoDB.

Another planned development is auditing controller actions. Not every action results in a change to the database, but even such actions are sometimes worth tracking (e.g. login & logout, searches, sending email, etc).

Tracking database updates and controller actions would cover most auditing needs for web applications.

Integration of Aquarium AOP library (*http://aquarium.rubyforge.org/*) to enable auditing of any relevant method call will also be investigated.

Resources

You can get simple_audit from its github page
http://github.com/gtarnovan/simple_audit

Aspect Oriented Programming with Aquarium
http://aquarium.rubyforge.org/