RUBY STORIES

Wifi pass: xx

Toilet is upstairs

Food & drink

Heating

Rock-Solid Migrations

David Hrachovy, CEO at PrimeHammer

david@primehammer.com

Introduction

- expected knowledge
- everyone writes migrations, db holds valuable data
- schema and data migrations

About me

hlídačky.cz
Potřebujete hlídání dětí?
5520 hlídaček na výběr!
Kde sháníte hlídání?
Nejoblíbenější: Praha Brno Olomouc Plzeň Liberec Ostrava

PRIMEHAMMER

Specific examples

Write efficient migrations

Bad code

```
class ChangeStatus < ActiveRecord::Migration
  def up
    Product.find(:all) do |p|
      p.update_attributes(status: 10)
    end
  end
end
```

Good code:

```
class ChangeStatus < ActiveRecord::Migration
  def up
    Product.find_each do |p|
      p.update_attribute(:status, 10)
    end
  end
end
```

Better code:

```
class ChangeStatus < ActiveRecord::Migration
  def up
    Product.update_all(status: 10)
  end
end
```

Even better code:

```
namespace :db_maintenance do
  desc 'Fix product status'
  task fix_product_status: :environment do
    Product.update_all(status: 10)
    puts 'done.'
  end
end
```

- TIP: test suspicious data migration with large tables

Refactor

Bad code:

```ruby
class AddNewCountToUsers < ActiveRecord::Migration
  def up
    add_column :users, :new_count, :integer
    execute "UPDATE users SET new_count = count;"
    remove_column :users, :count
  end
end
```

Good code:

```ruby
class AddNewCountToUsers < ActiveRecord::Migration
  def up
    rename_column :users, :count, :new_count
  end
end
```

Use reversible methods

Bad code:

```
def up
  remove_column :people, :name
end

def down
  add_column :people, :name, :string
end
```

Good code:

```
def change
  remove_column :people, :name, :string
end
```

Using ActiveRecord DSL instead of raw SQL is good

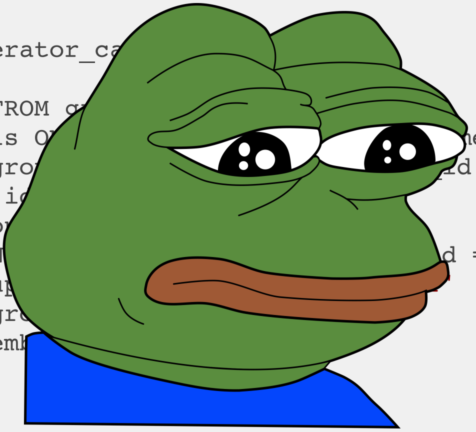
List of reversible methods

```
add_column
add_foreign_key
add_index
add_reference
add_timestamps
change_column_default # (must supply a :from and :to option)
drop_table # (must supply a block)
remove_column # (must supply a type)
remove_foreign_key # (must supply a second table)
...
and more
```

Complex data migrations

```
def up
  execute <<-SQL
    UPDATE groups SET moderator_ca
      WHERE id IN (
        SELECT groups.id FROM g
        INNER JOIN controls O                        ent_g
        INNER JOIN  user_gro                       d = us
        WHERE user_groups.i
          (SELECT user_gro
           LEFT OUTER JOIN                         d = mem
           WHERE user_group
           GROUP BY user_gr
           HAVING COUNT(memb
    SQL
end

def down
  execute "UPDATE groups SET moderator_can_access = 'f';"
end
```



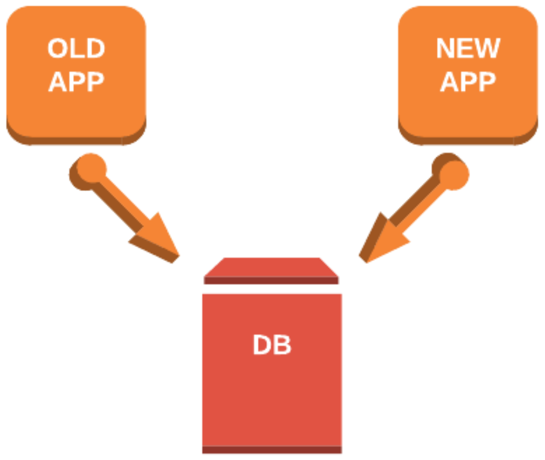
Complex data migrations

- always write test for multiple scenarios
- test it on staging
- explicitely give QA person instructions
- do backup
- write `down` method and test rollback
- document the intentions (code review)
- rake task

Downtime

Problem during deploy

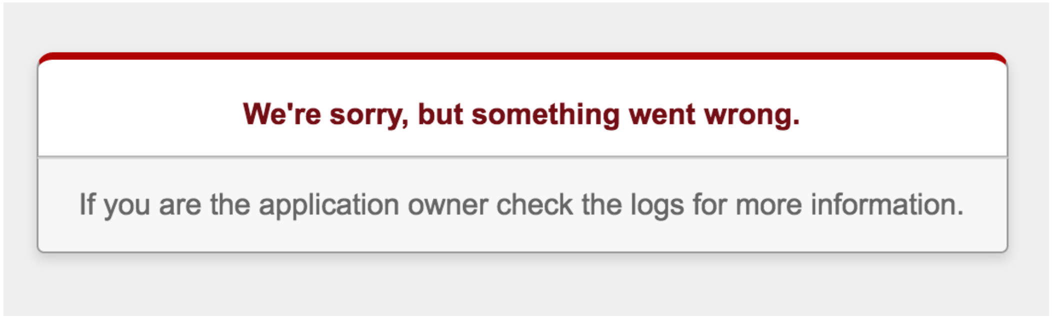
Typical deploy: run migrations & restart servers



Result: two versions of an app at the same time

Example: Dropping a column

```
class RemoveTitleFromUsers < ActiveRecord::Migration
  def change
    remove_column :users, :title, :string
  end
end
```


We're sorry, but something went wrong.

If you are the application owner check the logs for more information.

- old app may still save data into the column therefore raises `undefined method title`

FIX: maintenance mode or zero downtime migration

One of the solutions

Any migration being deployed should be compatible with the code that is already running.

General steps:

- make code compatible with migration you need to run, deploy
- run migration

Safe column drop

1. remove parts of code that touch title

   > ```
   > <%= user.title %>
   > ```

2. Deploy

3. Run the migration

> ```
> class RemoveTitleFromUsers < ActiveRecord::Migration
>   def change
>     remove_column :users, :title, :string
>   end
> end
> ```

Some unsafe migrations

```
changing the type of a column
renaming a table
renaming a column
removing a column
and more
```

avoid premature optimization

(maybe clean up DB every 6 months)

When something goes wrong

If on production

- Tell someone as soon as possible
- don't modify already pushed migration on production
- check other environments

If not on production

- Warn everybody (rebuild db)
- Delete the migration from source control

Wrap up

Checklist

- it's efficient
- it's reversible
- it's small
- old app can use it
- complex data migration is well tested
- code review
- use staging

References

No More Lost Data by Noah Gibbs

Strong migrations used by Instacart

http://www.simononsoftware.com/why-ruby-on-rails-migrations-dont-work/

The End. Questions?

david@primehammer.com