Jaywalking in Traffic

Safe Migrations at Scale

Brad Urani
Staff Engineer

PROCORE™
CLOUD-BASED CONSTRUCTION SOFTWARE

What is Scale?

20,000,000 rows fetched / sec
30,000 transactions / sec
6 TB


Changing the landscape of construction software
Stop managing systems and start empowering your teams.
Watch the Video
INTRODUCING THE JOBSITE PROCORE'S HUB FOR ORIGINAL CONTENT
PROCORE
SALES CHAT SESSION

+

PostgreSQL

+

People

Squad

- ✓ “Feel like a mini-startup”
- ✓ Self-organizing
- ✓ Cross-functional
- ✓ 5-7 engineers, less than 10
- ✓ Stable

14 Squads working on one of the biggest Rails apps in the world

Goals

- 5-10 Deploys per day
- Web devs are responsible for their own schema changes
- Minimal involvement by the Site Reliability squad

- Add / Remove Tables
- Add / Remove Columns
- Adding Constraints (unique, not null, FK)
- Add Indexes
- Adding Defaults
- Moving Data

```
> rails generate migration create_products
```

```ruby
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.text :description
      t.timestamps
    end
  end
end
```

```
procore/
| .capistrano/
| .git/
| app/
| bin/
| config/
| db/
| | initial_projects/
| | migrate/
| | | 20101129202324_rename_destroyed_on_punch_item_change.rb*
| | | 20101210100000_add_draft_to_meetings.rb*
| | | 20101213215844_create_submittal_log_types.rb*
| | | 20101216212843_expire_thumbnails.rb*
| | | 20110107181838_vendor_contact_relationship.rb*
| | | 20110114200341_add_position_to_unr.rb*
| | | 20110118002045_update_custom_option_foreign_keys.rb*
| | | 20110120221939_fix_rfi_time_resolved.rb*
| | | 20110121194455_add_company_id_to_insurance.rb*
| | | 20110124170100_remove_civics.rb*
| | | 20110124190158_add_avatar_to_contact.rb*
| | | 20110126191734_add_materials_retainage.rb*
| | | 20110126212112_add_avatar_to_company_directory.rb*
| | | 20110127234642_add_show_line_items_option_to_project.rb*
| | | 20110201014824_remove_changes.rb*
| | | 20110201200830_remove_selections.rb*
| | | 20110202010548_add_checkout_to_folder.rb*
| | | 20110203175206_create_procore_errors.rb*
| | | 20110204164432_contractize_existing_payment_application_liquid_templates.rb*
| | | 20110204174951_add_checkout_to_company.rb*
| | | 20110211175714_create_generic_tool_subscriptions.rb*
| | | 20110216211227_make_charts_polymorphic.rb*
| | | 20110302200911_new_fields_on_programs.rb*
| | | 20110322180723_rename_pay_app_accounting_method.rb*
| | | 20110323153208_create_markups_and_line_item_types.rb*
```

```
> rake db:migrate
```

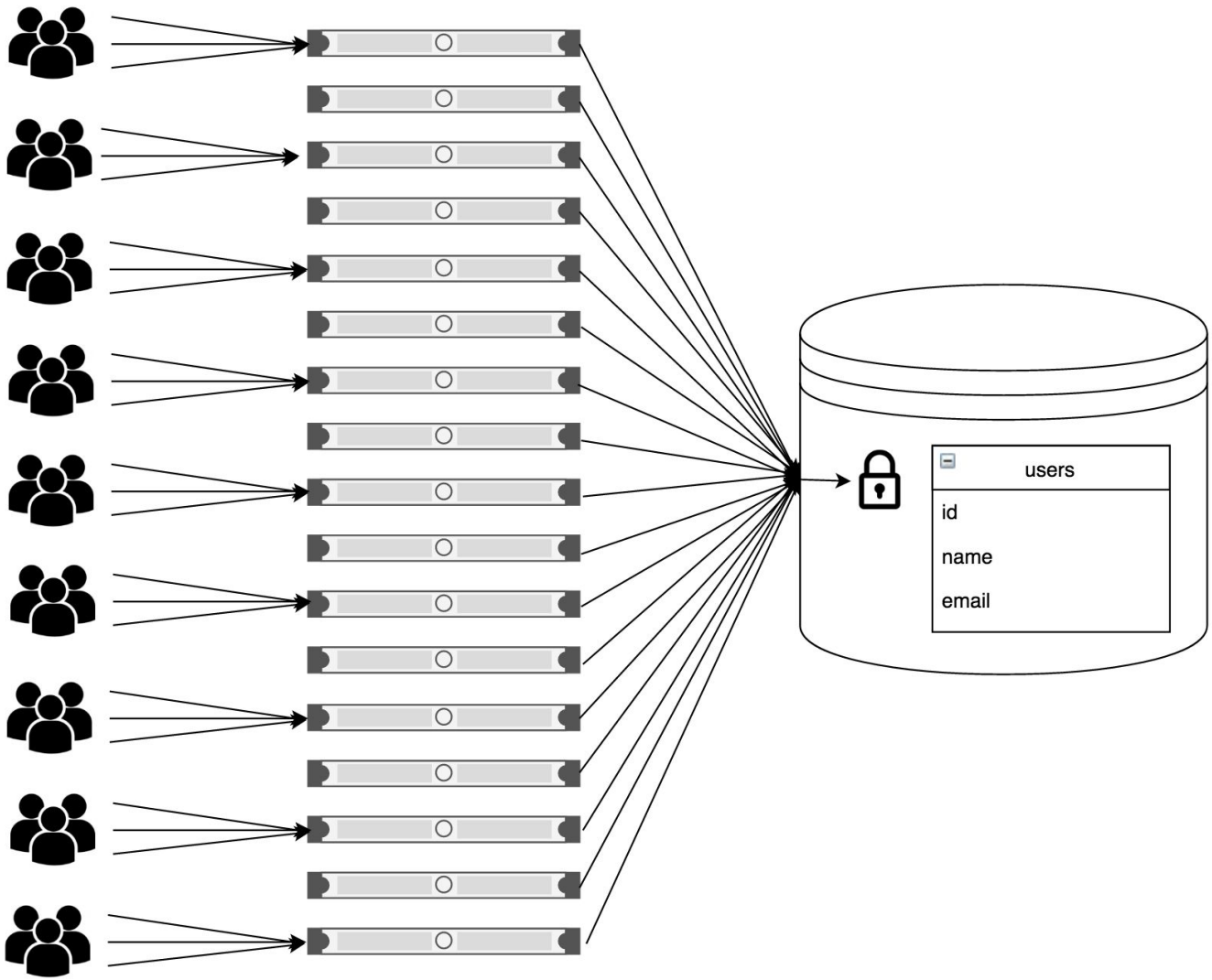
```
== 20160609000551 AddErpJobJobWithCostCodesAndCategoriesSyncableId: migrating -
-- add_column(:erp_jobs, :job_with_cost_codes_and_categories_syncable_id, :integer, {:null=>true})
   -> 0.0006s
-- add_foreign_key(:erp_jobs, :erp_syncables, {:column=>:job_with_cost_codes_and_categories_syncable_
id, :unique=>true})
   -> 0.0009s
== 20160609000551 AddErpJobJobWithCostCodesAndCategoriesSyncableId: migrated (0.0016s) -

== 20160610194840 AddColumnErpConnectionsStandardCostCodeListSyncableId: migrating -
-- add_column(:erp_connections, :standard_cost_code_list_syncable_id, :integer, {:null=>true})
   -> 0.0006s
-- add_foreign_key(:erp_connections, :erp_syncables, {:column=>:standard_cost_code_list_syncable_id,
:unique=>true})
   -> 0.0010s
== 20160610194840 AddColumnErpConnectionsStandardCostCodeListSyncableId: migrated (0.0016s) -

== 20160610205100 AddColumnErpConnectionsStandardCategorySyncableId: migrating -
-- add_column(:erp_connections, :standard_category_list_syncable_id, :integer, {:null=>true})
   -> 0.0005s
-- add_foreign_key(:erp_connections, :erp_syncables, {:column=>:standard_category_list_syncable_id, :
unique=>true})
   -> 0.0008s
== 20160610205100 AddColumnErpConnectionsStandardCategorySyncableId: migrated (0.0014s) -

== 20160614230649 RenameColumnErpJobJobSyncable: migrating -  =================
-- rename_column(:erp_jobs, :job_with_cost_codes_and_categories_syncable_id, :job_syncable_id)
   -> 0.0023s
== 20160614230649 RenameColumnErpJobJobSyncable: migrated (0.0023s) -  ========
```

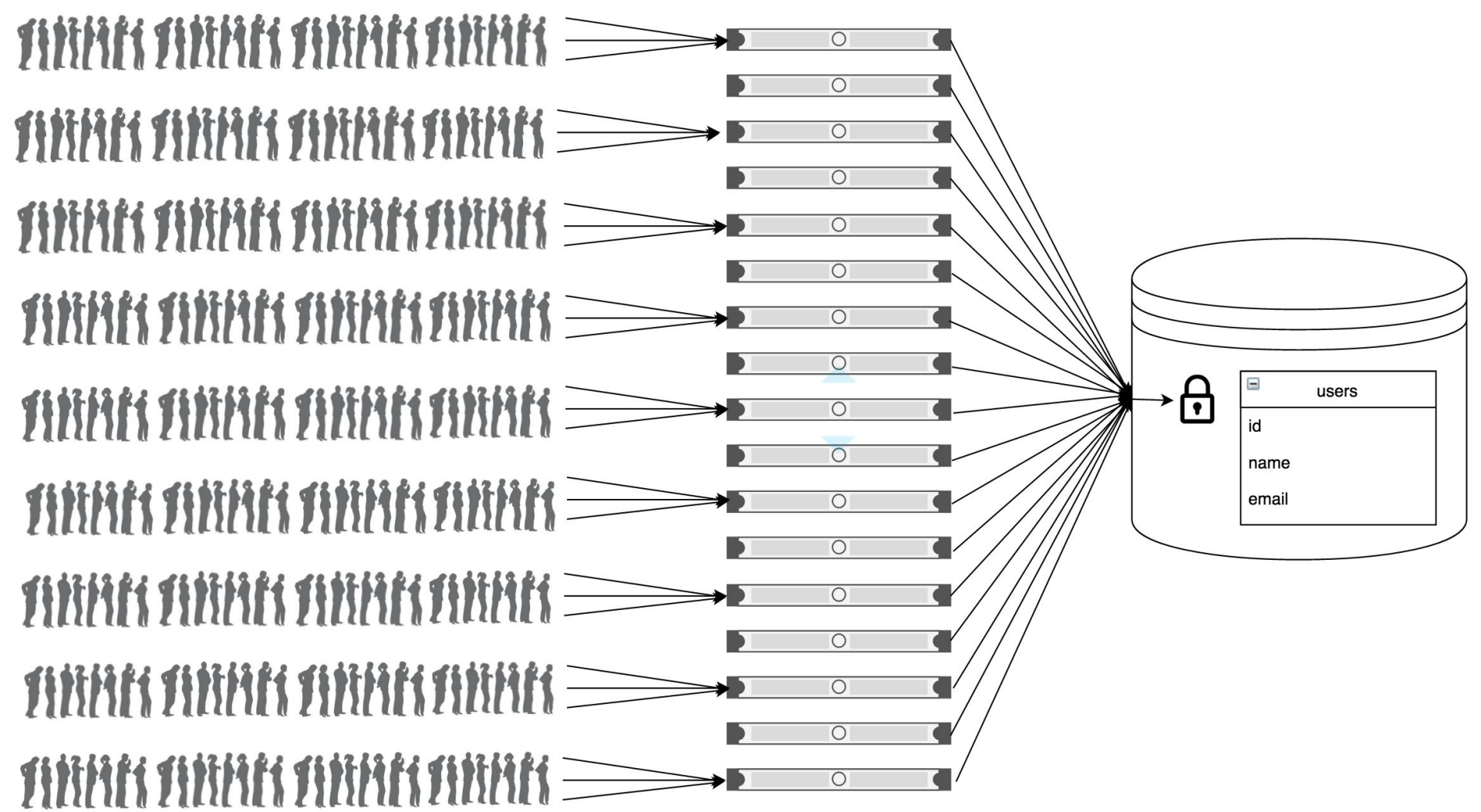
```ruby
class AddTableUsers < ActiveRecord::Migration

  def up
    add_table :users do |t|
      t.string :name, null: false
      t.string :email, null: false
    end
    add_index :users, :email, unique: true
  end

  def down
    remove_table :users
  end

end
```

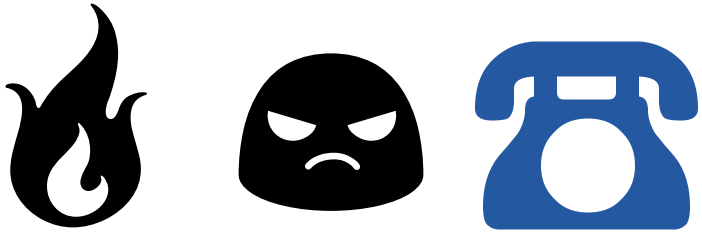
```ruby
class AddColumnIsAdmin < ActiveRecord::Migration

  def up
    add_column :users, :is_admin, :boolean, null: false, default: false
  end

  def down
    remove_column :users, :is_admin
  end

end
```

Is it safe?


FUNNYCUTEGIFS


users
id
name
email


users
id
name
email



https://www.postgresql.org/docs/9.1/static/explicit-locking.html

2 Ways to fail

- Waiting for lock
- During Query

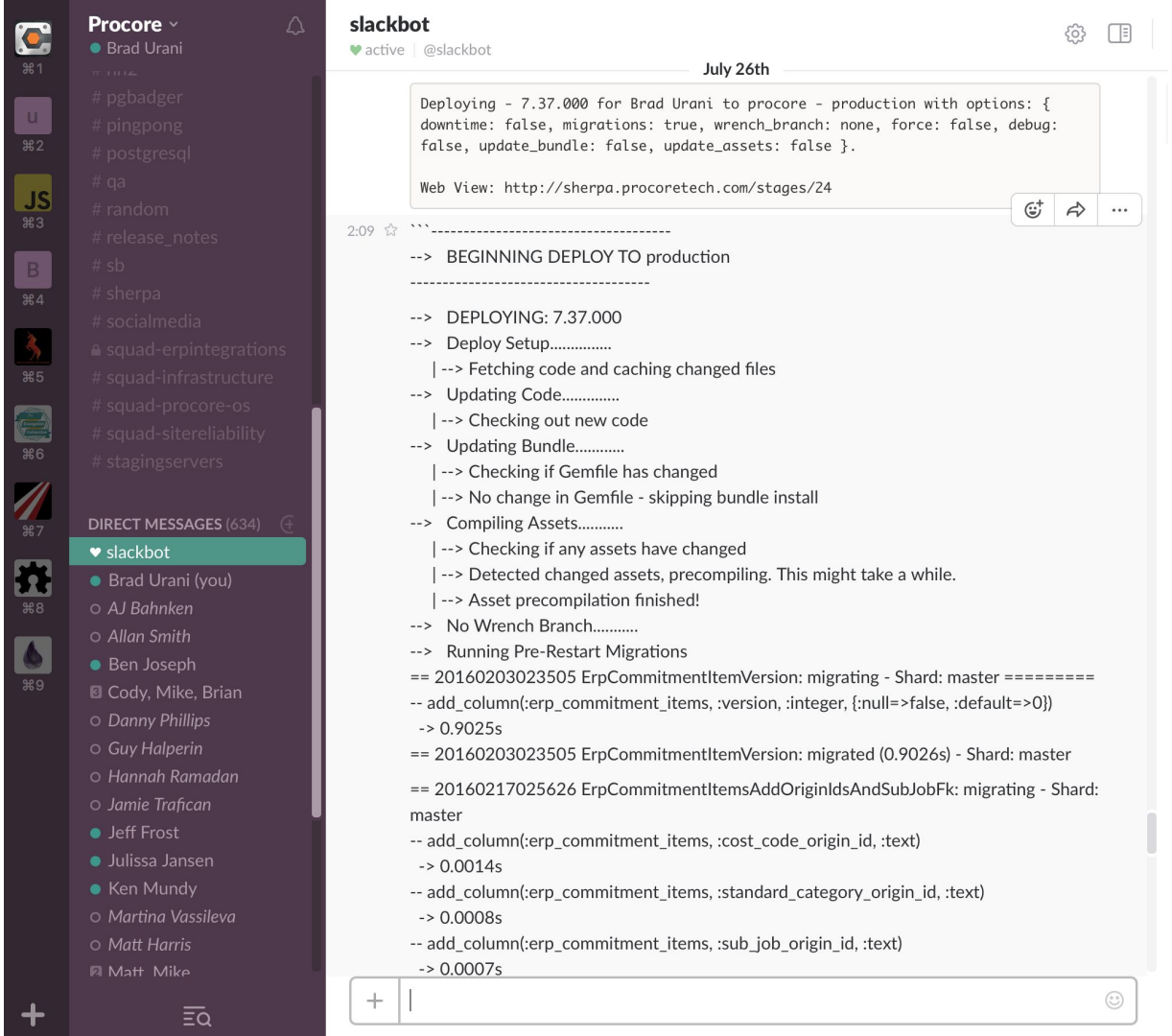
```ruby
class AddColumnIsAdmin < ActiveRecord::Migration

  def up
    execute "SET LOCAL lock_timeout = '5s'"
    add_column :users, :is_admin, :boolean, null: false, default: false
  end

  def down
    remove_column :users, :is_admin
  end

end
```

🔥🔥🔥🔥🔥🔥 TIMEOUT!!

https://github.com/procore/handcuffs

```
# config/initializers/handcuffs

Handcuffs.configure do |config|
  config.phases = [:pre_restart, :downtime, :post_restart]
  config.default_phase = :pre_restart
end
```

```
rake handcuffs:migrate[pre_restart]

rake handcuffs:migrate[downtime]

rake handcuffs:migrate[post_restart]

rake handcuffs:migrate[all]
```


Procore
Brad Urani
pgbadger
pingpong
postgresql
qa
random
release_notes
sb
sherpa
socialmedia
squad-erpintegrations
squad-infrastructure
squad-procore-os
squad-sitereliability
stagingservers
DIRECT MESSAGES (634)
slackbot
Brad Urani (you)
AJ Bahnken
Allan Smith
Ben Joseph
Cody, Mike, Brian
Danny Phillips
Guy Halperin
Hannah Ramadan
Jamie Trafican
Jeff Frost
Julissa Jansen
Ken Mundy
Martina Vassileva
Matt Harris
slackbot
active
@slackbot
July 26th
Deploying - 7.37.000 for Brad Urani to procore - production with options: { downtime: false, migrations: true, wrench_branch: none, force: false, debug: false, update_bundle: false, update_assets: false }.
Web View: http://sherpa.procoretech.com/stages/24
2:09
--> BEGINNING DEPLOY TO production
--> DEPLOYING: 7.37.000
--> Deploy Setup.....
| --> Fetching code and caching changed files
--> Updating Code.....
| --> Checking out new code
--> Updating Bundle.....
| --> Checking if Gemfile has changed
| --> No change in Gemfile - skipping bundle install
--> Compiling Assets.....
| --> Checking if any assets have changed
| --> Detected changed assets, precompiling. This might take a while.
| --> Asset precompilation finished!
--> No Wrench Branch.....
--> Running Pre-Restart Migrations
== 20160203023505 ErpCommitmentItemVersion: migrating - Shard: master =========
-- add_column(:erp_commitment_items, :version, :integer, {:null=>false, :default=>0})
-> 0.9025s
== 20160203023505 ErpCommitmentItemVersion: migrated (0.9026s) - Shard: master
== 20160217025626 ErpCommitmentItemsAddOriginIdsAndSubJobFk: migrating - Shard: master
-- add_column(:erp_commitment_items, :cost_code_origin_id, :text)
-> 0.0014s
-- add_column(:erp_commitment_items, :standard_category_origin_id, :text)
-> 0.0008s
-- add_column(:erp_commitment_items, :sub_job_origin_id, :text)
-> 0.0007s

A Multi-Part Migration Strategy

```ruby
class AddColumnIsAdmin < ActiveRecord::Migration

  def up
    add_column :users, :is_admin, :boolean
  end

end
```

```ruby
class BackfillUserIsAdmin < ActiveRecord::Migration

  phase :post_restart

  def up
    User.update_all(is_admin: false)
  end
end
```


Nothing to see here.
Please disperse.

```ruby
class BackfillUserIsAdmin < ActiveRecord::Migration

  phase :post_restart

  disable_ddl_transaction!

  def up
    User.find_in_batches(batch_size: 1000) do |users|
      User.where(id: users.map(&:id)).update_all(is_admin: false)
    end
  end
end
```

Potential Problems

- Still nullable
- Still no default value

Race conditions!!!!

```ruby
class BackfillWithLockAddConstraints < ActiveRecord::Migration

  phase :post_restart

  def up
    execute <<-SQL
      LOCK TABLE users IN ACCESS EXCLUSIVE MODE;
    SQL

    User.where(is_admin: nil).update_all(is_admin: false)

    change_column_default :users, :is_admin, false
    change_column_null    :users, :is_admin, false
  end

end
```

Requested Lock Mode	Current Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

- `ACCESS EXCLUSIVE` lock prevents race conditions
- Changing to non-null with default doesn't change that much
- `post_restart` minimizes time when code / db differ

5 Part Deploy

```ruby
class AddColumnIsAdmin < ActiveRecord::Migration

  def up
    add_column :users, :is_admin, :boolean
  end

end
```

```ruby
class BackfillUserIsAdmin < ActiveRecord::Migration

  phase :post_restart

  disable_ddl_transaction!

  def up
    User.find_in_batches(batch_size: 1000) do |foos|
      User.where(id: user.map(&:id)).update_all(is_admin: false)
    end
  end
end
```

```ruby
class CreateOptionalIndex < ActiveRecord::Migration

  phase :post_restart

  disable_ddl_transaction!

  def change
    add_index :users,
      :is_admin,
      where: "is_admin IS NULL",
      algorithm: :concurrently
  end

end
```

```ruby
class BackfillWithLockAddConstraints < ActiveRecord::Migration

  phase :post_restart

  def up
    execute <<-SQL
      LOCK TABLE users IN ACCESS EXCLUSIVE MODE;
    SQL

    User.where(is_admin: nil).update_all(is_admin: false)

    change_column_default :users, :is_admin, false
    change_column_null    :users, :is_admin, false
  end

  def down
    change_column_null    :users, :is_admin, true
    change_column_default :users, :is_admin, nil
  end
end
```

```ruby
class DropOptionalIndex < ActiveRecord::Migration

  phase :post_restart

  disable_ddl_transaction!

  def change
    remove_index :users,
      :is_admin,
      where: "is_admin IS NULL",
      algorithm: :concurrently
  end
end
```

Multiple Deploys

Rename:

1. Create a new column
2. Create code that writes to both columns
3. Write to both columns
4. Backfill data from the old column to the new column
5. Deploy code that only writes to new column
6. Drop the old column

Operations

Does it remove something used in old code?

- Can't be pre_restart

Does it add something used by new code

- Can't be post_restart

Does it lock?

- More than 5 seconds?
  - Can it be broken up?
    - Yes
      - Multi-part pre- and post-restart
    - No
      - Downtime
- Less than 5 seconds
  - Are you sure?
    - Lock timeout

create_table

create_table

pre_restart

- Not referenced by old code

drop_table

drop_table

post_restart

- Referenced by old code

Or downtime to be safe. Rolling restarts can cause errors

add_column

(nullable, no default)

add_column

(nullable, no default)

`pre_restart` with timeout 

- Not referenced by old code

add_column

(not nullable, with default)

add_column

(not nullable, with default)

downtime 

- Or break it up

(unless it's empty)

remove_column

remove_column

post_restart (as of rails 4)

- Can't run `pre_restart` because code references it
- Can run `post_restart` because INSERTS don't include all columns

change_column_null

change_column_null

`Non-Nullable` ➡ `Nullable` **`pre_restart`**

`Nullable` ➡ `Non-Nullable` **`downtime`** or multi-part

- Must be backfilled or empty
- Locks

change_column_default

change_column_default

`Default ➡ No Default` **post_restart (either really)**

`No Default ➡ Default` **pre_deploy**

- But mostly this is done while making it non-nullable

change_column

change_column

- Data types?
- Validations?

Depends

Err on the side on downtime - every case different

add_foreign_key

add_foreign_key

downtime 

- ACCESS EXCLUSIVE on BOTH sides

Exceptions

- During `create_table`: `pre_restart`
- During `create_column`, nullable FK: `pre_restart`
- **Be sure there are no violations!!!!**
- **With no index it will require scan, so you probably want an index in PK side**

remove_foreign_key

remove_foreign_key

pre_restart

add_index

(not unique)

add_index

(not unique)

post_restart

- Use with `disable_ddl_transaction!` and `algorithm: concurrently`

add_index

(unique)

add_index

(unique)

post_restart

- Use with `disable_ddl_transaction!` and `algorithm: concurrently`

Order

```
201606129876_add_table_foo    pre_restart

201606234567_add_index_bar    post_restart

201606346543_add_move_data    post_restart

201606487654_add_table_baz    pre_restart

201606587654_add_default      post_restart
```

Rollback?

Tradeoff ?

Preserve Autonomy?

- Focus on Training
- Focus on Tooling

Tradeoff

1 Feature, 1 Deploy

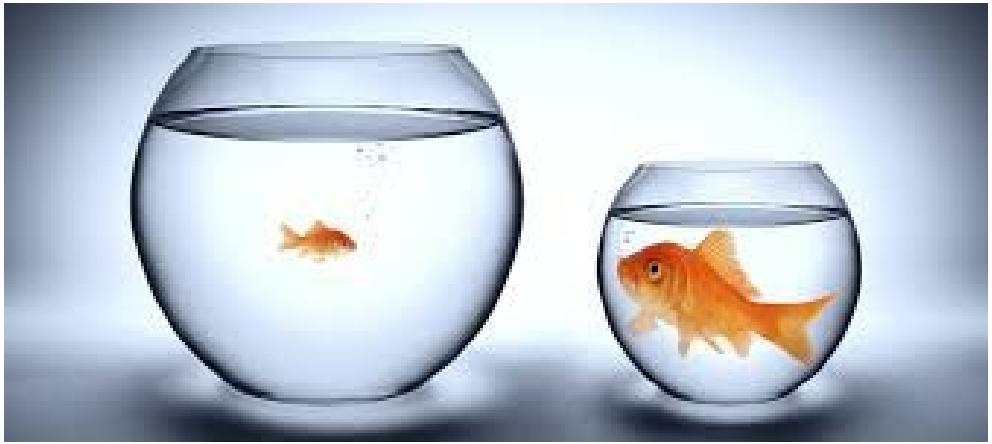
- Can't Eliminate all Lock Contention

Tradeoff

Speed

- Requires some risk

Not one size fits all



Resources

- https://blog.codeship.com/rails-migrations-zero-downtime/
- https://github.com/procore/handcuffs
- https://github.com/ankane/strong_migrations

Who am I?

I tweet at

 **@BradUrani**

Connect with me

 **linkedin.com/in/bradurani**

My seldomly updated blog

fractalbanana.com

I work in Santa Barbara at

PROCORE™
CLOUD-BASED CONSTRUCTION SOFTWARE