

**rails db:migrate:safely**

HI, I'M MATT.

# MATT DUSZYNSKI



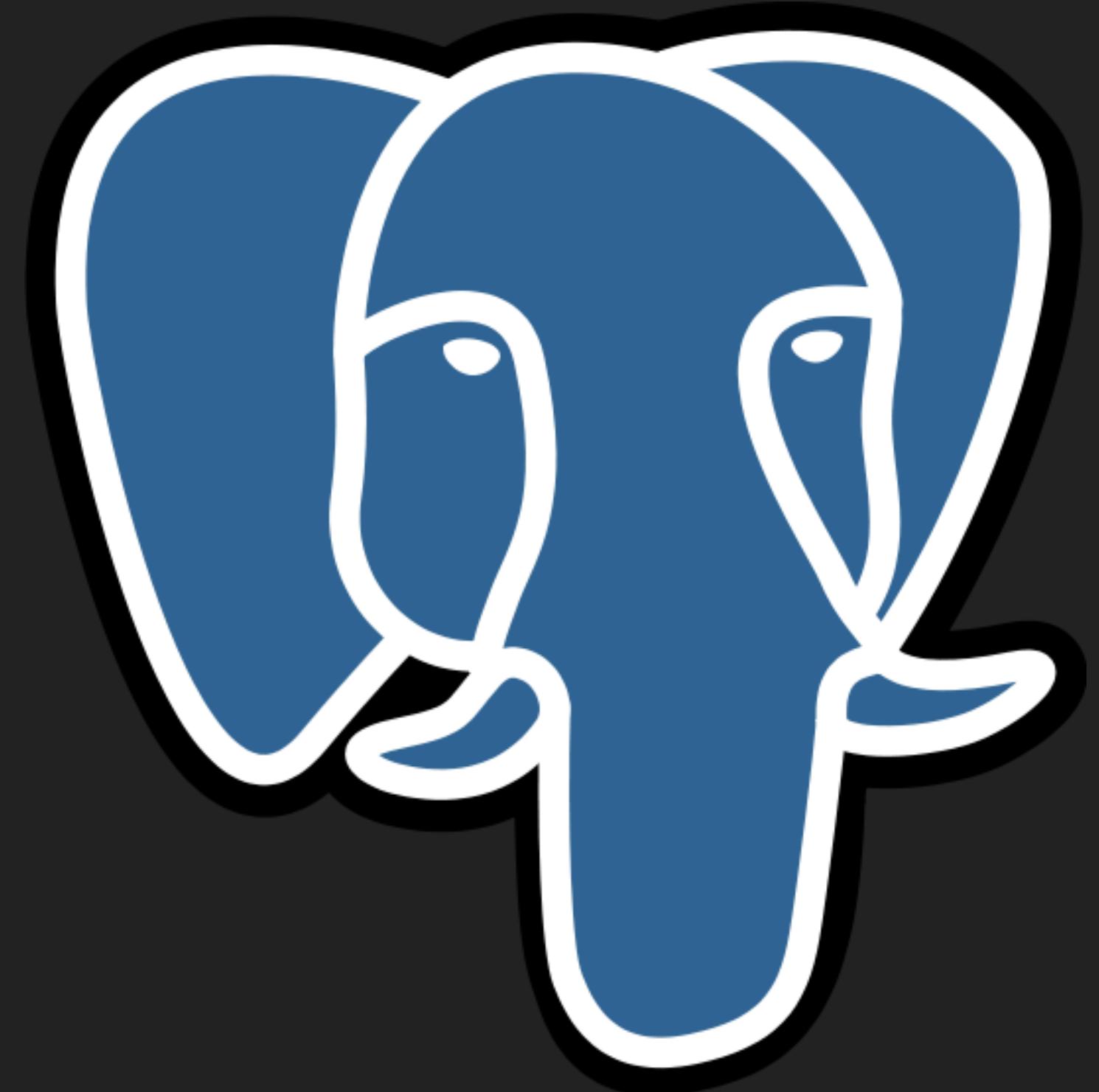
@BSDZUNK



GITHUB.COM/DZUNK

**wm**<sup>®</sup>

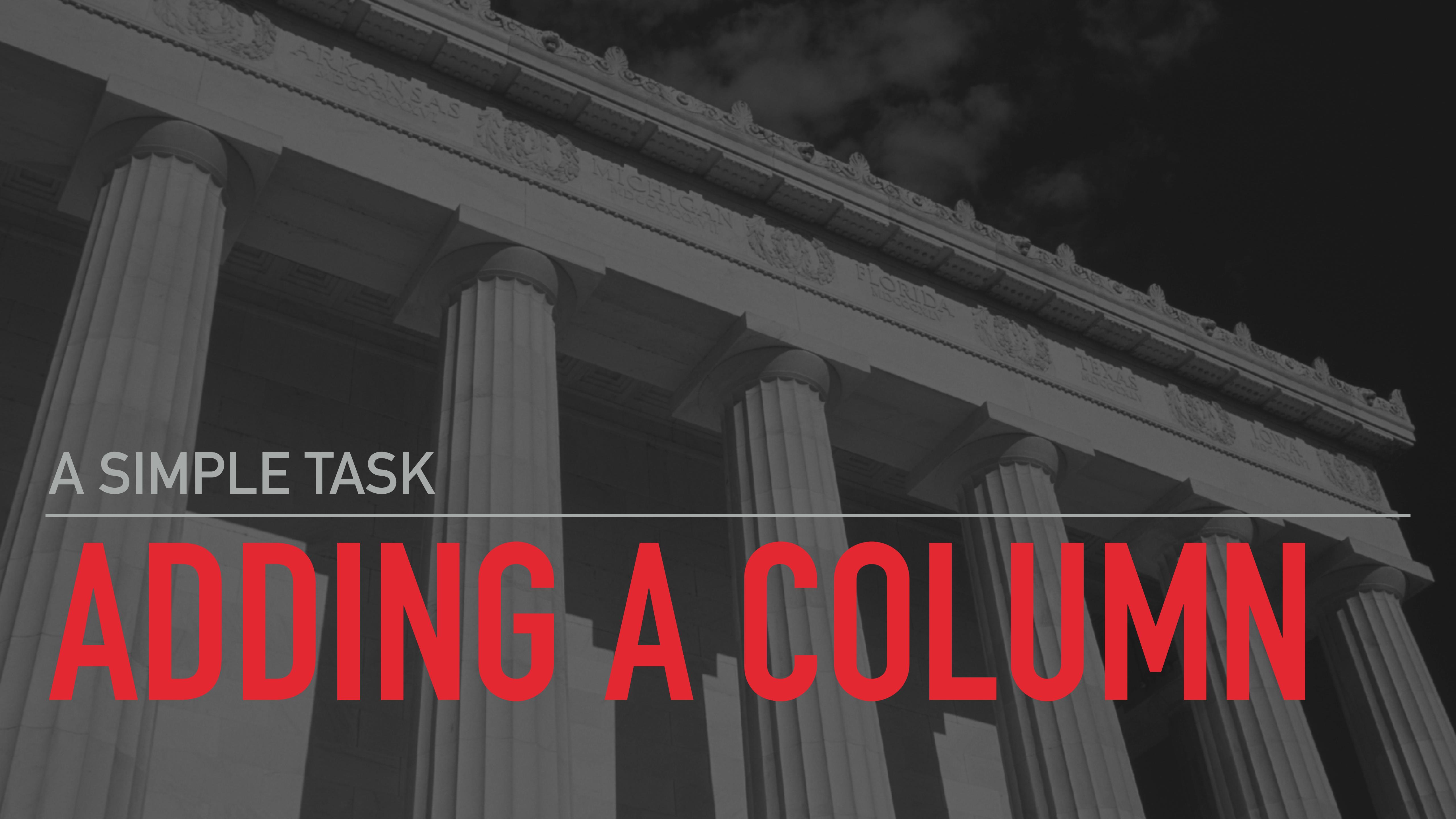




# 50 Million Records

50 Records





A SIMPLE TASK

---

# ADDING A COLUMN

## ADDING A COLUMN

```
class AddActiveToUsers < ActiveRecord::Migration[5.2]
  def change
    add_column :users, :active, :boolean, default: true
  end
end
```



UNDER THE HOOD

---

# LOCKS

## LOCKS

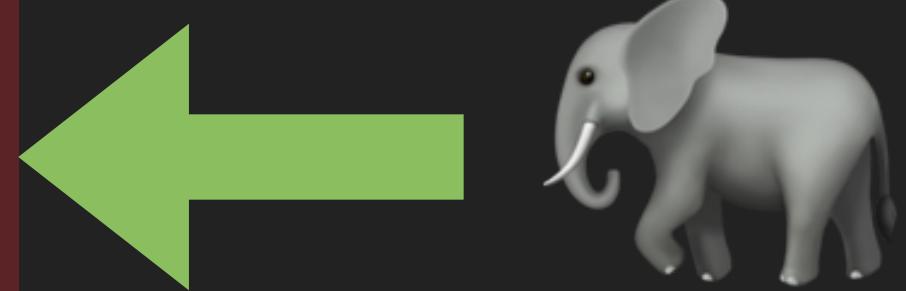
- ▶ Locks are a mechanism for ensuring multiple operations don't update the same row at the same time.
- ▶ There are 8 different lock modes, ranging from ACCESS SHARE (anyone can read and write data) to ACCESS EXCLUSIVE (no one else is permitted to read data).
- ▶ Certain database migrations will obtain an ACCESS EXCLUSIVE lock, and prevent the rest of your application from reading data until the migration completes.

<b>id</b>	<b>username</b>	<b>email</b>
1	alice	alice@example.com
2	bob	bob@example.com
3	charlie	charlie@foo.org
4	doug	doug@bar.net

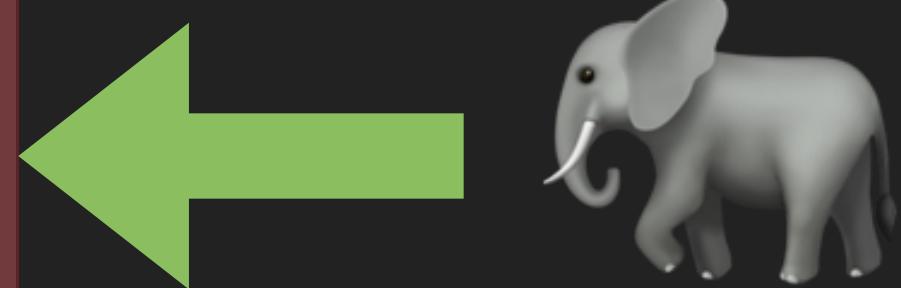


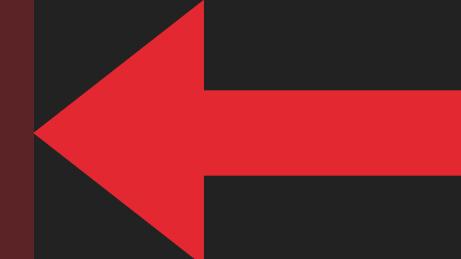
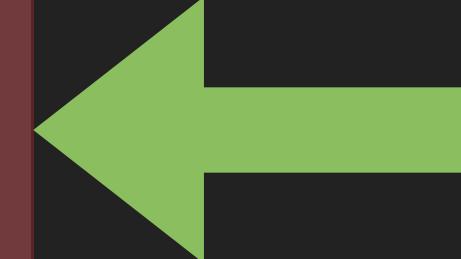
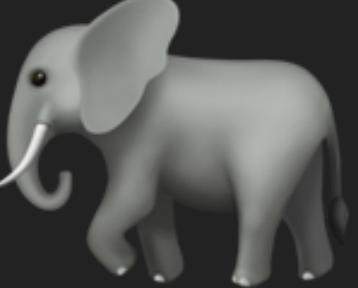


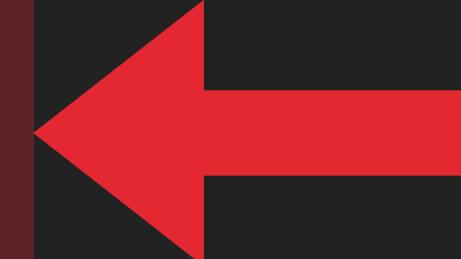
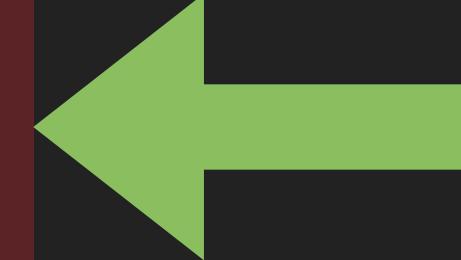
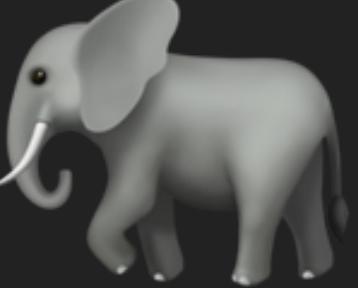
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	

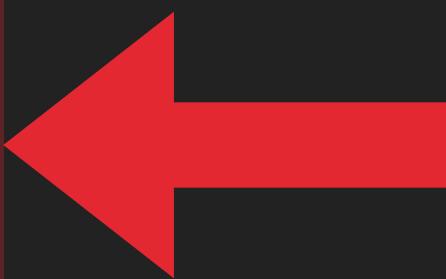
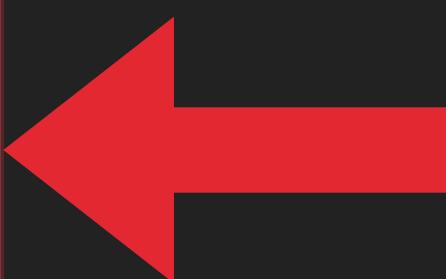
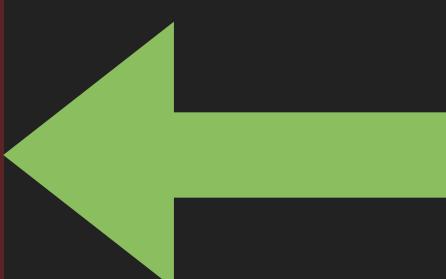
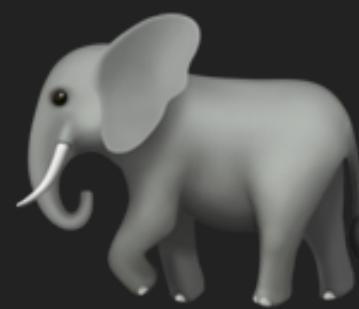


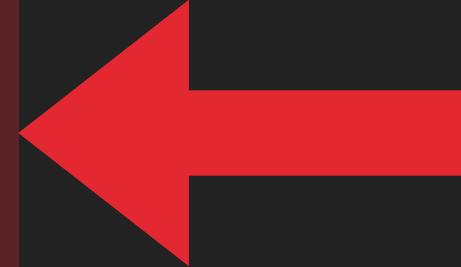
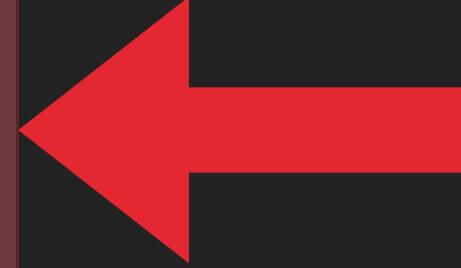
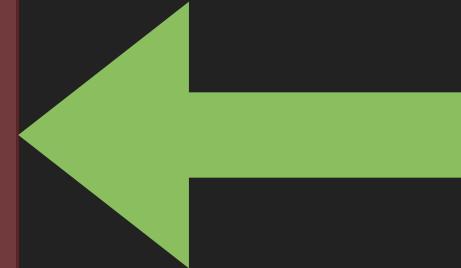
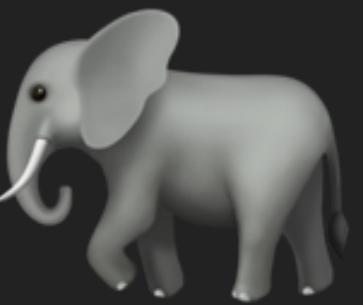
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	true
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	



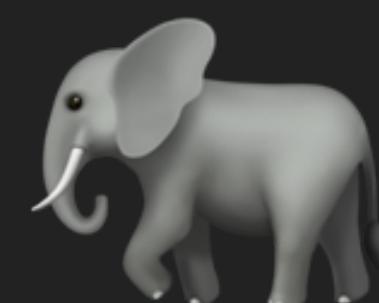
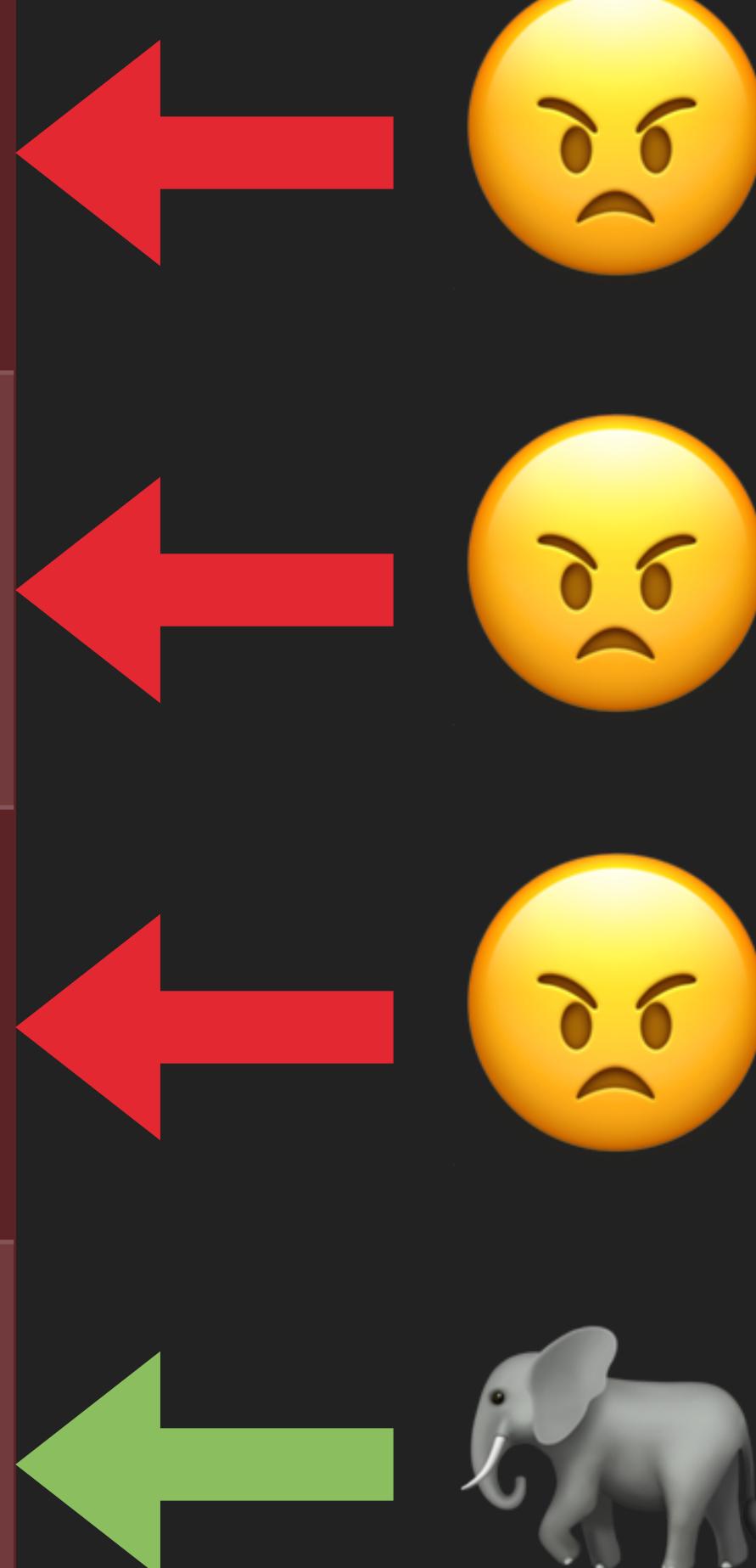
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org		
4	doug	doug@bar.net		

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net		

id	username	email	active	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	 
4	doug	doug@bar.net		

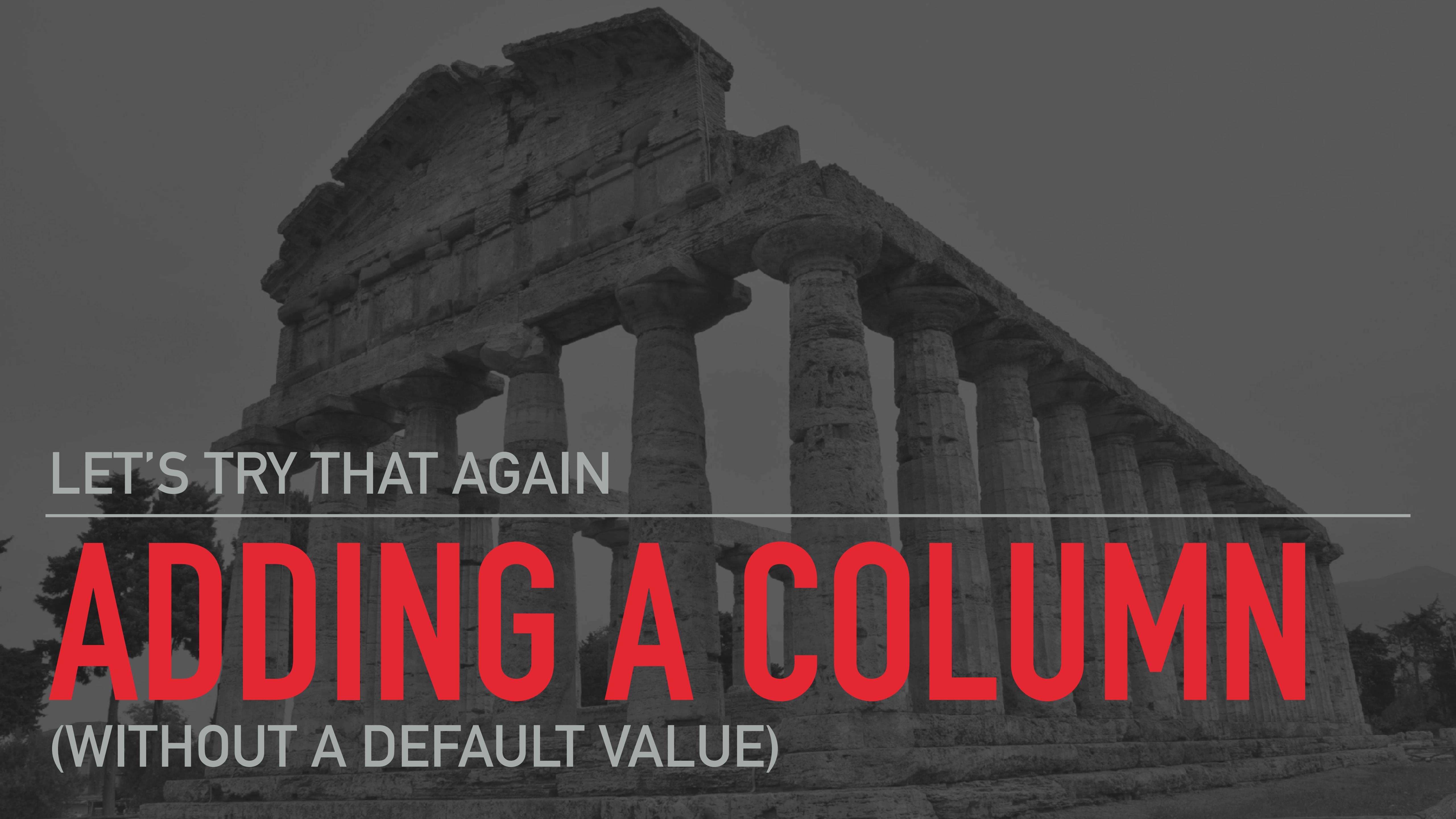
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net	true	 

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	true
3	charlie	charlie@foo.org	true
4	doug	doug@bar.net	true



<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	
2	bob	bob@example.com	true	
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net	true	

DON'T ADD COLUMNS  
WITH A DEFAULT VALUE.



LET'S TRY THAT AGAIN

---

# ADDING A COLUMN

(WITHOUT A DEFAULT VALUE)

LET'S TRY THAT AGAIN

---

## ADDING A COLUMN (WITHOUT A DEFAULT VALUE)

```
class AddActiveToUsers < ActiveRecord::Migration[5.2]
  def change
    add_column :users, :active, :boolean, default: true
  end
end
```

LET'S TRY THAT AGAIN

---

## ADDING A COLUMN (WITHOUT A DEFAULT VALUE)

```
class AddActiveToUsers < ActiveRecord::Migration[5.2]
  def up
    add_column :users, :active, :boolean
    change_column_default :users, :active, true
    User.update_all(active: true)
  end

  def down
    remove_column :users, :active
  end
end
```



UNDER THE HOOD

---

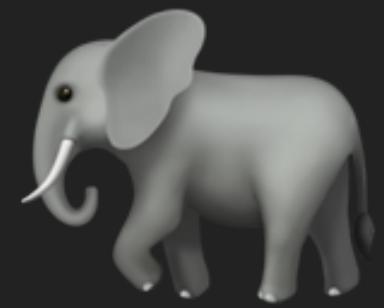
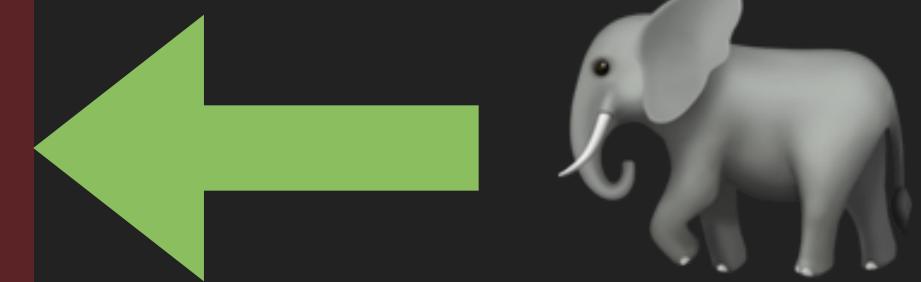
# TRANSACTIONS

## TRANSACTIONS

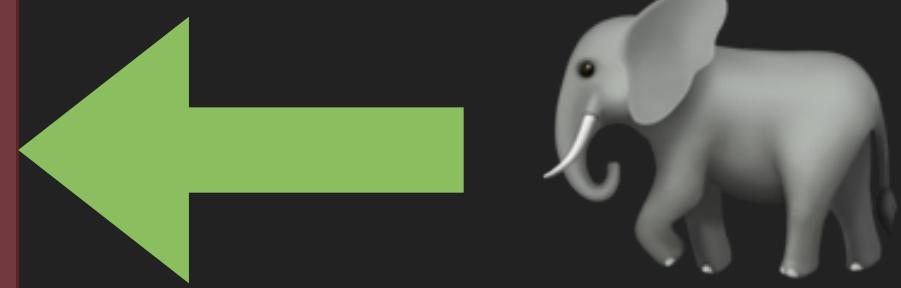
- ▶ Transactions combine multiple database operations into a single, “all-or-nothing” operation.
- ▶ They provide four guarantees: atomicity, consistency, isolation, and durability (“ACID”). Consistency and isolation are guaranteed by locks.
- ▶ When a row is being updated, an exclusive lock is issued, and no one else can update that same row until the first update is complete.
- ▶ Locks are issued on a first-come, first-served basis, and live for the duration of a transaction, even if the statement that requested the lock has already executed.
- ▶ Migrations are automatically wrapped in a transaction.

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	
2	bob	bob@example.com	
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	

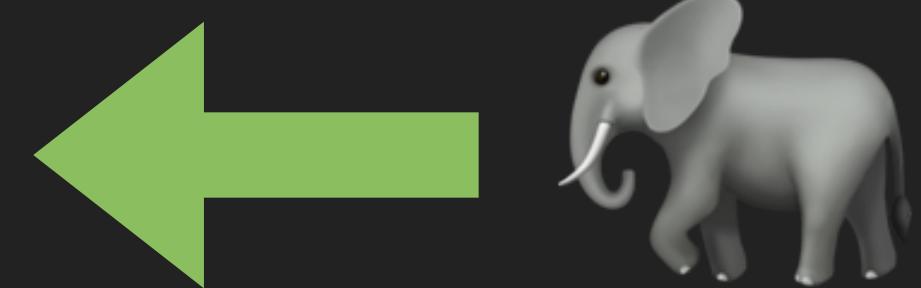
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	

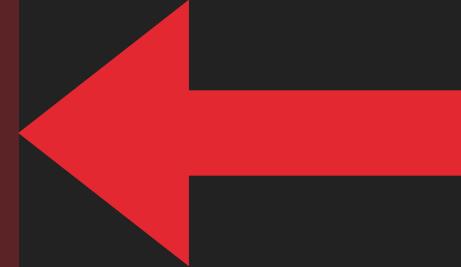
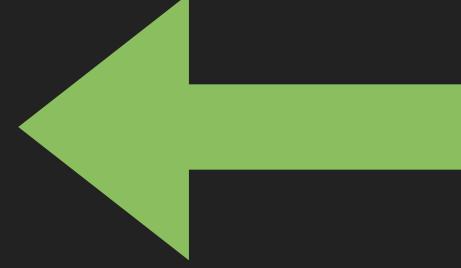
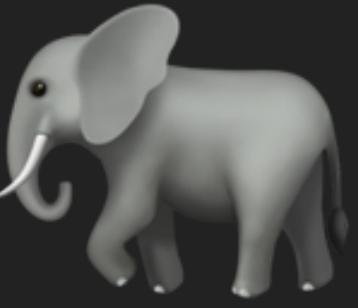


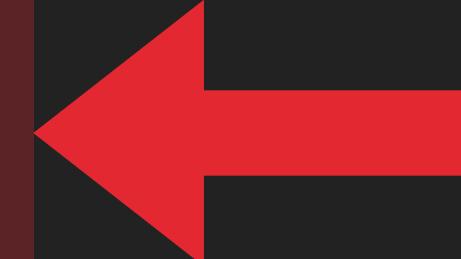
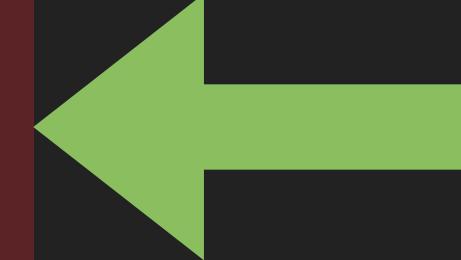
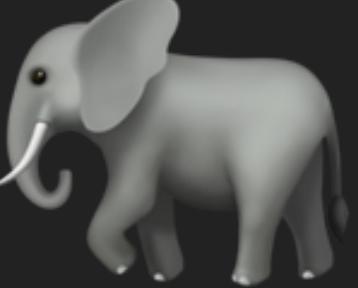
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	true
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	

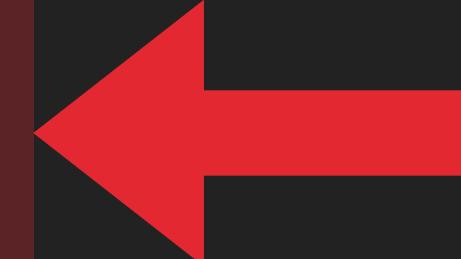
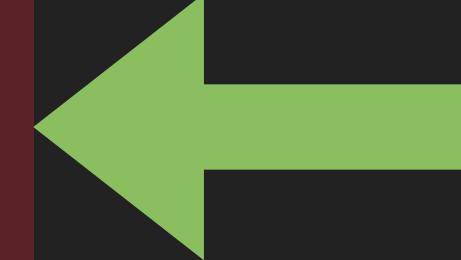
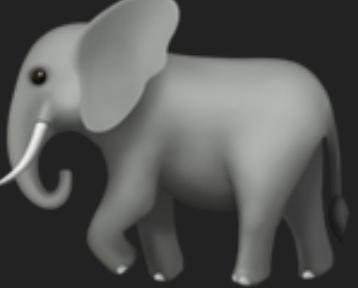
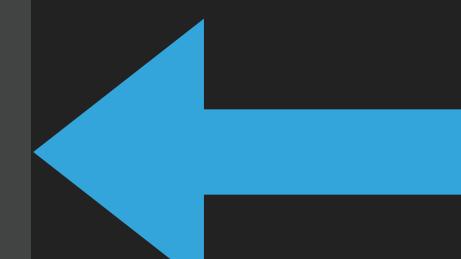


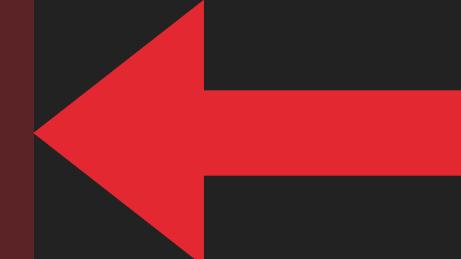
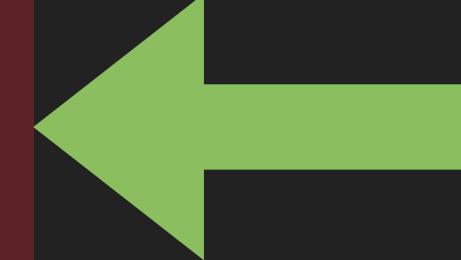
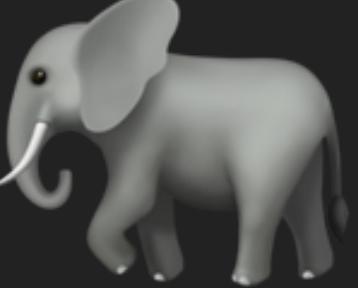
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>
1	alice	alice@example.com	true
2	bob	bob@example.com	true
3	charlie	charlie@foo.org	
4	doug	doug@bar.net	

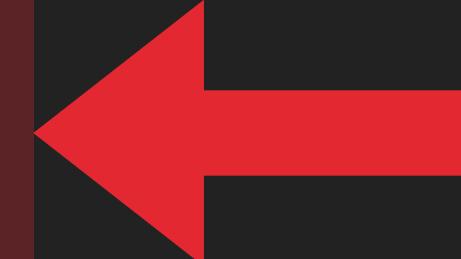
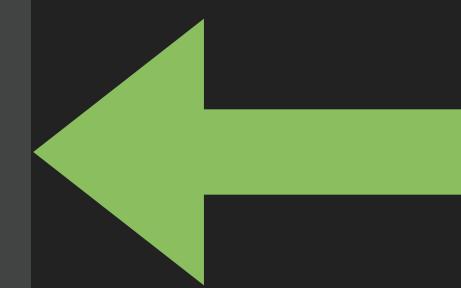
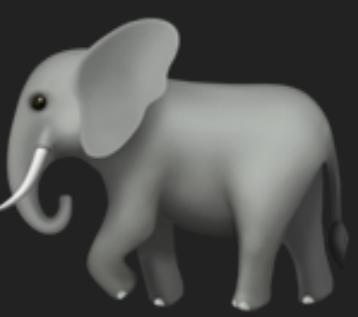


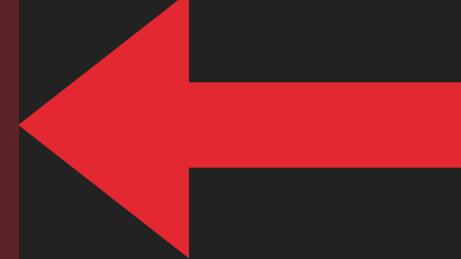
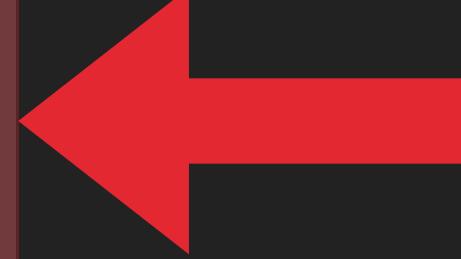
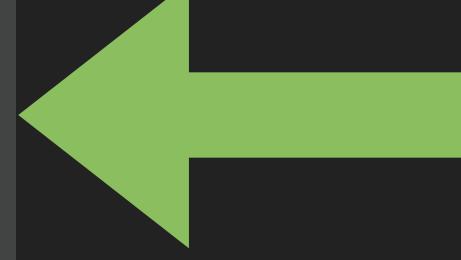
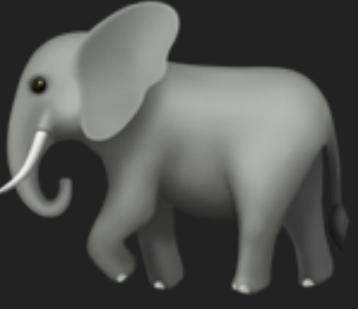
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org		
4	doug	doug@bar.net		

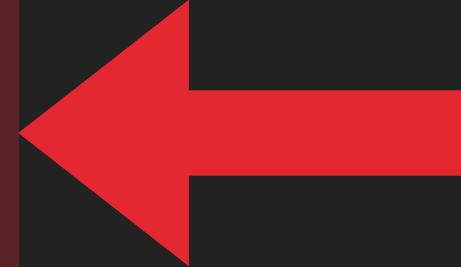
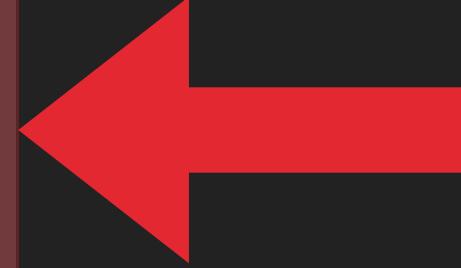
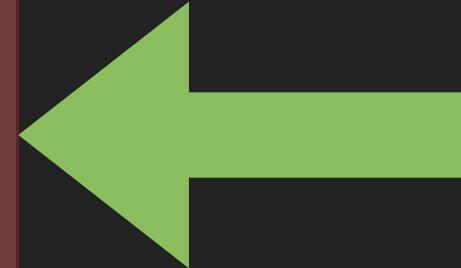
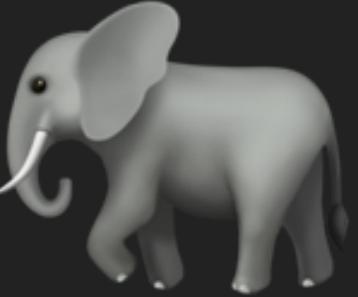
<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net		

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	 
4	doug	doug@bar.net		

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net		

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net		 

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net		 

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	 
2	bob	bob@example.com	true	 
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net	true	 

<b>id</b>	<b>username</b>	<b>email</b>	<b>active</b>	
1	alice	alice@example.com	true	
2	bob	bob@example.com	true	
3	charlie	charlie@foo.org	true	
4	doug	doug@bar.net	true	

<b>id</b>	<b>username</b>	<b>last_login_at</b>	<b>active</b>	
1	alice	1 day ago	true	
2	bob	2 days ago	true	
3	charlie	4 hours ago	true	
4	doug	1 year ago	true	

DON'T BACKFILL DATA  
INSIDE A TRANSACTION.

## ADDING A COLUMN (THE CORRECT WAY)

```
class AddActiveToUsers < ActiveRecord::Migration[5.2]
  def up
    add_column :users, :active, :boolean
    change_column_default :users, :active, true
    User.update_all(active: true)
  end

  def down
    remove_column :users, :active
  end
end
```

## ADDING A COLUMN (THE CORRECT WAY)

```
class MarkAllUsersActive < ActiveRecord::Migration[5.2]
  disable_ddl_transaction!

  def up
    User.in_batches do |batch|
      User.transaction { batch.update_all(active: true) }
    end
  end
end
```

DON'T MIX SCHEMA  
AND DATA CHANGES.

FOR POSTGRES ONLY

ADDING AN INDEX

FOR POSTGRES ONLY

---

## ADDING AN INDEX

```
class AddIndexToActive < ActiveRecord::Migration[5.2]
  def change
    add_index :users, :active
  end
end
```





FOR POSTGRES ONLY

---

## ADDING AN INDEX

```
class AddIndexToActive < ActiveRecord::Migration[5.2]
  disable_ddl_transaction!

  def change
    add_index :users, :active, algorithm: :concurrently
  end
end
```

**DO ADD POSTGRES  
INDICES CONCURRENTLY.**



CASE STUDY

---

# AUDITS



1. Write actual changes to model



2. Write audit record

## AUDITING

---

# OVERVIEW

# AUDITS

- ▶ Needed to update a compound index on the audits table.
- ▶ Drop old index.
- ▶ Add new index concurrently.
- ▶ No locks, no transactions.
- ▶ What could go wrong?

**EVERYTHING.**

## AUDITS

```
class UpdateAuditableIndex < ActiveRecord::Migration[5.2]
  disable_ddl_transaction!

  def change
    remove_index :audits, name: 'auditable_index'

    add_index :audits,
      ['auditable_type', 'auditable_id'],
      name: 'auditable_index',
      algorithm: :concurrently
  end
end
```



UNDER THE HOOD

---

**CONCURRENCY**

# LITTLE'S LAW

$$L = \lambda W$$

Concurrency

Throughput

Response Time

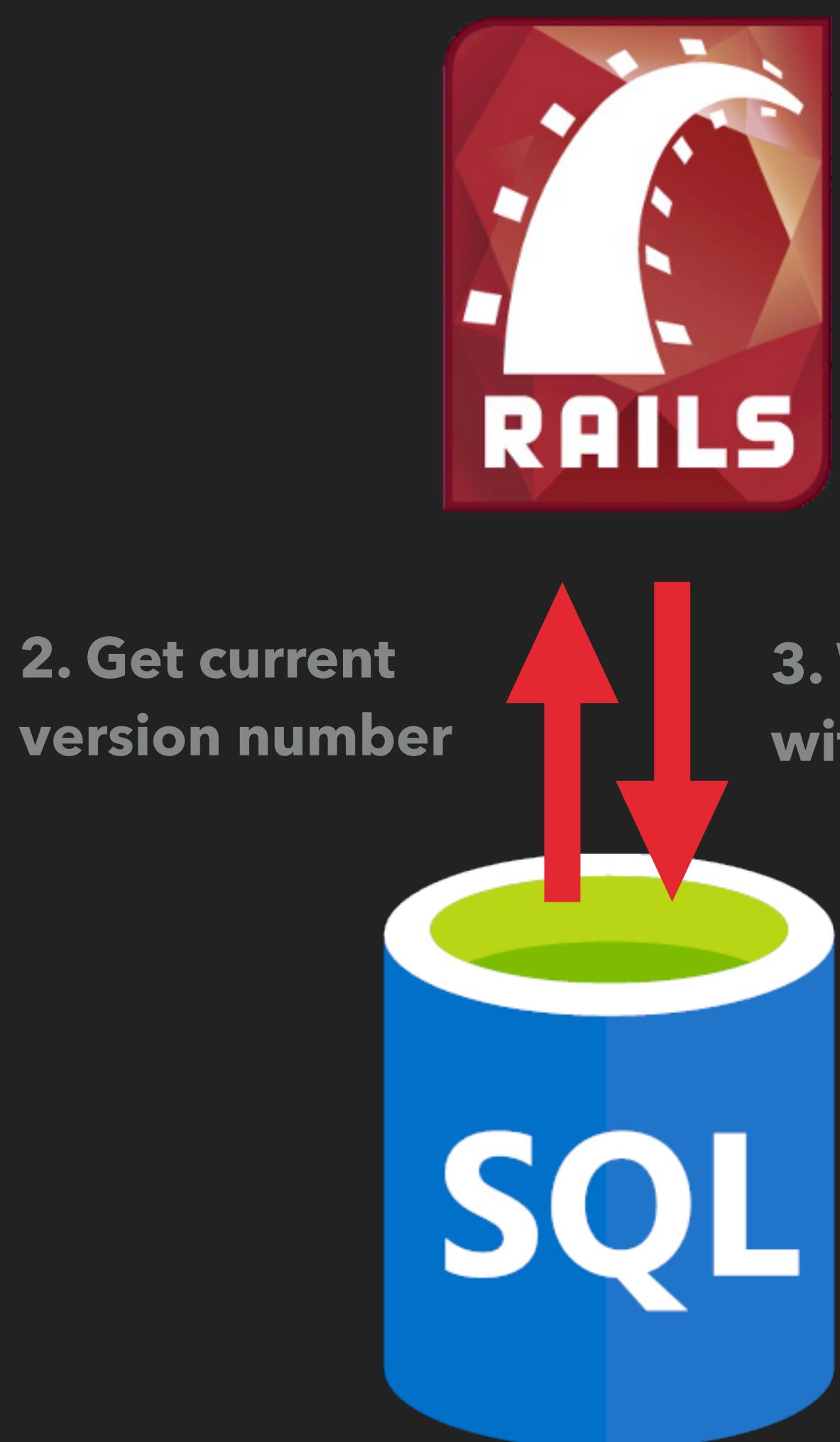
$$4 = 100 * 40\text{ms}$$

Concurrent requests

Req's per second \* Avg. response time

## CONCURRENCY

- ▶ Every application has a theoretical maximum level of concurrency it can support at any given time.
- ▶ Your database obeys the same principles. How fast your queries are, and how large your connection pool is, determines how many queries you can concurrently handle.
- ▶ Requests start queueing when they arrive faster than your application, or its database, can respond to them.
- ▶ If a database operation blocks many requests for a long time, your entire application will grind to a halt.

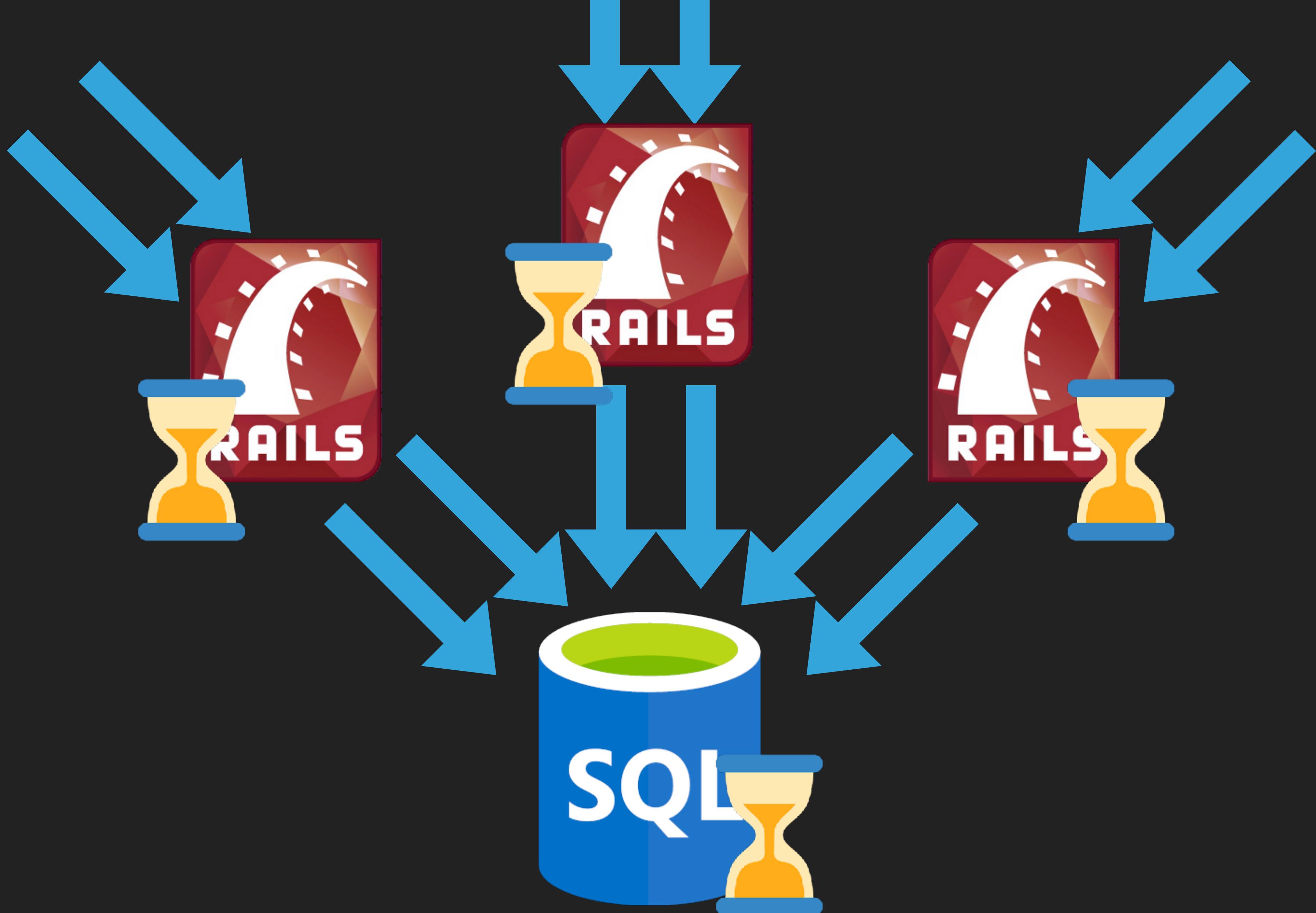


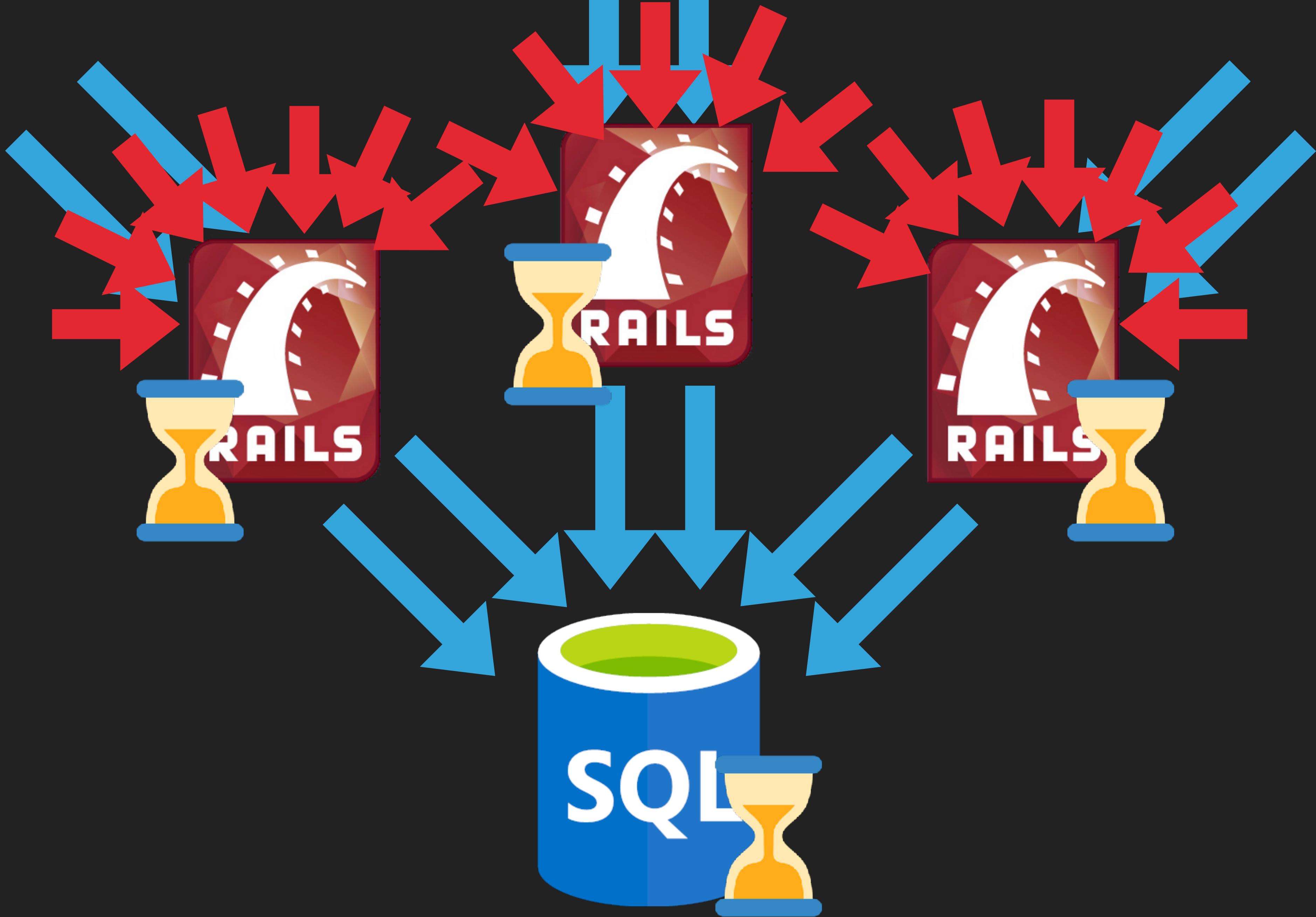
## AUDITING

---

# OVERVIEW

BUT ACCURATE THIS TIME





RULE NO. 4

DO TEST DATABASE  
PERFORMANCE.



YES, THAT IS A FRYING PAN

---

# TOOLS AND RESOURCES

### GEMS

- ▶ Static analysis will warn in advance about certain unsafe migrations.
- ▶ Catch problems at dev time, not deploy time.
- ▶ [ankane/strong\\_migrations](#)
- ▶ [LendingHome/zero\\_downtime\\_migrations](#)
- ▶ Not technically a gem, but: [Gitlab migration helpers](#)

# APPLICATION PERFORMANCE MONITORING

- ▶ Understanding your application's baseline performance is critical to understanding how migrations will change its performance characteristics.
- ▶ [New Relic](#)
- ▶ [Skylight](#)
- ▶ [Scout](#)
- ▶ [AppSignal](#)

**YOU.**

# YOU

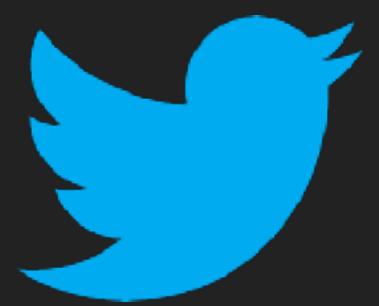
- ▶ You are your own best resource. Static analysis can only go so far. Know your application, and which migrations are potentially dangerous.
- ▶ Keep up with changes to your database engine. There are plenty of improvements being made in this area.
- ▶ Practice writing safe and reliable migrations when the stakes are lower, before you need to apply them to your production database.
- ▶ It's better to listen to that nagging voice in the back of your head than feel the icy dread of taking down your production system.

## TAKEAWAYS

- ▶ **DON'T** add columns with a default value.
- ▶ **DON'T** backfill data inside a transaction.
- ▶ **DON'T** mix schema and data changes in the same migration.
- ▶ **DO** add Postgres indexes concurrently.
- ▶ **DO** monitor and test database performance before, during, and after migrations.

UPTIME BEGINS AT \$HOME

# QUESTIONS?



@BSDZUNK



GITHUB.COM/DZUNK