# Migrating from MySQL to PostgreSQL

Paul Gross


http://www.pgrs.net

@pgr0ss

# Braintree is a payment gateway

People enter their credit cards on a website

Merchants pass those along to us

We verify and charge the cards

# Merchants rely on us

When we are down, our merchants cannot process credit cards

Downtime is a big deal!

# Braintree architecture

Ruby on Rails application

Load balancer

Apache app servers with passenger

MySQL

# Deployment with schema changes

Push new code to app servers

Put up maintenance page

Make database schema changes

Take down maintenance page

# We are down for as long as the schema changes take place

MySQL is really slow at migrating large tables

This is the primary reason we migrated to PostgreSQL

# Table with 10 columns and 1 million rows

| | MySQL | PostgreSQL |
|---|---|---|
| **Adding a column** | 26 seconds | 400 milliseconds |
| **Adding an index** | 30 seconds | 60 seconds |

## PostgreSQL can add indexes without locking tables

```
CREATE INDEX CONCURRENTLY index_name ON table (column);
```

# Hacks to work around MySQL

oak-online-alter-table

Builds a ghost table with new schema

Copies all data

Replaces original table

We started seeing deadlocks with MySQL

```
CREATE TABLE foo (id INT) ENGINE INNODB;

-- session 1                 session 2
-------------------------------------------
BEGIN;
-------------------------------------------
                             BEGIN;
-------------------------------------------
INSERT INTO foo VALUES (1);
-------------------------------------------
                             INSERT INTO foo VALUES (2);
-------------------------------------------
UPDATE actions SET id = 2 WHERE id = 1;
-------------------------------------------
                             UPDATE actions SET id = 4 WHERE id = 2;
```

# Researching the migration

Minimal downtime

Map data types

Set primary key sequences based on the number of rows

# We needed a migration like rsync

Initial migration while the site is up

Delta migration while the site is up

Take the site down

Run delta migration one last time

Bring the site back up with PostgreSQL

# We looked at tools

Most were one shot and took a long time

Most were very complex

So we decided to write our own

Use ActiveRecord

# Assumptions of our migration script

Tables have a primary key

Tables have an updated_at column

updated_at has an index

# ActiveRecord Primer

```ruby
class Cat < ActiveRecord::Base
    set_table_name :cats
end

Cat.create!(
    :name => "Kasha",
    :domesticated => true,
    :adopted_at => Time.now
)

kasha = Cat.find(1)

puts kasha.domesticated?
#=> true

puts kasha.adopted_at
#=> Tue Mar 22 15:34:36 -0400 2011
```

# Migration script

```
class MysqlModelBase < ActiveRecord::Base
  establish_connection(
    :adapter => "mysql",
    :username => "username",
    :database => "ourdb",
    :encoding => "UTF8"
  )
end

class PGModelBase < ActiveRecord::Base
  establish_connection(
    :adapter => "postgresql",
    :username => "username",
    :database => "ourdb",
    :encoding => "UTF8"
  )
end
```

```ruby
MysqlModelBase.connection.tables.each do |table|
    last_updated_at = PGModelBase.connection.select_value(<<-SQL)
        SELECT MAX(updated_at) FROM #{table}
    SQL

    updated_ids = MysqlModelBase.connection.select_values(<<-SQL)
        SELECT id FROM #{table}
        WHERE updated_at >= '#{last_updated_at}'
    SQL

    PGModelBase.connection.execute(<<-SQL)
        DELETE FROM #{table}
        WHERE id IN (#{updated_ids.join(",")})
    SQL

    ...
```

```
mysql_ids = MysqlModelBase.connection.select_values(<<-SQL)
    SELECT id FROM #{table}
SQL

postgres_ids = PGModelBase.connection.select_values(<<-SQL)
    SELECT id FROM #{table}
SQL

deleted_record_ids = postgres_ids - mysql_ids

PGModelBase.connection.execute(<<-SQL)
    DELETE FROM #{table}
    WHERE id IN (#{deleted_record_ids.join(",")})
SQL

...
```

```
new_record_ids = mysql_ids - postgres_ids

mysql_model_class = Class.new(MysqlModelBase) do
    set_table_name table
end

postgres_model_class = Class.new(PGModelBase) do
    set_table_name table
end

mysql_models = mysql_model_class.find(new_record_ids)
mysql_models.each do |mysql_model|
    # ActiveRecord maps data types
    postgres_model_class.create!(mysql_model.attributes)
end

...
```

```
PGModelBase.connection.execute(<<-SQL)
    SELECT SETVAL('#{table}_id_seq',
        (SELECT MAX(id) FROM #{table}))
SQL
```

# The real script

github.com/braintree/mysql_to_postgresql

Processes tables in parallel

Pulls back data in groups instead of all at once

Handles tables without an updated_at

Prints debug messages as it runs

# MySQL is case insensitive

PostgreSQL is not

We had many places in our app which relied on that behavior

# Default solution

```
SELECT * FROM people
WHERE LOWER(name) = LOWER(?);

CREATE INDEX lower_name_idx ON people ((LOWER(title)));
```

# Better solution for us

```
CREATE TABLE people (name CITEXT);
```

postgresql.org/docs/current/static/citext.html

# MySQL has an implicit ordering

SELECT statements returned rows ordered by id

PostgreSQL has non-deterministic ordering

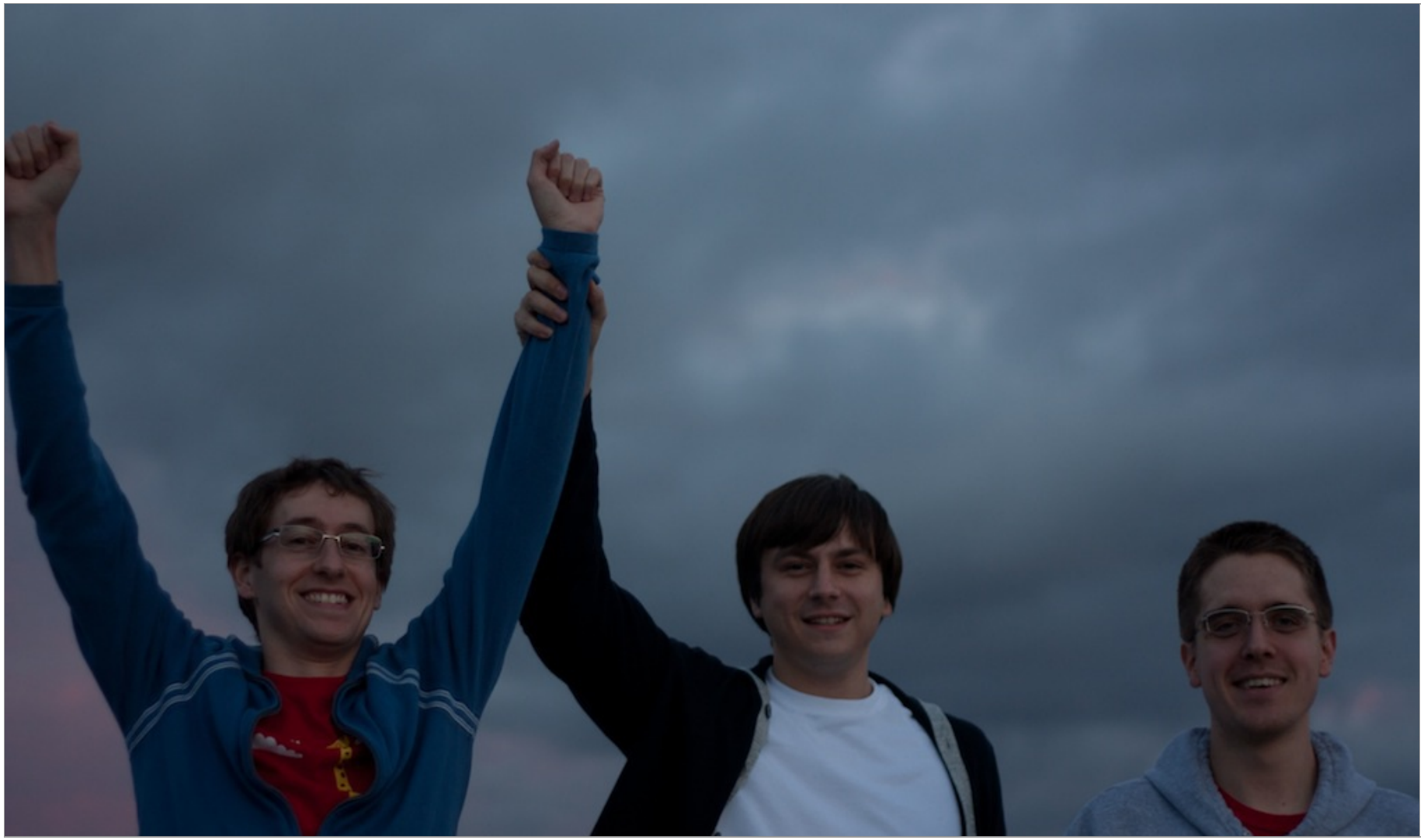
We added ORDER BY clauses when the order mattered

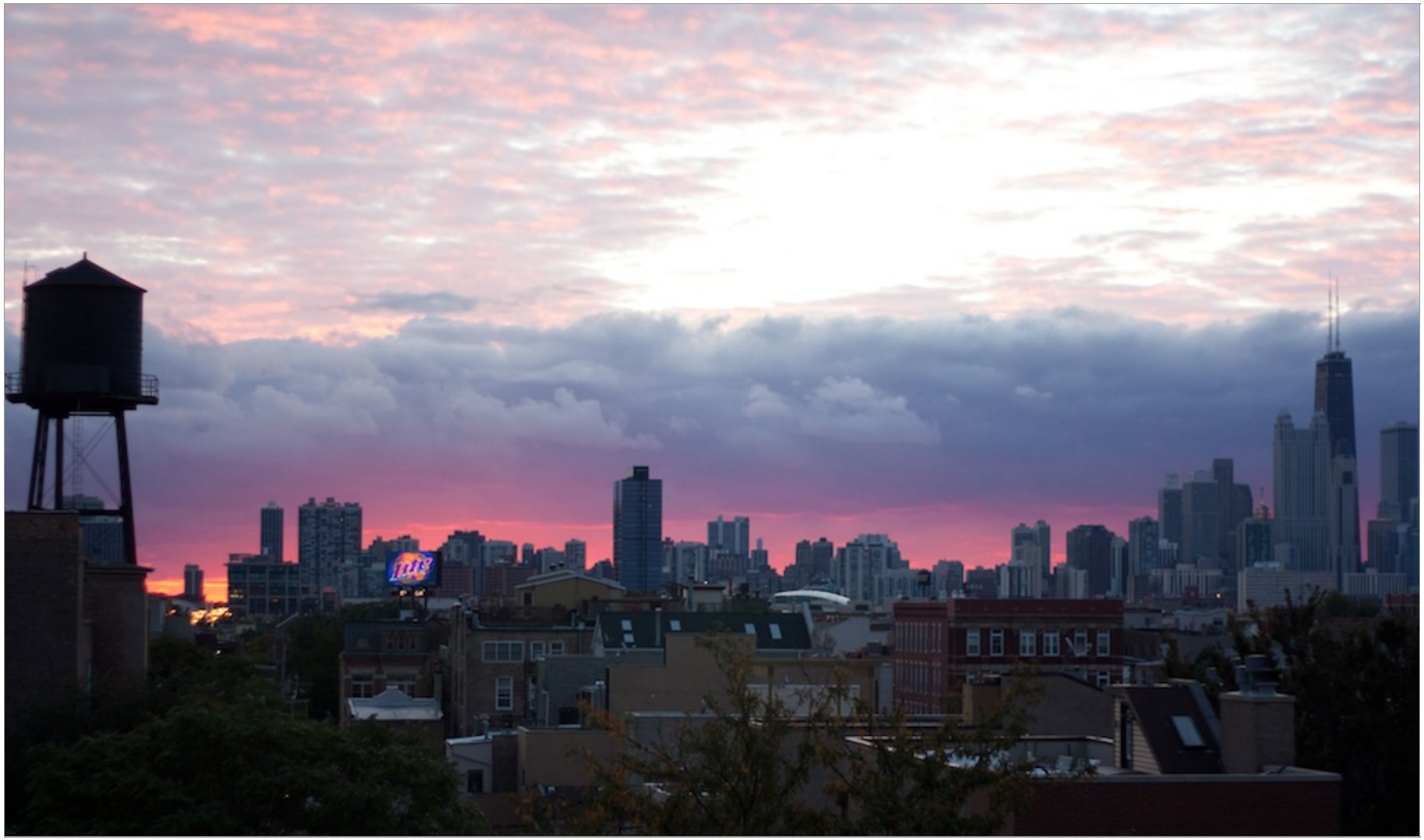
# The big day

3am on Saturday night

Migrated millions of rows across many tables

Total downtime was under 5 minutes





# Aftermath

Worked great at first, then started seeing performance problems

Our app and schema were optimized for MySQL, not PostgreSQL

# Sequences

MySQL does not have sequences, so we had them as rows in a table

```
CREATE FUNCTION next_sequence_value(sequence_id INT(11))
    RETURNS INT
BEGIN
    UPDATE sequences
    SET value = last_insert_id(value + 1)
    WHERE id = sequence_id;
RETURN last_insert_id();
```

PostgreSQL version of same function

```
CREATE FUNCTION next_sequence_value(integer)
         RETURNS integer AS $$
    UPDATE sequences
    SET value = value + 1
    WHERE id = $1
    RETURNING value;
$$ LANGUAGE SQL;
```

These performed horribly; moved them to PostgreSQL sequences

# Hardware

We had MySQL and PostgreSQL running on the same server

Write Ahead Log was on same partition

Fsync would flush way too much data and disk writes would spike

# Hardware

Moved PostgreSQL to its own server

More memory, faster hard drives

Moved the Write Ahead Log to its own partition

Moved the Write Ahead Log to its own hard drives

# Queries

PostgreSQL executes queries differently from MySQL

Spent time studying explain plans

Adding new indexes

Rewriting expensive queries

# Today

We can do deploys with database schema changes in under 30 seconds

Add new indexes while the site is up

We can hold requests, migrate the database, and then let the requests through

Merchants will see a few slow requests, but they will see zero downtime

Questions?