Going beyond the Django ORM limitations with Postgres

@craigkerstiens

heroku

PSA: Macs

Postgres.app

PSA #2

Postgres Weekly

http://postgresweekly.com

Why Postgres

"Its the emacs of databases"

http://www.craigkerstiens.com/2012/04/30/why-postgres/
https://speakerdeck.com/craigkerstiens/postgres-demystified

TLDR

Datatypes
Conditional Indexes
Transactional DDL
Foreign Data Wrappers
Concurrent Index Creation
Extensions
Common Table Expressions

Fast Column Addition
Listen/Notify
Table Inheritance
Per Transaction sync replication
Window functions
NoSQL inside SQL
Momentum

Limitations?

Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we've had to make design decisions on which features to support and which assumptions we can make safely.

Datatypes


< Postgres >
\ ^__^
\ (oo)_______
(__)\ )\/\
||----w |
|| ||

Datatypes

UUID

date

boolean

timestamptz

array

interval

integer

bigint

smallint

line

XML

enum

char

money

serial

bytea

point

float

inet

polygon

numeric

circle

varchar

cidr

tsquery

timetz

text

path

time

timestamp

box

macaddr

tsvector

Datatypes

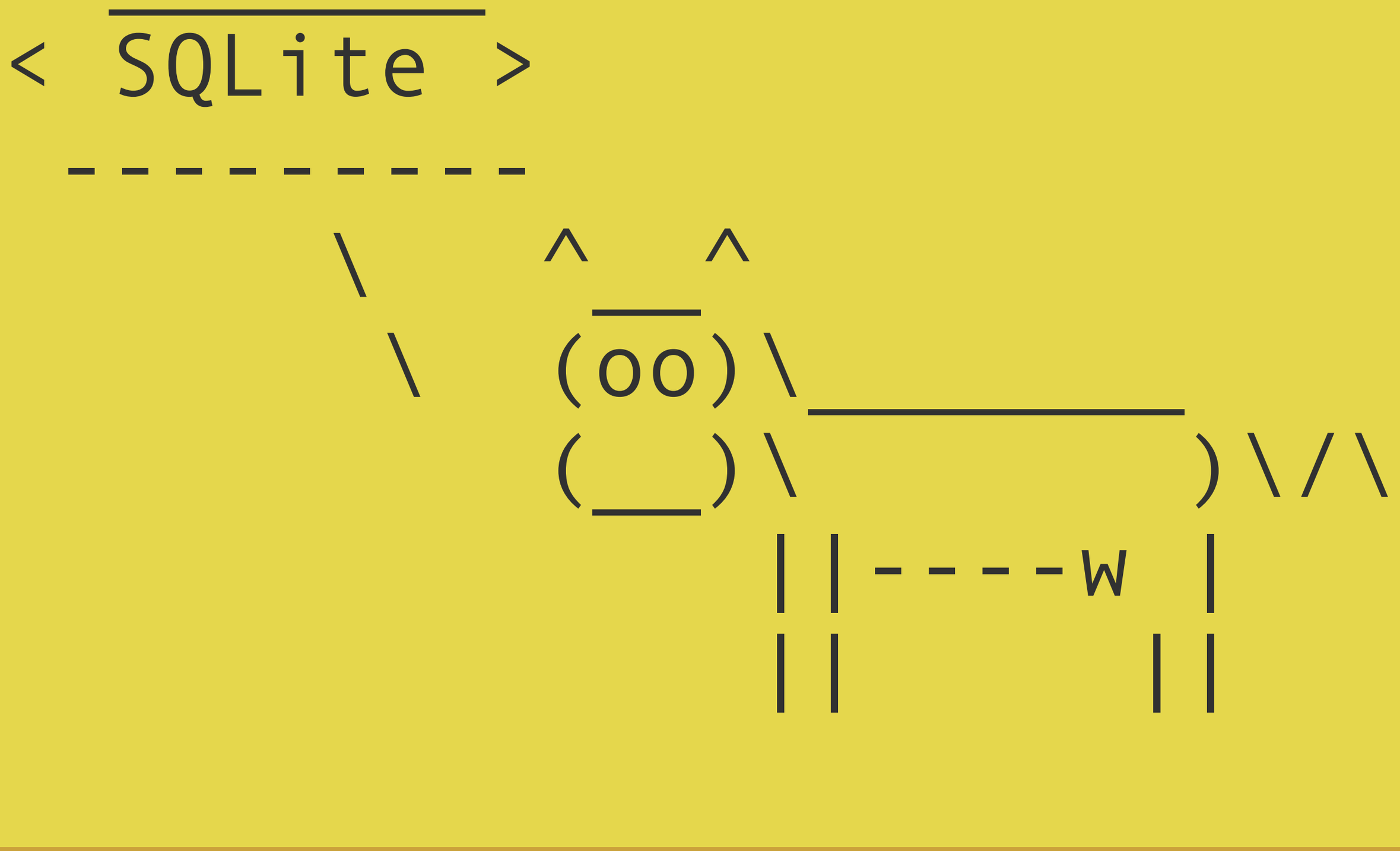
UUID
boolean
date
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector

Datatypes

UUID
date
boolean
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector

Datatypes

UUID
date
boolean
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector


< SQLite >
\ ^ ^
\ (oo)_____
(__)\ )\/\
||----w |
|| ||

Datatypes

null

integer

text

real

blob

Datatypes

UUID

date

boolean

timestamptz

array

interval

integer

bigint

smallint

line

XML

enum

char

money

serial

bytea

point

float

inet

polygon

numeric

circle

varchar

cidr

tsquery

timetz

text

path

time

timestamp

box

macaddr

tsvector

Indexes

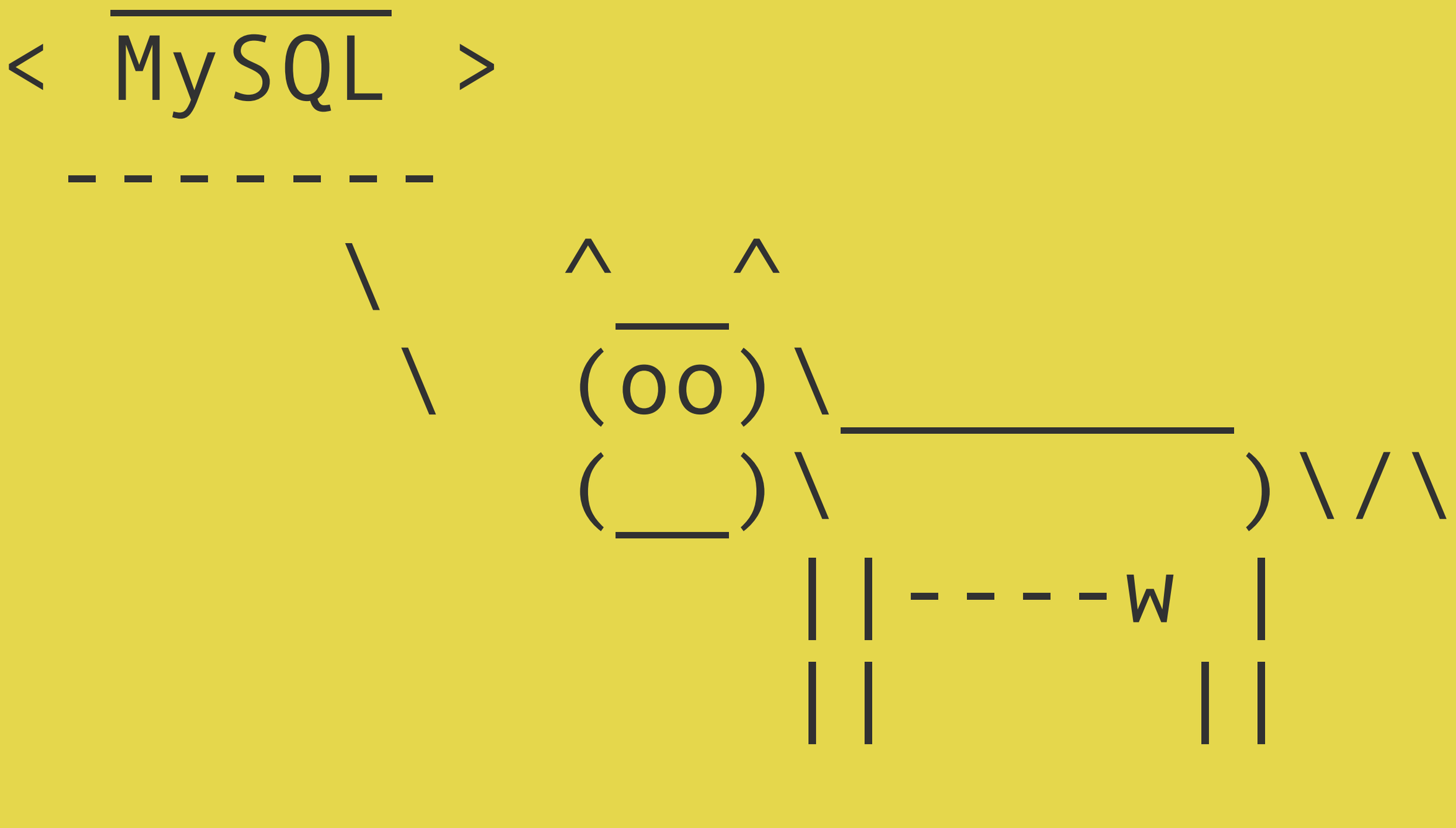

< Postgres >
\ ^ ^
\ (oo)_____
(__)\ )\/\
||----w |
|| ||

Indexes

Multiple Types

B-Tree, Gin, Gist, KNN, SP-Gist

CREATE INDEX CONCURRENTLY


MySQL
\ ^ ^
\ (oo)_____
(__)\ )\/\
||----w |
|| ||

Indexes

They exist

Digging In

Arrays

```
CREATE TABLE item (
    id serial NOT NULL,
    name varchar (255),
    tags varchar(255) [],
    created_at timestamp
);
```

Arrays

```
CREATE TABLE item (
    id serial NOT NULL,
    name varchar (255),
    tags varchar(255) [],
    created_at timestamp
);
```

Arrays

```
INSERT INTO item
VALUES (1, 'Django Pony',
'{“Programming”,”Animal”}', now());
```

```
INSERT INTO item
VALUES (2, 'Ruby Gem',
'{“Programming”,”Jewelry”}', now());
```

Arrays

```
INSERT INTO item
VALUES (1, 'Django Pony',
'{“Programming”,”Animal”}', now());
```

```
INSERT INTO item
VALUES (2, 'Ruby Gem',
'{“Programming”,”Jewelry”}', now());
```

Django

```
pip install djorm-ext-pgarray
pip install djorm-ext-expressions

models.py:
```

```
from djorm_pgarray.fields import ArrayField
from djorm_expressions.models import ExpressionManager

class Item(models.Model):
    name = models.CharField(max_length=255)
    tags = ArrayField(dbtype="varchar(255)")
    created_at = models.DateTimeField(auto_now=True)
```

Django

```
pip install djorm-ext-pgarray
pip install djorm-ext-expressions

models.py:
```

```
from djorm_pgarray.fields import ArrayField
from djorm_expressions.models import ExpressionManager

class Item(models.Model):
    name = models.CharField(max_length=255)
    tags = ArrayField(dbtype="varchar(255)")
    created_at = models.DateTimeField(auto_now=True)
```

Django

```
i = Item(
    name='Django Pony',
    tags=['Programming','Animal'])
i.save()

qs = Item.objects.where(
    SqlExpression("tags", "@>", ['programming'])
)
```

Django

```
i = Item(
    name='Django Pony',
    tags=['Programming','Animal'])
i.save()

qs = Item.objects.where(
    SqlExpression("tags", "@>", ['programming'])
)
```

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

isn

ltree

fuzzystrmatch

cube

dict_int

earthdistance

dict_xsyn

unaccent

btree_gist

tablefunc

pgrowlocks

NoSQL in your SQL

NoSQL in your SQL

```
CREATE EXTENSION hstore;
CREATE TABLE item (
    id integer NOT NULL,
    name varchar(255),
    data hstore,
);
```

NoSQL in your SQL

```
CREATE EXTENSION hstore;
CREATE TABLE item (
    id integer NOT NULL,
    name varchar(255),
    data hstore,
);
```

NoSQL in your SQL

```
INSERT INTO items
VALUES (
 1,
 'Pony',
 'rating => "4.0", color => “Pink”',
);
```

NoSQL in your SQL

```
INSERT INTO items
VALUES (
 1,
 'Pony',
 'rating => "4.0", color => “Pink”',
);
```

Django

```
pip install django-hstore

from django.db import models
from django_hstore import hstore

class Item(models.Model):
    name = models.CharField(max_length=250)
    data = hstore.DictionaryField(db_index=True)
    objects = hstore.Manager()

    def __unicode__(self):
        return self.name
```

Django

```
pip install django-hstore

from django.db import models
from django_hstore import hstore

class Item(models.Model):
    name = models.CharField(max_length=250)
    data = hstore.DictionaryField(db_index=True)
    objects = hstore.Manager()

    def __unicode__(self):
        return self.name
```

Django

```
Item.objects.create(
    name='Django Pony',
    data={'rating': '5'})
```

```
Item.objects.create(
    name='Pony',
    data={'color': 'pink', 'rating': '4'})
```

Django

```
Item.objects.create(
    name='Django Pony',
    data={'rating': '5'})
```

```
Item.objects.create(
    name='Pony',
    data={'color': 'pink', 'rating': '4'})
```

Django

```
colored_ponies =
  Product.objects.filter(data__contains='color')
print colored_ponies[0].data['color']
```

```
favorite_ponies =
  Product.objects.filter(data__contains={'rating': '5'})
print colored_ponies[0]
```

Django

```
colored_ponies =
  Product.objects.filter(data__contains='color')
print colored_ponies[0].data['color']
```

```
favorite_ponies =
  Product.objects.filter(data__contains={'rating': '5'})
print colored_ponies[0]
```

JSON

JSON

w/ PLV8

JSON

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

```
create or replace function
js(src text) returns text as $$
  return eval(
  "(function() { " + src + "})"
  )();
$$ LANGUAGE plv8;
```

JSON

```
SELECT
    '{"id":1,"email":
      "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

```
create or replace function
js(src text) returns text as $$
  return eval(
    "(function() { " + src + "})"
  )();
$$ LANGUAGE plv8;
```

JS Injection in DB: Bad Idea

Queueing

Pub/Sub?

Postgres a great Queue

Postgres a great Queue

Not With Polling

Trunk

```
pip install celery trunk

psql < sql/*.sql
celery worker -A tasks --loglevel=info

ipython -i tasks.py
>>> add.delay(2, 2)
```

Trunk

```
pip install celery trunk
```

```
psql < sql/*.sql
celery worker -A tasks --loglevel=info

ipython -i tasks.py
>>> add.delay(2, 2)
```

Trunk

```
from celery import Celery

celery = Celery('tasks')
celery.config_from_object({
    'BROKER_URL': 'trunk.transport.Transport://
localhost/trunk',
})

@celery.task
def add(x, y):
    return x + y
```

Trunk

```
from celery import Celery

celery = Celery('tasks')
celery.config_from_object({
    'BROKER_URL': 'trunk.transport.Transport://
localhost/trunk',
})

@celery.task
def add(x, y):
    return x + y
```

Text Search

Searching Text

Searching Text

Lucene

Elastic Search

Sphinx

Solr

Searching Text

Lucene

Elastic Search

Postgres

Sphinx

Solr

Full Text Search

```
CREATE TABLE posts (
  id serial,
  title varchar(255),
  content text,
  tags varchar(255)[],
  post_text tsvector
);

CREATE INDEX posttext_gin ON
posts USING GIN(post_text);

CREATE TRIGGER update_posttext
BEFORE INSERT OR UPDATE ON posts
FOR EACH ROW EXECUTE PROCEDURE
tsvector_update_trigger(
  'PostText','english',title, content, tags);
```

Full Text Search

```
CREATE TABLE posts (
  id serial,
  title varchar(255),
  content text,
  tags varchar(255)[],
  post_text tsvector
);

CREATE INDEX posttext_gin ON
posts USING GIN(post_text);

CREATE TRIGGER update_posttext
BEFORE INSERT OR UPDATE ON posts
FOR EACH ROW EXECUTE PROCEDURE
tsvector_update_trigger(
  'PostText','english',title, content, tags);
```

Django

```
from djorm_pgfulltext.models import SearchManager
from djorm_pgfulltext.fields import VectorField
from django.db import models

class Posts(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()

    search_index = VectorField()

    objects = SearchManager(
        fields = ('title', 'content'),
        config = 'pg_catalog.english',
        search_field = 'search_index',
        auto_update_search_field = True
    )
```

Django

```
from djorm_pgfulltext.models import SearchManager
from djorm_pgfulltext.fields import VectorField
from django.db import models

class Posts(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()

    search_index = VectorField()

    objects = SearchManager(
        fields = ('title', 'content'),
        config = 'pg_catalog.english',
        search_field = 'search_index',
        auto_update_search_field = True
    )
```

Django

```
Post.objects.search("documentation & about")
```

```
Post.objects.search("about | documentation")
```

Django

```
Post.objects.search("documentation & about")
```

```
Post.objects.search("about | documentation")
```

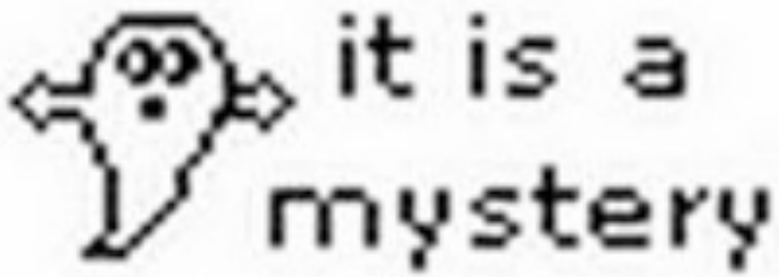
Indexes

Indexes

B-Tree
Generalized Inverted Index (GIN)
Generalized Search Tree (GIST)
K Nearest Neighbors (KNN)
Space Partitioned GIST (SP-GIST)

Indexes

Which do I use?



Indexes

B-Tree
Generalized Inverted Index (GIN)
Generalized Search Tree (GIST)
K Nearest Neighbors (KNN)
Space Partitioned GIST (SP-GIST)

Generalized Inverted Index (GIN)

- Use with multiple values in 1 column
- Array/hStore

Generalized Search Tree (GIST)

- Full text search
- Shapes
- PostGIS

Indexes

B-Tree
Generalized Inverted Index (GIN)
Generalized Search Tree (GIST)
K Nearest Neighbors (KNN)
Space Partitioned GIST (SP-GIST)

GeoSpatial

GeoDjango

http://geodjango.org/

https://speakerdeck.com/pyconslides/location-location-location

Read Only

Lanyrd is undergoing maintenance.

Sorry about this, we are in **read only** mode. You can browse the site as normal but you won't be able to sign in or make changes.

Read Only

Django

```
pip install django-db-tools

settings.py

MIDDLEWARE_CLASSES = (
    # ...
    'dbtools.middleware.ReadOnlyMiddleware',
    # ...
)

READ_ONLY_MODE = True
```

Django

```
pip install django-db-tools

settings.py

MIDDLEWARE_CLASSES = (
    # ...
    'dbtools.middleware.ReadOnlyMiddleware',
    # ...
)

READ_ONLY_MODE = True
```

Django

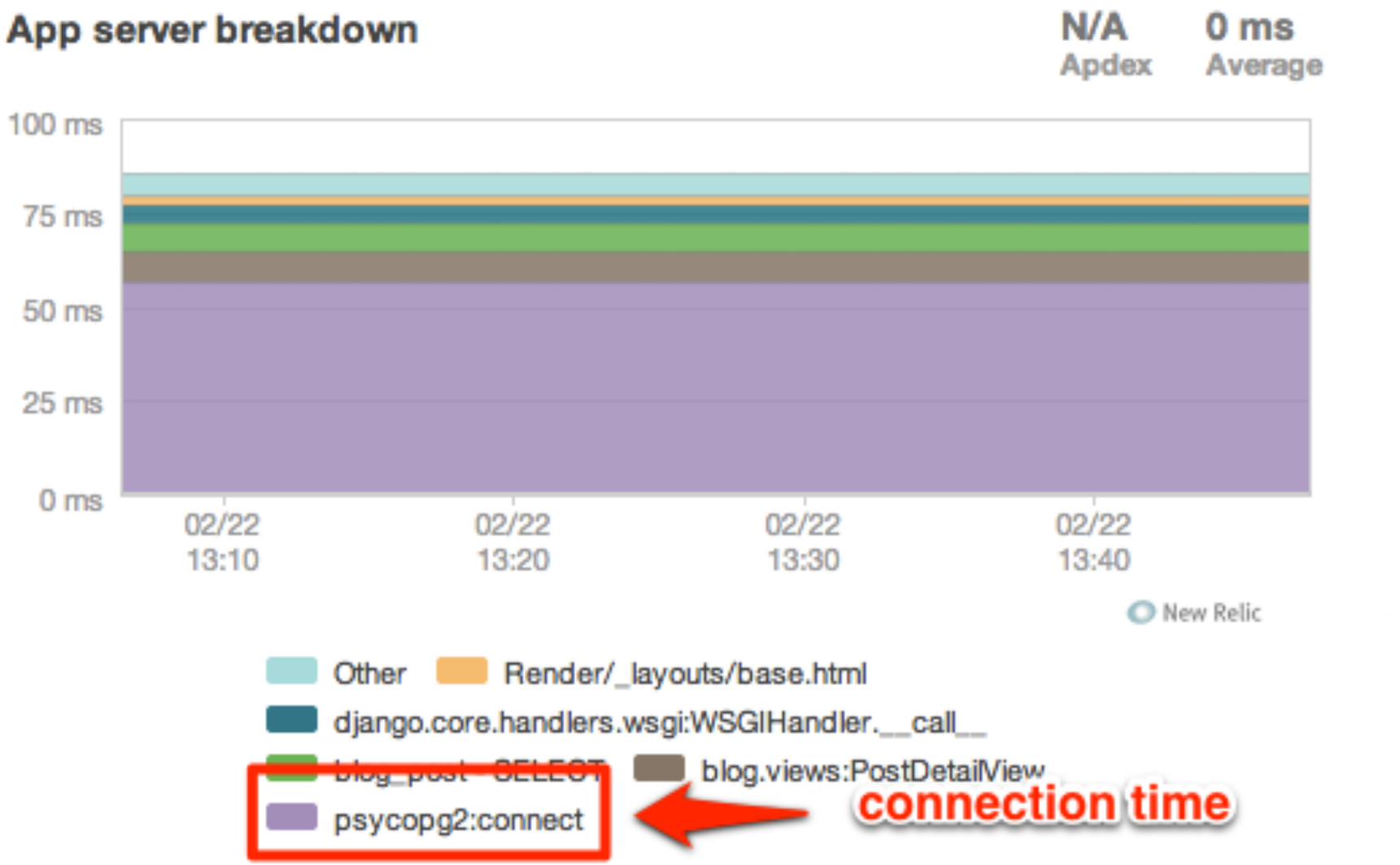
```
pip install django-db-tools

settings.py

MIDDLEWARE_CLASSES = (
    # ...
    'dbtools.middleware.ReadOnlyMiddleware',
    # ...
)

READ_ONLY_MODE = os.environ[‘READ_ONLY_MODE']
```

Django

```
pip install django-db-tools

settings.py

MIDDLEWARE_CLASSES = (
    # ...
    'dbtools.middleware.ReadOnlyMiddleware',
    # ...
)

READ_ONLY_MODE = os.environ[‘READ_ONLY_MODE']
```

One More

Connections



Connections

django-postgrespool
djorm-ext-pool
django-db-pool

django-postgrespool

```
import dj_database_url
import django_postgrespool

DATABASE = { 'default': dj_database_url.config() }
DATABASES['default']['ENGINE'] = 'django_postgrespool'


SOUTH_DATABASE_ADAPTERS = {
    'default': 'south.db.postgresql_psycopg2'
}
```

django-postgrespool

```
import dj_database_url
import django_postgrespool

DATABASE = { 'default': dj_database_url.config() }
DATABASES['default']['ENGINE'] = 'django_postgrespool'


SOUTH_DATABASE_ADAPTERS = {
    'default': 'south.db.postgresql_psycopg2'
}
```

Limitations?

Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we've had to make design decisions on which features to support and which assumptions we can make safely.

Postgres

Its great

Django ORM

Its not so bad

Thanks!

http://www.speakerdeck.com/craigkerstiens