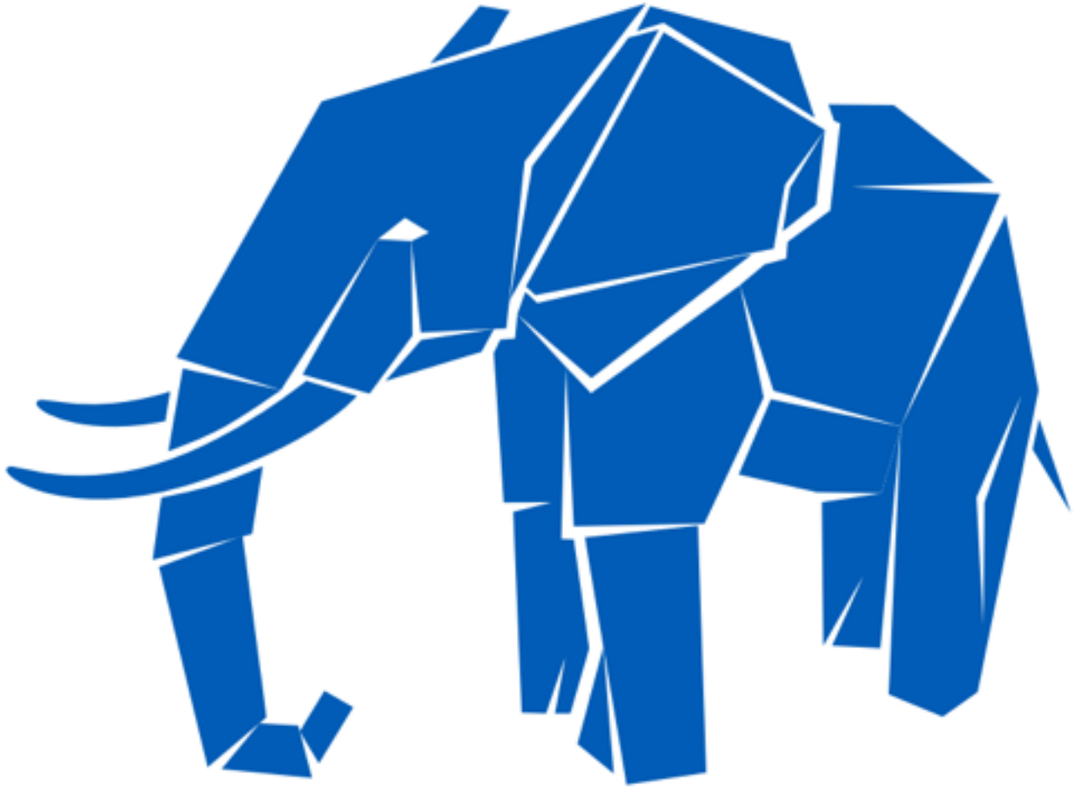
Postgres Demystified

Craig Kerstiens
@craigkerstiens
http://www.craigkerstiens.com

https://speakerdeck.com/u/craigkerstiens/p/postgres-demystified

Postgres Demystified

Postgres Demystified



herokupostgres

Getting Setup

Postgres.app

Agenda

Brief History
Developing w/ Postgres
Postgres Performance
Querying

Postgres History

Postgres
PostgresQL

Post Ingress
Around since 1989/1995
Community Driven/Owned

MVCC

Each query sees transactions committed before it

Locks for writing don't conflict with reading

Why Postgres

Why Postgres

“ its the emacs of databases”

Developing w/ Postgres

Basics

psql is your friend

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
boolean
date
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
numeric
polygon
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
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

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

Datatypes

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

Datatypes

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

Datatypes

```
INSERT INTO items
VALUES (1, 'Ruby Gem', '{“Programming”,”Jewelry”}', now());

INSERT INTO items
VALUES (2, 'Django Pony', '{“Programming”,”Animal”}', now());
```

Datatypes

UUID
boolean
timestamptz
date
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 wish list

email

url

zip

phone

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

NoSQL in your SQL

```
CREATE EXTENSION hstore;
CREATE TABLE users (
    id integer NOT NULL,
    email character varying(255),
    data hstore,
    created_at timestamp without time zone,
    last_login timestamp without time zone
);
```

hStore

```
INSERT INTO users
VALUES (
  1,
  'craig.kerstiens@gmail.com',
  'sex => "M", state => “California”',
  now(),
  now()
);
```

JSON

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

9.2

JSON

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

V8 w/ PLV8

9.2

JSON

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

V8 w/ PLV8

Bad Idea

9.2

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;
```

Bad Idea

9.2

Range Types

9.2

Range Types

```
CREATE TABLE talks (room int, during tsrange);
INSERT INTO talks VALUES
    (3, '[2012-09-24 13:00, 2012-09-24 13:50)');
```

9.2

Range Types

```
CREATE TABLE talks (room int, during tsrange);
INSERT INTO talks VALUES
    (3, '[2012-09-24 13:00, 2012-09-24 13:50)');


ALTER TABLE talks ADD EXCLUDE USING gist (during WITH &&);
INSERT INTO talks VALUES
    (1108, '[2012-09-24 13:30, 2012-09-24 14:00)');
ERROR:  conflicting key value violates exclusion constraint
"talks_during_excl"
```

9.2

Full Text Search

Full Text Search

TSVECTOR - Text Data
TSQUERY - Search Predicates

Specialized Indexes and Operators

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

PostGIS

PostGIS

1. New datatypes i.e. (2d/3d boxes)

PostGIS

1. New datatypes — i.e. (2d/3d boxes)
2. New operators — i.e. SELECT foo && bar ...

PostGIS

1. New datatypes i.e. (2d/3d boxes)
2. New operators i.e. SELECT foo && bar ...
3. Understand relationships and distance

i.e. person within location, nearest distance

Performance

Sequential Scans

Sequential Scans

They're Bad

Sequential Scans

They're Bad (most of the time)

Indexes

Indexes

They're Good

Indexes

They're Good (most of the time)

Indexes

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

Indexes

B-Tree

- Default
- Usually want this

Indexes

Generalized Inverted Index (GIN)

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

Indexes

Generalized Search Tree (GIST)

- Full text search
- Shapes

Understanding Query Perf

Given

```
SELECT last_name
FROM employees
WHERE salary >= 50000;
```

Explain

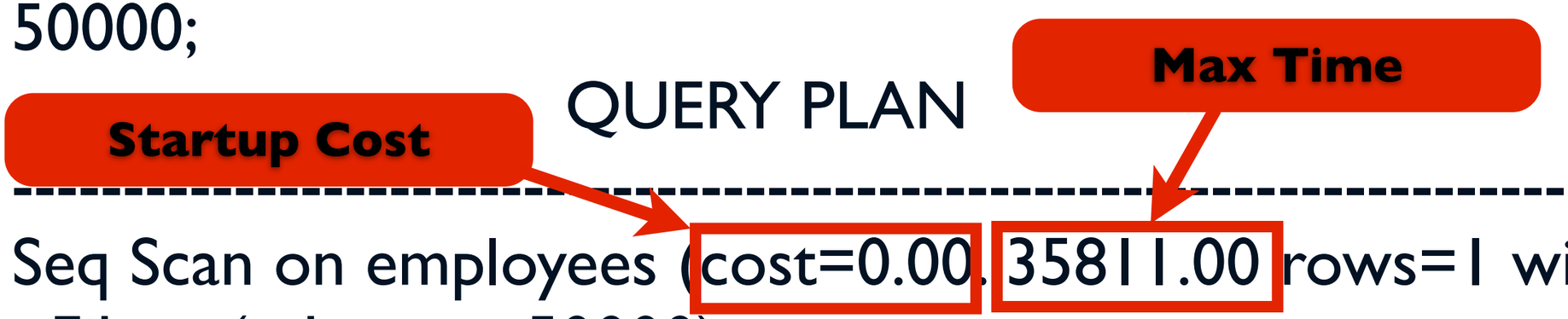
```
# EXPLAIN SELECT last_name FROM employees WHERE salary >=
50000;
                          QUERY PLAN
-----------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6)
  Filter: (salary >= 50000)
(3 rows)
```

Explain

```
# EXPLAIN SELECT last_name FROM employees WHERE salary >=
50000;
                          QUERY PLAN
-------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6)
  Filter: (salary >= 50000)
(3 rows)
```

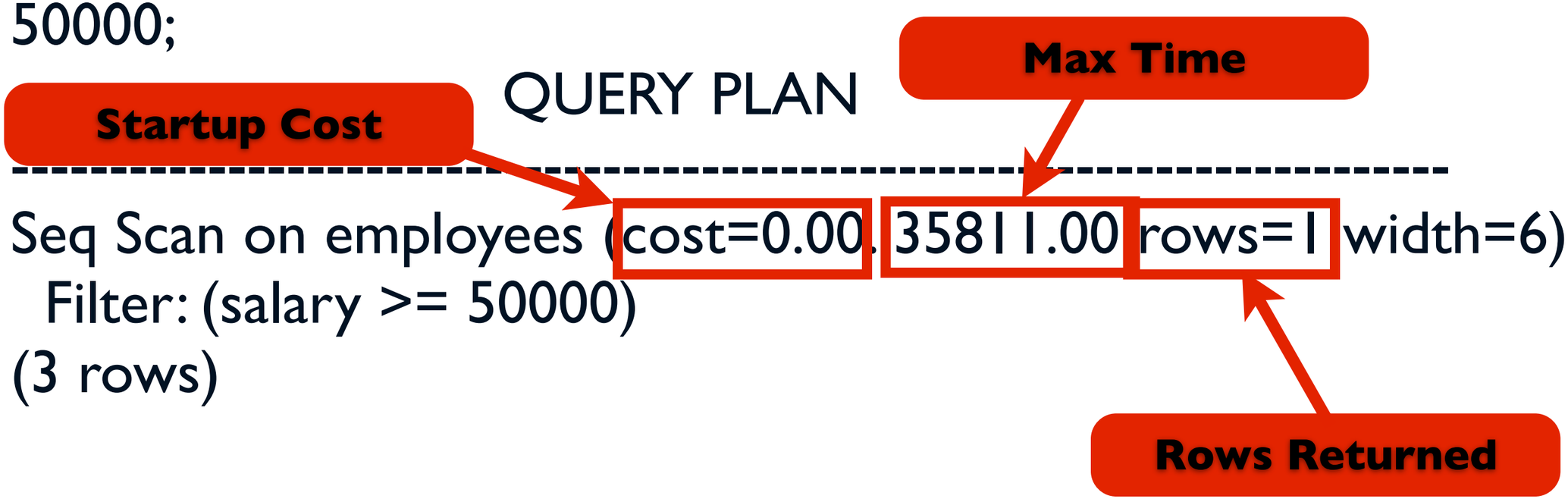
Explain

```
# EXPLAIN SELECT last_name FROM employees WHERE salary >=
50000;
                        QUERY PLAN
-----------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6)
  Filter: (salary >= 50000)
(3 rows)
```



Explain

```
# EXPLAIN SELECT last_name FROM employees WHERE salary >=
50000;
                          QUERY PLAN
-------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6)
  Filter: (salary >= 50000)
(3 rows)
```

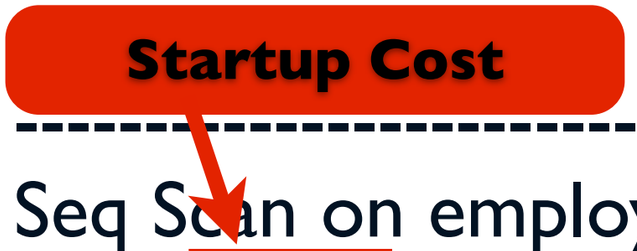


Explain

```
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                QUERY PLAN
-----------------------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6) (actual
time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)
```

Explain

```
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6) (actual
time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)
```

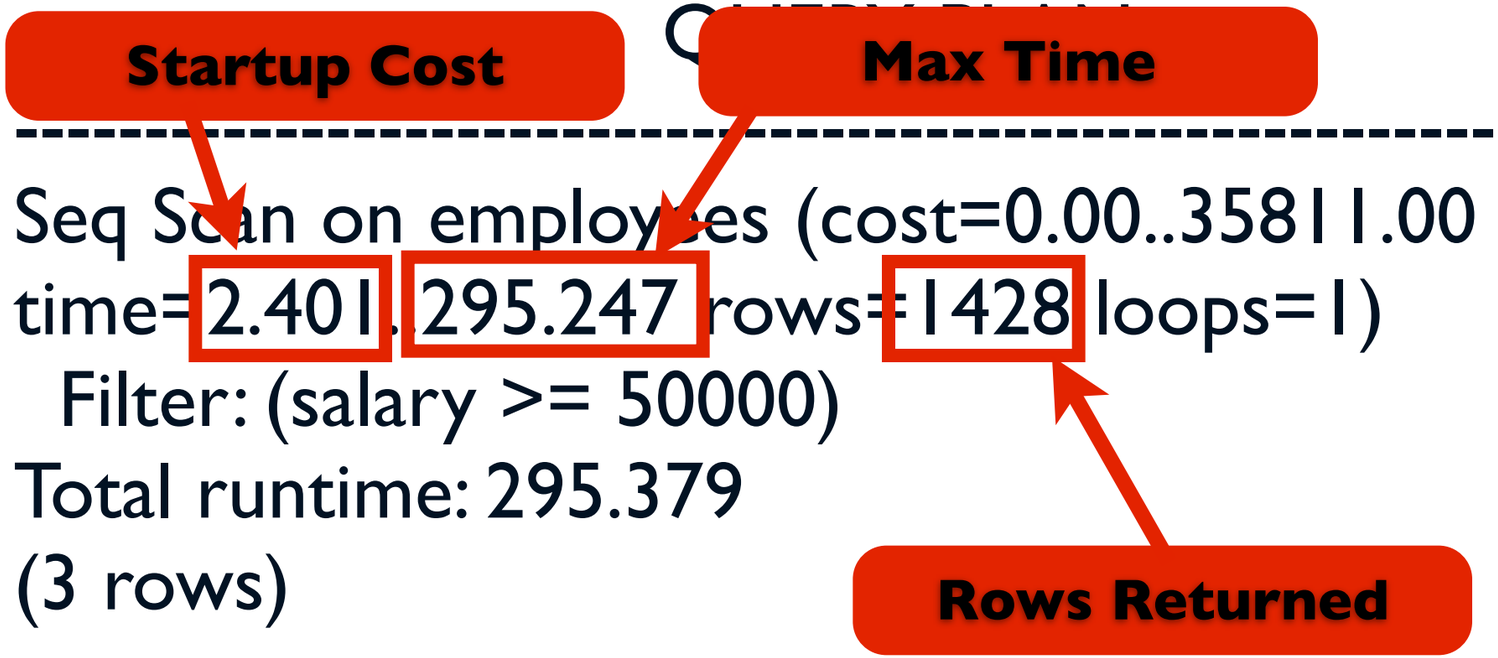


Explain

```
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6) (actual
time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)
```

Explain

```
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                   QUERY PLAN
-------------------------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6) (actual
time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)
```



Explain

```
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                   QUERY PLAN
-------------------------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1 width=6) (actual
time=2.401..295.247 rows=1428 loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)
```

Startup Cost

Max Time

Rows Returned

```
# CREATE INDEX idx_emps ON employees (salary);
```

```
# CREATE INDEX idx_emps ON employees (salary);
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                QUERY PLAN
-----------------------------------------------------------------------------
Index Scan using idx_emps on employees (cost=0.00..8.49 rows=1
width=6) (actual time = 0.047..1.603 rows=1428 loops=1)
  Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

```
# CREATE INDEX idx_emps ON employees (salary);
# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
                                QUERY PLAN
-----------------------------------------------------------------------------
Index Scan using idx_emps on employees (cost=0.00..8.49 rows=1
width=6) (actual time = 0.047..1.603 rows=1428 loops=1)
  Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

Indexes Pro Tips

Indexes Pro Tips

CREATE INDEX CONCURRENTLY

Indexes Pro Tips

CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

Indexes Pro Tips

CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

SELECT * WHERE foo LIKE '%bar% is BAD

Indexes Pro Tips

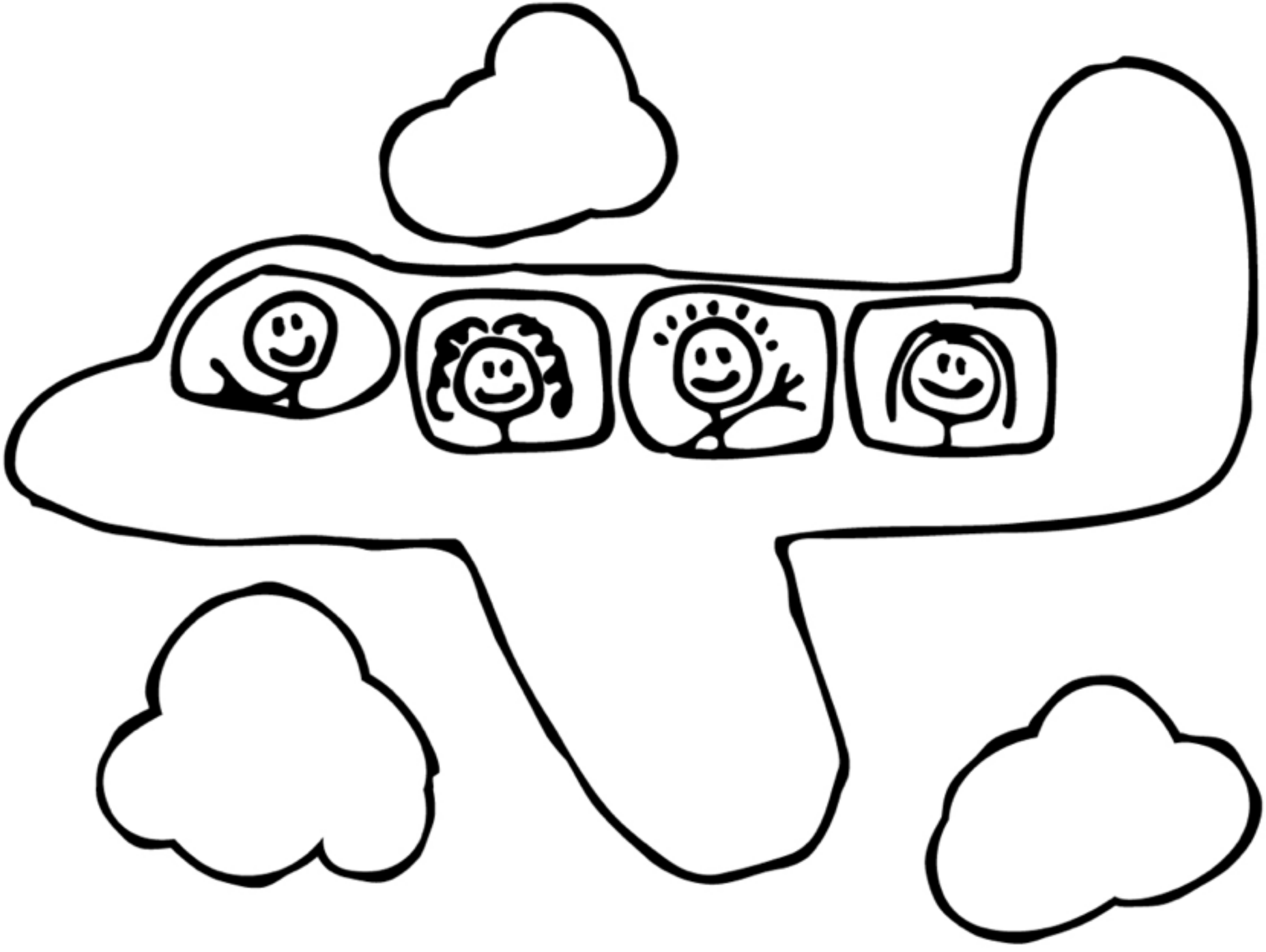
CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

SELECT * WHERE foo LIKE '%bar% is BAD
SELECT * WHERE Food LIKE 'bar%' is OKAY

Extensions

dblink
hstore
uuid-ossp
trigram
pgstattuple
citext
pgcrypto
pgrowlocks
isn
ltree
fuzzystrmatch
cube
earthdistance
dict_int
tablefunc
unaccent
btree_gist
dict_xsyn



Index Hit Rate

```
SELECT
  relname,
  100 * idx_scan / (seq_scan + idx_scan),
  n_live_tup
FROM pg_stat_user_tables
ORDER BY n_live_tup DESC;
```

Cache Hit Rate

```
SELECT
 relname::text,
 heap_blks_read + heap_blks_hit as reads,
 round(100 * heap_blks_hit / (heap_blks_hit + heap_blks_read)) as
hit_pct,
 round(100 * idx_blks_hit / (idx_blks_hit + idx_blks_read)) as idx_hit_pct
FROM  pg_statio_user_tables
WHERE heap_blks_hit + heap_blks_read + idx_blks_hit + idx_blks_read
> 0
ORDER BY 2 DESC;
```

pg_stats_statements

9.2

pg_stats_statements

```
$ select * from pg_stat_statements where query ~ 'from users where email';

userid             | 16384
dbid               | 16388
query              | select * from users where email = ?;
calls              | 2
total_time         | 0.000268
rows               | 2
shared_blks_hit    | 16
shared_blks_read   | 0
shared_blks_dirtied | 0
shared_blks_written | 0
local_blks_hit     | 0
local_blks_read    | 0
local_blks_dirtied | 0
local_blks_written | 0
temp_blks_read     | 0
temp_blks_written  | 0
time_read          | 0
time_write         | 0
```

9.2

pg_stats_statements

9.2

pg_stats_statements

```
SELECT query, calls, total_time, rows, 100.0 * shared_blks_hit /
               nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
          FROM pg_stat_statements ORDER BY total_time DESC LIMIT 5;
```

```
query        | UPDATE pgbench_branches SET bbalance = bbalance + ?
WHERE bid = ?;
calls        | 3000
total_time   | 9609.00100000002
rows         | 2836
hit_percent | 99.9778970000200936
```

9.2

Querying

Window Functions

Example:

Biggest spender by state

Window Functions

```
SELECT
  email,
  users.data->'state',
  sum(total(items)),
  rank() OVER
    (PARTITION BY users.data->'state'
    ORDER BY sum(total(items)) desc)
FROM
  users, purchases
WHERE purchases.user_id = users.id
GROUP BY 1, 2;
```

Window Functions

```
SELECT
  email,
  users.data->'state',
  sum(total(items)),
  rank() OVER
    (PARTITION BY users.data->'state'
    ORDER BY sum(total(items)) desc)
FROM
  users, purchases
WHERE purchases.user_id = users.id
GROUP BY 1, 2;
```

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

Fuzzystrmatch

Fuzzystrmatch

```
SELECT soundex('Craig'), soundex('Will'), difference('Craig', 'Will');
```

Fuzzystrmatch

```
SELECT soundex('Craig'), soundex('Will'), difference('Craig', 'Will');
```

```
SELECT soundex('Craig'), soundex('Greg'), difference('Craig', 'Greg');
SELECT soundex('Willl'), soundex('Will'), difference('Willl', 'Will');
```

Moving Data Around

```
\copy (SELECT * FROM users) TO '~/users.csv';

\copy users FROM '~/users.csv';
```

db_link

```
SELECT dblink_connect('myconn', 'dbname=postgres');
SELECT * FROM dblink('myconn','SELECT * FROM foo') AS t(a int, b text);

 a |    b
---+---------
 1 | example
 2 | example2
```

Foreign Data Wrappers

oracle

mysql

odbc

twitter

sybase

redis

jdbc

files

couch

s3

www

ldap

informix

Foreign Data Wrappers

```
CREATE EXTENSION redis_fdw;

CREATE SERVER redis_server
    FOREIGN DATA WRAPPER redis_fdw
    OPTIONS (address '127.0.0.1', port '6379');

CREATE FOREIGN TABLE redis_db0 (key text, value text)
    SERVER redis_server
    OPTIONS (database '0');

CREATE USER MAPPING FOR PUBLIC
    SERVER redis_server
    OPTIONS (password 'secret');
```

Query Redis from Postgres

```
SELECT *
FROM redis_db0;

SELECT
  id,
  email,
  value as visits
FROM
  users,
  redis_db0
WHERE ('user_' || cast(id as text)) = cast(redis_db0.key as text)
    AND cast(value as int) > 10;
```

Readability

Readability

```
WITH top_5_products AS (
  SELECT products.*, count(*)
  FROM products, line_items
 WHERE products.id = line_items.product_id
  GROUP BY products.id
  ORDER BY count(*) DESC
  LIMIT 5
)

SELECT users.email, count(*)
FROM users, line_items, top_5_products
WHERE line_items.user_id = users.id
  AND line_items.product_id = top_5_products.id
GROUP BY 1
ORDER BY 1;
```

Common Table Expressions

```
WITH top_5_products AS (
  SELECT products.*, count(*)
  FROM products, line_items
 WHERE products.id = line_items.product_id
  GROUP BY products.id
  ORDER BY count(*) DESC
  LIMIT 5
)

SELECT users.email, count(*)
FROM users, line_items, top_5_products
WHERE line_items.user_id = users.id
  AND line_items.product_id = top_5_products.id
GROUP BY 1
ORDER BY 1;
```

Brief History
Developing w/ Postgres
Postgres Performance
Querying

Extras

Extras

Listen/Notify

Extras

Listen/Notify

Per Transaction Synchronous Replication

Extras

Listen/Notify
Per Transaction Synchronous Replication
Drop index concurrently

Postgres - 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