This slide intentionally left blank

~~Cheap Parlor Tricks in psql~~

~~Good PostgreSQL Advice~~

~~Top 10 Things You Should Do in PostgreSQL~~

Random and-totally-not-thrown-together PostgreSQL Talk #84

Tony Drake, Senior Web Developer, MOBI Wireless Management

Inby.rb, January 2017

Topics of Discussion

- Four cool things you can do with psql
- Do’s and Don’ts for PostgreSQL
- ~~Couple~~ A “neat” query feature of PostgreSQL

Cool thing with psql: \e

- Use \e to be dropped into a text editor
- Prepopulated with the previously ran query/command
- Save and exit to run the query
- Uses the system’s $EDITOR environment variable

- Great for writing/editing large queries!

Cool thing with psql: \timing

- Run “\timing” to turn timing mode on
- Shows the time each query takes to execute

Cool thing with psql: \watch

- Append \watch to the end of your query
- Query will be executed repeatedly ever few seconds

- SELECT now() **-** query_start, state, query FROM pg_stat_activity \watch

Cool thing with psql: \copy (CSV export)

- Use the \copy command to send a query to a file as a CSV
- \copy (SELECT * FROM users) To '/tmp/test.csv' With CSV HEADER
- Does not require super user access (unlike “COPY TO”)

Do’s and Don’t’s with PostgreSQL *(A disclaimer)*

- **Totally not scientific**
- **Based on experience**
- **YMMV**
- Restrictions apply
- Do not take while pregnant
- Valid only in the continental United States
- Void where prohibited
- See a doctor if this lasts longer than four hours

Do – Index Concurrently

- Normal indexes lock the table while they are being applied
- Concurrently takes longer to apply, but won’t lock the table
- **WAITS FOR LONG RUNNING TRANSACTIONS TO FINISH FIRST**

```
class AddStatusIndexToPosts < ActiveRecord::Migration
  disable_ddl_transaction!

  def change
    add_index :posts, :status_id, algorithm: :concurrently
  end
end
```

Don’t – Over index tables

- Just because you have an index on a table, doesn’t mean it’s being actively used
- Indexes take up space just like tables. Unused index = bloat and waste
- Note that tables have to be a certain size before PostgreSQL deems it necessary to use them. Sometimes a sequence scan of a small table is just as fast.

```
SELECT *
FROM pg_stat_all_indexes
WHERE schemaname = 'public'
ORDER BY indexrelname DESC, idx_scan ASC;
```

Do – Apply indexes to foreign keys
(When they will be actually used)

- Userbelongs to a Group
  - group_id lives on users

- Apply an index to this column if
  - There will be a need to ask “I need all the users that belong to this group”
  - You will be actively joining users to groups in everyday queries

Don’t – Use like or ilike with btree indexes

- Using “like” or “ilike” CANNOT USE BTREE INDEXES
  - Ok I lied… it can sorta use them if a wildcard is on the far right
  - SELECT … WHERE foo like ’bar%’
- pg_trgm provides features for specialized GIN and GIST indexes that ilike and like can use!
- https://www.postgresql.org/docs/current/static/pgtrgm.html

Do – Create indexes for sorting

- Do you sort by *and* filter by a certain column a lot?
  - Example: publish_on for a blogs table
  - SELECT *
    FROM blogs
    WHERE publish_on <= now()
    ORDER BY publish_on DESC
- PostgreSQL must sort the entire resultset after filtering the initial rows
- You can even create an index for the sort order of the column
  - CREATE INDEX foo ON blogs (publish_on DESC)

Don’t – Use subqueries when you can join

- SELECT * FROM widgets
  WHERE owner_id IN(
  	SELECT id FROM users WHERE group_id = 2
)

- SELECT * FROM widgets
  JOIN users ON widgets.owner_id = users.id
  WHERE users.group_id = 2

Do – Create specialized indexes

- SELECT * FROM employees WHERE LOWER(cost_center) IN(?)
- Does not use a standard index on cost_center
- CREATE INDEX foo ON employees (LOWER(cost_center))

Don’t – Use hstore

- Use JSON or JSONB instead
- Queryable
- Stores datatypes
- No need for an extension

- Gotchas:
  - Can take up more physical space on disk
  - Can’t SELECT DISTINCT * with JSON (JSONB is ok)

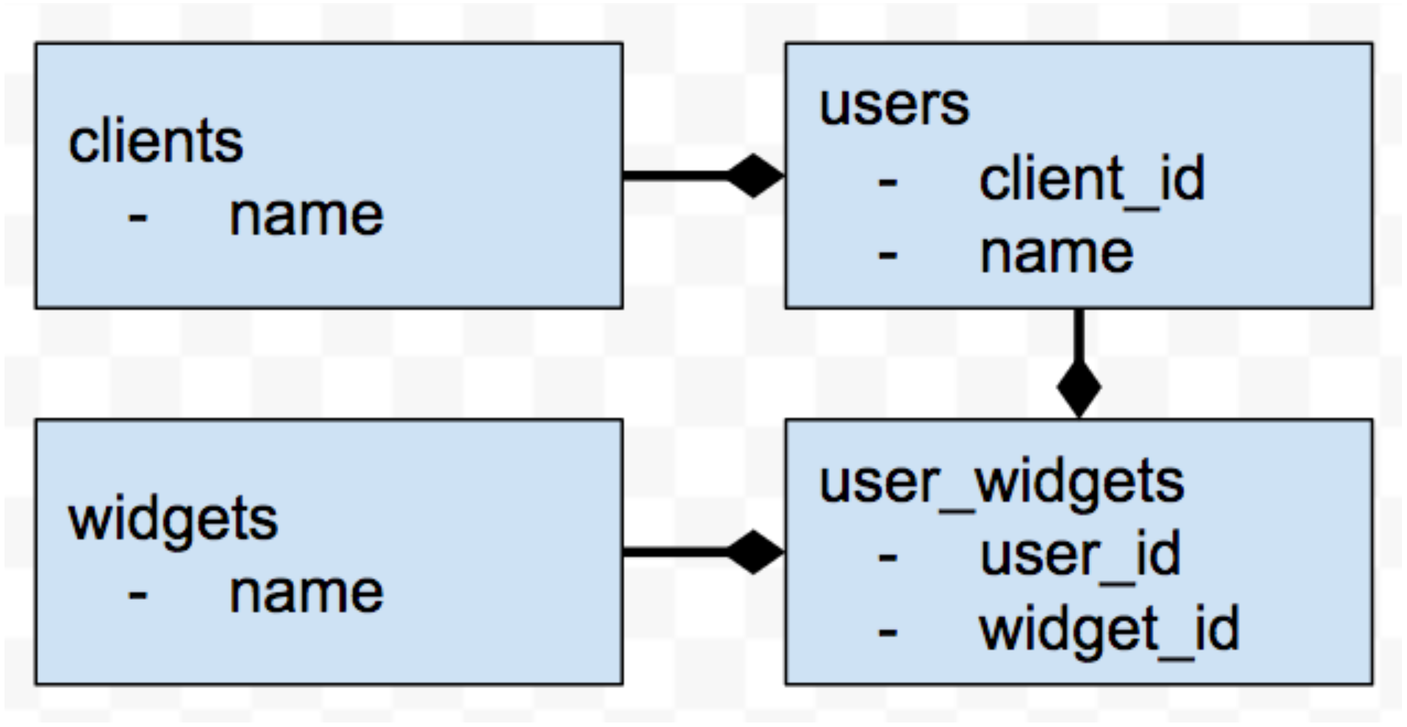
Do – Use explain analyze

- “Why is this query slow”?
- ”Is my query using indexes”?

- EXPLAIN – Gives the projected query plain
- EXPLAIN ANALYZE – Executes query, gives exact plan, and timings

- https://explain.depesz.com/

Don’t – Over normalize



What widgets are actually being used by a client?

```
class Client
  has_many :users
  has_many :widgets, through: :users
end

class User
  belongs_to :client
  has_many :widgets
end

class UserWidget
  belongs_to :user
  belongs_to :widget
end
```

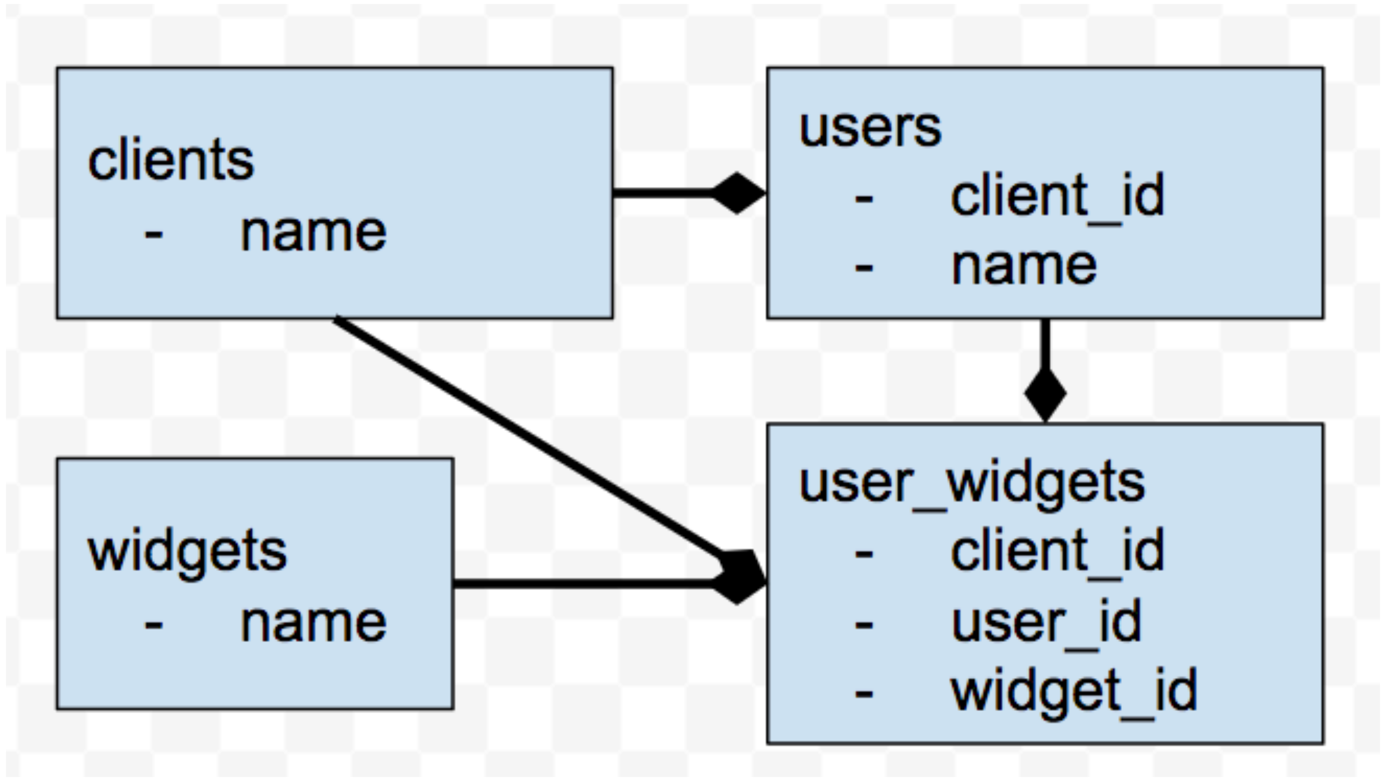
Don’t – Over normalize

```
client.widgets.pluck(“DISTINCT widget_id”)

SELECT DISTINCT widget_id
FROM user_widgets
JOIN users ON clients.user_id = users.id
WHERE user_widgets.user_id = users.id
```

So… what happens if a client has… 10,000 users?

Don’t – Over normalize



What widgets are actually being used by a client?

```
class Client
  has_many :users
  has_many :widgets
end

class User
  belongs_to :client
  has_many :widgets
end

class UserWidget
  belongs_to :user
  belongs_to :widget
end
```

Don’t – Over normalize

```
client.widgets.pluck(“DISTINCT widget_id”)

SELECT DISTINCT widget_id
FROM user_widgets
WHERE user_widgets.client_id = 123
```

Neat Feature – The WITH Clause

```
WITH unhappy_users AS (
  SELECT count(*) user_count
  FROM complaints
  GROUP BY user_id
  HAVING count(*) > 3
)
SELECT ROUND(
  (unhappy_users.user_count / (SELECT count(*) FROM users) * 100) , 1
)
FROM unhappy_users
```

Neat Feature – The WITH Clause

```
WITH user_client_relation AS (
  SELECT  u.client_id AS user_client_id, uw.user_id
  FROM users u, user_widgets uw
  WHERE uw.user_id = u.id
)
UPDATE user_widgets
SET client_id = user_client_relation.user_client_id
FROM user_client_relation
WHERE user_widgets.user_id = user_client_relation.user_id
```

Other tidbits

- Use integers instead of strings to filter when possible
- Always specify an ORDER BY for large, paginated datasets
- Any ALTER TABLE requires a full table lock
- Window functions
  - https://www.postgresql.org/docs/current/static/tutorial-window.html

The End

Gets these slides at https://github.com/t27duck/showandtell