Test Your Database with pgTAP

David E. Wheeler
david@kineticode.com
http://justatheory.com/

PostgreSQL Conference West 2008

Working on a New App

Working on a New App

```
CREATE TABLE users (
  nick  TEXT    PRIMARY KEY,
  pass  TEXT    NOT NULL,
);
```

Working on a New App

```
CREATE TABLE users (
  nick  TEXT    PRIMARY KEY,
  pass  TEXT    NOT NULL,
);

CREATE UNIQUE INDEX udx_users_nick
ON users(LOWER(nick));
```

Working on a New App

```
CREATE TABLE users (
  nick  TEXT    PRIMARY KEY,
  pass  TEXT    NOT NULL,
);

CREATE UNIQUE INDEX udx_users_nick
ON users(LOWER(nick));
```

- Bah! Two indexes!

Lookup a User

Lookup a User

```
SELECT *
  FROM users
 WHERE LOWER(nick) = LOWER(?);
```

Lookup a User

```
SELECT *
  FROM users
 WHERE LOWER(nick) = LOWER(?);
```

- I finally got sick of this

Lookup a User

```
SELECT *
  FROM users
 WHERE LOWER(nick) = LOWER(?);
```

- I finally got sick of this
- What to do?

- Well, I've got these great tools…


CITEXT

Shave Me!

Inspiration Strikes!

Inspiration Strikes!

- Hey, I know!

Inspiration Strikes!

- Hey, I know!
- I can implement CITEXT 2.0!

Inspiration Strikes!

- Hey, I know!
- I can implement CITEXT 2.0!
- (Arm waving)

Inspiration Strikes!

- Hey, I know!
- I can implement CITEXT 2.0!
- (Arm waving)
- Hey, it compiles!

Inspiration Strikes!

- Hey, I know!
- I can implement CITEXT 2.0!
- (Arm waving)
- Hey, it compiles!
- Now what?

Inspiration Strikes!

- Hey, I know!
- I can implement CITEXT 2.0!
- (Arm waving)
- Hey, it compiles!
- Now what?
- Write tests, of course

Writing Tests

Writing Tests

- Followed varlena.com example

Writing Tests

- Followed varlena.com example
- Wrote pure SQL test script

Writing Tests

- Followed varlena.com example
- Wrote pure SQL test script
- Hard to know if values correct

Writing Tests

- Followed varlena.com example
- Wrote pure SQL test script
- Hard to know if values correct
- Perl tests: Output easy to interpret

Writing Tests

- Followed varlena.com example
- Wrote pure SQL test script
- Hard to know if values correct
- Perl tests: Output easy to interpret
- Why not output something simple?

First Stab at SQL Testing

First Stab at SQL Testing

```
\set QUIET 1
\set ON_ERROR_STOP 1
\pset format unaligned
\pset tuples_only
\pset pager

SELECT 'a'::citext = 'a'::citext;
SELECT 'a'::citext = 'A'::citext;
SELECT 'à'::citext = 'À'::citext;
```

First Pass at Output

First Pass at Output

```
% psql try -f test.sql
 t
 t
 t
```

First Pass at Output

```
% psql try -f test.sql
 t
 t
 t
```

- Cool!

First Pass at Output

```
% psql try -f test.sql
t
t
t
```

- Cool!
- Inspired by Perl test output: easy to scan

First Pass at Output

```
% psql try -f test.sql
t
t
t
```

- Cool!
- Inspired by Perl test output: easy to scan
- But…could it be better?


ExposedPlanet.com
TAP


ExposedPlanet.com
TAP
Shave Me!

Inspiration Strikes Again!

Inspiration Strikes Again!

- Hey, I know!

Inspiration Strikes Again!

- Hey, I know!
- I can port Test::More from Perl

Inspiration Strikes Again!

- Hey, I know!
- I can port Test::More from Perl
- Test::More outputs TAP

Inspiration Strikes Again!

- Hey, I know!
- I can port Test::More from Perl
- Test::More outputs TAP
- Test output will be easy to read

OMG TAP WTF?

OMG TAP WTF?

> Test Anything Protocol (TAP) is a general purpose format for transmitting the result of test programs to a thing which interprets and takes action on those results. Though it is language agnostic, it is primarily used by Perl modules.
>
> —Wikipedia

What is TAP

What is TAP

- What does that mean in practice?

What is TAP

- What does that mean in practice?
- Test output easy to interpret

What is TAP

- What does that mean in practice?
- Test output easy to interpret
  - By humans

What is TAP

- What does that mean in practice?
- Test output easy to interpret
  - By humans
  - By computers

What is TAP

- What does that mean in practice?
- Test output easy to interpret
  - By humans
  - By computers
  - By aliens

What is TAP

- What does that mean in practice?
- Test output easy to interpret
  - By humans
  - By computers
  - By aliens
  - By gum!

TAP Output

TAP Output

- So what does it look like?

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

- So what does it look like?

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
not ok 4 - Should have a state
#   Failed test 'Should have a state'
#   at t/try.t line 12.
ok 5 - Should be able to reset
# Looks like you failed 1 test of 5.
```

TAP Output

TAP Output

- Passing tests even easier to grok

TAP Output

- Passing tests even easier to grok

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
ok 4 - Should have a state
ok 5 - Should be able to reset
```

TAP Output

- Passing tests even easier to grok

```
% perl -Ilib t/try.t
1..5
ok 1 - use FSA::Rules;
ok 2 - Create FSA::Rules object
ok 3 - Start the machine
ok 4 - Should have a state
ok 5 - Should be able to reset
```

- Think they passed?

Writing Tests

Writing Tests

- And how are those tests written?

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

```
use Test::More tests => 5;
use_ok 'FSA::Rules';
ok my $r = FSA::Rules->new,
   'Create FSA::Rules object';
ok $r->start, 'Start the machine';
ok $r->curr_state, 'Should have a state';
ok $r->reset, 'Should be able to reset';
```

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

```
use Test::More tests => 5;
```

```
use_ok 'FSA::Rules';
ok my $r = FSA::Rules->new,
   'Create FSA::Rules object';
ok $r->start, 'Start the machine';
ok $r->curr_state, 'Should have a state';
ok $r->reset, 'Should be able to reset';
```

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

```
use Test::More tests => 5;
use_ok 'FSA::Rules';
ok my $r = FSA::Rules->new,
    'Create FSA::Rules object';
ok $r->start, 'Start the machine';
ok $r->curr_state, 'Should have a state';
ok $r->reset, 'Should be able to reset';
```

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

```
use Test::More tests => 5;
use_ok 'FSA::Rules';
ok my $r = FSA::Rules->new,
    'Create FSA::Rules object';
ok $r->start, 'Start the machine';
ok $r->curr_state, 'Should have a state';
ok $r->reset, 'Should be able to reset';
```

Writing Tests

- And how are those tests written?
- Simple declarative syntax:

```
use Test::More tests => 5;
use_ok 'FSA::Rules';
ok my $r = FSA::Rules->new,
    'Create FSA::Rules object';
ok $r->start, 'Start the machine';
ok $r->curr_state, 'Should have a state';
ok $r->reset, 'Should be able to reset';
```

A Quick SQL ok()

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
            ELSE 'not ' END)
        || 'ok '
        || NEXTVAL('__tc__')
        || CASE $2 WHEN '' THEN ''
           ELSE COALESCE(
                ' - ' || $2, ''
           ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
              ELSE COALESCE(
                   ' - ' || $2, ''
              ) END;
$$ LANGUAGE SQL;
```

A Quick SQL ok()

```
\echo 1..3
CREATE TEMP SEQUENCE __tc__;
CREATE FUNCTION ok ( boolean, text )
RETURNS TEXT AS $$
    SELECT (CASE $1 WHEN TRUE THEN ''
                ELSE 'not ' END)
           || 'ok '
           || NEXTVAL('__tc__')
           || CASE $2 WHEN '' THEN ''
           ELSE COALESCE(
                ' - ' || $2, ''
           ) END;
$$ LANGUAGE SQL;
```

Ugly Function, But Look!

Ugly Function, But Look!

```
\set QUIET 1
\set ON_ERROR_STOP 1
\pset format unaligned
\pset tuples_only
\pset pager

SELECT ok( 'a' = 'a', '"a" should eq "a"' );
SELECT ok( 'a' = 'a', '"a" should eq "A"' );
SELECT ok( 'à' = 'À', '"à" should eq "À"' );
```

Drum Roll, Please

Drum Roll, Please

```
% psql try -f ~/Desktop/try.sql
1..3
ok 1 - "a" should eq "a"
ok 2 - "a" should eq "A"
ok 3 - "à" should eq "À"
```

Drum Roll, Please

```
% psql try -f ~/Desktop/try.sql
1..3
ok 1 - "a" should eq "a"
ok 2 - "a" should eq "A"
ok 3 - "à" should eq "À"
```

- There it is, pure TAP output

Drum Roll, Please

```
% psql try -f ~/Desktop/try.sql
1..3
ok 1 - "a" should eq "a"
ok 2 - "a" should eq "A"
ok 3 - "à" should eq "À"
```

- There it is, pure TAP output
- But if it was that easy, couldn't I port Test::More?

Drum Roll, Please

```
% psql try -f ~/Desktop/try.sql
1..3
ok 1 - "a" should eq "a"
ok 2 - "a" should eq "A"
ok 3 - "à" should eq "À"
```

- There it is, pure TAP output
- But if it was that easy, couldn’t I port Test::More?
- Yes

Introducing pgTAP

Introducing pgTAP

- Includes most Test::More functions

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean
  - is() — Value comparison

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean
  - is() — Value comparison
  - isnt() — NOT is()

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean
  - is() — Value comparison
  - isnt() — NOT is()
  - cmp_ok() — Compare values with specific operator

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean
  - is() — Value comparison
  - isnt() — NOT is()
  - cmp_ok() — Compare values with specific operator
  - matches() — Regex comparison

Introducing pgTAP

- Includes most Test::More functions
  - ok() — Boolean
  - is() — Value comparison
  - isnt() — NOT is()
  - cmp_ok() — Compare values with specific operator
  - matches() — Regex comparison
  - imatches() — Case-insensitive regex comparison

Introducing pgTAP

Introducing pgTAP

- Includes schema testing functions:

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist
  - has_pk() — Named table has a primary key

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist
  - has_pk() — Named table has a primary key
  - has_column() — Table has named column

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist
  - has_pk() — Named table has a primary key
  - has_column() — Table has named column
  - col_not_null() — Named column is NOT NULL

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist
  - has_pk() — Named table has a primary key
  - has_column() — Table has named column
  - col_not_null() — Named column is NOT NULL
  - col_type_is() — Named column is specified data type

Introducing pgTAP

- Includes schema testing functions:
  - has_table() — Named table exist
  - has_pk() — Named table has a primary key
  - has_column() — Table has named column
  - col_not_null() — Named column is NOT NULL
  - col_type_is() — Named column is specified data type
  - can_ok() — Has a function with signature

Introducing pgTAP

Introducing pgTAP

- Includes Test controls

Introducing pgTAP

- Includes Test controls
  - plan() — How many tests?

Introducing pgTAP

- Includes Test controls
  - plan() — How many tests?
  - no_plan() — Unknown number of tests

Introducing pgTAP

- Includes Test controls
  - plan() — How many tests?
  - no_plan() — Unknown number of tests
  - diag() — Diagnostic output

Introducing pgTAP

- Includes Test controls
  - plan() — How many tests?
  - no_plan() — Unknown number of tests
  - diag() — Diagnostic output
  - finish() — Test finished, report!

Introducing pgTAP

Introducing pgTAP

- Plus some extras

Introducing pgTAP

- Plus some extras
  - alike() — LIKE comparison

Introducing pgTAP

- Plus some extras
  - alike() — LIKE comparison
  - ialike() — ILIKE comparison

Introducing pgTAP

- Plus some extras
  - alike() — LIKE comparison
  - ialike() — ILIKE comparison
  - throws_ok() — Test error-throwing code

Introducing pgTAP

- Plus some extras
  - alike() — LIKE comparison
  - ialike() — ILIKE comparison
  - throws_ok() — Test error-throwing code
  - lives_ok() — Test for no errors

pgTAP Basics

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Basics

```
\set QUIET on
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
\pset tuples_only
\pset format unaligned
BEGIN;

-- Plan the tests.
SELECT plan(1);
--SELECT * FROM no_plan();

SELECT ok( true, 'Truth should be true' );

-- Clean up.
SELECT * FROM finish();
ROLLBACK;
```

pgTAP Practices

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );

SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

pgTAP Practices

```
SELECT has_table( 'users' );
SELECT has_pk( 'users' );
SELECT has_column( 'users', 'surname' );
SELECT col_not_null( 'users', 'name' );
SELECT col_type_is( 'users', 'name', 'text' );
```

```
SELECT fk_ok(
    'contacts', 'user_id',
    'users',    'id'
);
```

A pgTAP Loop

A pgTAP Loop

```
SELECT plan(5);

CREATE TEMPORARY TABLE try (
    a CITEXT,
    b TEXT
);

INSERT INTO try
VALUES ('a', 'a'), ('b', 'b'),
       ('a', 'a'), ('à', 'à'),
       ('Bjørn', 'Bjørn');

SELECT is(
    UPPER( a ),
    UPPER( b ),
    'UPPER(''' || a || ''')'
) FROM try;
```

A pgTAP Loop

```
SELECT plan(5);

CREATE TEMPORARY TABLE try (
    a CITEXT,
    b TEXT
);

INSERT INTO try
VALUES ('a', 'a'), ('b', 'b'),
       ('a', 'a'), ('à', 'à'),
       ('Bjørn', 'Bjørn');

SELECT is(
    UPPER( a ),
    UPPER( b ),
    'UPPER(''' || a || ''')'
) FROM try;
```

A pgTAP Loop

```
SELECT plan(5);

CREATE TEMPORARY TABLE try (
    a CITEXT,
    b TEXT
);

INSERT INTO try
VALUES ('a', 'a'), ('b', 'b'),
       ('a', 'a'), ('à', 'à'),
       ('Bjørn', 'Bjørn');

SELECT is(
    UPPER( a ),
    UPPER( b ),
    'UPPER(''' || a || ''')'
) FROM try;
```

A pgTAP Loop

```
SELECT plan(5);

CREATE TEMPORARY TABLE try (
    a CITEXT,
    b TEXT
);
```

```
INSERT INTO try
VALUES ('a', 'a'), ('b', 'b'),
       ('a', 'a'), ('à', 'à'),
       ('Bjørn', 'Bjørn');
```

```
SELECT is(
    UPPER( a ),
    UPPER( b ),
    'UPPER(''' || a || ''')'
) FROM try;
```

A pgTAP Loop

```
SELECT plan(5);

CREATE TEMPORARY TABLE try (
    a CITEXT,
    b TEXT
);

INSERT INTO try
VALUES ('a', 'a'), ('b', 'b'),
       ('a', 'a'), ('à', 'à'),
       ('Bjørn', 'Bjørn');

SELECT is(
    UPPER( a ),
    UPPER( b ),
    'UPPER(''' || a || ''')'
) FROM try;
```

pgTAP Loop Output

pgTAP Loop Output

```
1..5
ok 1 - UPPER("a")
ok 2 - UPPER("b")
ok 3 - UPPER("a")
ok 4 - UPPER("à")
ok 5 - UPPER("Bjørn")
```

Validate your Schema

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

```
SELECT throws_ok(
    'SELECT 1 / 0',
    NULL,
    'divide by 0'
);
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

```
SELECT throws_ok(
    'SELECT 1 / 0',
    NULL,
    'divide by 0'
);
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');

SELECT throws_ok(
    'SELECT 1 / 0',
    NULL,
    'divide by 0'
);

SELECT lives_ok( 'SELECT 1/1', 'it’s alive' );
```

Validate your Schema

```
SELECT is( 'a', 'a', '"a" should eq "a"' );
SELECT isnt( 'a', 'b', '"a" should ne "b"' );
SELECT matches( 'foo', 'o$', '"foo" ~ "o$"');
SELECT like( 'foo', 'f%', '"foo" ~~ "f%"');
```

```
SELECT throws_ok(
    'SELECT 1 / 0',
    NULL,
    'divide by 0'
);
```

```
SELECT lives_ok( 'SELECT 1/1', 'it's alive' );
```

Test Your Schema

Test Your Schema

- Say you use triggers in your database

Test Your Schema

- Say you use triggers in your database

```
CREATE FUNCTION hash_pass() RETURNS TRIGGER AS $$
BEGIN
    NEW.pass := MD5( NEW.pass );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_users_pass
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW EXECUTE PROCEDURE hash_pass();
```

Test Your Schema

- Say you use triggers in your database

```
CREATE FUNCTION hash_pass() RETURNS TRIGGER AS $$
BEGIN
    NEW.pass := MD5( NEW.pass );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_users_pass
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW EXECUTE PROCEDURE hash_pass();
```

Validate Your Schema

Validate Your Schema

- Now test it!

Validate Your Schema

- Now test it!

```
INSERT INTO users ( nick, pass )
VALUES ('theory', 's0up3rs3c!7' );
```

Validate Your Schema

- Now test it!

```
INSERT INTO users ( nick, pass )
VALUES ('theory', 's0up3rs3c!7' );
```

```
SELECT is(
    pass,
    MD5('s0up3rs3c!7'),
    'Password should be hashed'
) FROM users WHERE nick = 'theory';
```

Validate Your Schema

- Now test it!

```
INSERT INTO users ( nick, pass )
VALUES ('theory', 's0up3rs3c!7' );
```

```
SELECT is(
    pass,
    MD5('s0up3rs3c!7'),
    'Password should be hashed'
) FROM users WHERE nick = 'theory';
```

Validate Your Schema

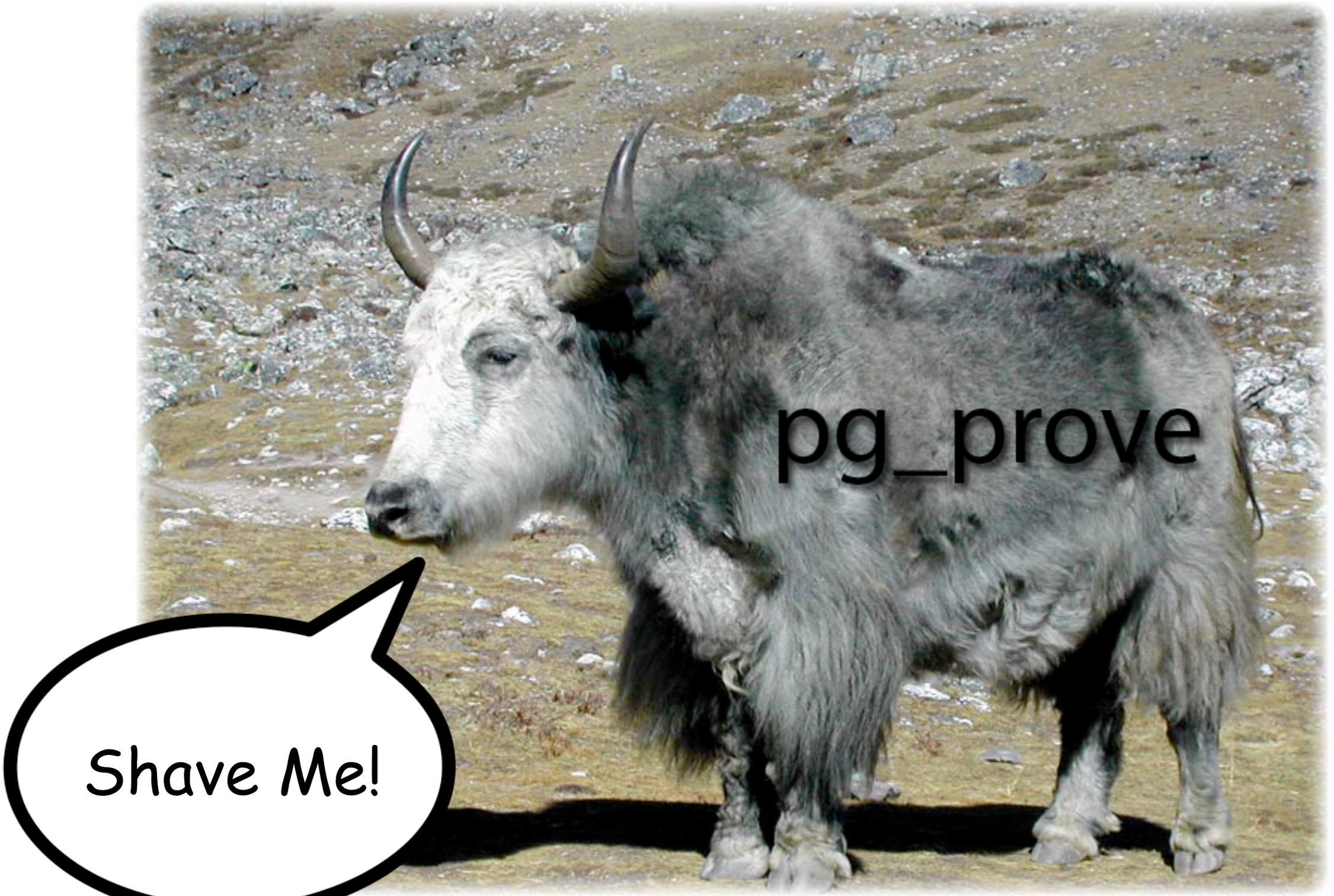
- Now test it!

```
INSERT INTO users ( nick, pass )
VALUES ('theory', 's0up3rs3c!7' );
```

```
SELECT is(
    pass,
    MD5('s0up3rs3c!7'),
    'Password should be hashed'
) FROM users WHERE nick = 'theory';
```

- This got me to thinking...


pg_prove


pg_prove
Shave Me!

TAP::Harness

TAP::Harness

- TAP designed to be harnessed

TAP::Harness

- TAP designed to be harnessed
- Harness can collect TAP

TAP::Harness

- TAP designed to be harnessed
- Harness can collect TAP
- Analyze output

TAP::Harness

- TAP designed to be harnessed
- Harness can collect TAP
- Analyze output
- Output report

TAP::Harness

- TAP designed to be harnessed
- Harness can collect TAP
- Analyze output
- Output report
- Could Perl's TAP::Harness work with pgTAP output

TAP::Harness

- TAP designed to be harnessed
- Harness can collect TAP
- Analyze output
- Output report
- Could Perl's TAP::Harness work with pgTAP output
- YES!

Using pg_prove

Using pg_prove

- pg_prove distributed with pgTAP

Using pg_prove

- pg_prove distributed with pgTAP
- Supports standard prove and pg_* options

Using pg_prove

- pg_prove distributed with pgTAP
- Supports standard prove and pg_* options

```
% pg_prove -d try -U postgres test_*.sql --verbose
test_foo.....
1..5
ok 1 - UPPER("a")
ok 2 - UPPER("b")
ok 3 - UPPER("a")
ok 4 - UPPER("à")
ok 5 - UPPER("Bjørn")
ok
All tests successful.
Files=1, Tests=5,  1 wallclock secs
( 0.01 usr  0.00 sys +  0.01 cusr  0.00 csys =  0.02 CPU)
Result: PASS
```

Using pg_prove

- pg_prove distributed with pgTAP
- Supports standard prove and pg_* options

```
% pg_prove -d try -U postgres test_*.sql --verbose
test_foo........
1..5
ok 1 - UPPER("a")
ok 2 - UPPER("b")
ok 3 - UPPER("a")
ok 4 - UPPER("à")
ok 5 - UPPER("Bjørn")
ok
All tests successful.
Files=1, Tests=5,  1 wallclock secs
( 0.01 usr  0.00 sys +  0.01 cusr  0.00 csys =  0.02 CPU)
Result: PASS
```

Hrm...

Hrm...

- If TAP::Harness/prove works...

Hrm...

- If TAP::Harness/prove works…
- Couldn’t we mix

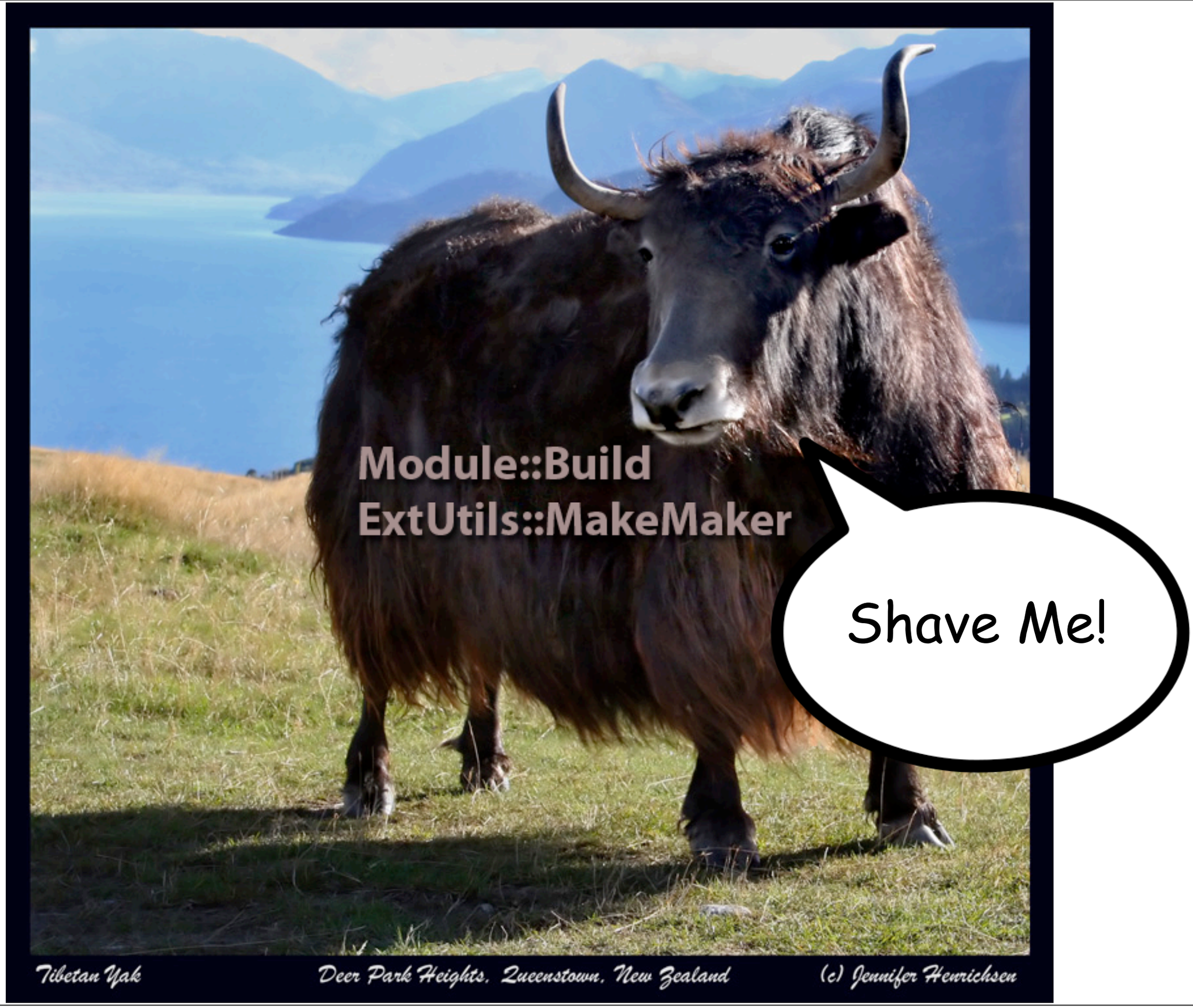
Hrm...

- If TAP::Harness/prove works…
- Couldn't we mix
  - Perl tests

Hrm…

- If TAP::Harness/prove works…
- Couldn't we mix
  - Perl tests
  - pgTAP tests


Module::Build
ExtUtils::MakeMaker
Tibetan Yak
Deer Park Heights, Queenstown, New Zealand
(c) Jennifer Henrichsen


Module::Build
ExtUtils::MakeMaker
Shave Me!
Tibetan Yak
Deer Park Heights, Queenstown, New Zealand
(c) Jennifer Henrichsen

Build Integration

Build Integration

- Perl tests output TAP

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP
- Why not mix 'em up?

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP
- Why not mix 'em up?
- Submitted patches:

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP
- Why not mix 'em up?
- Submitted patches:
  - TAP::Harness (accepted, released)

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP
- Why not mix ’em up?
- Submitted patches:
  - TAP::Harness (accepted, released)
  - Module::Build (accepted, released)

Build Integration

- Perl tests output TAP
- pgTAP tests output TAP
- Why not mix ’em up?
- Submitted patches:
  - TAP::Harness (accepted, released)
  - Module::Build (accepted, released)
  - ExtUtils::MakeMaker (C’mon, Schwern!)

Build Integration

Build Integration

- Mix and match tests!

Build Integration

- Mix and match tests!
- Let’s give it a try…

Build Integration

- Mix and match tests!
- Let’s give it a try…

```
try%
```

PostgreSQL Module Testing

PostgreSQL Module Testing

- PostgreSQL supports regression tests

PostgreSQL Module Testing

- PostgreSQL supports regression tests
- Output is fully verbose

PostgreSQL Module Testing

- PostgreSQL supports regression tests
- Output is fully verbose
- Tests pass when out `diff`'ed against a file

PostgreSQL Module Testing

- PostgreSQL supports regression tests
- Output is fully verbose
- Tests pass when out `diff`'ed against a file
- No conditional expressions

PostgreSQL Module Testing

- PostgreSQL supports regression tests
- Output is fully verbose
- Tests pass when out `diff`'ed against a file
- No conditional expressions
- No TODO tests

PostgreSQL Module Testing

- PostgreSQL supports regression tests
- Output is fully verbose
- Tests pass when out `diff`'ed against a file
- No conditional expressions
- No TODO tests
- Noisy Ouput

Would you rather read this...

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)
```

```
SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)
```

```
SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

- Which pass?

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)
```

```
SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

- Which pass?
- Which fail?

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)
```

```
SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

- Which pass?
- Which fail?
- I can't tell either

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

- Which pass?
- Which fail?
- I can't tell either
- Have to trust expected output file

Would you rather read this...

```
% psql -Xd try -f pg.sql
CREATE TEMP TABLE srt (
   name CITEXT
);
CREATE TABLE
INSERT INTO srt (name)
VALUES ('aardvark'),
       ('AAA'),
       ('aba'),
       ('ABC'),
       ('abd');
INSERT 0 5
SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA'::text;
 aaa
-----
 aaa
(1 row)
```

```
SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::varchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name =
'AAA'::bpchar;
 aaa
-----
 aaa
(1 row)

SELECT LOWER(name) as aaa FROM
srt WHERE name = 'AAA';
 aaa
-----
 aaa
(1 row)
```

- Which pass?
- Which fail?
- I can't tell either
- Have to trust expected output file
- Separate files to maintain!

Or this?

Or this?

```
% ~/dev/Kineticode/pgtap/trunk/pg_prove -d try pgtap.sql --verbose
pgtap....
1..4
ok 1 - Should find "AAA"::text
ok 2 - Should find "AAA"::varchar
ok 3 - Should find "AAA"::bpchar
ok 4 - Should find "AAA"
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.00 usr  0.00 sys +  0.01
cusr  0.00 csys =  0.01 CPU)
Result: PASS
```

Or this?

```
% ~/dev/Kineticode/pgtap/trunk/pg_prove -d try pgtap.sql --verbose
pgtap....
1..4
ok 1 - Should find "AAA"::text
ok 2 - Should find "AAA"::varchar
ok 3 - Should find "AAA"::bpchar
ok 4 - Should find "AAA"
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.00 usr  0.00 sys +  0.01
cusr  0.00 csys =  0.01 CPU)
Result: PASS
```

- Which pass?

Or this?

```
% ~/dev/Kineticode/pgtap/trunk/pg_prove -d try pgtap.sql --verbose
pgtap....
1..4
ok 1 - Should find "AAA"::text
ok 2 - Should find "AAA"::varchar
ok 3 - Should find "AAA"::bpchar
ok 4 - Should find "AAA"
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.00 usr  0.00 sys +  0.01
cusr  0.00 csys =  0.01 CPU)
Result: PASS
```

- Which pass?
- Which fail?

Or this?

```
% ~/dev/Kineticode/pgtap/trunk/pg_prove -d try pgtap.sql --verbose
pgtap....
1..4
ok 1 - Should find "AAA"::text
ok 2 - Should find "AAA"::varchar
ok 3 - Should find "AAA"::bpchar
ok 4 - Should find "AAA"
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.00 usr  0.00 sys +  0.01
cusr  0.00 csys =  0.01 CPU)
Result: PASS
```

- Which pass?
- Which fail?
- What was tested?

Or this?

```
% ~/dev/Kineticode/pgtap/trunk/pg_prove -d try pgtap.sql --verbose
pgtap....
1..4
ok 1 - Should find "AAA"::text
ok 2 - Should find "AAA"::varchar
ok 3 - Should find "AAA"::bpchar
ok 4 - Should find "AAA"
ok
All tests successful.
Files=1, Tests=4,  0 wallclock secs ( 0.00 usr  0.00 sys +  0.01
cusr  0.00 csys =  0.01 CPU)
Result: PASS
```

- Which pass?
- Which fail?
- What was tested?
- I can tell, too

Module Integration

Module Integration

- Running under `installcheck` is Tricky

Module Integration

- Running under `installcheck` is Tricky
- Must have pg_regress load PL/pgSQL

Module Integration

- Running under `installcheck` is Tricky
- Must have pg_regress load PL/pgSQL
- Makefile:

Module Integration

- Running under `installcheck` is Tricky
- Must have pg_regress load PL/pgSQL
- Makefile:

```
MODULES = citext
DATA_built = citext.sql
DATA = uninstall_citext.sql
REGRESS = citext
REGRESS_OPTS = --load-language plpgsql

ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/citext
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

Module Integration

- Running under `installcheck` is Tricky
- Must have pg_regress load PL/pgSQL
- Makefile:

```
MODULES = citext
DATA_built = citext.sql
DATA = uninstall_citext.sql
REGRESS = citext
REGRESS_OPTS = --load-language plpgsql

ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/citext
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

Module Integration

Module Integration

- Must set all variables in the test files

Module Integration

- Must set all variables in the test files

```
\unset ECHO
\pset format unaligned
\pset tuples_only true
\pset pager

\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true

BEGIN;
\i pgtap.sql

SELECT plan(1);
SELECT pass('W00t!');
SELECT * FROM finish();
```

Module Integration

- Must set all variables in the test files

```
\unset ECHO
\pset format unaligned
\pset tuples_only true
\pset pager

\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true

BEGIN;
\i pgtap.sql

SELECT plan(1);
SELECT pass('W00t!');
SELECT * FROM finish();
```

Module Integration

- Must set all variables in the test files

```
\unset ECHO
\pset format unaligned
\pset tuples_only true
\pset pager

\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true

BEGIN;
\i pgtap.sql

SELECT plan(1);
SELECT pass('W00t!');
SELECT * FROM finish();
```

Module Integration

- Must set all variables in the test files

```
\unset ECHO
\pset format unaligned
\pset tuples_only true
\pset pager

\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true

BEGIN;
\i pgtap.sql

SELECT plan(1);
SELECT pass('W00t!');
SELECT * FROM finish();
```

Module Integration

- Must set all variables in the test files

```
\unset ECHO
\pset format unaligned
\pset tuples_only true
\pset pager

\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true

BEGIN;
\i pgtap.sql

SELECT plan(1);
SELECT pass('W00t!');
SELECT * FROM finish();
```

Module Integration

Module Integration

- Include unset in expected output file

Module Integration

- Include unset in expected output file

```
\unset ECHO
1..1
ok 1 - W00t!
```

pgTAP Downsides

pgTAP Downsides

- Slower than pure SQL regression tests

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though
- Table results must be contorted into scalars

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though
- Table results must be contorted into scalars

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though
- Table results must be contorted into scalars

```
SELECT is(
    ARRAY( SELECT name FROM srt ORDER BY name ),
    ARRAY['AAA', 'aardvark', 'aba', 'ABC', 'abc'],
    'The words should be case-insensitively sorted'
);
```

pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though
- Table results must be contorted into scalars

```
SELECT is(
    ARRAY( SELECT name FROM srt ORDER BY name ),
    ARRAY['AAA', 'aardvark', 'aba', 'ABC', 'abc'],
    'The words should be case-insensitively sorted'
);
```

- Tom hates it

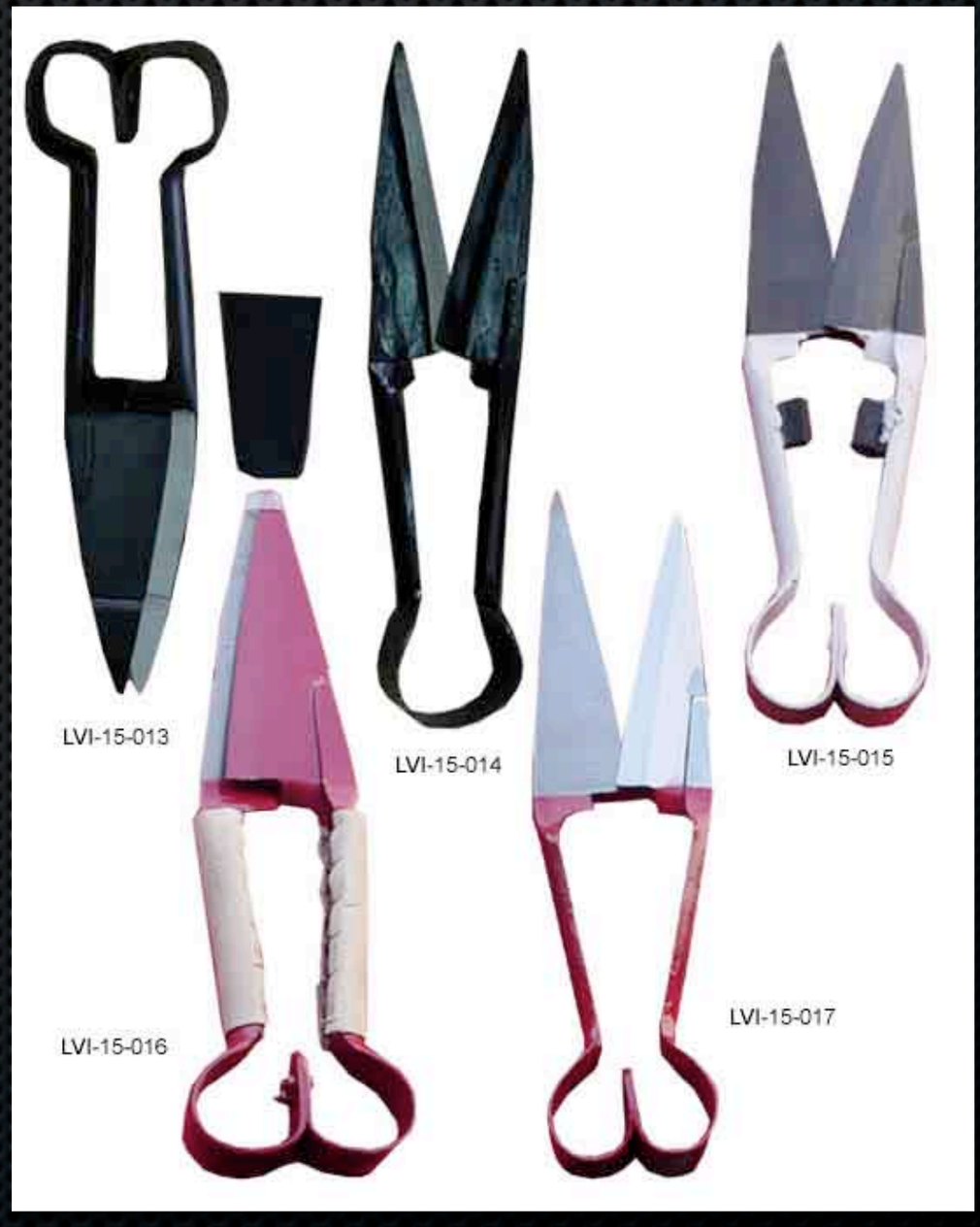
pgTAP Downsides

- Slower than pure SQL regression tests
  - 4-5 times slower
  - Faster than Perl tests, though
- Table results must be contorted into scalars

```
SELECT is(
    ARRAY( SELECT name FROM srt ORDER BY name ),
    ARRAY['AAA', 'aardvark', 'aba', 'ABC', 'abc'],
    'The words should be case-insensitively sorted'
);
```

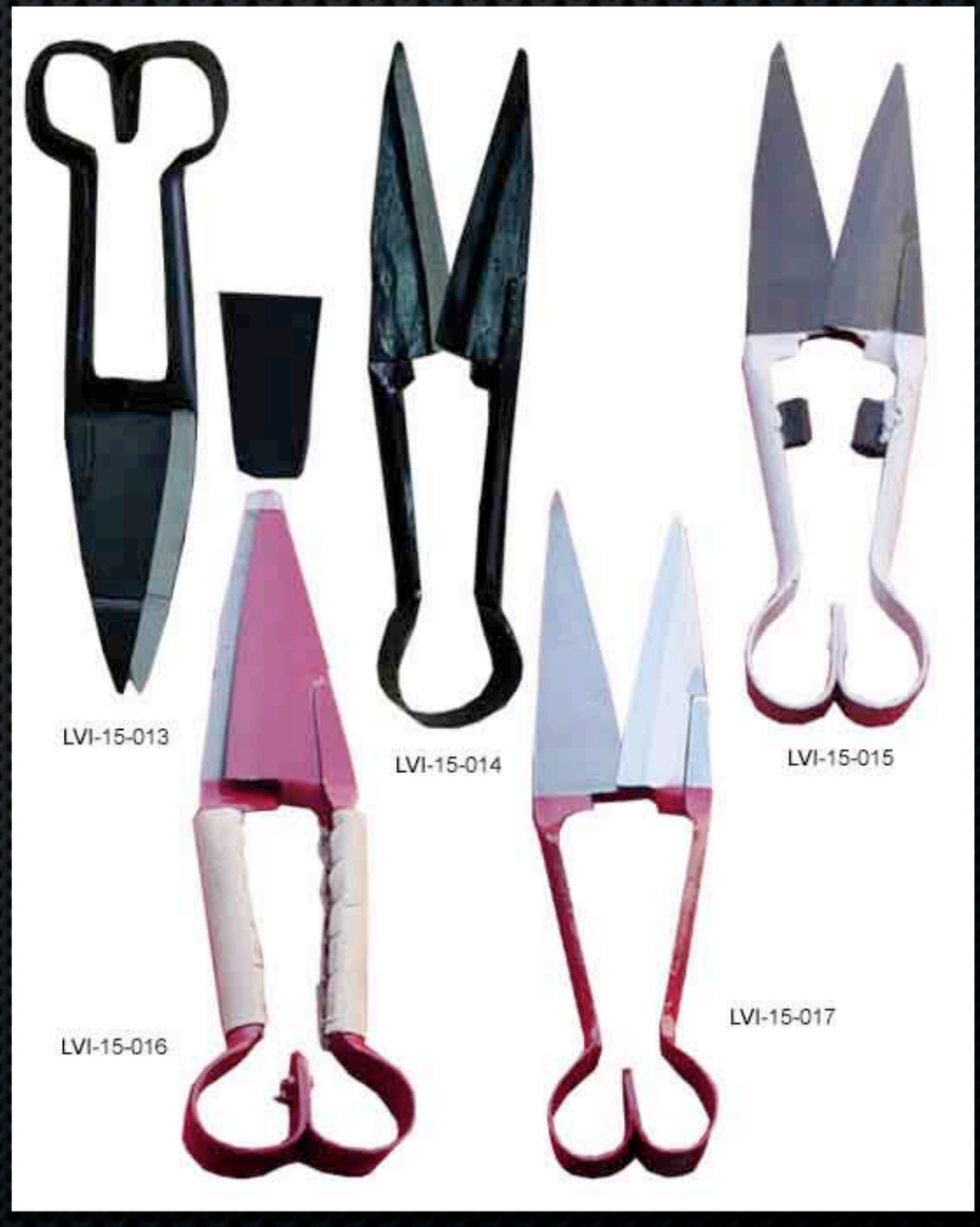
- Tom hates it
- (Doesn’t bother me)

The Future



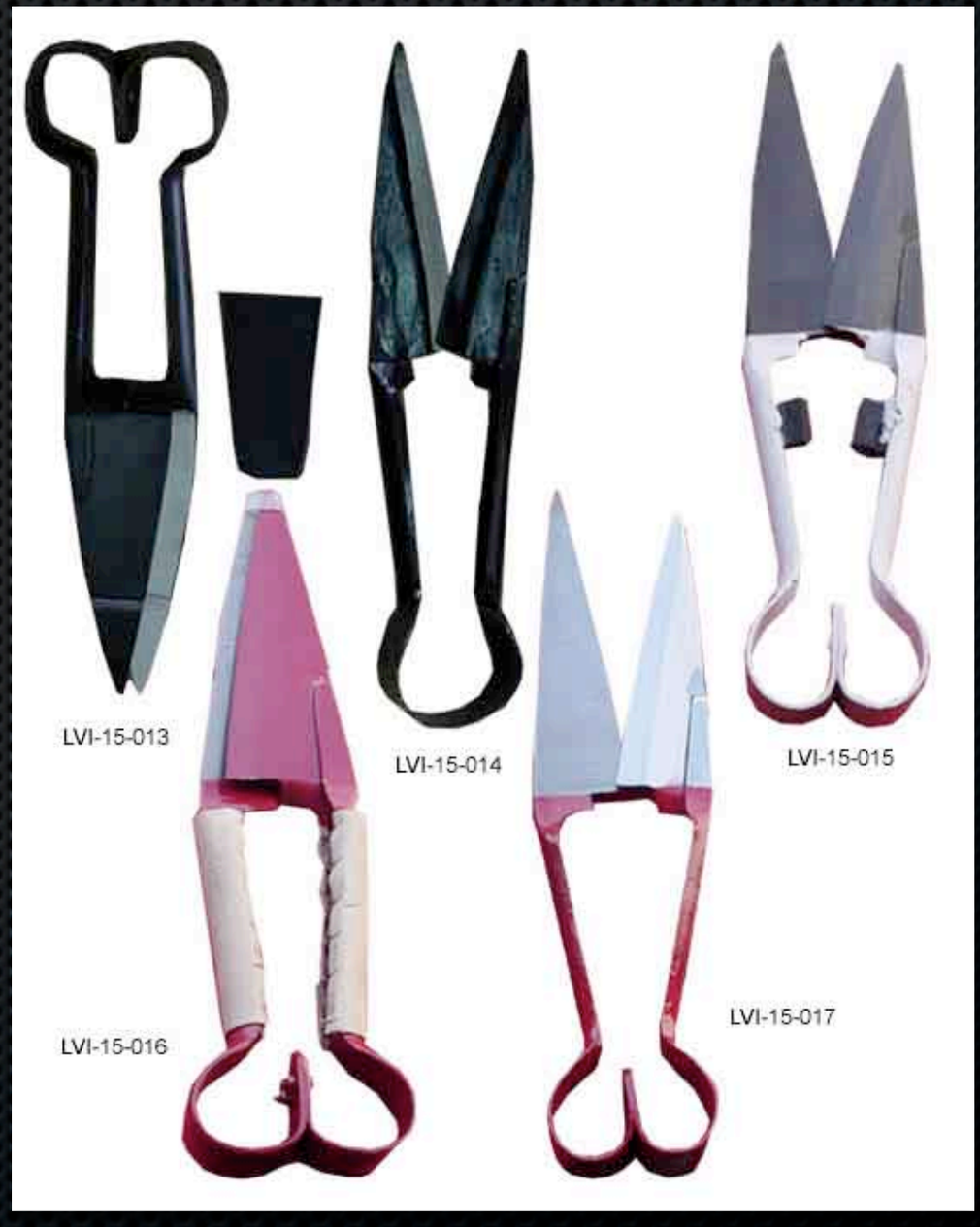
The Future

- Conditionals?



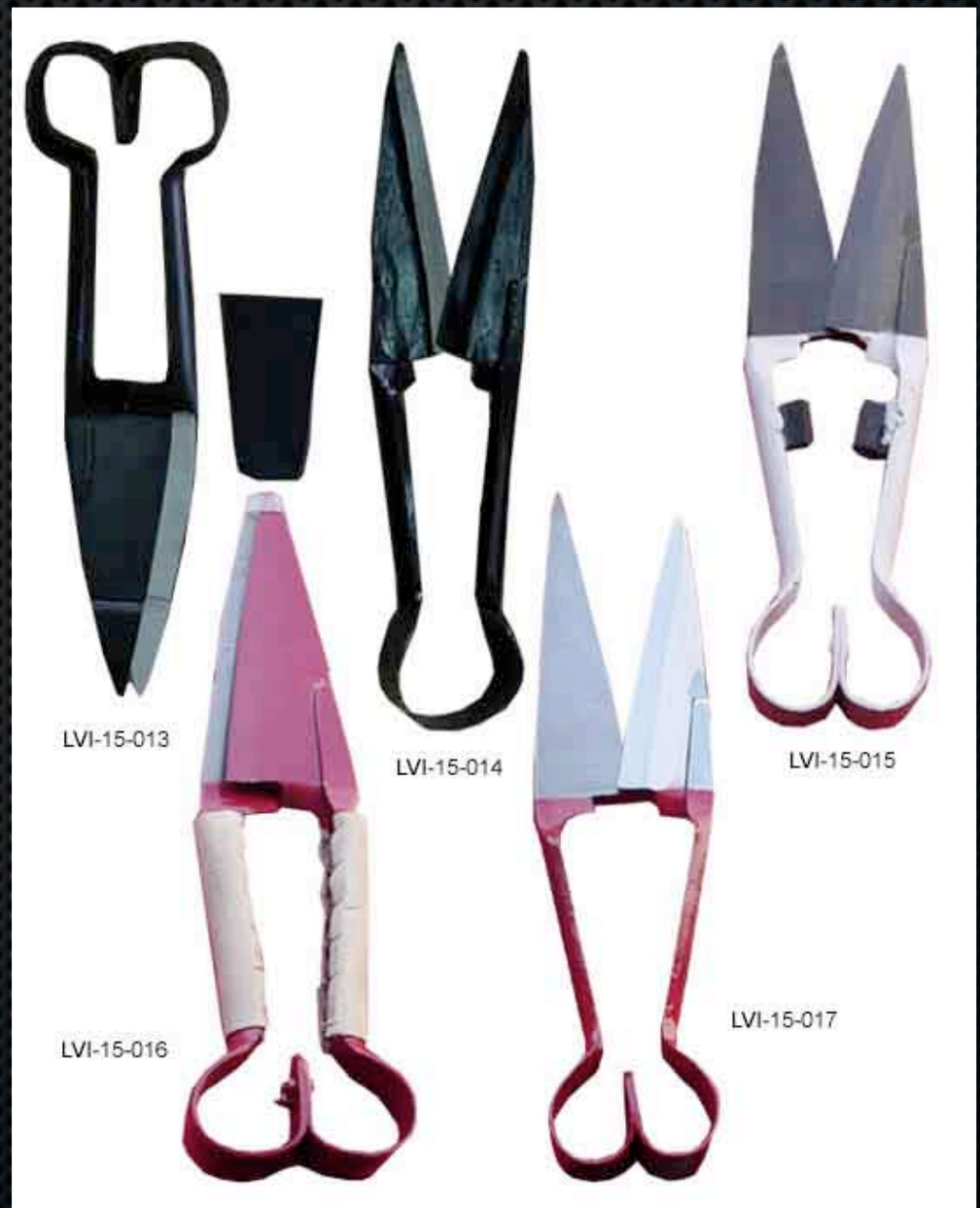
The Future

- Conditionals?
- Faster performance?



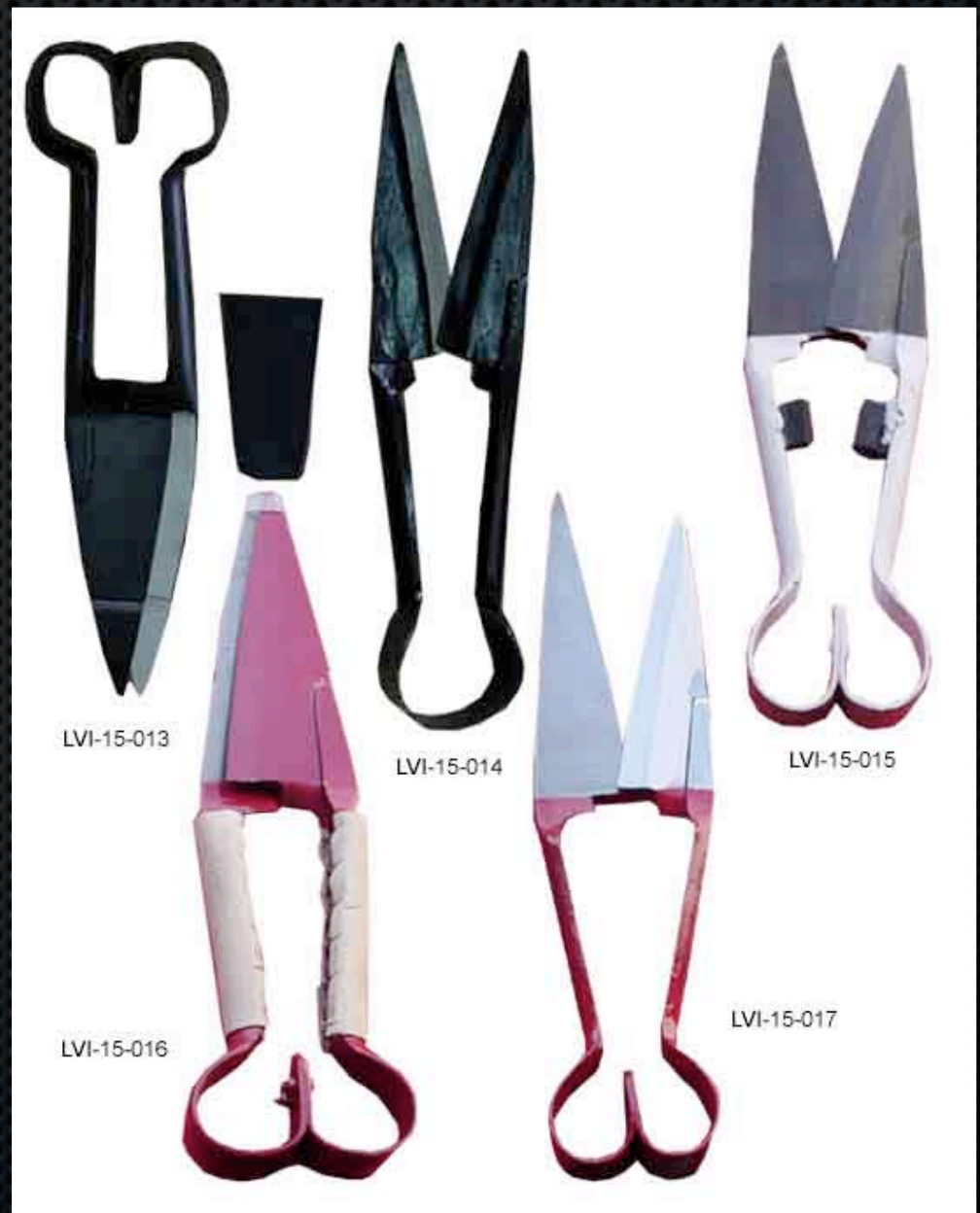
The Future

- Conditionals?
- Faster performance?
- Port slow bits to C?



The Future

- Conditionals?
- Faster performance?
  - Port slow bits to C?
- Wanna help?



Help Out!

http://pgtap.projects.postgresql.org/

Help Out!

http://pgtap.projects.postgresql.org/

http://pgtap.projects.postgresql.org/

Give it a Try!

http://pgtap.projects.postgresql.org/

Give it a Try!

Thank You!