Test your Database with pgTAP

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

YAPC::NA 2008 Lightning Talk

Working on a New App

Working on a New App

```
CREATE TABLE users (
   id       SERIAL PRIMARY KEY,
   login    TEXT   NOT NULL,
   password TEXT   NOT NULL
);
```

Working on a New App

```
CREATE TABLE users (
  id       SERIAL PRIMARY KEY,
  login    TEXT   NOT NULL,
  password TEXT   NOT NULL
);

CREATE UNIQUE INDEX udx_users_login
ON users(LOWER(login));
```

Lookup a User

Lookup a User

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

Lookup a User

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

- I finally got sick of this

Lookup a User

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

- I finally got sick of this
- Decided to implement case-insensitive TEXT type

Lookup a User

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

- I finally got sick of this
- Decided to implement case-insensitive TEXT type
- Got it to compile, now what?

Lookup a User

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

- I finally got sick of this
- Decided to implement case-insensitive TEXT type
- Got it to compile, now what?
- Tests!

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 ~/Desktop/try.sql
t
t
t
```

First Pass at Output

```
% psql try -f ~/Desktop/try.sql
t
t
```

- Cool!

First Pass at Output

```
% psql try -f ~/Desktop/try.sql
t
t
t
```

- Cool!
- Inspired by TAP: easy to scan

First Pass at Output

```
% psql try -f ~/Desktop/try.sql
t
t
t
```

- Cool!
- Inspired by TAP: easy to scan
- But why not actually emit TAP?

A Quick ok()

A Quick 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()

Introducing pgTAP

- Includes most Test::More functions
  - ok()
  - is()

Introducing pgTAP

- Includes most Test::More functions
  - ok()
  - is()
  - isnt()

Introducing pgTAP

- Includes most Test::More functions
  - ok()
  - is()
  - isnt()
  - matches()

Introducing pgTAP

- Includes most Test::More functions
  - ok()
  - is()
  - isnt()
  - matches()
  - imatches()

Introducing pgTAP

Introducing pgTAP

- Includes Test controls

Introducing pgTAP

- Includes Test controls
  - plan()

Introducing pgTAP

- Includes Test controls
  - plan()
  - no_plan()

Introducing pgTAP

- Includes Test controls
  - plan()
  - no_plan()
  - diag()

Introducing pgTAP

- Includes Test controls
  - plan()
  - no_plan()
  - diag()
  - finish()

Introducing pgTAP

Introducing pgTAP

- Plus some extras

Introducing pgTAP

- Plus some extras
  - alike()

Introducing pgTAP

- Plus some extras
  - alike()
  - ialike()

Introducing pgTAP

- Plus some extras
  - alike()
  - ialike()
  - throws_ok()

Introducing pgTAP

- Plus some extras
  - alike()
  - ialike()
  - throws_ok()
  - lives_ok()

pgTAP Basics

pgTAP Basics

```
SET client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 client_min_messages = warning;
\set ON_ERROR_ROLLBACK true
\set ON_ERROR_STOP true
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 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%"');
```

pgTAP Practices

```
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'
);
```

pgTAP Practices

```
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' );
```

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

- Say you use triggers in your database

Validate Your Schema

- Say you use triggers in your database

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

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

Validate Your Schema

- Say you use triggers in your database

```
CREATE FUNCTION hash_pass() RETURNS TRIGGER AS $$
BEGIN
    NEW.password := MD5( NEW.password );
    RETURN NEW;
END;
$$ 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 ( login, password )
VALUES ('theory', 's0up3rs3c!7' );
```

Validate Your Schema

- Now test it!

```
INSERT INTO users ( login, password )
VALUES ('theory', 's0up3rs3c!7' );
```

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

Using pg_prove

Using pg_prove

- Harness your tests with pg_prove

Using pg_prove

- Harness your tests with pg_prove
- Supports Standard prove and pg_* options

Using pg_prove

- Harness your tests with pg_prove
- Supports Standard prove and pg_* options

```
% pg_prove -d try 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

- Harness your tests with pg_prove
- Supports Standard prove and pg_* options

```
% pg_prove -d try 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
```

Future

Future

- Improved build process

Future

- Improved build process
- Integration into Module::Build

Future

- Improved build process
- Integration into Module::Build
- Mixed SQL and Perl tests

Future

- Improved build process
- Integration into Module::Build
- Mixed SQL and Perl tests
- More test functions

Future

- Improved build process
- Integration into Module::Build
- Mixed SQL and Perl tests
- More test functions
- A function to test test functions

Give it a Try!

http://pgfoundry.org/projects/pgtap/

Give it a Try!

http://pgfoundry.org/projects/pgtap/

Thank You!