

pgTAP Best Practices

David E. Wheeler

Kineticcode, Inc.
PostgreSQL Experts, Inc.

PostgreSQL Conference West 2009

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

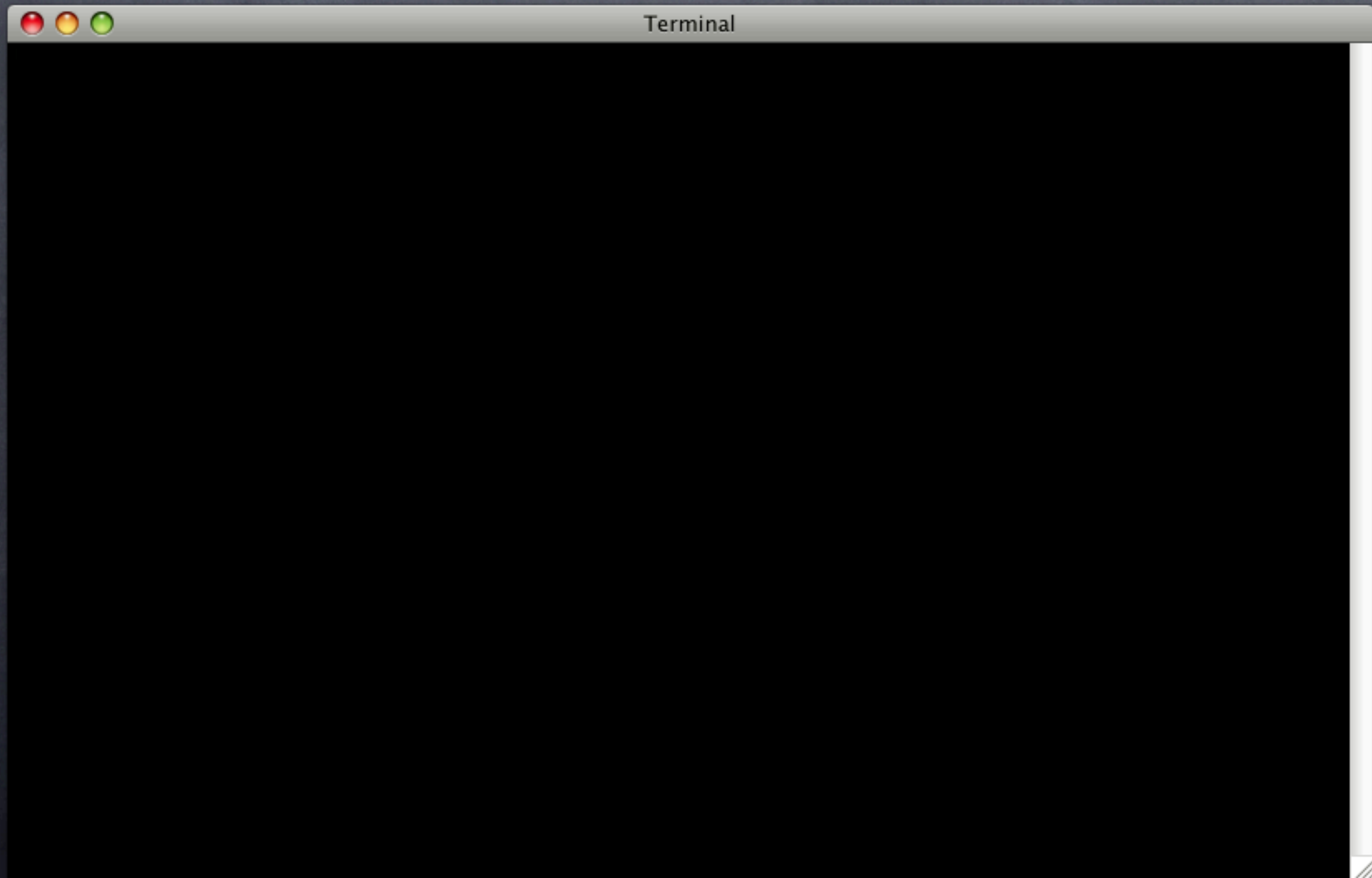
What is TAP?

- 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

```
Terminal
% 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

```
Terminal
% 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

```
Terminal
% 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

```
Terminal
% 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

```
Terminal
% 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.
```

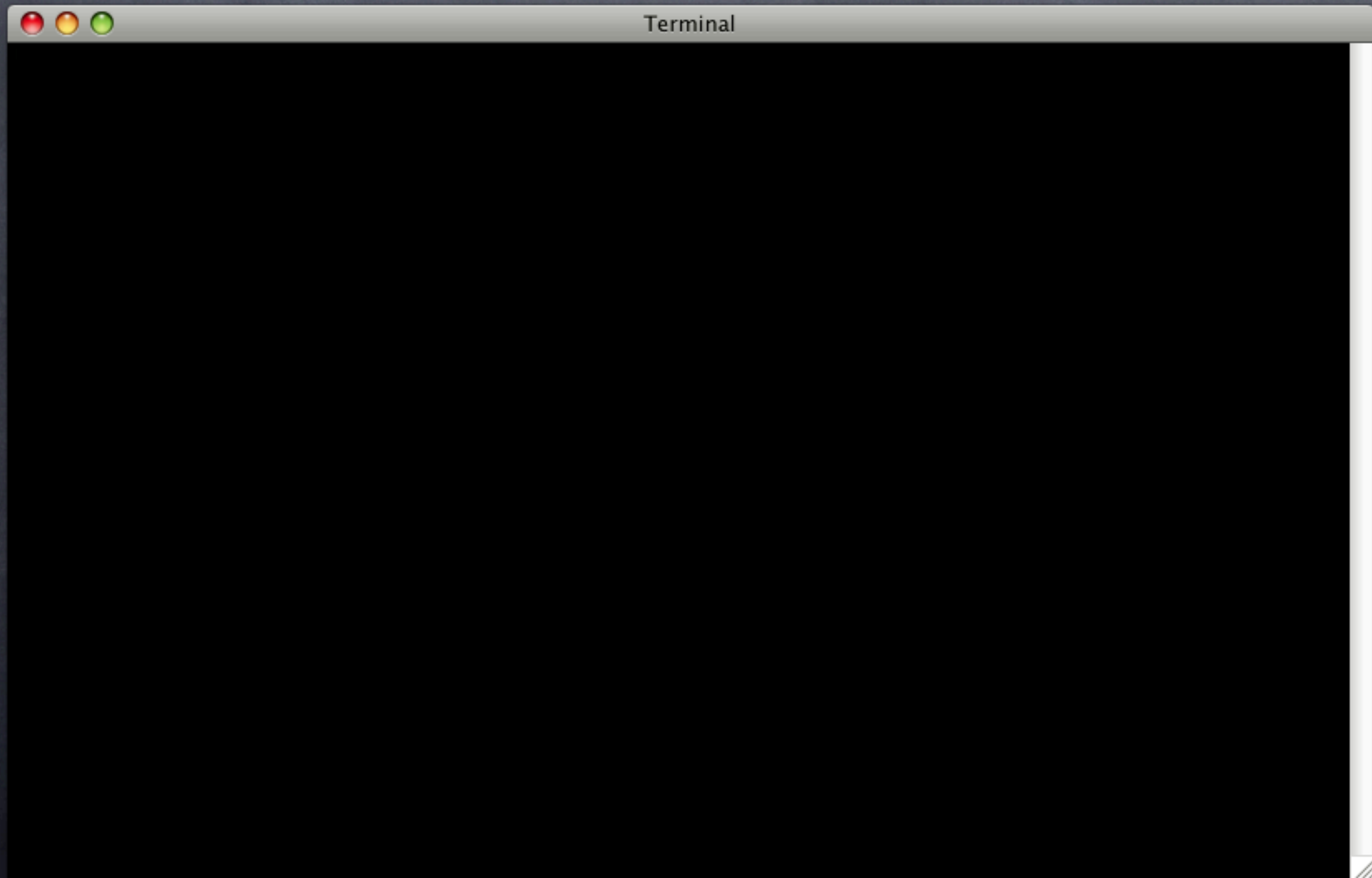

TAP Output

```
Terminal
% 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

```
Terminal
% 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.
```

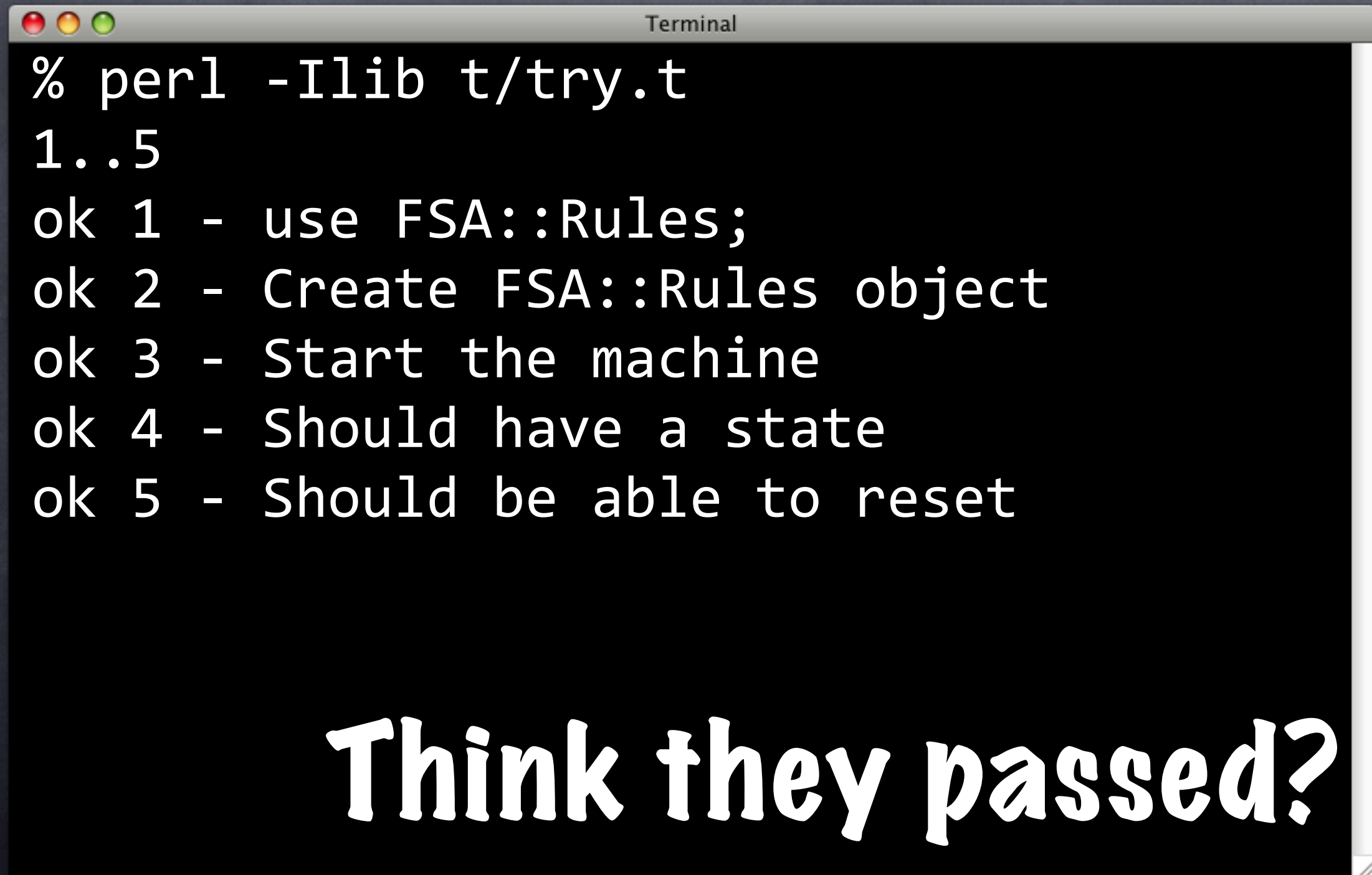

TAP Output



TAP Output

```
Terminal
% 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
```

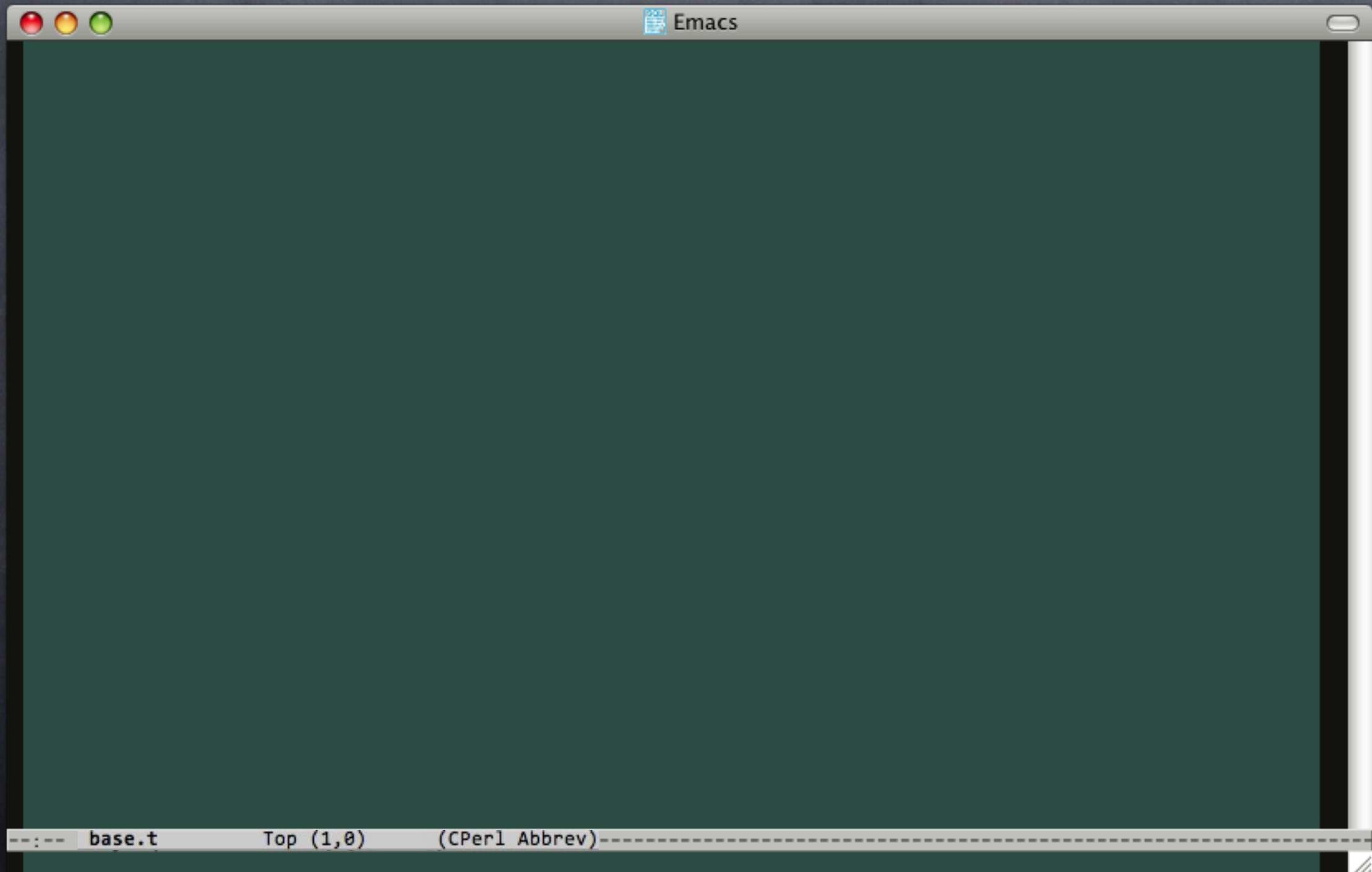

TAP Output

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal displays the output of a Perl script. The first line is the command "% perl -Ilib t/try.t". The second line is "1..5". The following five lines are test results, each starting with "ok" and a number, followed by a description of the test: "ok 1 - use FSA::Rules;", "ok 2 - Create FSA::Rules object", "ok 3 - Start the machine", "ok 4 - Should have a state", and "ok 5 - Should be able to reset".

```
Terminal
% 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

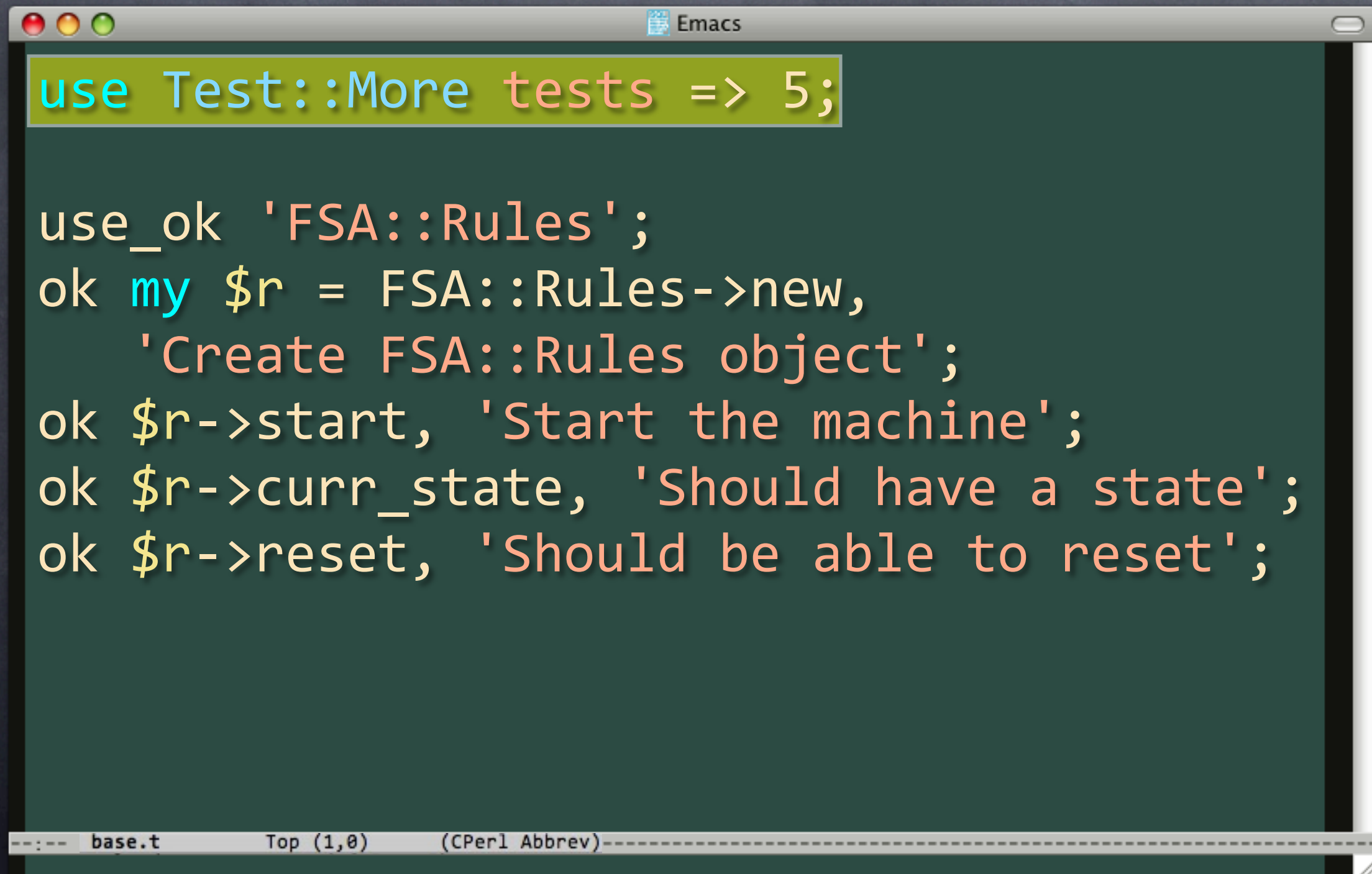


Writing Tests

```
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



```
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';
```

base.t Top (1,0) (CPerl Abbrev)

Writing Tests

```
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

```
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

```
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

```
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';
```

Simple declarative syntax

pgTAP

pgTAP

- Includes most Test::More functions

pgTAP

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

pgTAP

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

pgTAP

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

pgTAP

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

pgTAP

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

pgTAP

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

pgTAP

pgTAP

- **Includes Test controls**

pgTAP

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

pgTAP

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

pgTAP

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

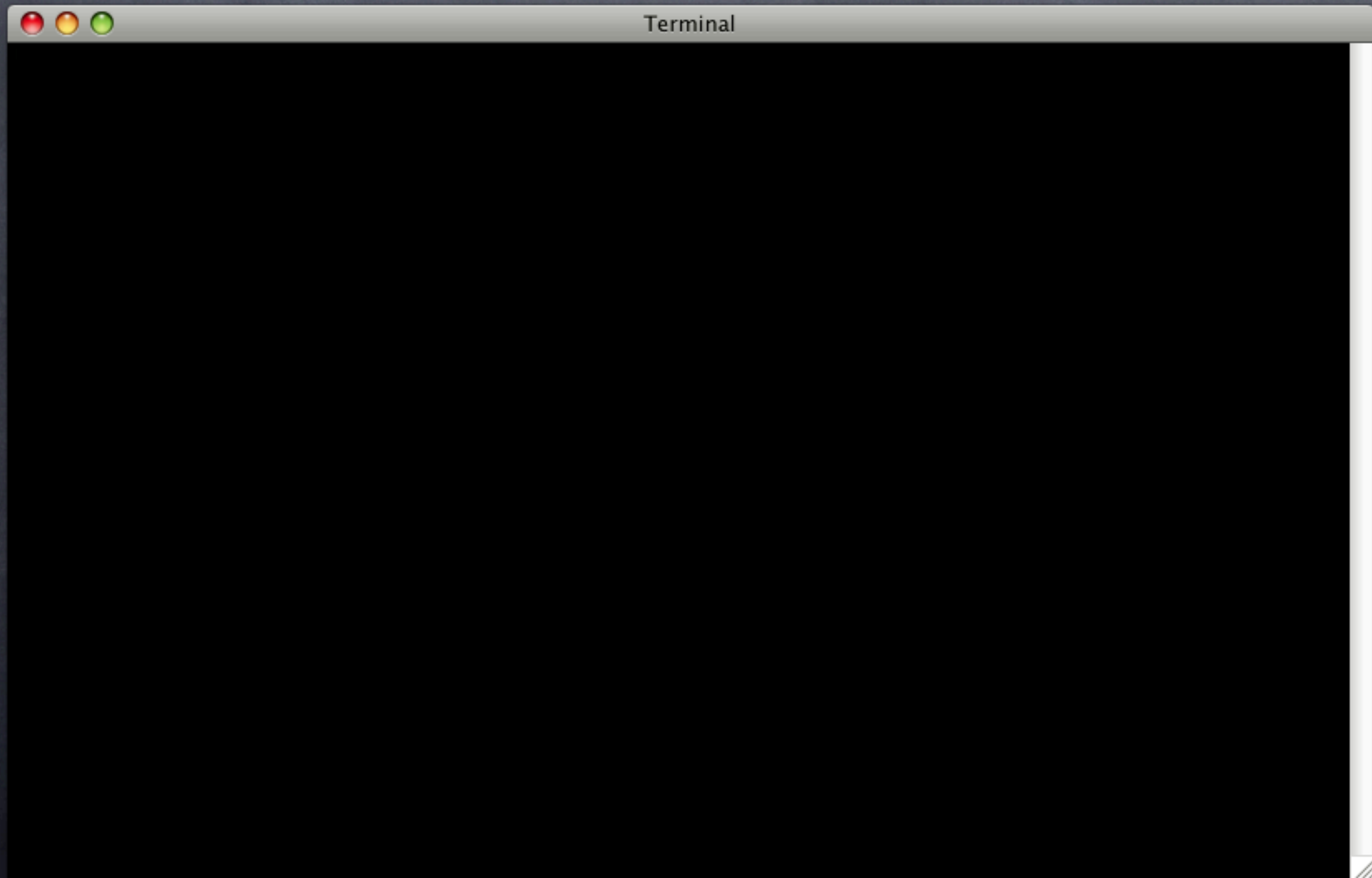
pgTAP

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

Follow Along!

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

Installing pgTAP



Installing pgTAP

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal content shows a shell prompt followed by the command to extract a tarball.

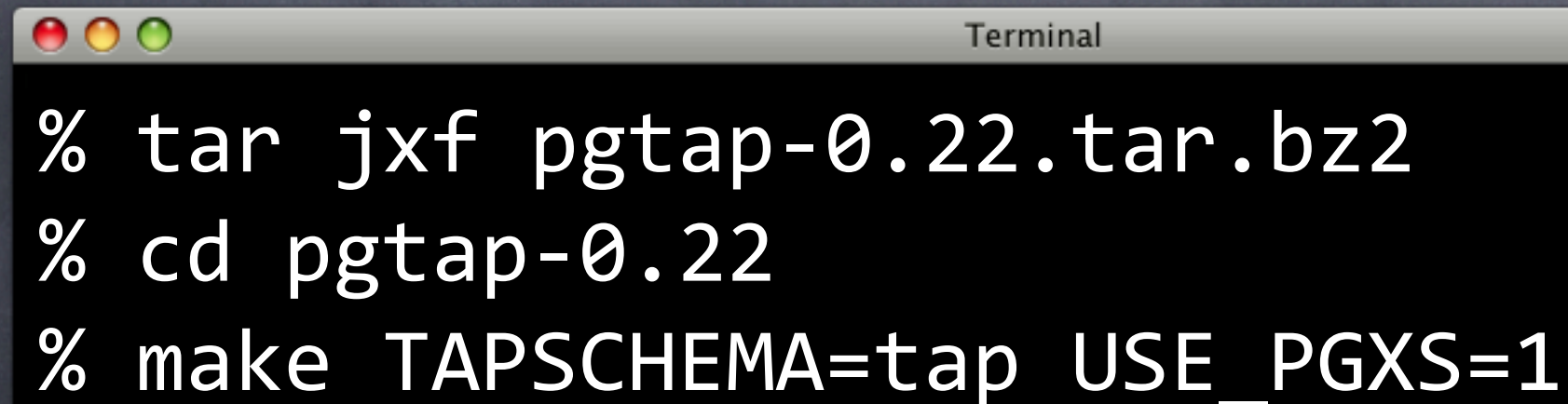
```
% tar jxf pgtap-0.22.tar.bz2
```


Installing pgTAP

A terminal window titled "Terminal" with three window control buttons (red, yellow, green) in the top-left corner. The terminal displays two lines of shell commands: "% tar jxf pgtap-0.22.tar.bz2" and "% cd pgtap-0.22".

```
Terminal  
% tar jxf pgtap-0.22.tar.bz2  
% cd pgtap-0.22
```


Installing pgTAP

A terminal window titled "Terminal" with three colored window control buttons (red, yellow, green) in the top-left corner. The terminal contains three lines of shell commands:

```
% tar jxf pgtap-0.22.tar.bz2
% cd pgtap-0.22
% make TAPSCHEMA=tap USE_PGXS=1
```


Installing pgTAP



Terminal

```
% tar jxf pgtap-0.22.tar.bz2  
% cd pgtap-0.22  
% make TAPSCHEMA=tap USE_PGXS=1
```

If in contrib...

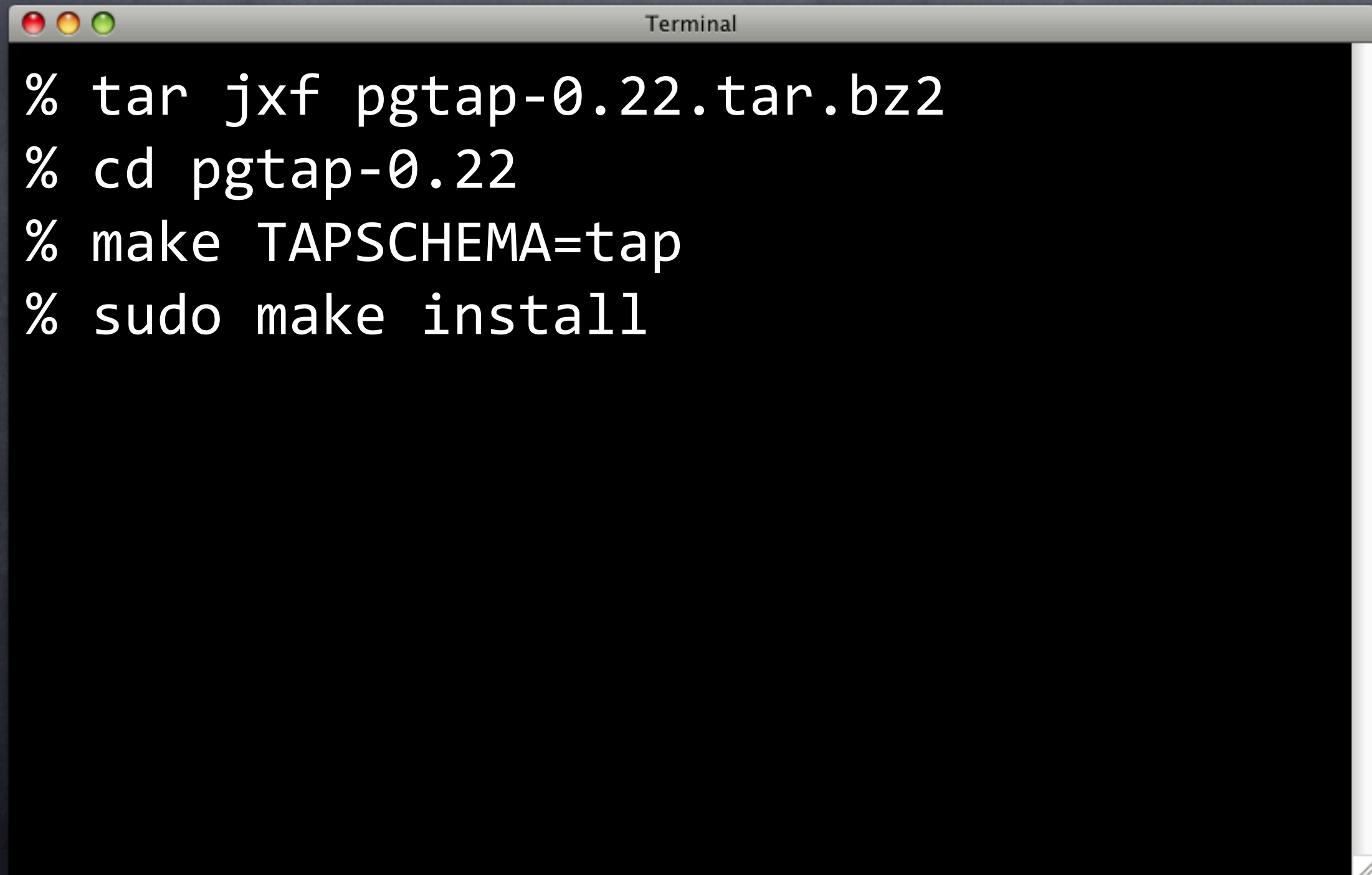
Installing pgTAP

A terminal window titled "Terminal" with three colored window control buttons (red, yellow, green) in the top-left corner. The terminal contains three lines of shell commands: "% tar jxf pgtap-0.22.tar.bz2", "% cd pgtap-0.22", and "% make TAPSCHEMA=tap".

```
Terminal
% tar jxf pgtap-0.22.tar.bz2
% cd pgtap-0.22
% make TAPSCHEMA=tap
```

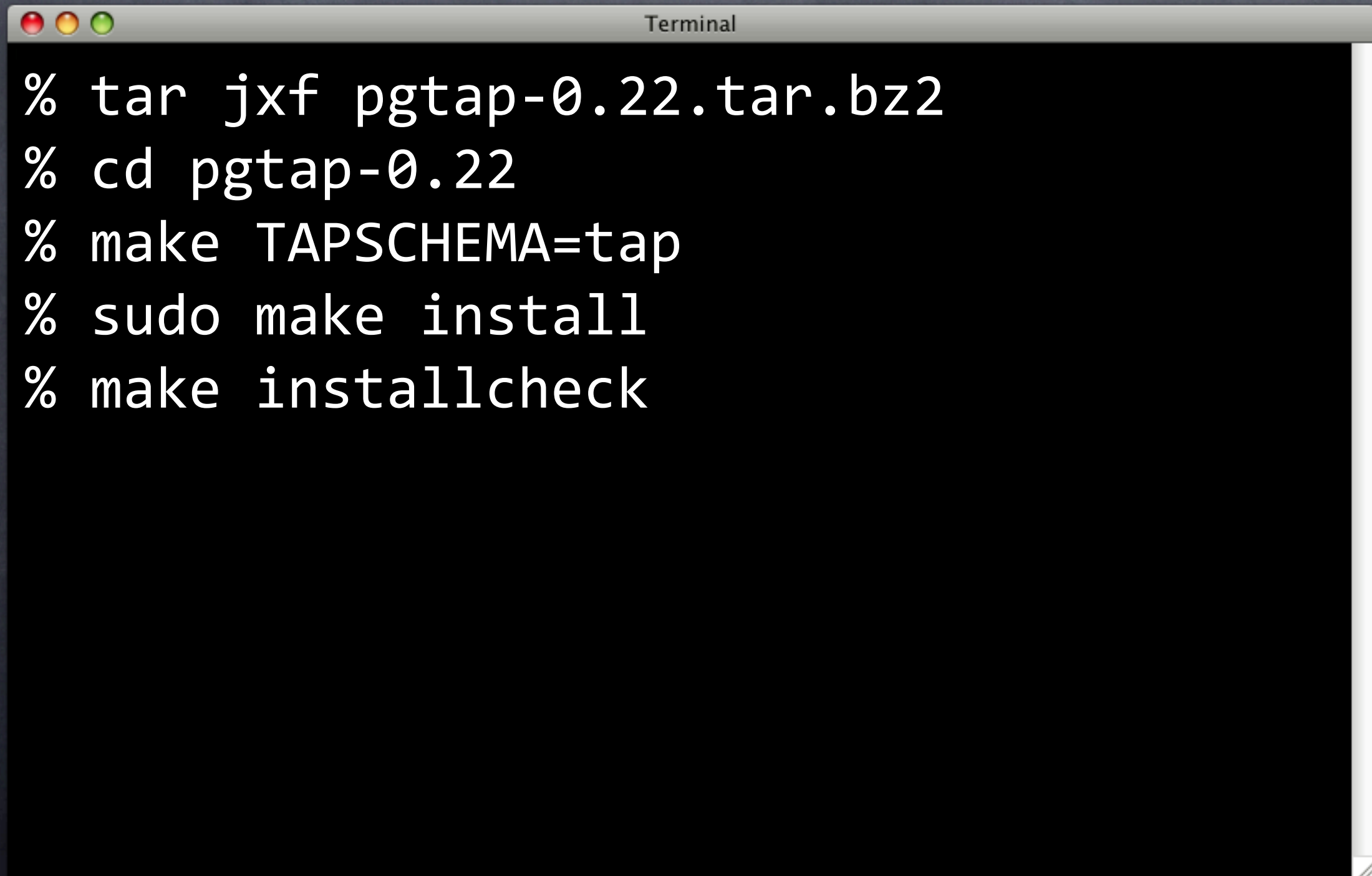
If in contrib...

Installing pgTAP

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal contains four lines of shell commands for installing pgTAP.

```
Terminal  
% tar jxf pgtap-0.22.tar.bz2  
% cd pgtap-0.22  
% make TAPSCHEMA=tap  
% sudo make install
```


Installing pgTAP

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal contains five lines of shell commands for installing pgTAP.

```
Terminal  
% tar jxf pgtap-0.22.tar.bz2  
% cd pgtap-0.22  
% make TAPSCHEMA=tap  
% sudo make install  
% make installcheck
```


Installing pgTAP

```
Terminal
% tar jxf pgtap-0.22.tar.bz2
% cd pgtap-0.22
% make TAPSCHEMA=tap
% sudo make install
% make installcheck
% psql -d template1 -d pgtap.sql
```

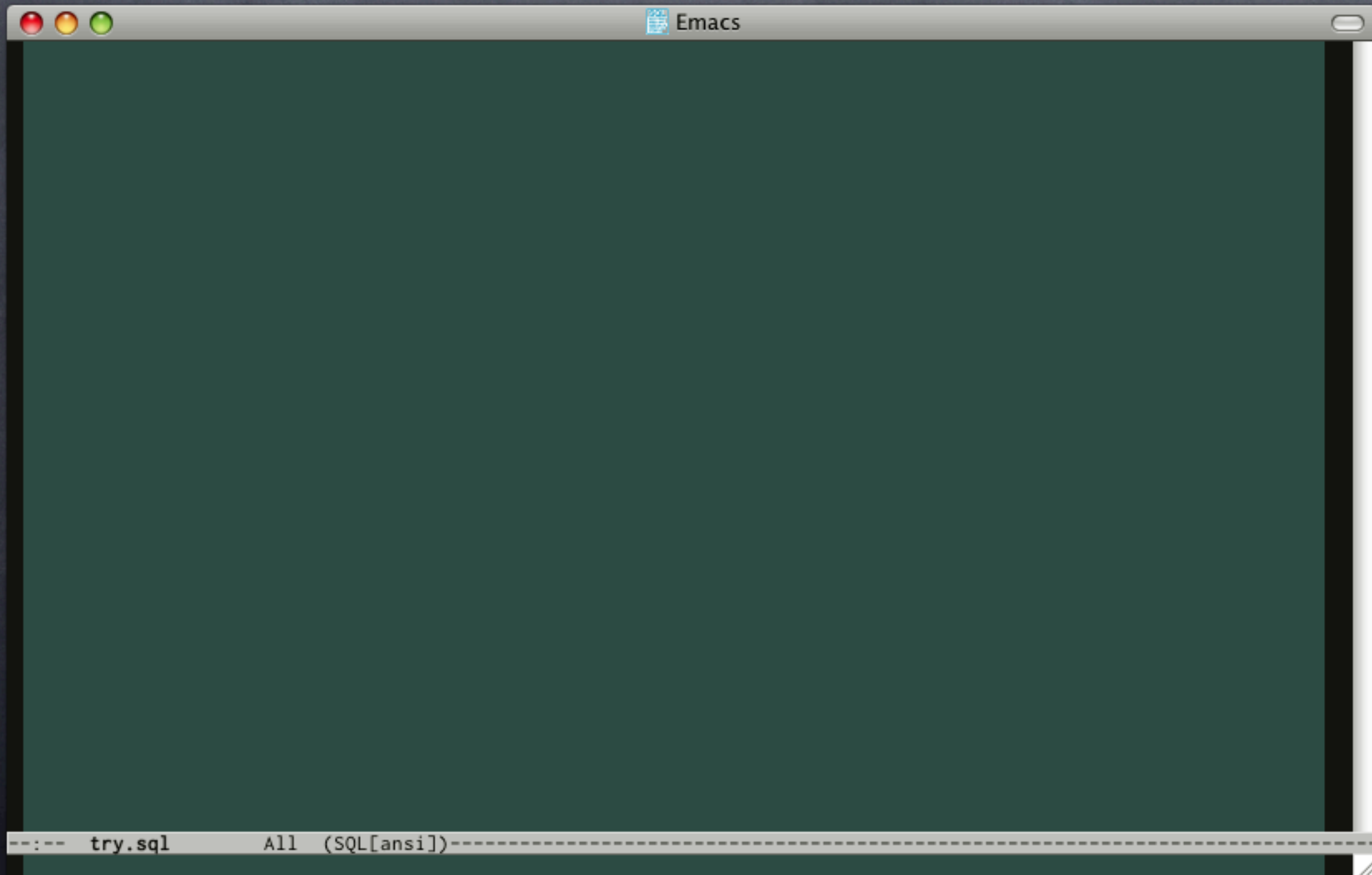

Installing pgTAP

A terminal window titled "Terminal" with a standard macOS-style title bar (red, yellow, green buttons). The terminal contains the following commands:

```
% tar jxf pgtap-0.22.tar.bz2
% cd pgtap-0.22
% make TAPSCHEMA=tap
% sudo make install
% make installcheck
% psql -d template1 -d pgtap.sql
```

Need postgresql-dev

pgTAP Basics

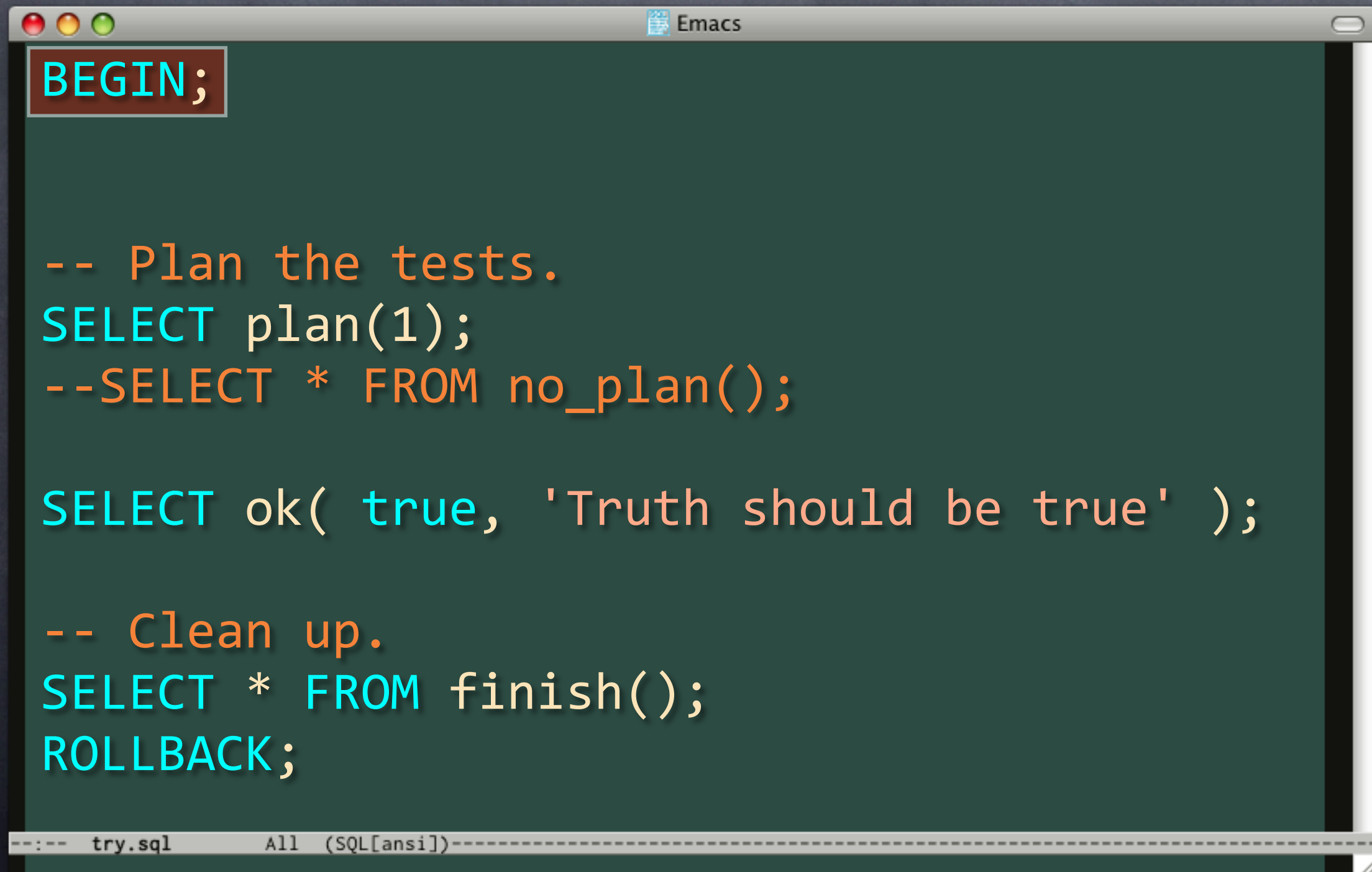


pgTAP Basics

```
BEGIN;  
  
-- Plan the tests.  
SELECT plan(1);  
--SELECT * FROM no_plan();  
  
SELECT ok( true, 'Truth should be true' );  
  
-- Clean up.  
SELECT * FROM finish();  
ROLLBACK;
```

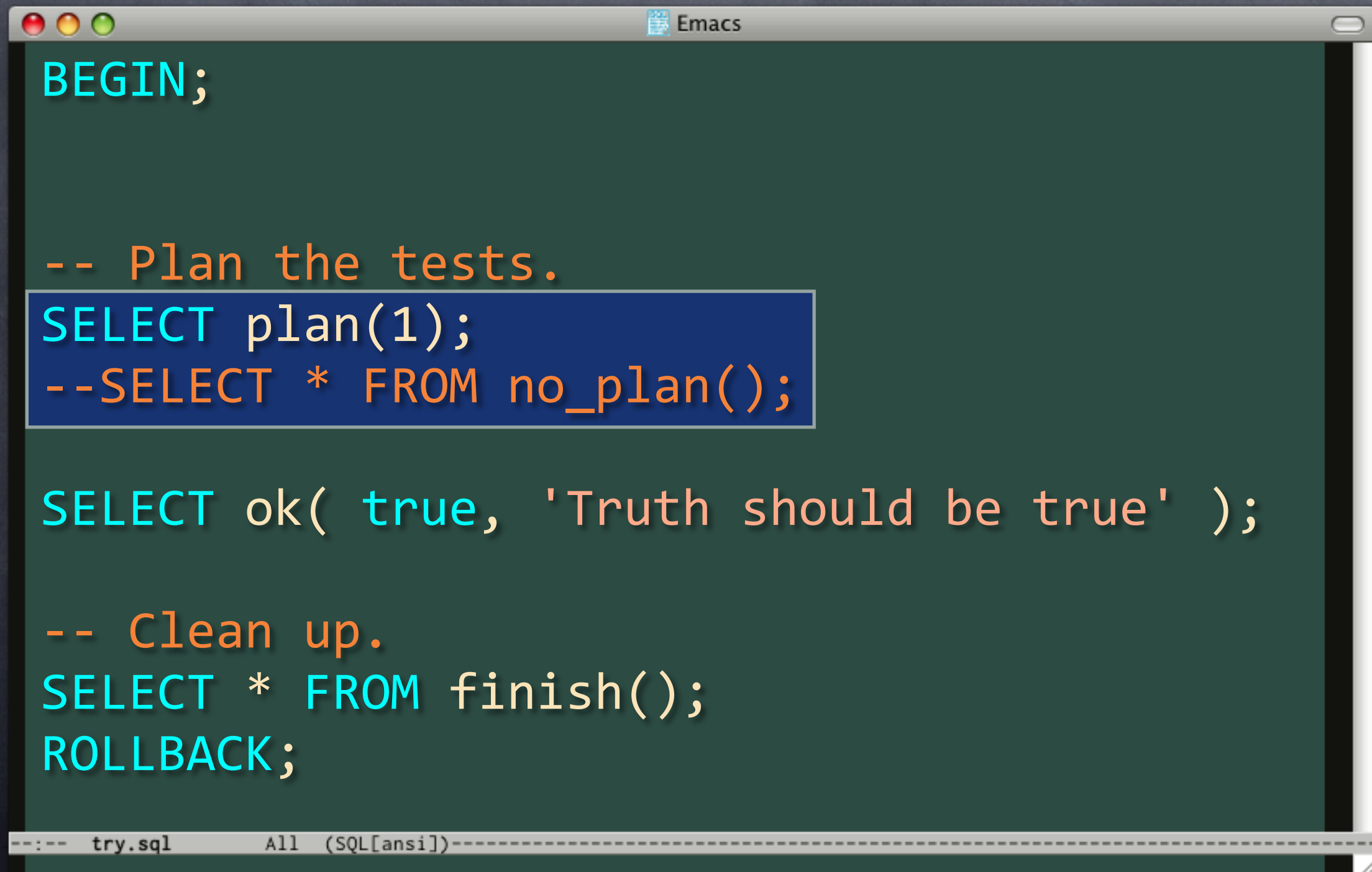
```
--:-- try.sql All (SQL[ansi])-----
```


pgTAP Basics

The image shows a screenshot of an Emacs window. The window title bar at the top says "Emacs" and has three colored window control buttons (red, yellow, green) on the left. The main content area is a dark green background with text in cyan and orange. The text is SQL code for a pgTAP test. The first line "BEGIN;" is highlighted with a dark red background. The code includes comments in orange, SQL commands in cyan, and a string literal in orange. The status bar at the bottom shows "--:-- try.sql All (SQL[ansi])-----".

```
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

The image shows a screenshot of an Emacs window with a dark green background. The window title bar at the top says "Emacs" and has three colored window control buttons (red, yellow, green) on the left. The main content area contains SQL code for a pgTAP test. The code is color-coded: keywords like "BEGIN", "SELECT", and "ROLLBACK" are in cyan, while comments and function names are in orange. A blue rectangular highlight is placed over the lines "SELECT plan(1);" and "--SELECT * FROM no_plan();". At the bottom of the window, a status bar shows "--:-- try.sql All (SQL[ansi])-----".

```
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

```
BEGIN;
```

```
-- Plan the tests.
```

```
SELECT plan(1);
```

```
--SELECT * FROM no_plan();
```

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

```
-- Clean up.
```

```
SELECT * FROM finish();
```

```
ROLLBACK;
```

```
--:-- try.sql All (SQL[ansi])-----
```


pgTAP Basics

```
BEGIN;
```

```
-- Plan the tests.
```

```
SELECT plan(1);
```

```
--SELECT * FROM no_plan();
```

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

```
-- Clean up.
```

```
SELECT * FROM finish();
```

```
ROLLBACK;
```

```
--:-- try.sql All (SQL[ansi])-----
```


pgTAP Basics

```
BEGIN;
```

```
-- Plan the tests.
```

```
SELECT plan(1);
```

```
--SELECT * FROM no_plan();
```

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

```
-- Clean up.
```

```
SELECT * FROM finish();
```

```
ROLLBACK;
```

```
--:-- try.sql All (SQL[ansi])-----
```


pgTAP Basics

```
BEGIN;
```

```
-- Plan the tests.
```

```
SELECT plan(1);
```

```
--SELECT * FROM no_plan();
```

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

```
-- Clean up.
```

```
SELECT * FROM finish();
```

```
ROLLBACK;
```

If TAPSCHEMA...

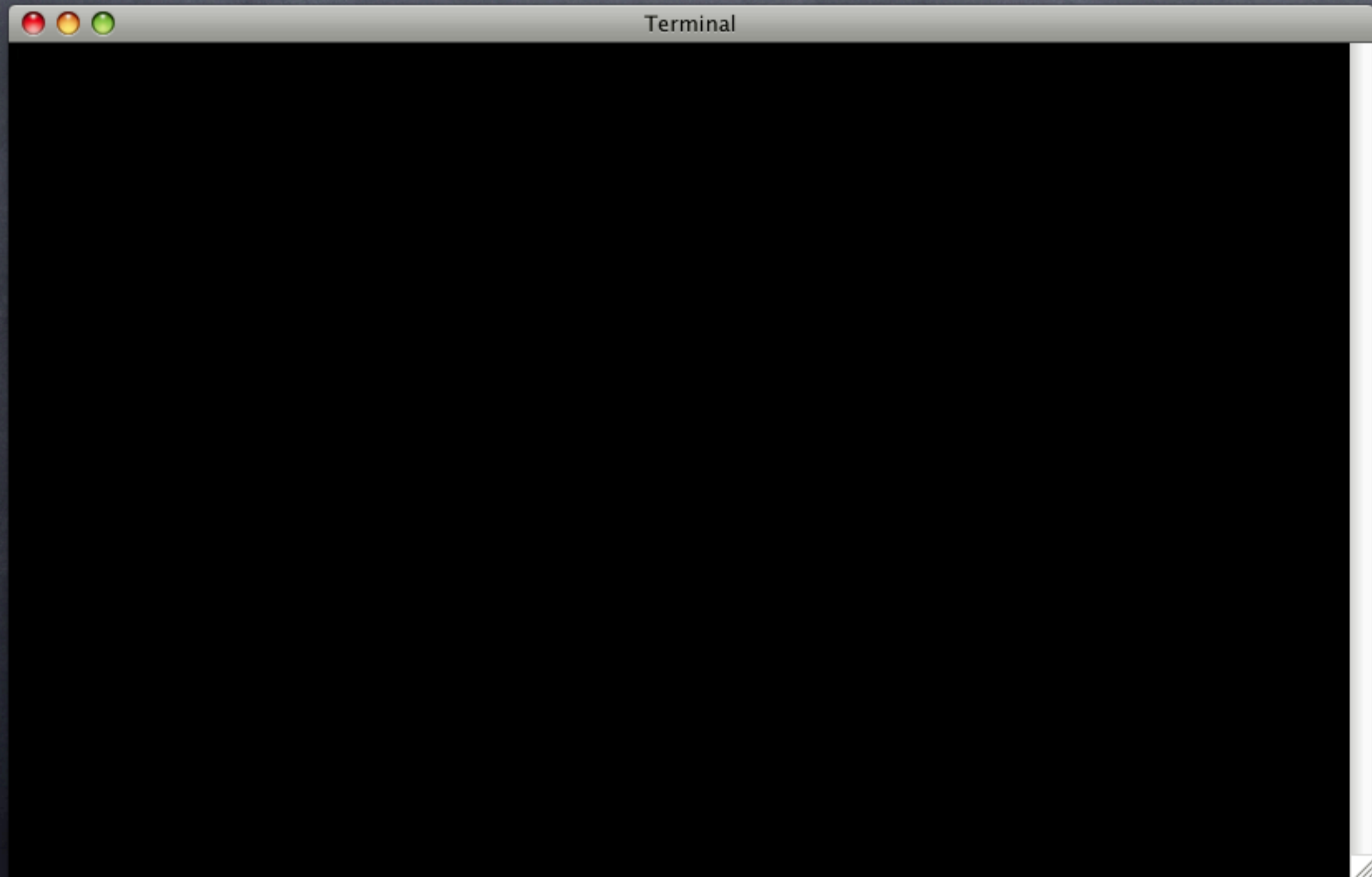
```
--:-- try.sql All (SQL[ansi])-----
```


pgTAP Basics

```
BEGIN;  
SET search_path TO tap, public;  
  
-- Plan the tests.  
SELECT plan(1);  
--SELECT * FROM no_plan();  
  
SELECT ok( true, 'Truth should be true' );  
  
-- Clean up.  
SELECT * FROM finish();  
ROLLBACK;
```

If TAPSCHEMA...

Running Tests



Running Tests

```
Terminal
% pg_prove -v -d try try.sql
try.sql .. psql:try.sql:5: ERROR:  function plan(integer) does
not exist
LINE 1: SELECT plan(1);
                ^
HINT:  No function matches the given name and argument types.
You might need to add explicit type casts.
Dubious, test returned 3 (wstat 768, 0x300)
No subtests run

Test Summary Report
-----
try.sql (Wstat: 768 Tests: 0 Failed: 0)
  Non-zero exit status: 3
  Parse errors: No plan found in TAP output
Files=1, Tests=0,  0 wallclock secs ( 0.02 usr +  0.00 sys =
0.02 CPU)
Result: FAIL
```

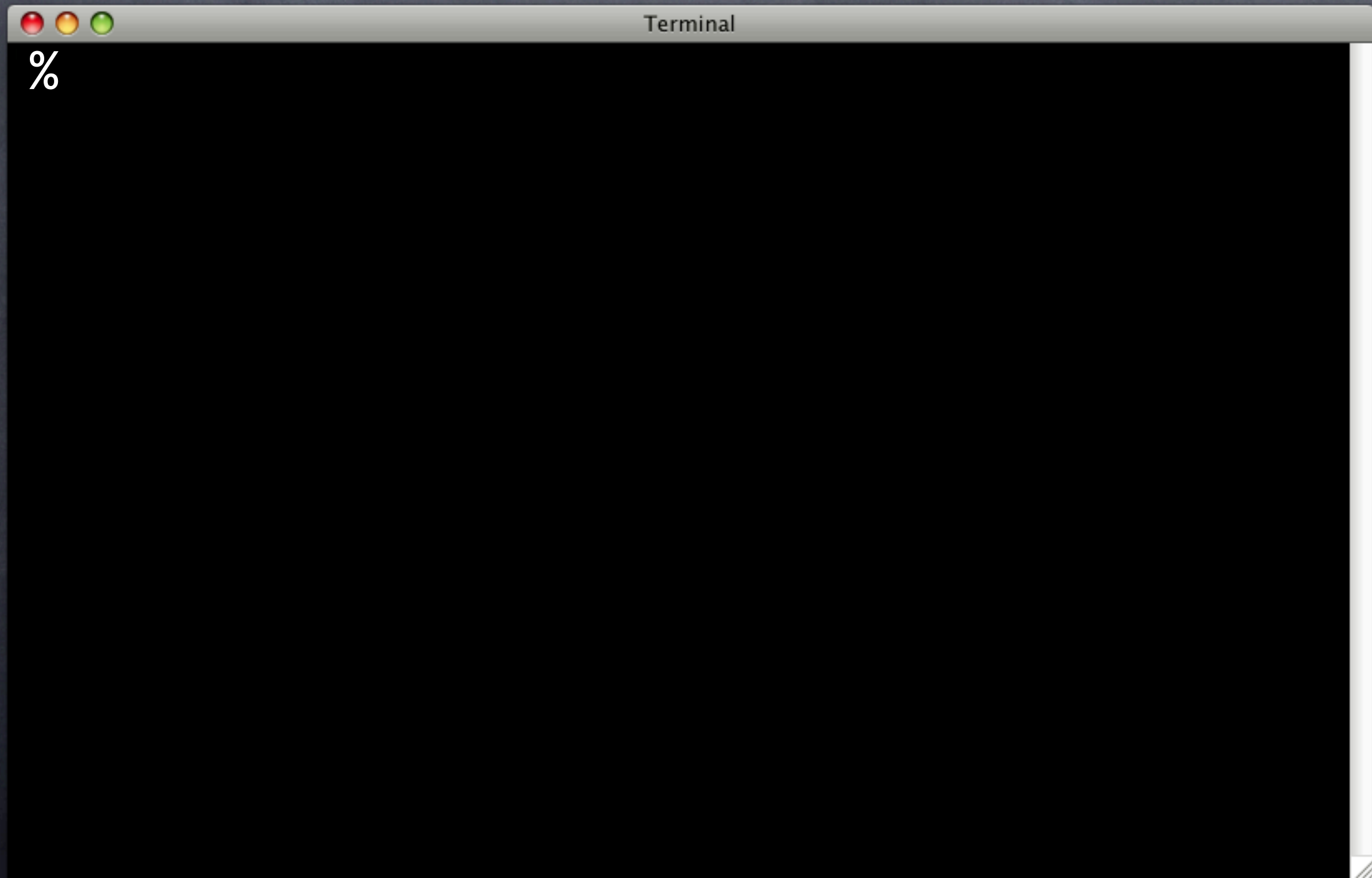

Running Tests

```
Terminal
% pg_prove -v -d try try.sql
try.sql .. psql:try.sql:5: ERROR:  function plan(integer) does
not exist
LINE 1: SELECT plan(1);
                ^
HINT:  No function matches the given name and argument types.
You might need to add explicit type casts.
Dubious, test returned 3 (wstat 768, 0x300)
No subtests run

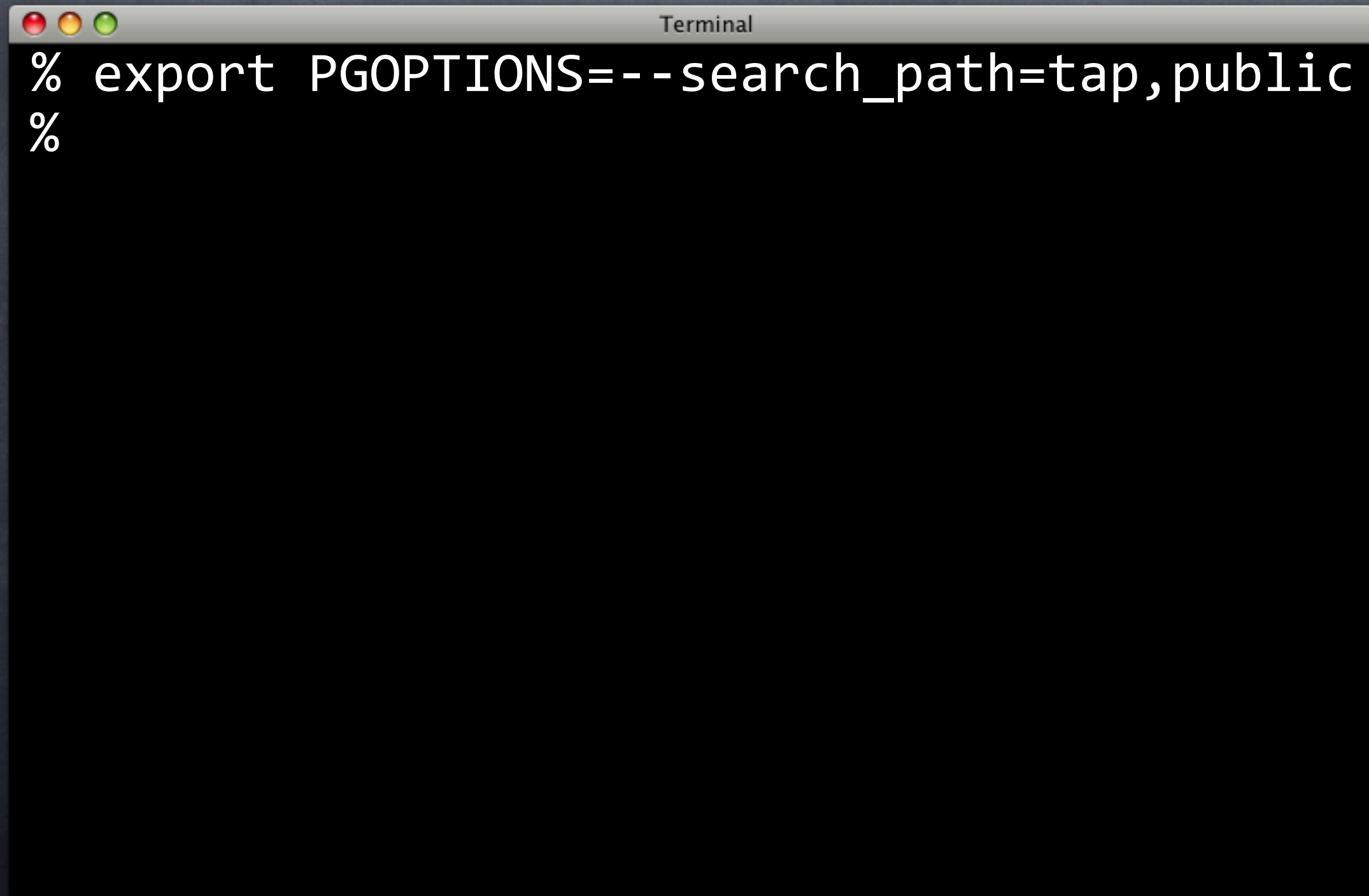
Test Summary Report
-----
try.sql (Wstat: 768 Tests: 0 Failed: 0)
  Non-zero exit status: 3
  Parse errors: No plan found in TAP output
Files=1, Tests=0,  0 wallclock secs ( 0.02 usr +  0.00 sys =
0.02 CPU)
Result: FAIL
```

Oops

Running Tests



Running Tests

A terminal window titled "Terminal" with three window control buttons (red, yellow, green) in the top-left corner. The terminal content shows a shell prompt followed by the command `export PGOPTIONS=--search_path=tap,public` and another shell prompt.

```
Terminal
% export PGOPTIONS=--search_path=tap,public
%
```


Running Tests

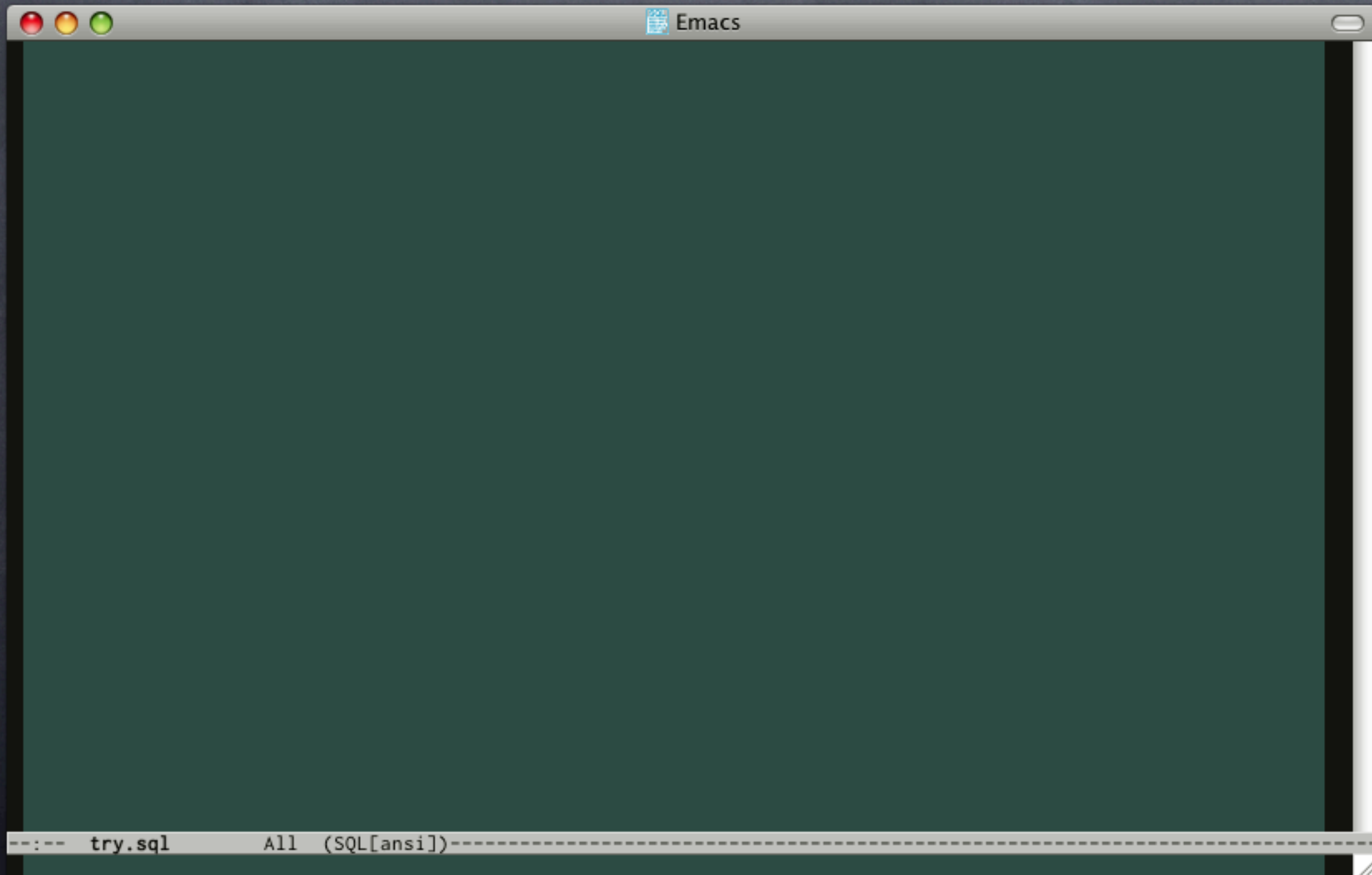
```
Terminal
% export PGOPTIONS=--search_path=tap,public
% pg_prove -v -d try try.sql
try.sql ..
1..1
ok 1 - Truth should be true
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```


Running Tests

```
Terminal
% export PGOPTIONS=--search_path=tap,public
% pg_prove -v -d try try.sql
try.sql ..
1..1
ok 1 - Truth should be true
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS

WOOT!
```


♥ Failure



♥ Failure

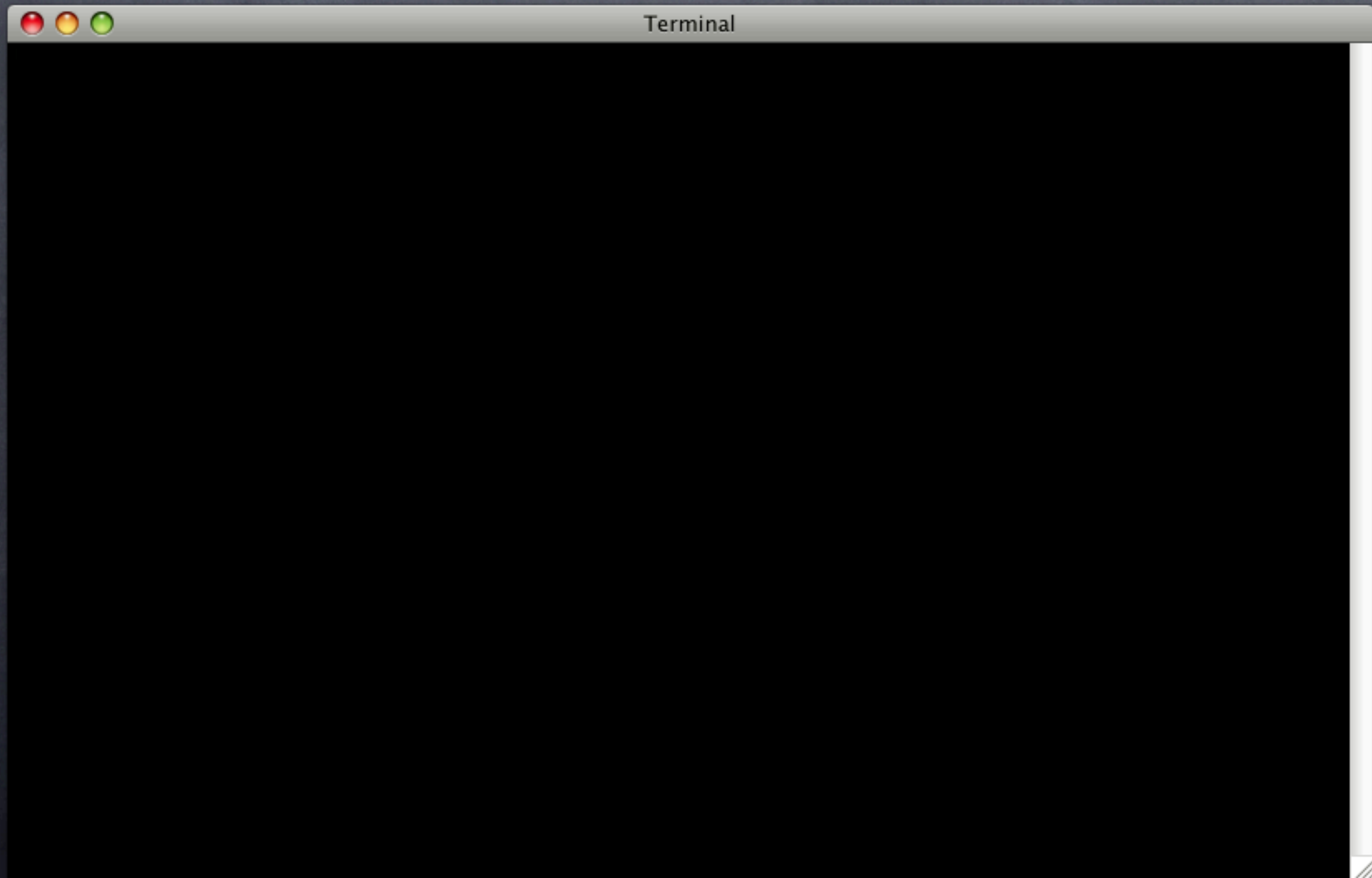
```
Emacs
BEGIN;

SELECT plan(2);
SELECT ok( 1 + 1 = 2 );
SELECT ok( 2 + 2 = 5 );

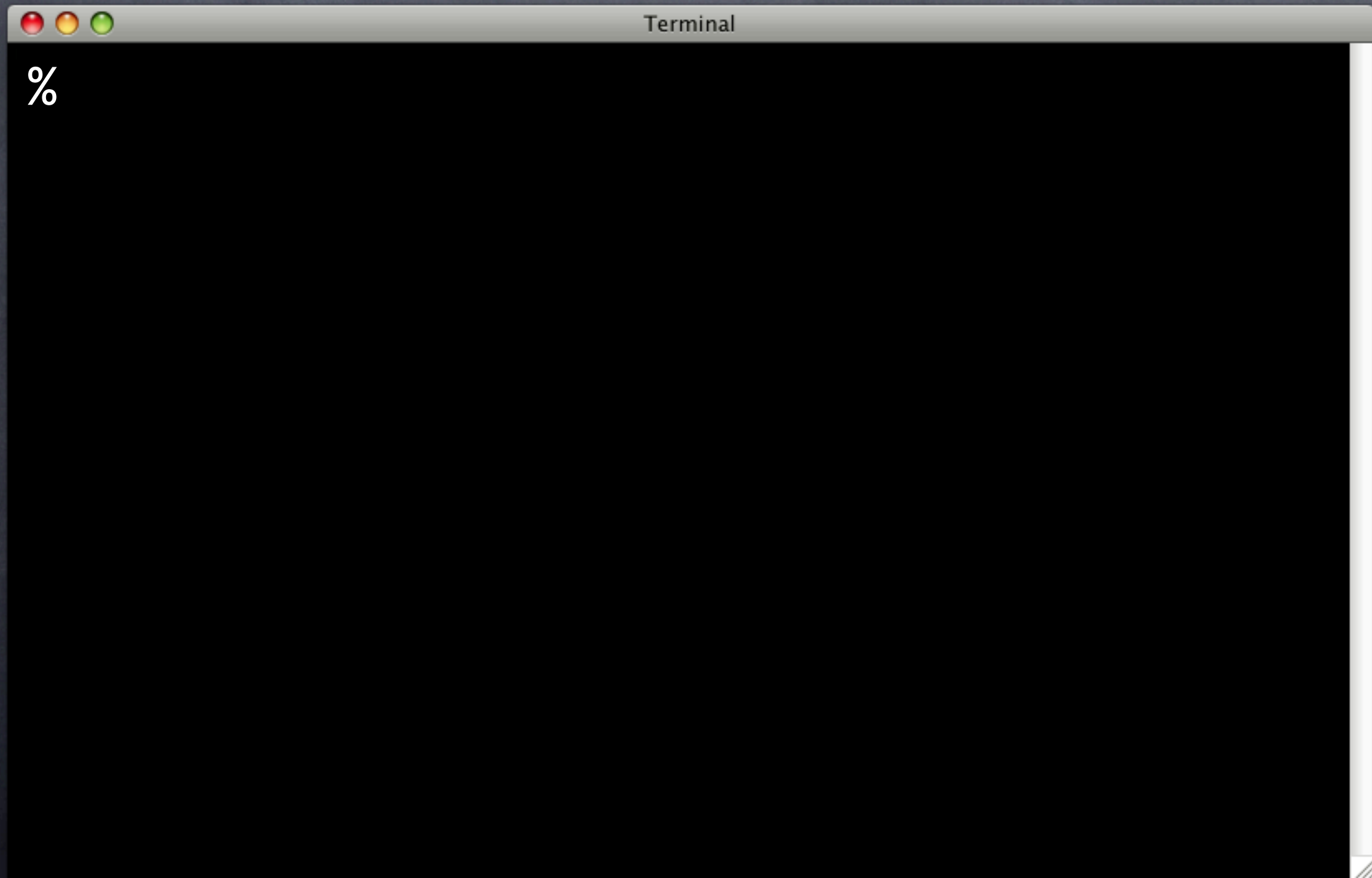
SELECT * FROM finish();
ROLLBACK;

---:-- try.sql      All (SQL[ansi])-----
```


Failing Tests



Failing Tests



Failing Tests

```
Terminal
% pg_prove -v -d try fail.sql
fail.sql ..
1..2
ok 1
not ok 2
# Failed test 2
# Looks like you failed 1 test of 2
Failed 1/2 subtests
```


Failing Tests

```
Terminal
% pg_prove -v -d try fail.sql
fail.sql ..
1..2
ok 1
not ok 2
# Failed test 2
# Looks like you failed 1 test of 2
Failed 1/2 subtests
```


Failing Tests

```
Terminal
% pg_prove -v -d try fail.sql
fail.sql ..
1..2
ok 1
not ok 2
# Failed test 2
# Looks like you failed 1 test of 2
Failed 1/2 subtests
```


Failing Tests

```
Terminal
% pg_prove -v -d try fail.sql
fail.sql ..
1..2
ok 1
not ok 2
# Failed test 2
# Looks like you failed 1 test of 2
Failed 1/2 subtests
```


Failing Tests

```
Terminal
% pg_prove -v -d try fail.sql
fail.sql ..
1..2
ok 1
not ok 2
# Failed test 2
# Looks like you failed 1 test of 2
Failed 1/2 subtests
```


Scalar Testing

Scalar Testing

- Where to start testing?

Scalar Testing

- Where to start testing?
- Start with one unit of code

Scalar Testing

- **Where to start testing?**
- **Start with one unit of code**
- **Maybe a custom data type**

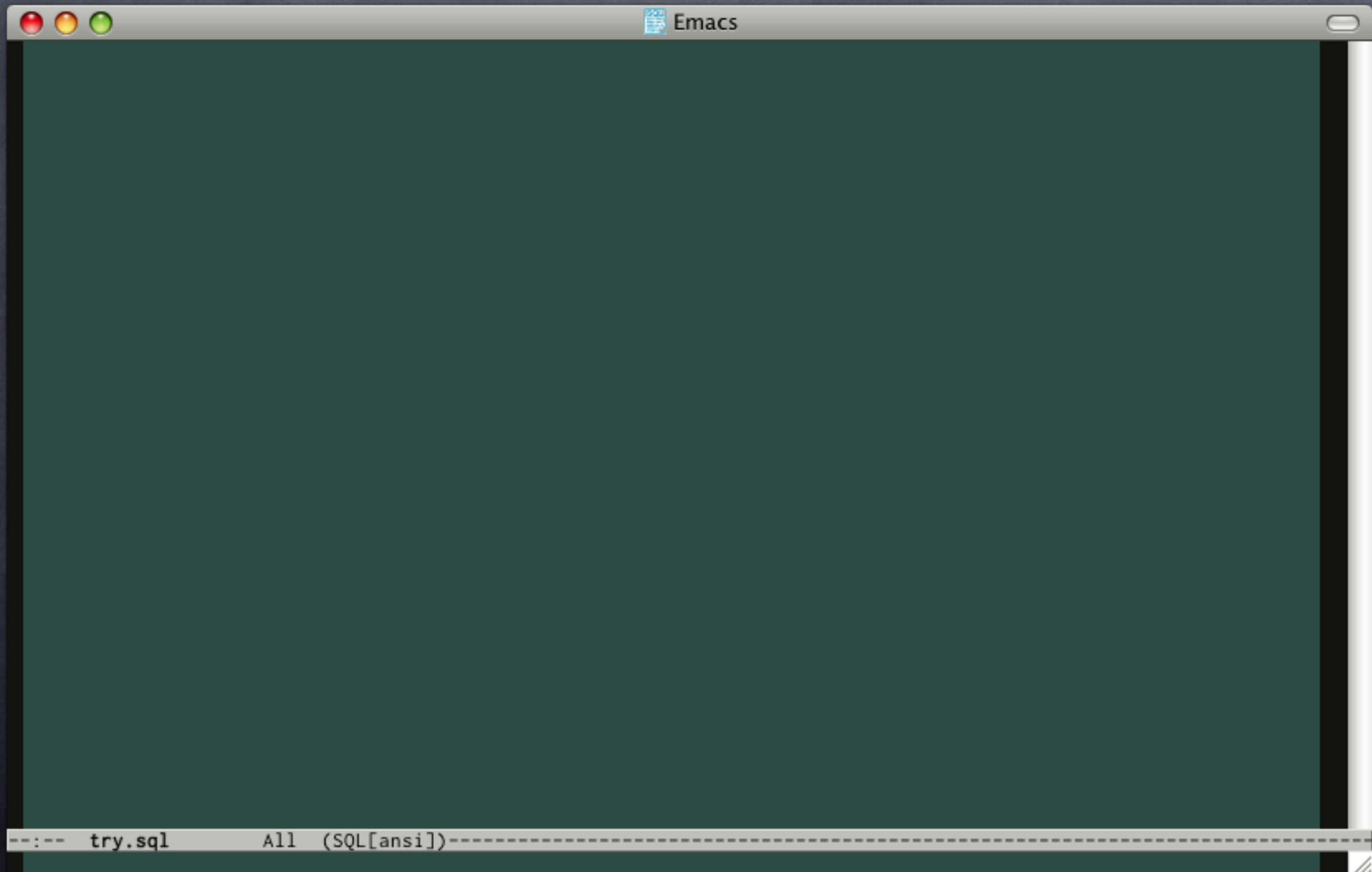
Scalar Testing

- **Where to start testing?**
- **Start with one unit of code**
- **Maybe a custom data type**
- **Obvious candidate for scalar testing**

Scalar Testing

- **Where to start testing?**
- **Start with one unit of code**
- **Maybe a custom data type**
- **Obvious candidate for scalar testing**
- **Testing scalar values**

Testing hstore

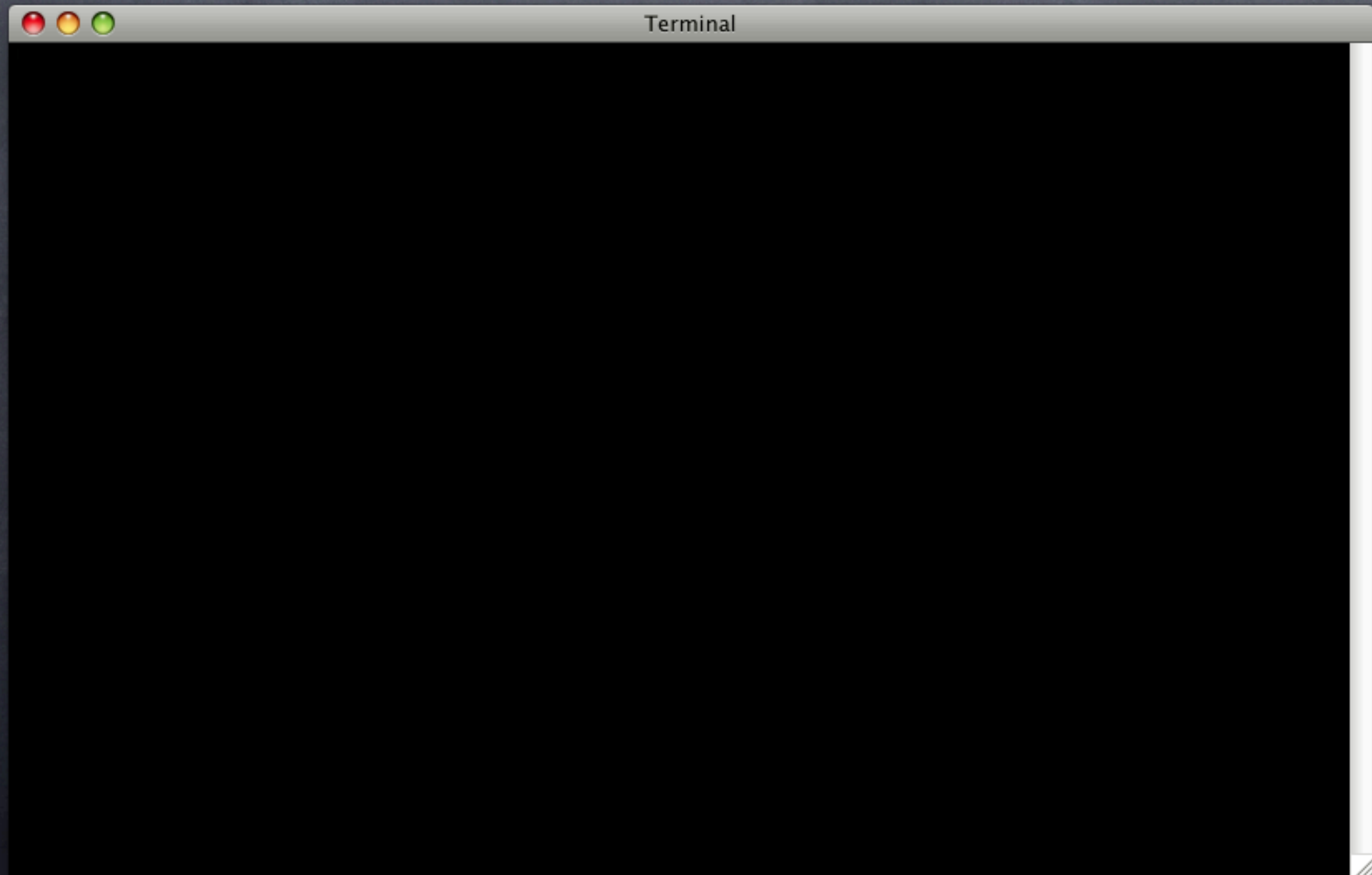


Testing hstore

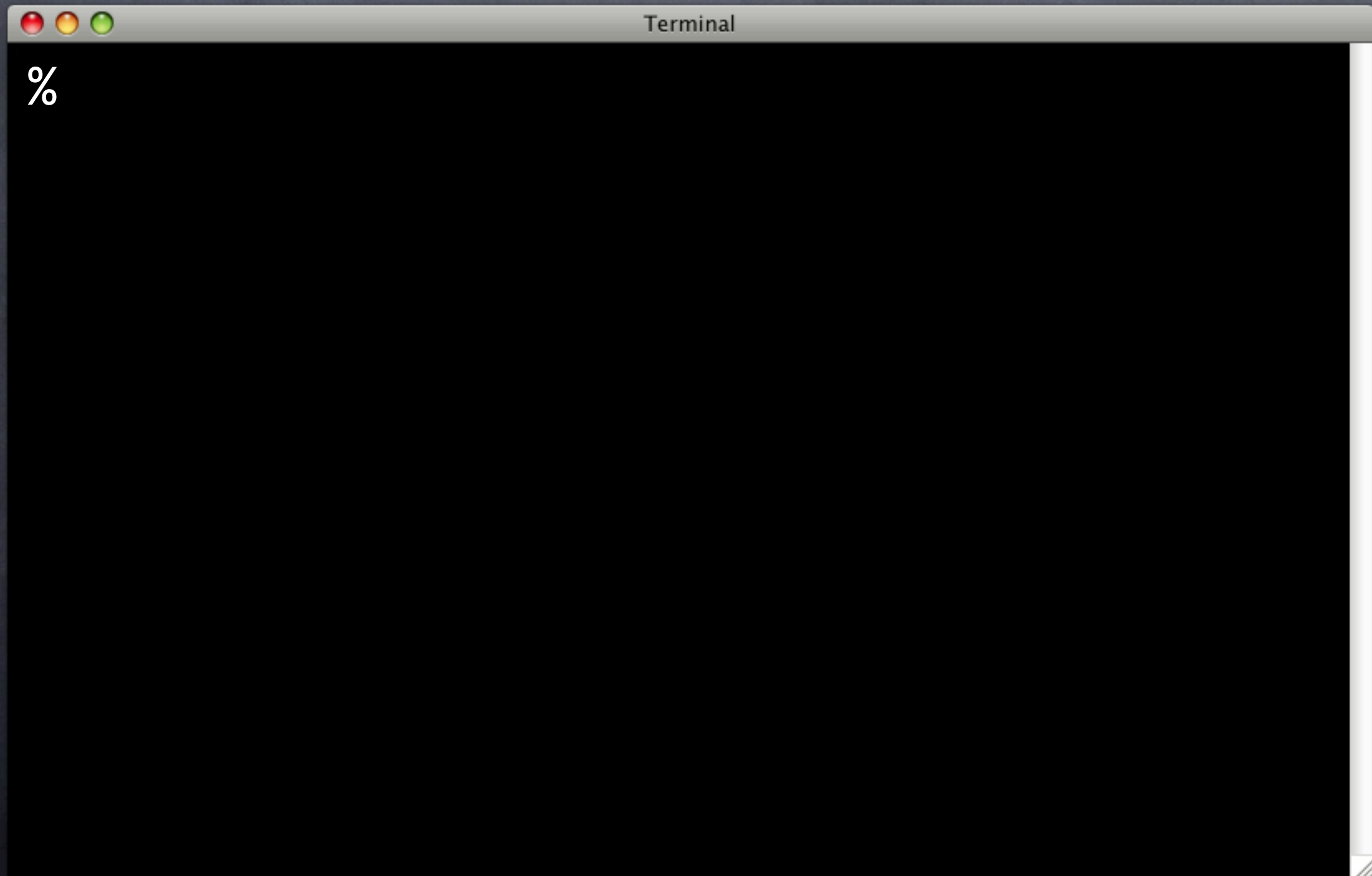
```
BEGIN;  
  
SELECT plan(2);  
SELECT ok( 'k => 1'::hstore IS NOT NULL );  
SELECT ok(  
    pg_typeof('k' => '2') = 'hstore'::regtype  
);  
  
SELECT * FROM finish();  
ROLLBACK;
```

--:-- try.sql All (SQL[ansi])-----

Testing hstore



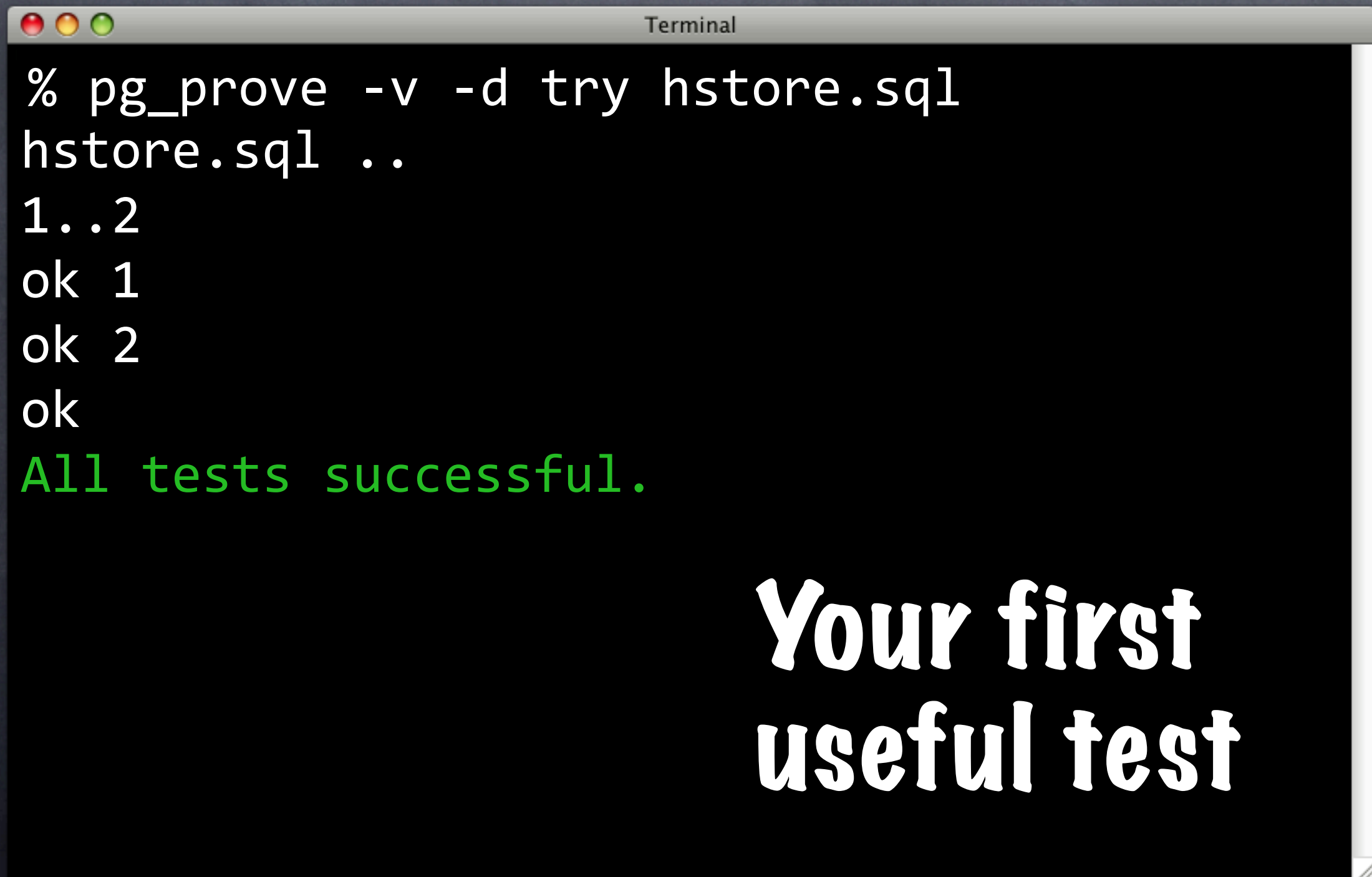
Testing hstore



Testing hstore

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..2
ok 1
ok 2
ok
All tests successful.
```

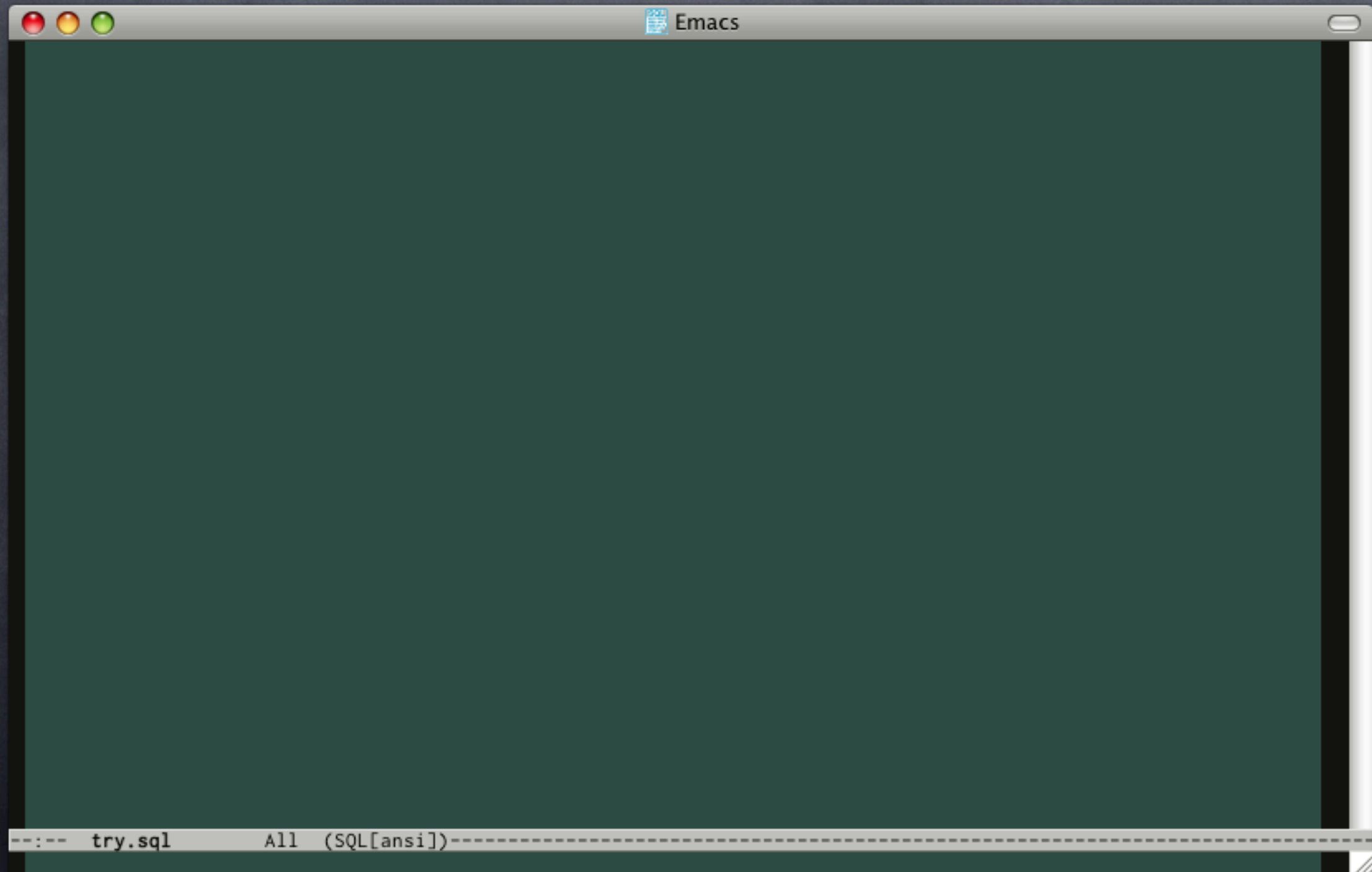

Testing hstore



```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..2
ok 1
ok 2
ok
All tests successful.
```

**Your first
useful test**

Describe Yourself

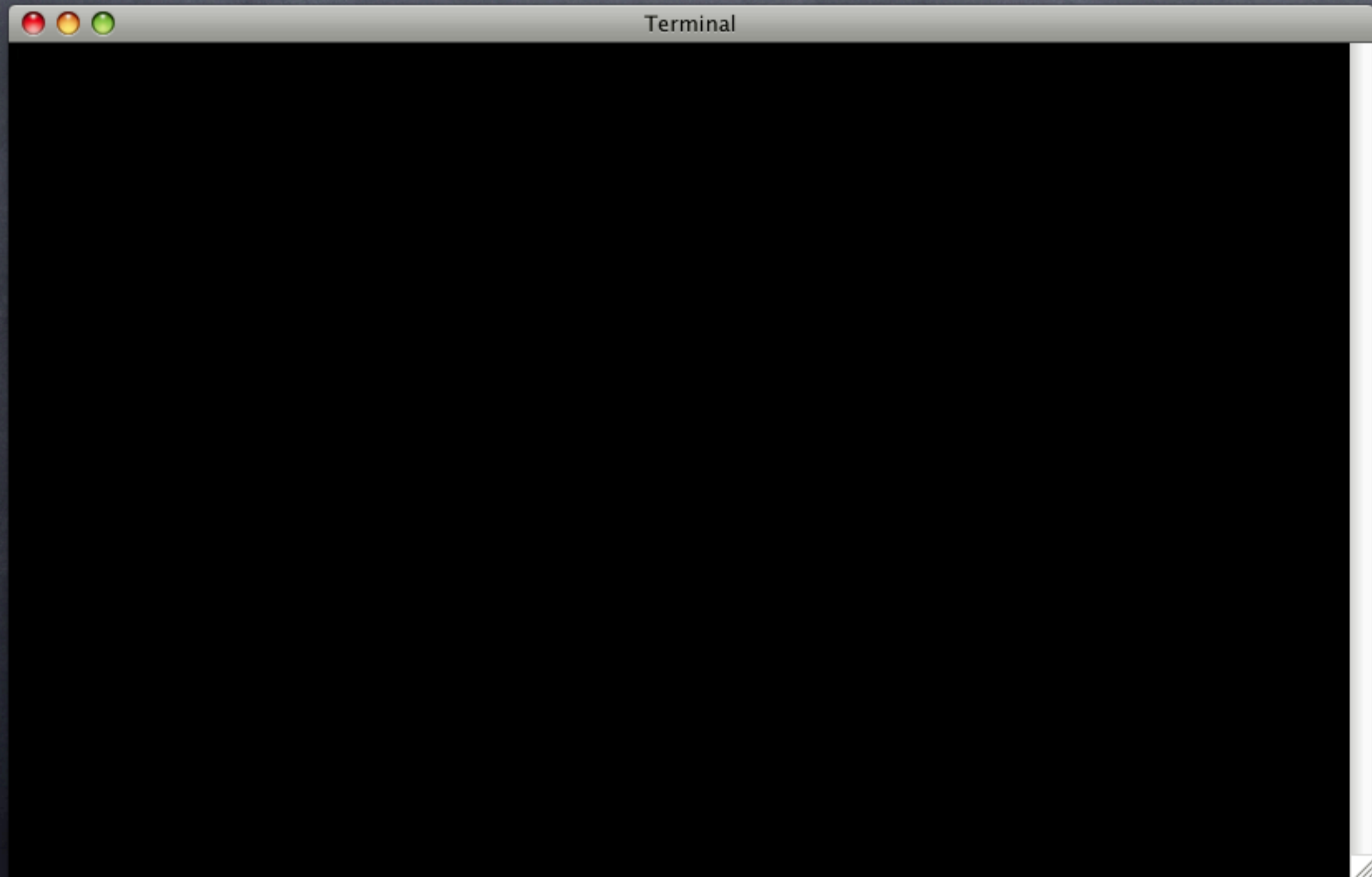


Describe Yourself

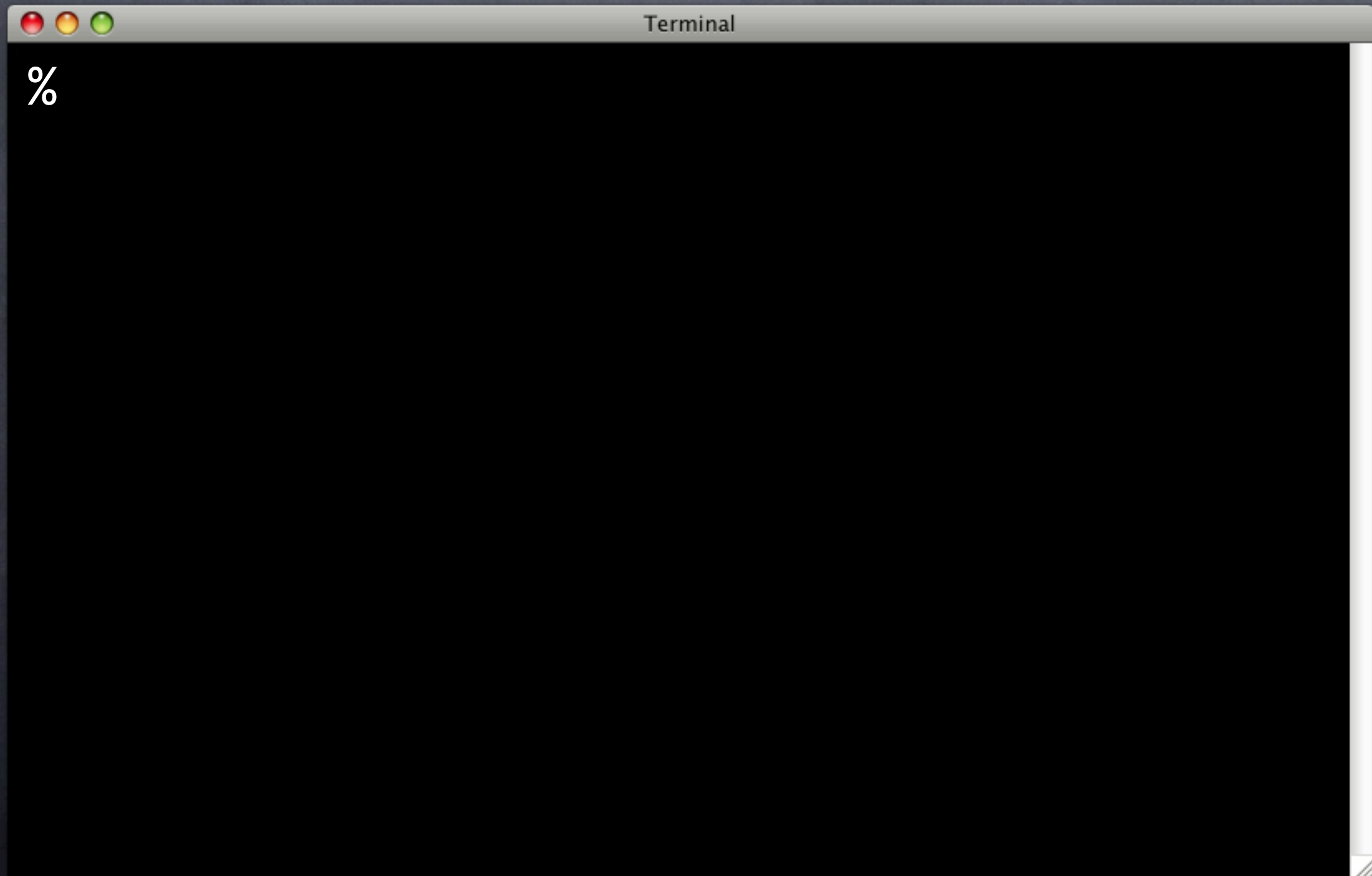
```
SELECT plan(2);
SELECT ok(
    'k => 1'::hstore IS NOT NULL,
    'hstore should work'
);
SELECT ok(
    pg_typeof('k' => '2') = 'hstore'::regtype,
    '=> should create hstore'
);

SELECT * FROM finish();
```


Describe Yourself



Describe Yourself



Describe Yourself

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..2
ok 1 - hstore should work
ok 2 - => should create hstore
ok
All tests successful.
```


Simplify, Simplify

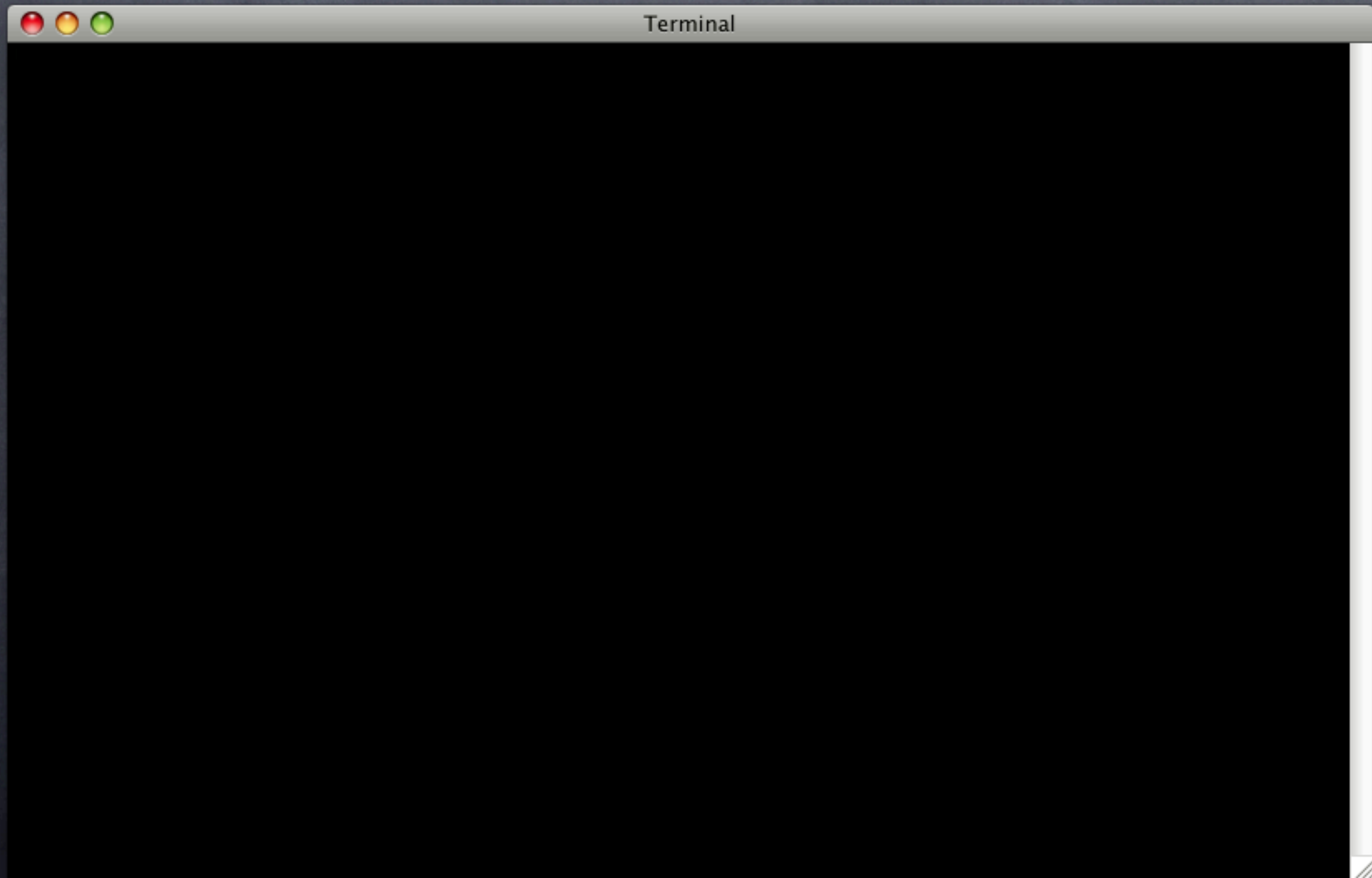
```
SELECT plan(2);
SELECT ok(
    'k => 1'::hstore IS NOT NULL,
    'hstore should work'
);
SELECT ok(
    pg_typeof('k' => '2') = 'hstore'::regtype,
    '=> should create hstore'
);
SELECT * FROM finish();
```


Simplify, Simplify

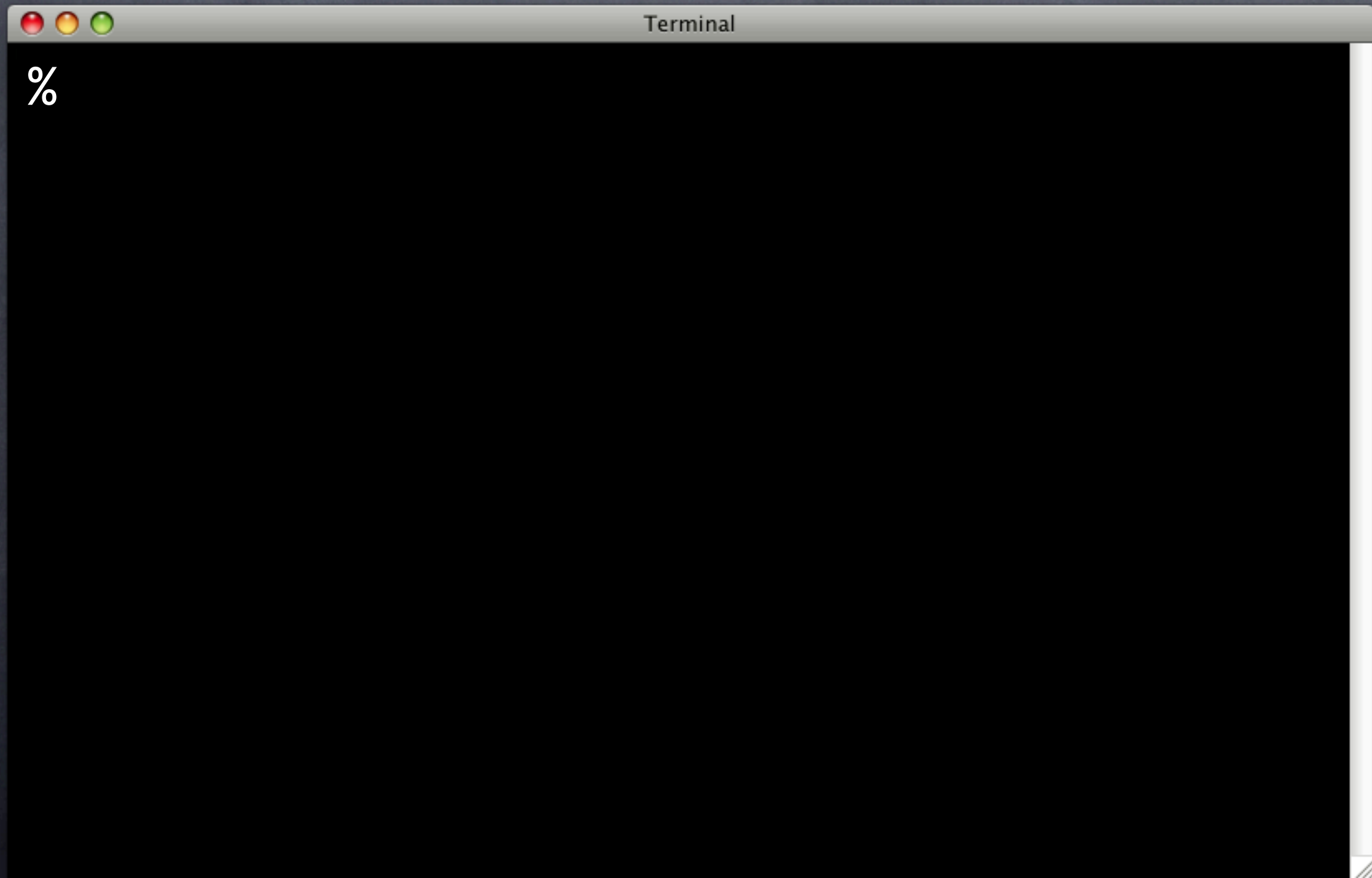
```
SELECT plan(2);
SELECT ok(
    'k => 1'::hstore IS NOT NULL,
    'hstore should work'
);
SELECT isa_ok(
    'k' => '2', 'hstore', '=> return value'
);
SELECT * FROM finish();
```

--- try.sql All (SQL[ansi])-----

Simplify, Simplify



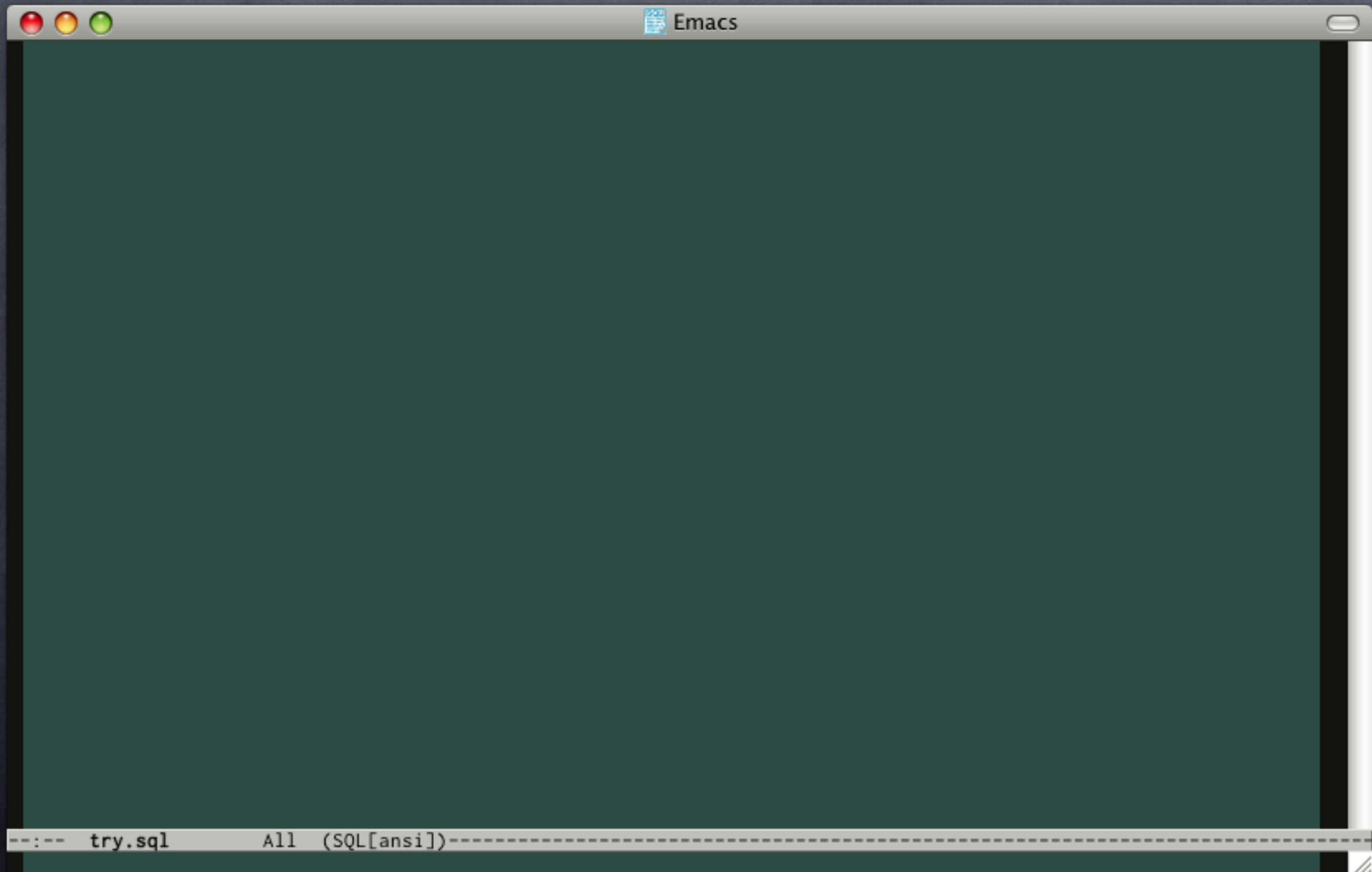
Simplify, Simplify



Simplify, Simplify

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..2
ok 1 - hstore should work
ok 2 - => return value isa hstore
ok
All tests successful.
```


Test the Manual



Test the Manual

```
SELECT plan(6);
```

```
SELECT ok( 'a=>x, b=>y'::hstore -> 'a' = 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

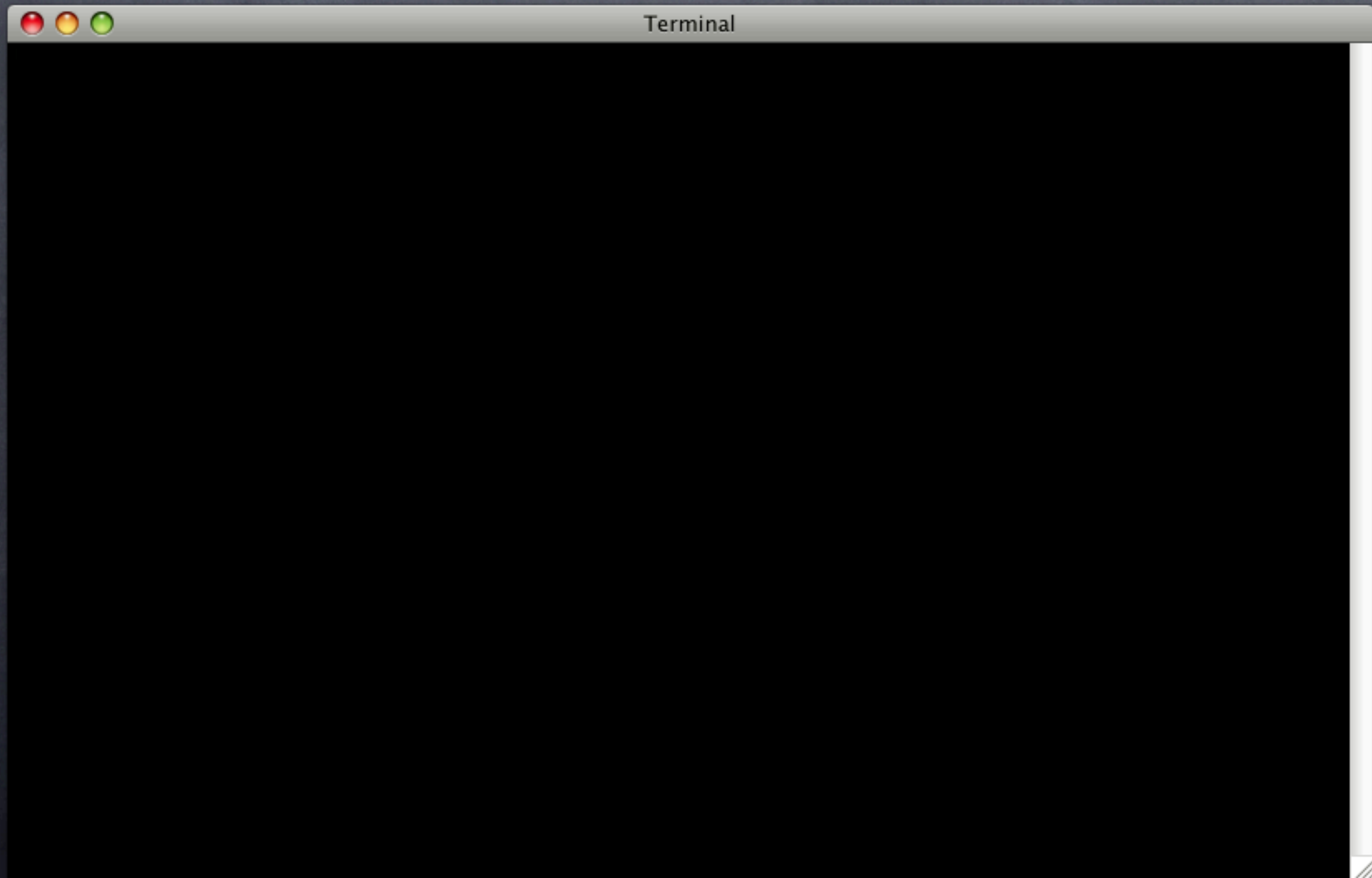
```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT ok( akeys('a=>1,b=>2') = '{a,c}', 'akeys()' );
```

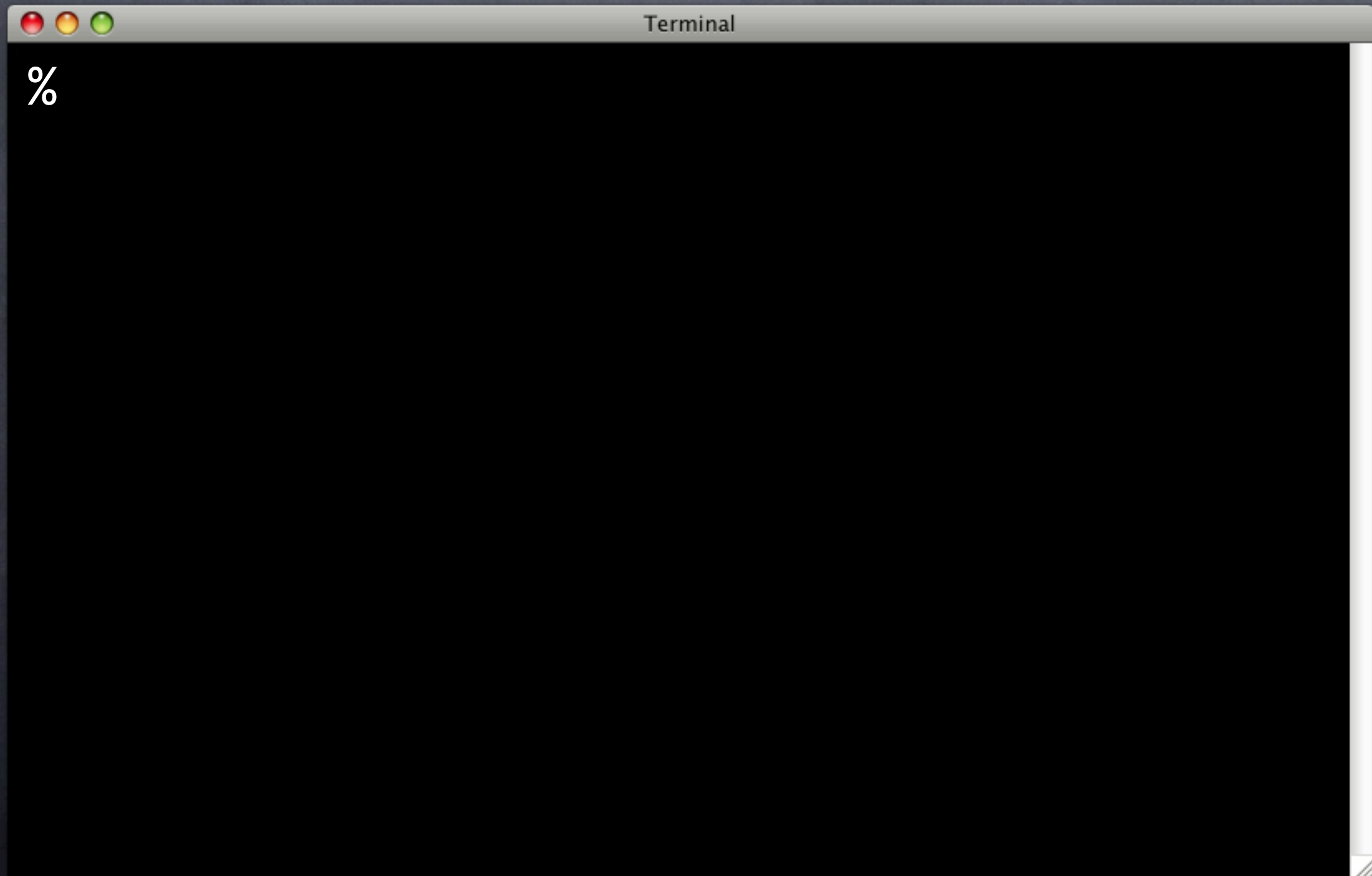
```
SELECT ok( avals('a=>1,b=>2') = '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


Test the Manual



Test the Manual



Test the Manual

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..6
ok 1 - op ->
ok 2 - op ?
ok 3 - op @>
ok 4 - op <@
not ok 5 - akeys()
# Failed test 5: "akeys()"
ok 6 - avals()
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Test the Manual

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..6
ok 1 - op ->
ok 2 - op ?
ok 3 - op @>
ok 4 - op <@
not ok 5 - akeys()
# Failed test 5: "akeys()"
ok 6 - avals()
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```

Um, why?

What is() It?

```
SELECT plan(6);
```

```
SELECT ok( 'a=>x, b=>y'::hstore -> 'a' = 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT ok( akeys('a=>1,b=>2') = '{a,c}', 'akeys()' );
```

```
SELECT ok( avals('a=>1,b=>2') = '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


What is() It?

```
SELECT plan(6);
```

```
SELECT is( 'a=>x, b=>y'::hstore -> 'a', 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

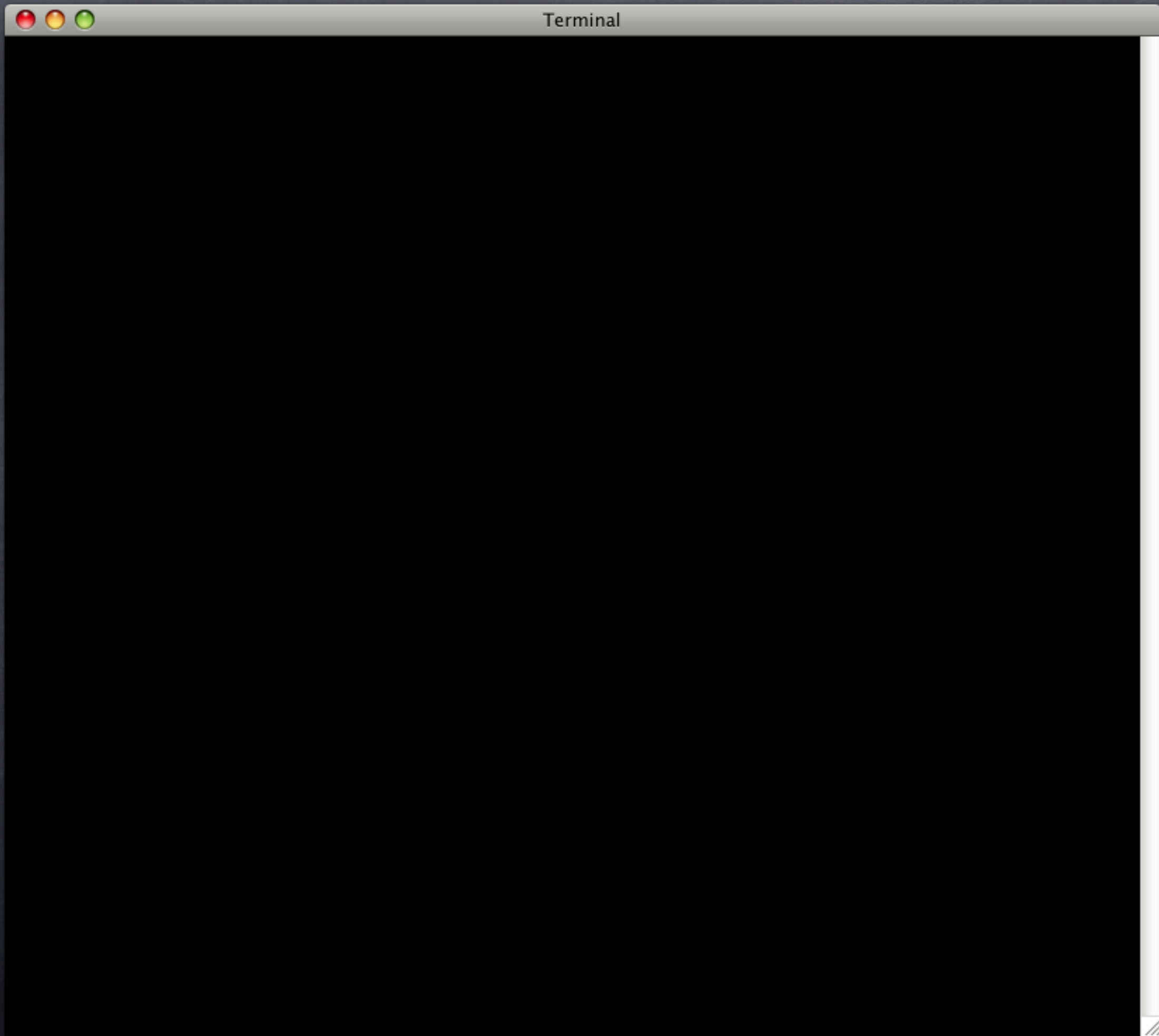
```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT is( akeys('a=>1,b=>2'), '{a,c}', 'akeys()' );
```

```
SELECT is( avals('a=>1,b=>2'), '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```



%


```
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..6
ok 1 - op ->
ok 2 - op ?
ok 3 - op @>
ok 4 - op <@
not ok 5 - akeys()
# Failed test 5: "akeys()"
#       have: {a,b}
#       want: {a,c}
ok 6 - avals()
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```



```
% pg_prove -v -d try hstore.sql
```

```
hstore.sql ..
```

```
1..6
```

```
ok 1 - op ->
```

```
ok 2 - op ?
```

```
ok 3 - op @>
```

```
ok 4 - op <@
```

```
not ok 5 - akeys()
```

```
# Failed test 5: "akeys()"
```

```
#           have: {a,b}
```

```
#           want: {a,c}
```

```
ok 6 - avals()
```

```
# Looks like you failed 1 test of 6
```

```
Failed 1/6 subtests
```



```
% pg_prove -v -d try hstore.sql
```

```
hstore.sql ..
```

```
1..6
```

```
ok 1 - op ->
```

```
ok 2 - op ?
```

```
ok 3 - op @>
```

```
ok 4 - op <@
```

```
not ok 5 - akeys()
```

```
# Failed test 5: "akeys()"
```

```
#           have: {a,b}
```

```
#           want: {a,c}
```

```
ok 6 - avals()
```

```
# Looks like you failed 1 test of 6
```

```
Failed 1/6 subtests
```

There it is

What is() It?

```
SELECT plan(6);
```

```
SELECT is( 'a=>x, b=>y'::hstore -> 'a', 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT is( akeys('a=>1,b=>2'), '{a,c}', 'akeys()' );
```

```
SELECT is( avals('a=>1,b=>2'), '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


What is() It?

```
SELECT plan(6);
```

```
SELECT is( 'a=>x, b=>y'::hstore -> 'a', 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT is( akeys('a=>1,b=>2'), '{a,c}', 'akeys()' );
```

```
SELECT is( avals('a=>1,b=>2'), '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


What is() It?

```
SELECT plan(6);
```

```
SELECT is( 'a=>x, b=>y'::hstore -> 'a', 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT is( akeys('a=>1,b=>2'), '{a,c}', 'akeys()' );
```

```
SELECT is( avals('a=>1,b=>2'), '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


What is() It?

```
SELECT plan(6);
```

```
SELECT is( 'a=>x, b=>y'::hstore -> 'a', 'x', 'op ->' );
```

```
SELECT ok( 'a=>1'::hstore ? 'a', 'op ?' );
```

```
SELECT ok( 'a=>b, b=>1'::hstore @> 'b=>1', 'op @>' );
```

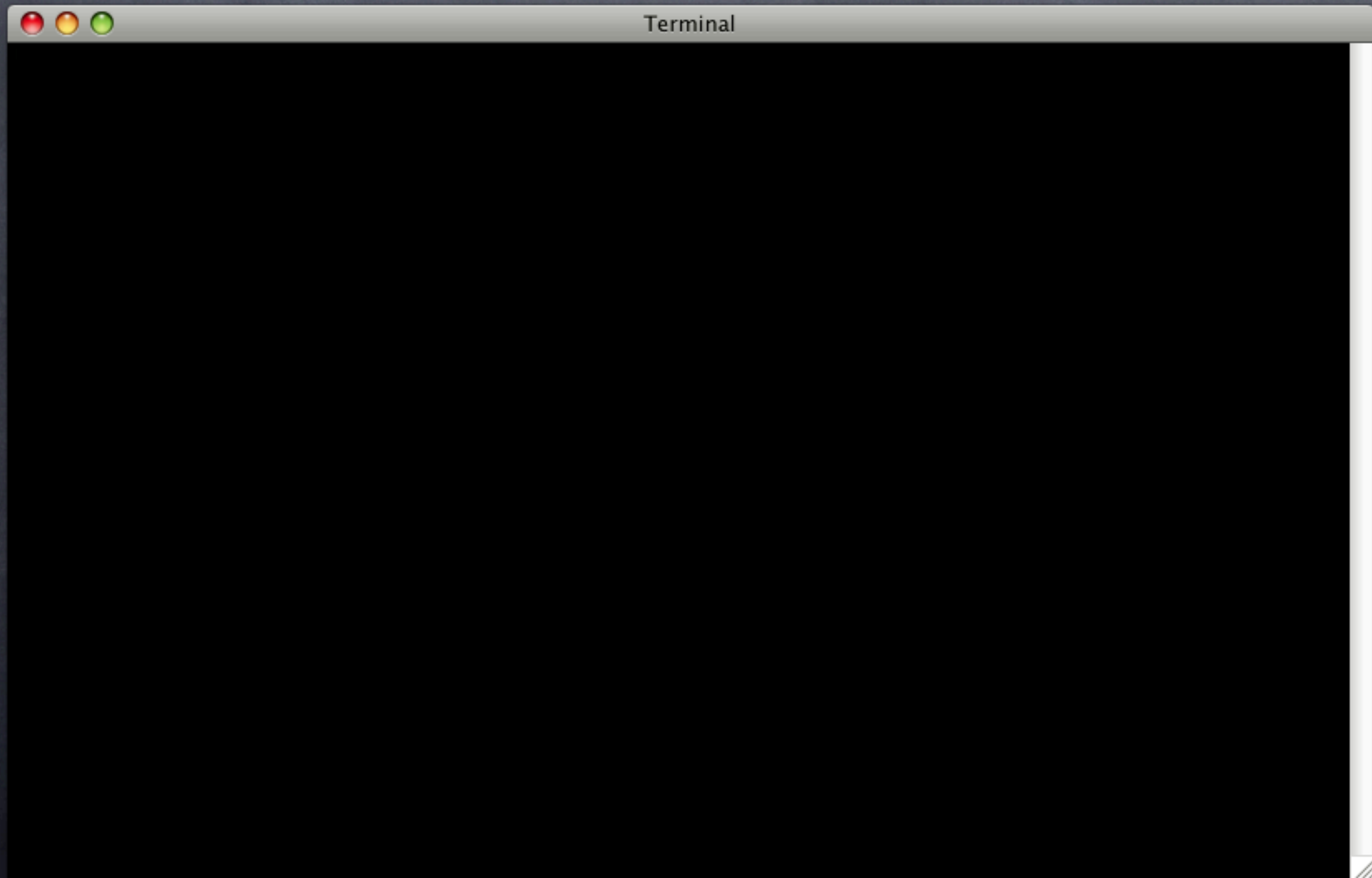
```
SELECT ok( NOT 'a=>c'::hstore <@ 'a=>b, b=>1', 'op <@' );
```

```
SELECT is( akeys('a=>1,b=>2'), '{a,b}', 'akeys()' );
```

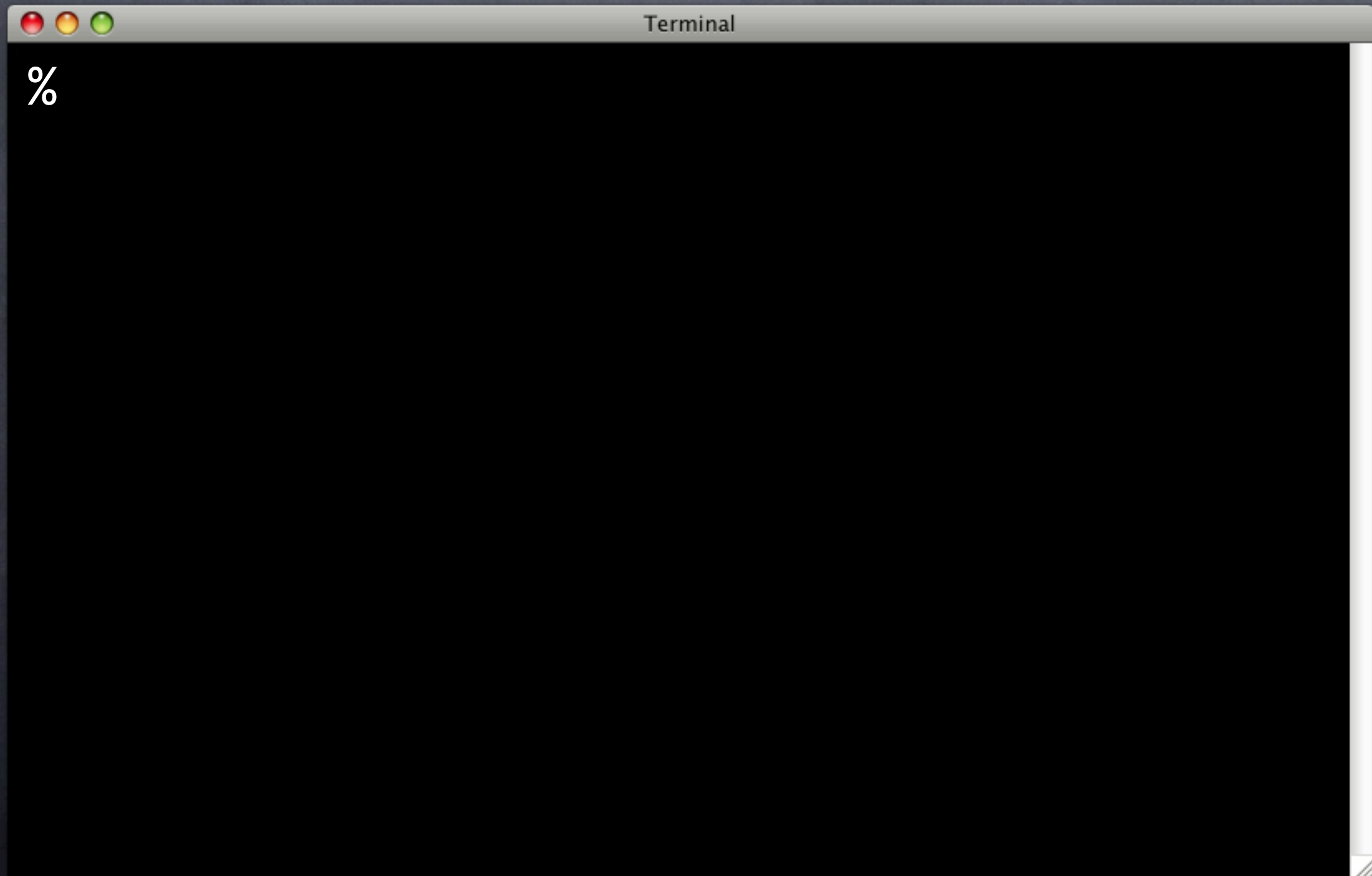
```
SELECT is( avals('a=>1,b=>2'), '{1,2}', 'avals()' );
```

```
SELECT * FROM finish();
```


What is() It?



What is() It?



What is() It?

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..6
ok 1 - op ->
ok 2 - op ?
ok 3 - op @>
ok 4 - op <@
ok 5 - akeys()
ok 6 - avals()
ok
All tests successful.
```


What is() It?

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..6
ok 1 - op ->
ok 2 - op ?
ok 3 - op @>
ok 4 - op <@
ok 5 - akeys()
ok 6 - avals()
ok
All tests successful.
```

Much better!

Repetition

Repetition

- **Want to test lots of values**

Repetition

- **Want to test lots of values**
- **Try to trick code with edge cases**

Repetition

- **Want to test lots of values**
- **Try to trick code with edge cases**
- **Writing same tests with different values**

Repetition

- Want to test lots of values
- Try to trick code with edge cases
- Writing same tests with different values
- Could get old fast

Repetition

- Want to test lots of values
- Try to trick code with edge cases
- Writing same tests with different values
- Could get old fast
- Use a table for repetitive tests



--:-- try.sql All (SQL[ansi])-----


```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
Emacs
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,      -- val -> arrow_rop
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,      -- val -> arrow_rop
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,      -- val -> arrow_rop
  arrow_val text,      --      = arrow_val
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,      -- val ? qmark_rop
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[], -- akeys(val) = akeys_val
  avals_val text[]
);
RESET client_min_messages;
```



```
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[]
);
RESET client_min_messages;
```



```
Emacs
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
  val hstore,
  arrow_rop text,
  arrow_val text,
  qmark_rop text,
  akeys_val text[],
  avals_val text[] -- avals(val) = avals_val
);
RESET client_min_messages;
```



```
Emacs
SET client_min_messages = warning;
CREATE TEMPORARY TABLE hs_tests (
    val hstore,
    arrow_rop text,
    arrow_val text,
    qmark_rop text,
    akeys_val text[],
    avals_val text[]
);
RESET client_min_messages;

-- Insert test values.
INSERT INTO hs_tests VALUES
('a=>1,b=>2', 'a', '1', 'a', '{a,b}', '{1,2}'),
('a=>1', 'a', '1', 'a', '{a}', '{1}')
;
```




--:-- try.sql All (SQL[ansi])-----


```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;
```

```
SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;
```

```
SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;
```

```
SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;
```

```
SELECT is(  
    val -> arrow_rop,  
    arrow_val,  
    quote_literal(val) || ' ? ' || quote_literal(arrow_val)  
) FROM hs_tests;
```

```
SELECT ok(  
    val ? qmark_rop,  
    quote_literal(val) || ' -> ' || quote_literal(qmark_rop)  
) FROM hs_tests;
```

```
SELECT is(  
    akeys(val), akeys_val,  
    'akeys(' || quote_literal(val) || ')'  
) FROM hs_tests;
```

```
SELECT is(  
    avals(val), avals_val,  
    'avals(' || quote_literal(val) || ')'  
) FROM hs_tests;
```

```
SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;
```

```
SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;
```

```
SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;
```

```
SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;
```

```
SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;
```

```
SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;
```

```
SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;
```

```
SELECT * FROM finish();
```



```
SELECT plan(COUNT(*)::int * 4) FROM hs_tests;

SELECT is(
  val -> arrow_rop,
  arrow_val,
  quote_literal(val) || ' ? ' || quote_literal(arrow_val)
) FROM hs_tests;

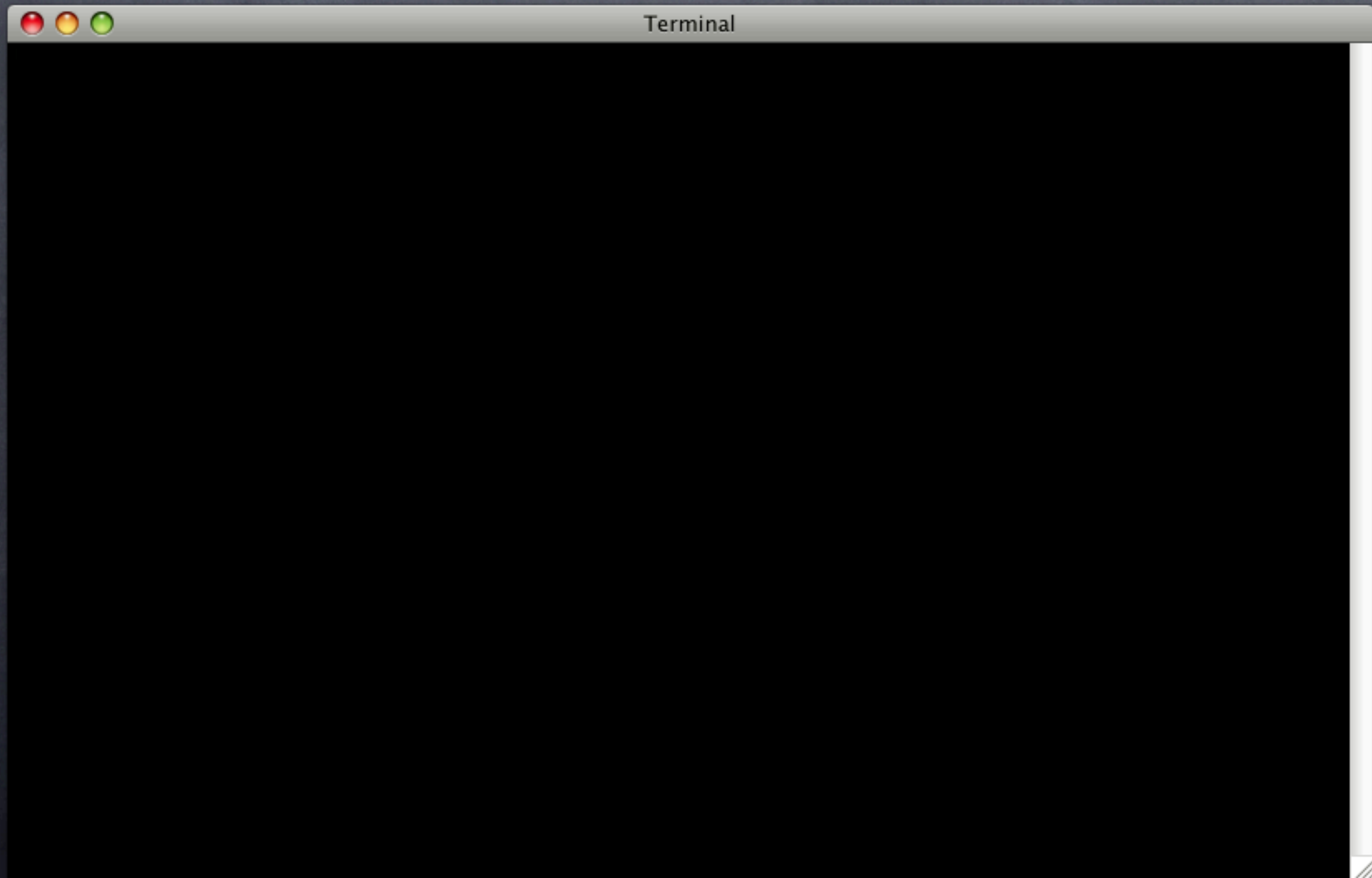
SELECT ok(
  val ? qmark_rop,
  quote_literal(val) || ' -> ' || quote_literal(qmark_rop)
) FROM hs_tests;

SELECT is(
  akeys(val), akeys_val,
  'akeys(' || quote_literal(val) || ')'
) FROM hs_tests;

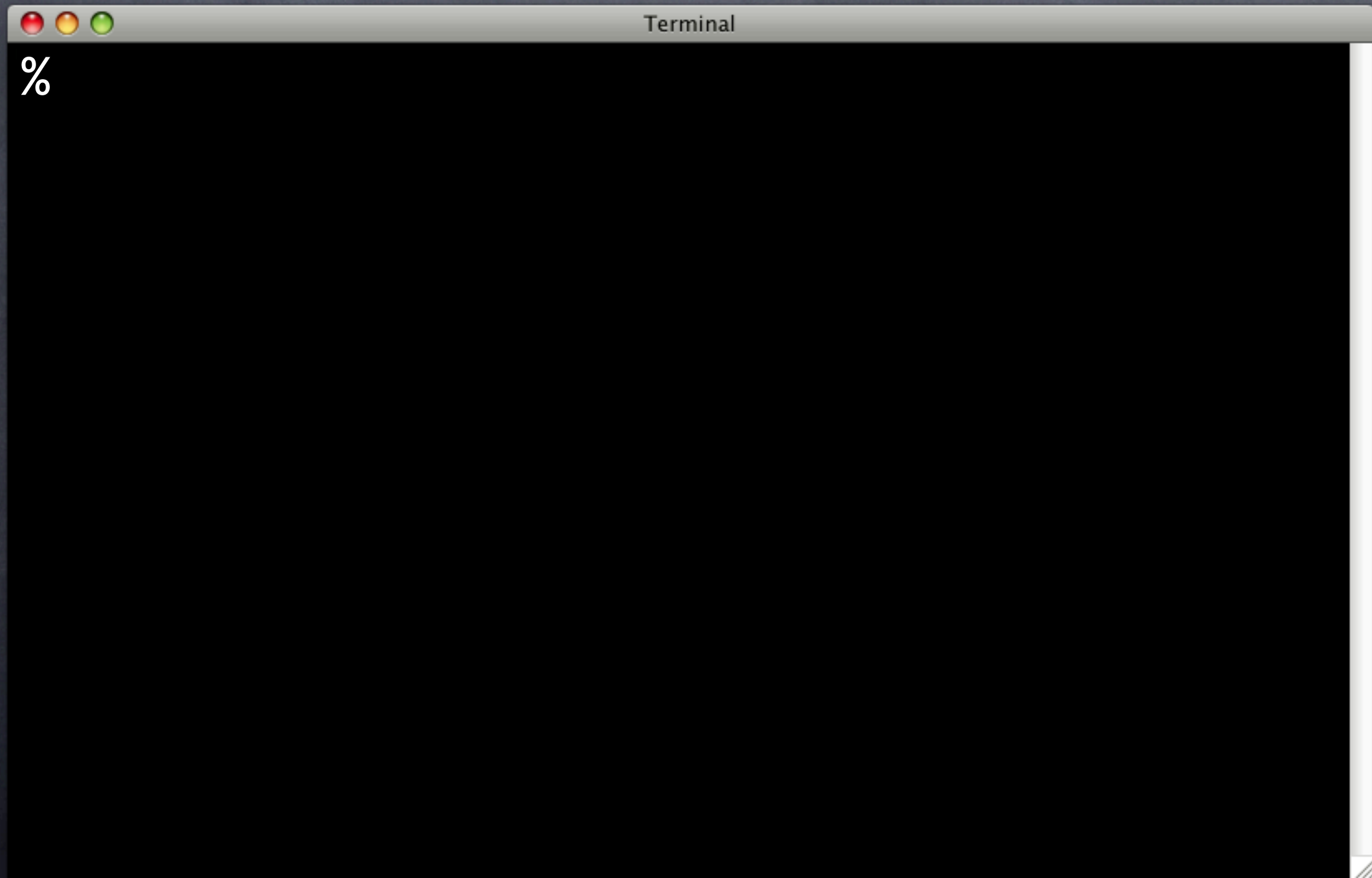
SELECT is(
  avals(val), avals_val,
  'avals(' || quote_literal(val) || ')'
) FROM hs_tests;

SELECT * FROM finish();
```


Repetition



Repetition



Repetition

```
Terminal
% pg_prove -v -d try hstore.sql
hstore.sql ..
1..8
ok 1 - '"a"=>"1", "b"=>"2"' ? '1'
ok 2 - '"a"=>"1"' ? '1'
ok 3 - '"a"=>"1", "b"=>"2"' -> 'a'
ok 4 - '"a"=>"1"' -> 'a'
ok 5 - akeys( '"a"=>"1", "b"=>"2"' )
ok 6 - akeys( '"a"=>"1"' )
ok 7 - avals( '"a"=>"1", "b"=>"2"' )
ok 8 - avals( '"a"=>"1"' )
ok
All tests successful.
```


Add Tests

```
Emacs
-- Insert test values.
INSERT INTO hs_tests VALUES
('a=>1,b=>2', 'a', '1', 'a', '{a,b}', '{1,2}'),
('a=>1', 'a', '1', 'a', '{a}', '{1}')
```

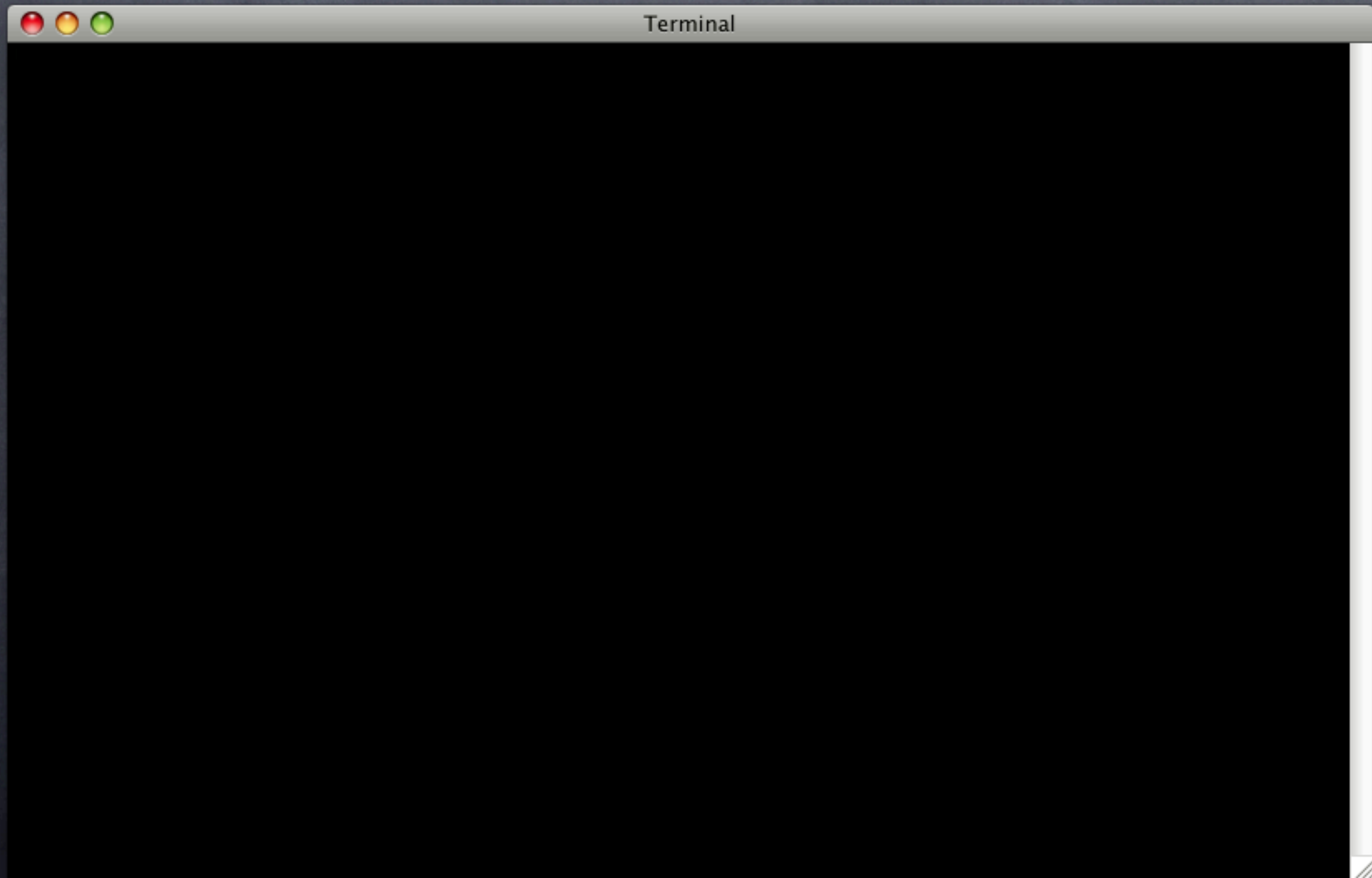
try.sql All (SQL[ansi])

Add Tests

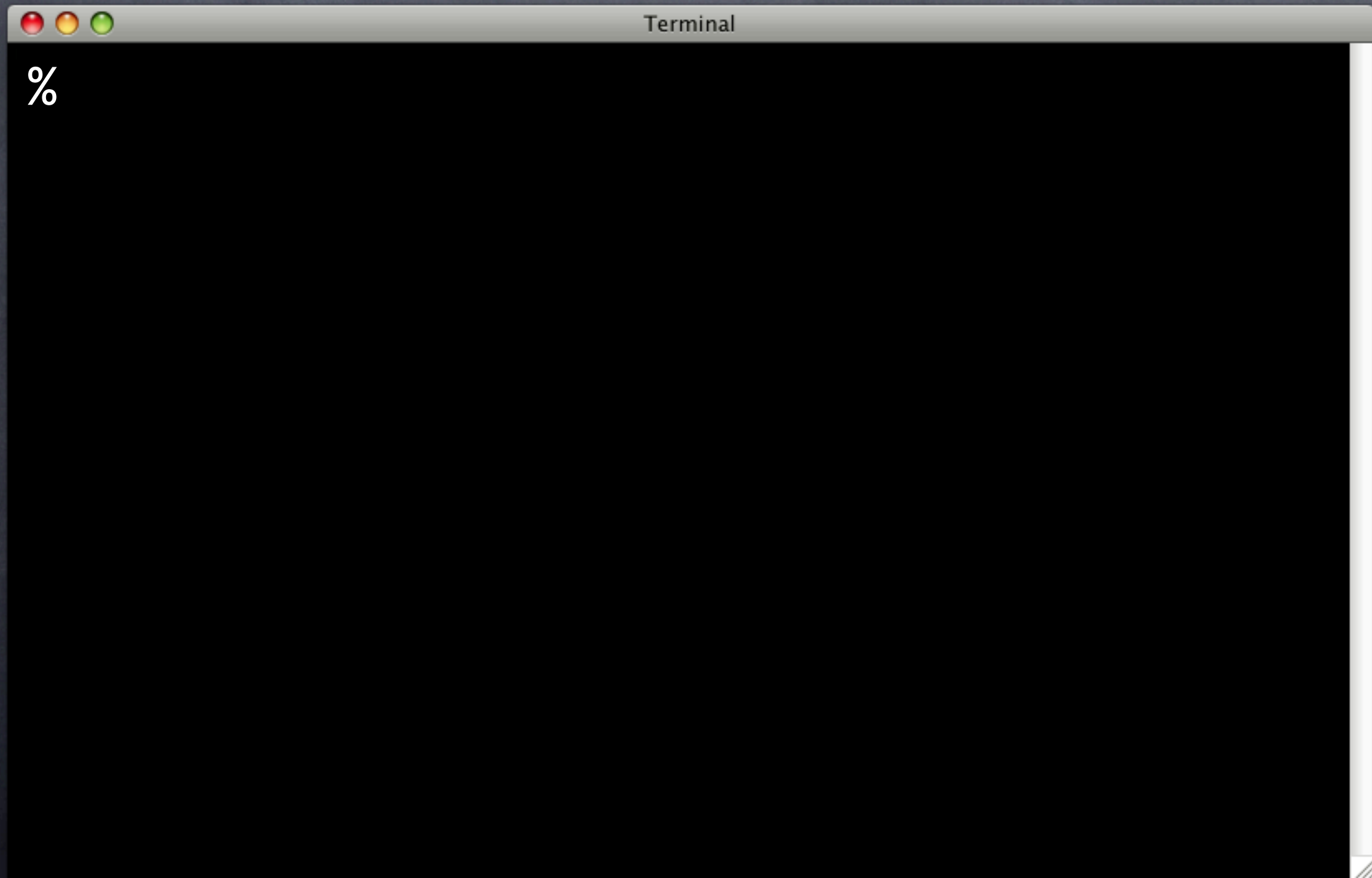
```
Emacs
-- Insert test values.
INSERT INTO hs_tests VALUES
('a=>1,b=>2', 'a', '1', 'a', '{a,b}', '{1,2}'),
('a=>1', 'a', '1', 'a', '{a}', '{1}'),
('foo=>bar', 'foo', 'bar', 'foo', '{foo}', '{bar}'),
('1=>a,2=>b', '1', 'a', '1', '{1,2}', '{a,b}'),
(
    'name=>"Tom Lane",rank=>"H@xi3",sn=>ou812',
    'rank', 'H@xi3',
    'name',
    '{sn,name,rank}',
    '{ou812,"Tom Lane",H@xi3}'
);

--- try.sql All (SQL[ansi])---
```


More Repetition



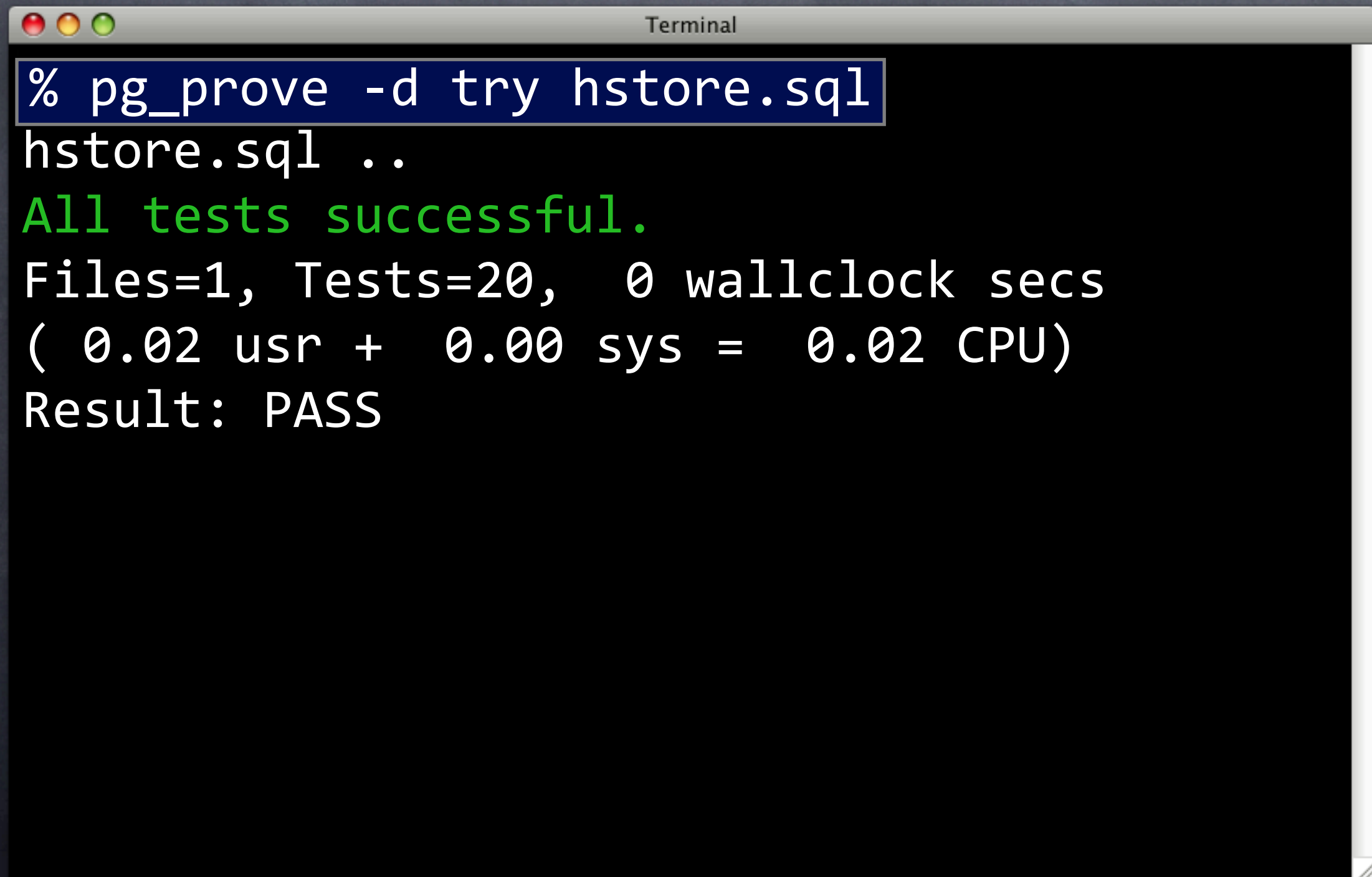
More Repetition



More Repetition

```
Terminal
% pg_prove -d try hstore.sql
hstore.sql ..
All tests successful.
Files=1, Tests=20,  0 wallclock secs
( 0.02 usr +  0.00 sys =  0.02 CPU)
Result: PASS
```


More Repetition



```
Terminal
% pg_prove -d try hstore.sql
hstore.sql ..
All tests successful.
Files=1, Tests=20, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```


More Repetition

```
Terminal
% pg_prove -d try hstore.sql
hstore.sql ..
All tests successful.
Files=1, Tests=20, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```


Skipping Tests

Skipping Tests

- ◉ Sometimes need to skip tests

Skipping Tests

- Sometimes need to skip tests
 - Platform dependencies

Skipping Tests

- Sometimes need to skip tests
 - Platform dependencies
 - Collation dependencies

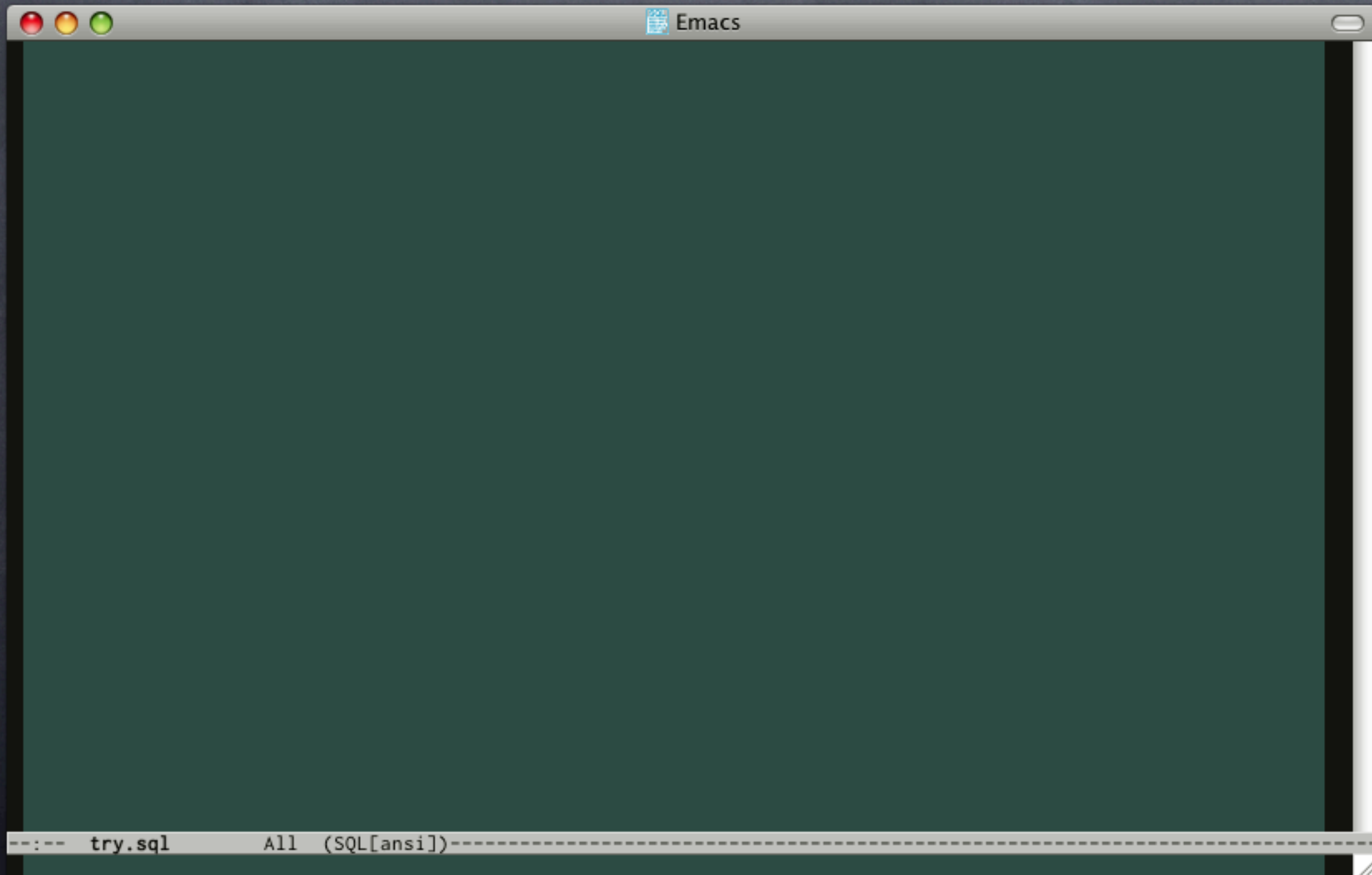
Skipping Tests

- ◉ Sometimes need to skip tests
 - ◉ Platform dependencies
 - ◉ Collation dependencies
 - ◉ Version dependencies

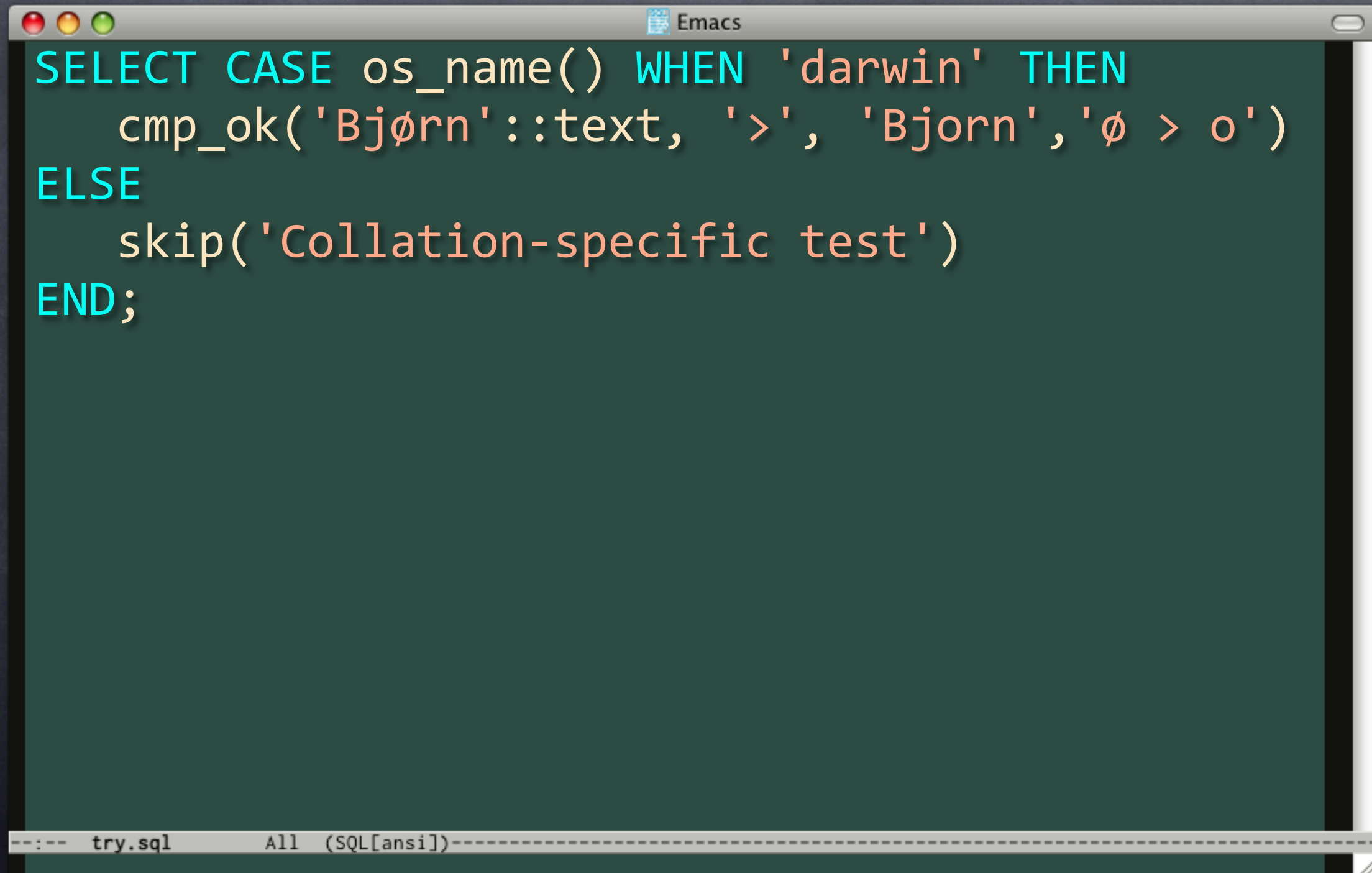
Skipping Tests

- Sometimes need to skip tests
 - Platform dependencies
 - Collation dependencies
 - Version dependencies
- Use `skip()`

Skipping Tests



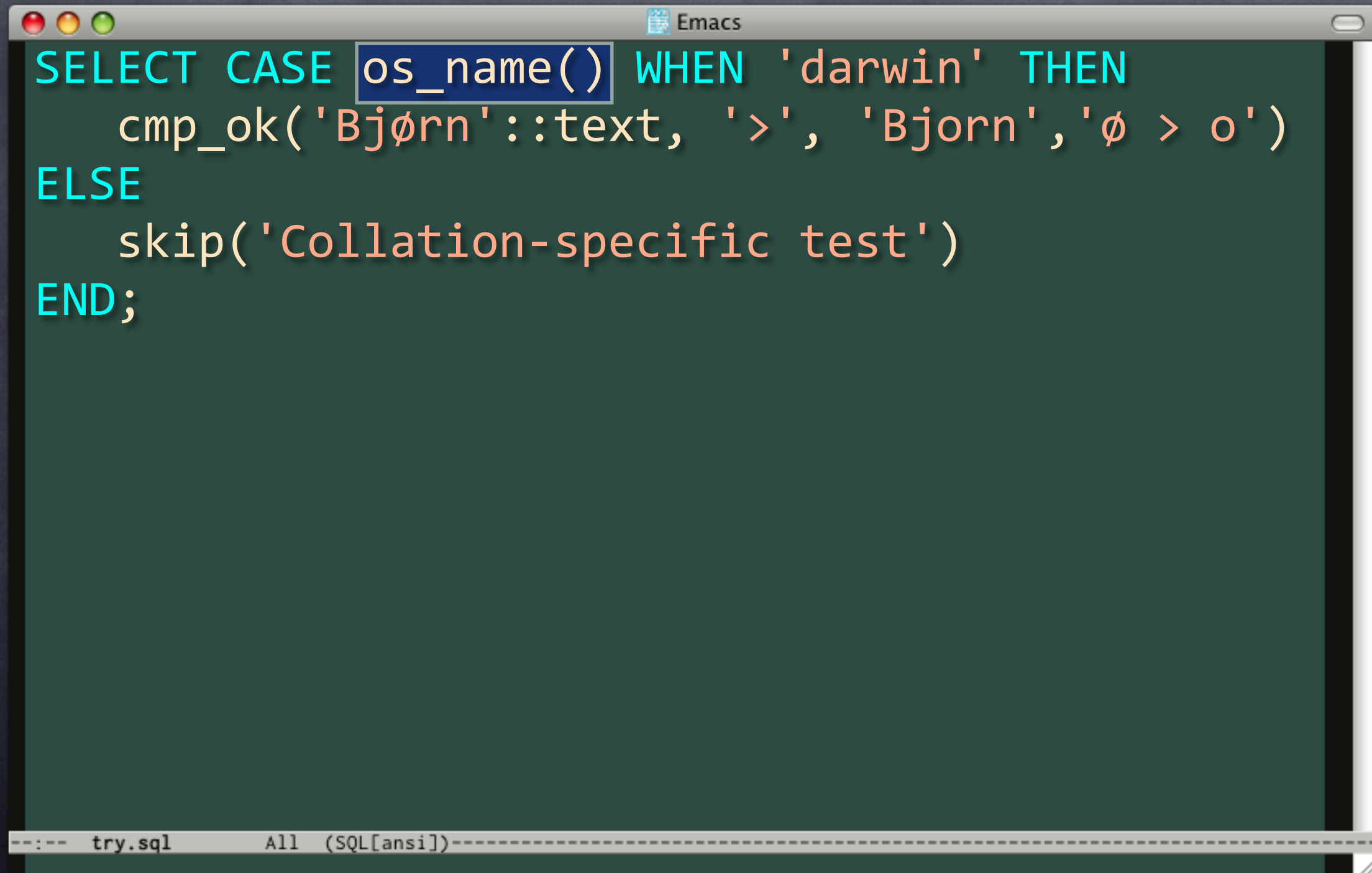
Skipping Tests



```
SELECT CASE os_name() WHEN 'darwin' THEN
    cmp_ok('Bjørn'::text, '>', 'Bjorn', 'ø > o')
ELSE
    skip('Collation-specific test')
END;
```

try.sql All (SQL[ansi])

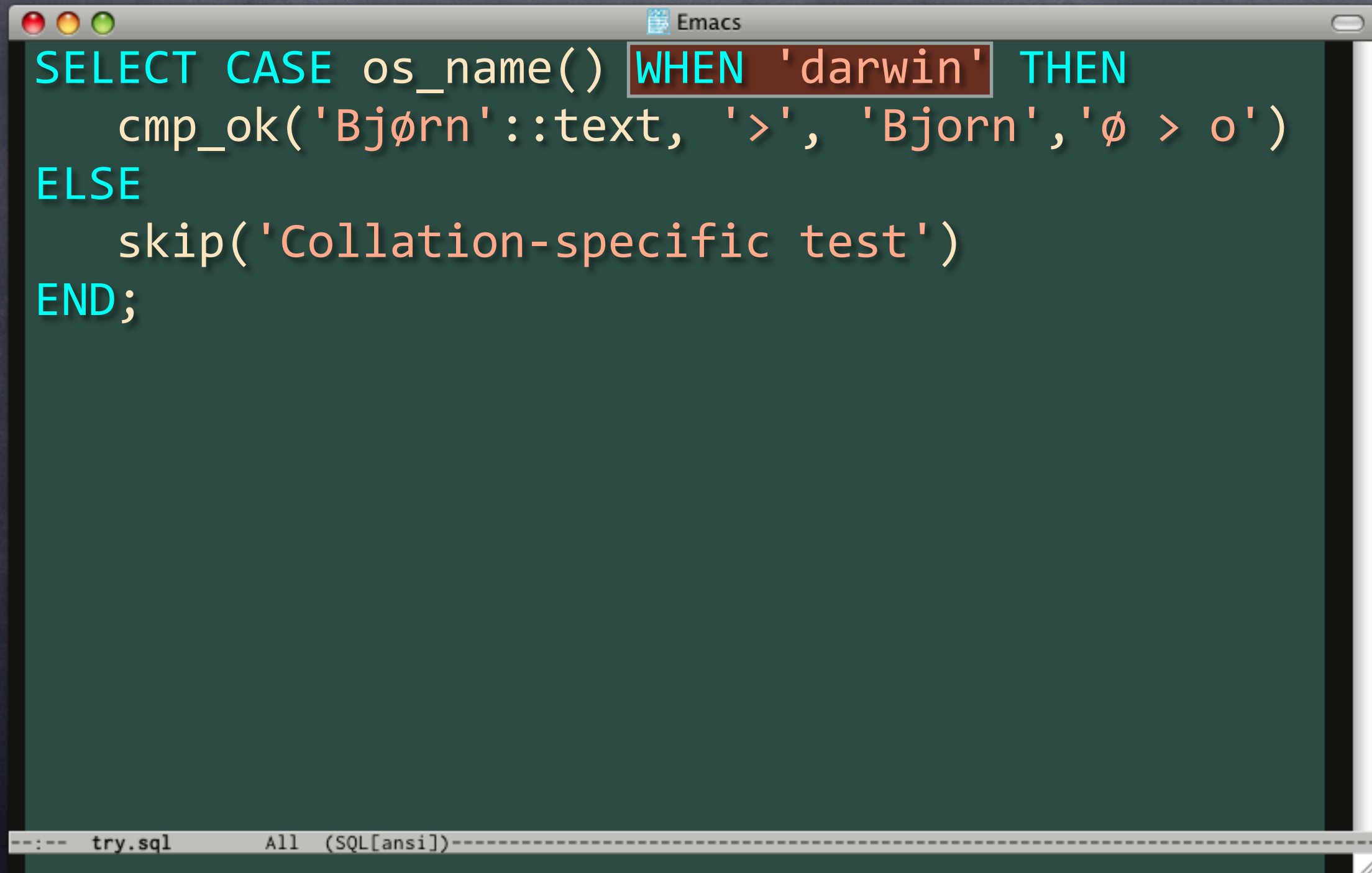
Skipping Tests



```
SELECT CASE os_name() WHEN 'darwin' THEN
    cmp_ok('Bjørn'::text, '>', 'Bjorn', 'ø > o')
ELSE
    skip('Collation-specific test')
END;
```

try.sql All (SQL[ansi])

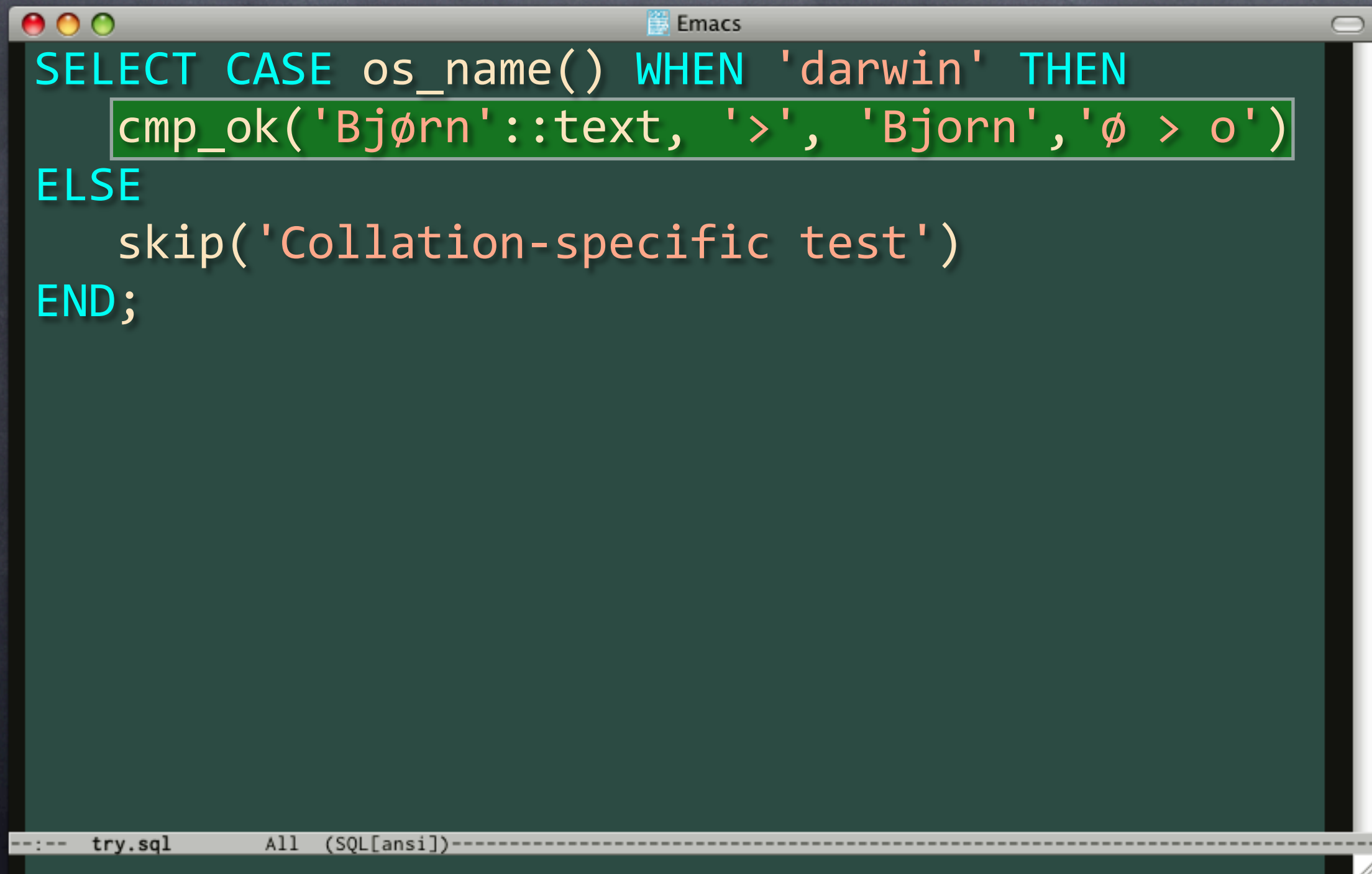
Skipping Tests



```
SELECT CASE os_name() WHEN 'darwin' THEN
    cmp_ok('Bjørn'::text, '>', 'Bjorn', 'ø > o')
ELSE
    skip('Collation-specific test')
END;
```

try.sql All (SQL[ansi])

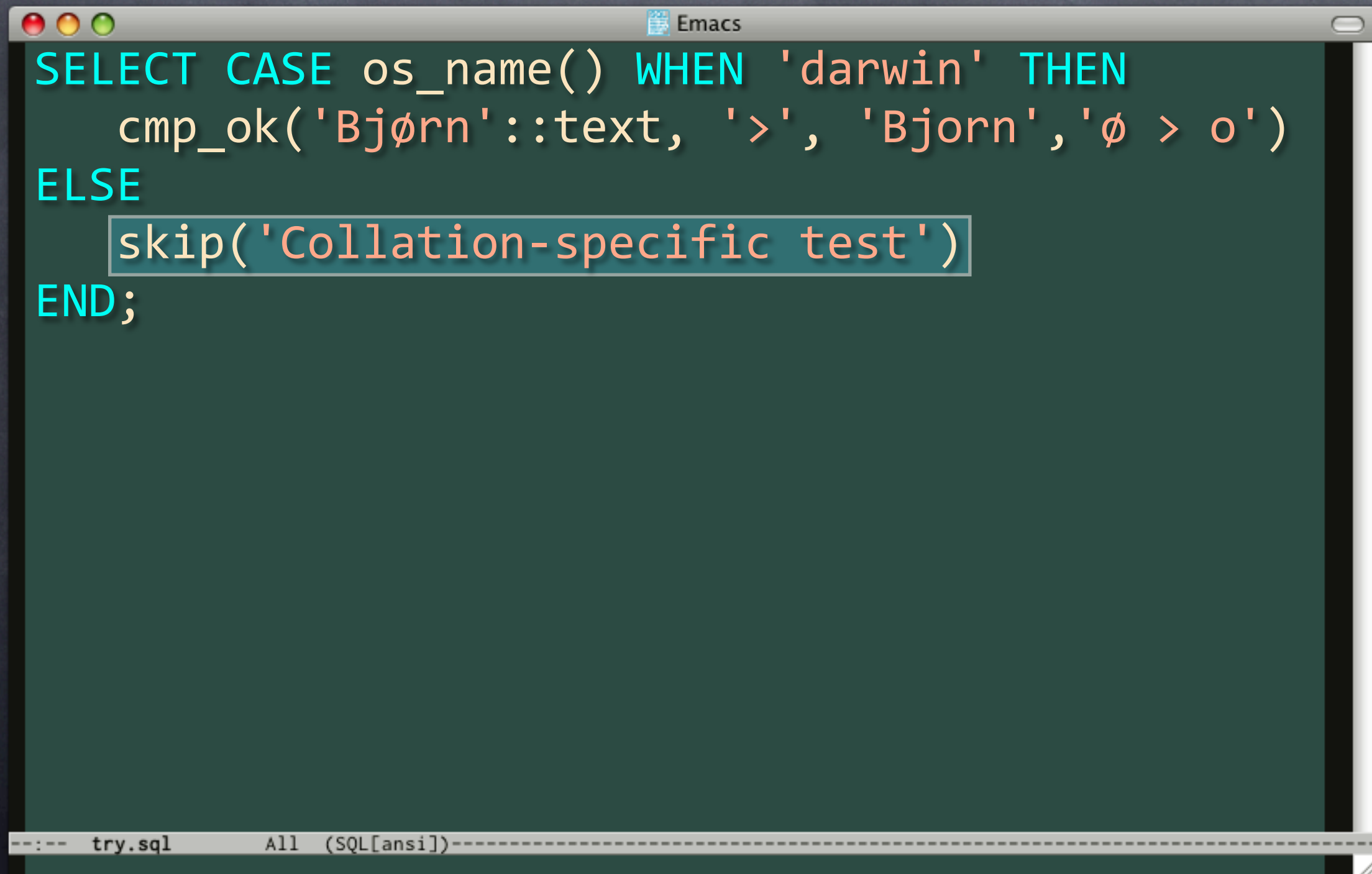
Skipping Tests



```
SELECT CASE os_name() WHEN 'darwin' THEN
    cmp_ok('Bjørn'::text, '>', 'Bjorn', 'ø > o')
ELSE
    skip('Collation-specific test')
END;
```

The screenshot shows an Emacs window with a dark green background. The code is displayed in a monospaced font with syntax highlighting: 'SELECT', 'CASE', 'WHEN', 'THEN', 'ELSE', and 'END;' are in cyan; 'os_name()', 'darwin', 'cmp_ok()', 'Bjørn', '>', 'Bjorn', 'ø > o', 'skip()', and 'Collation-specific test' are in orange. The line containing the `cmp_ok` function call is highlighted in green. The Emacs window title bar shows 'Emacs' and standard window control buttons. At the bottom, a status bar displays 'try.sql', 'All (SQL[ansi])', and a dashed line.

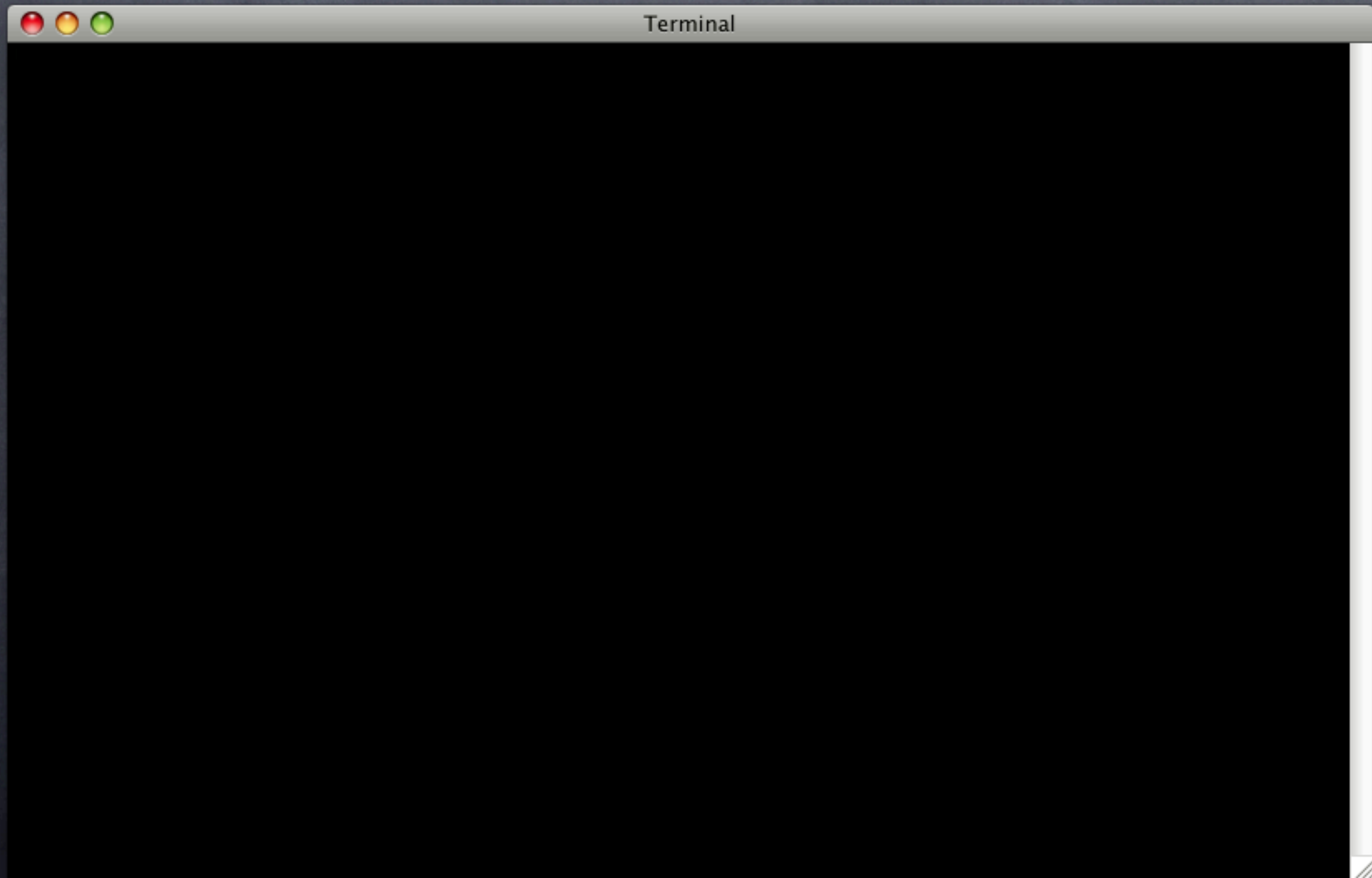
Skipping Tests



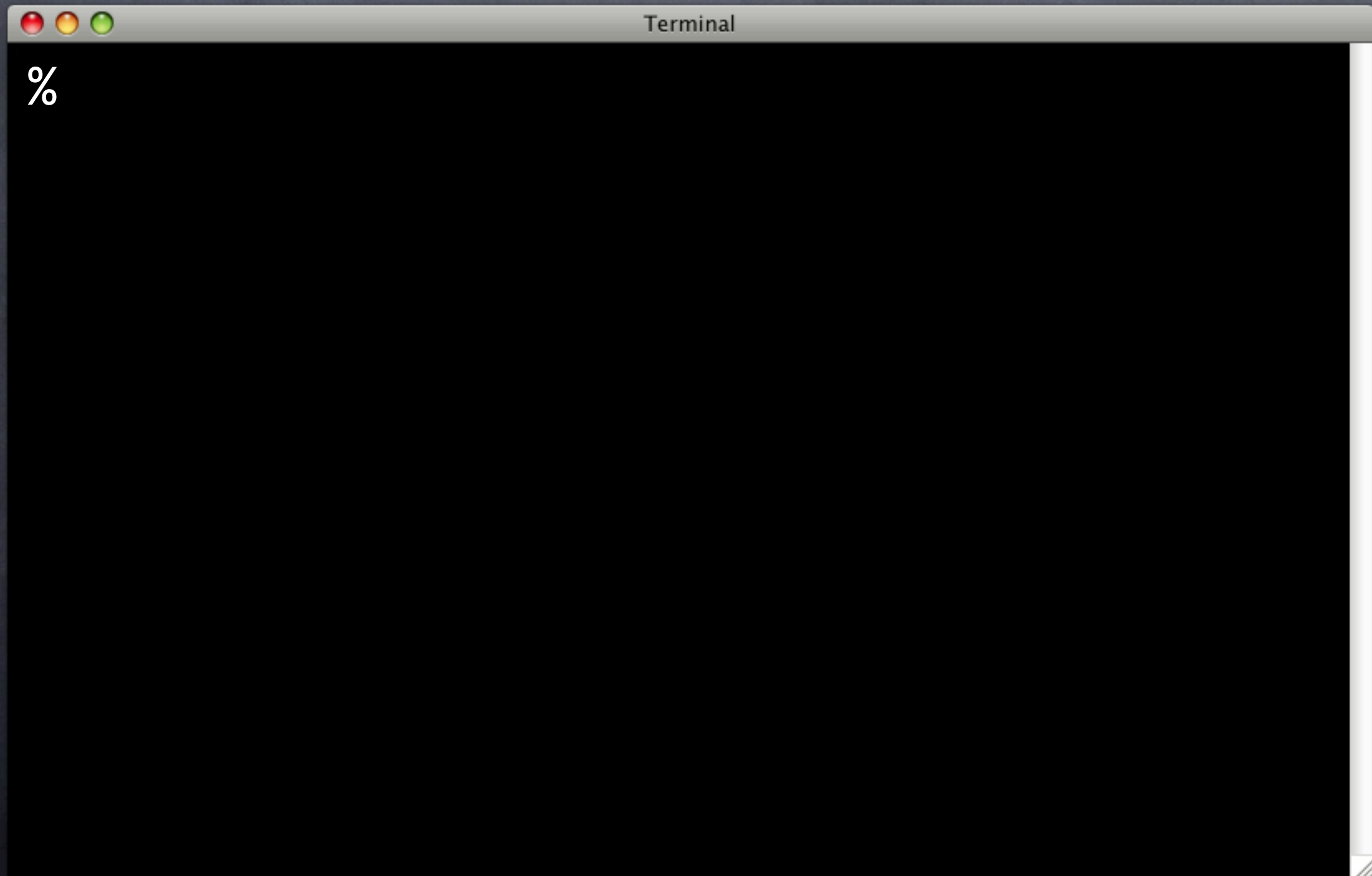
```
SELECT CASE os_name() WHEN 'darwin' THEN
  cmp_ok('Bjørn'::text, '>', 'Bjorn', 'ø > o')
ELSE
  skip('Collation-specific test')
END;
```

The screenshot shows an Emacs window with a dark green background. The code is displayed in a light blue/cyan font. The line `skip('Collation-specific test')` is highlighted with a light blue rectangular box. The Emacs window title bar shows the name 'Emacs' and standard window control buttons (red, yellow, green). At the bottom of the window, a status bar shows the file name 'try.sql', the cursor position 'All', and the mode '(SQL[ansi])'.

Skipping Tests



Skipping Tests



Skipping Tests

```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 -  $\emptyset > 0$ 
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.01 sys = 0.03 CPU)
Result: PASS
```


Skipping Tests

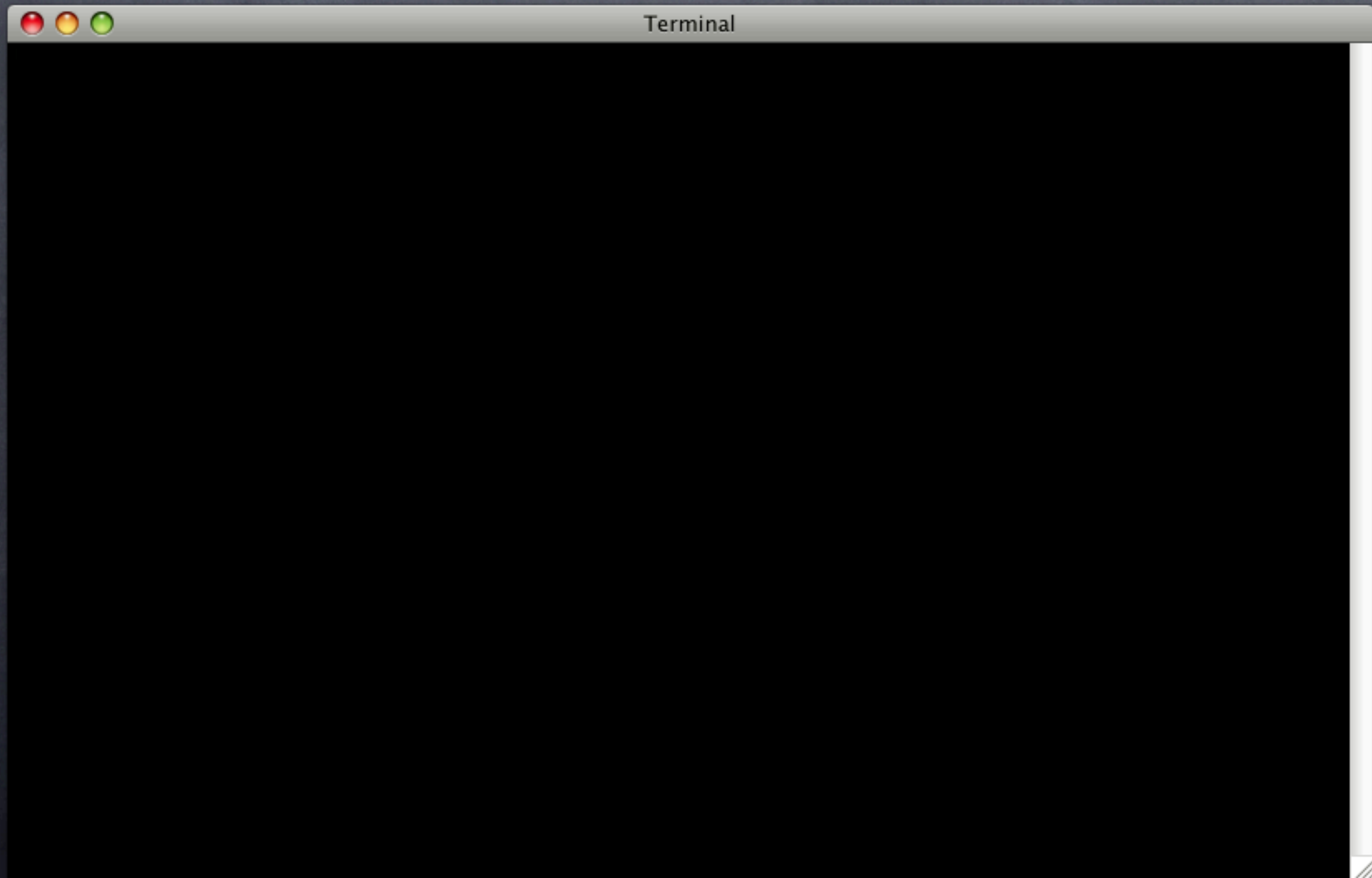
```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 -  $\emptyset$  > 0
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.01 sys = 0.03 CPU)
Result: PASS
```


Skipping Tests

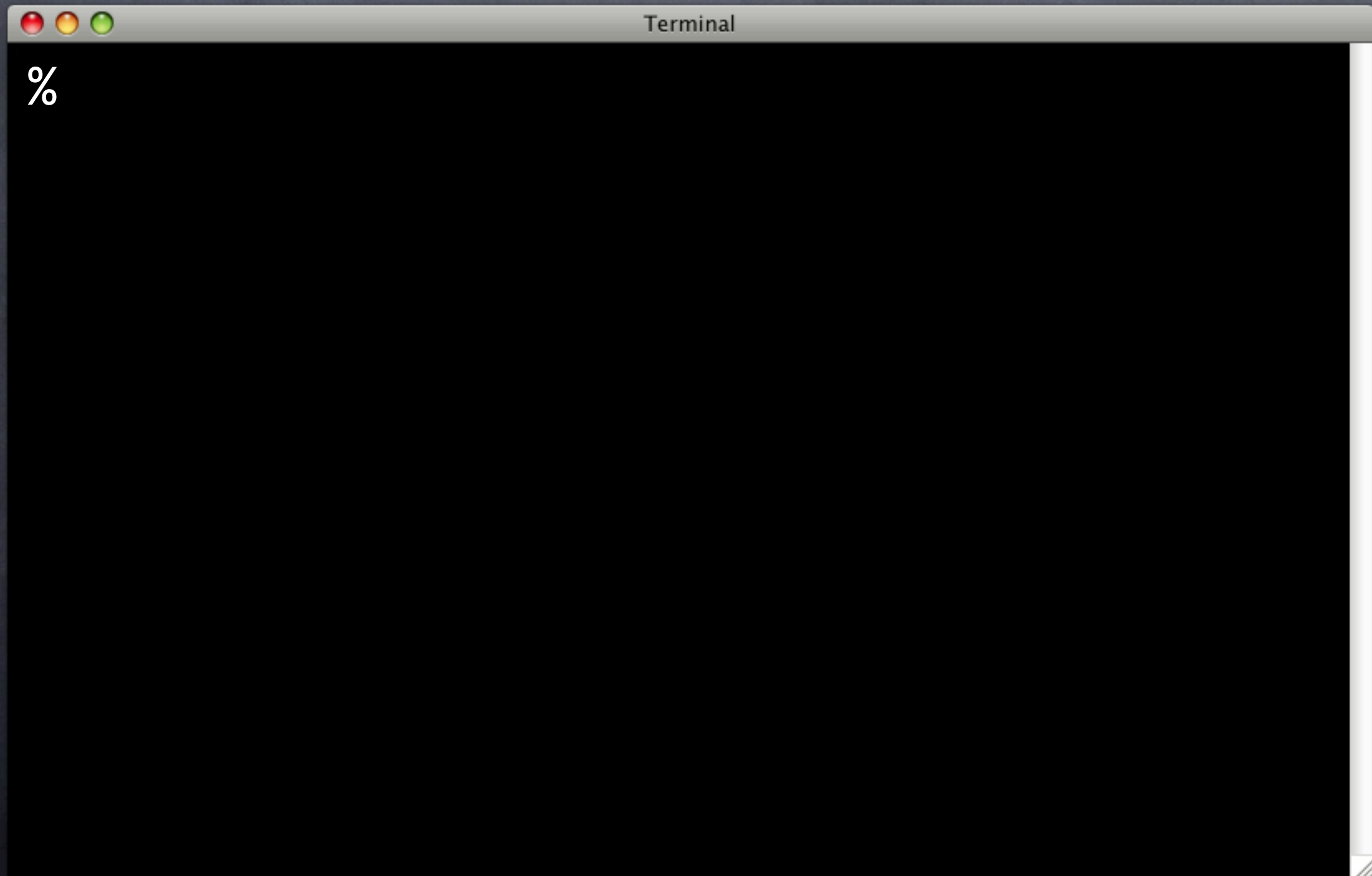
```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 -  $\emptyset$  > 0
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.01 sys = 0.03 CPU)
Result: PASS
```

✓ Darwin

Skipping Tests



Skipping Tests



Skipping Tests

```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 - SKIP: Collation-specific test
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```


Skipping Tests

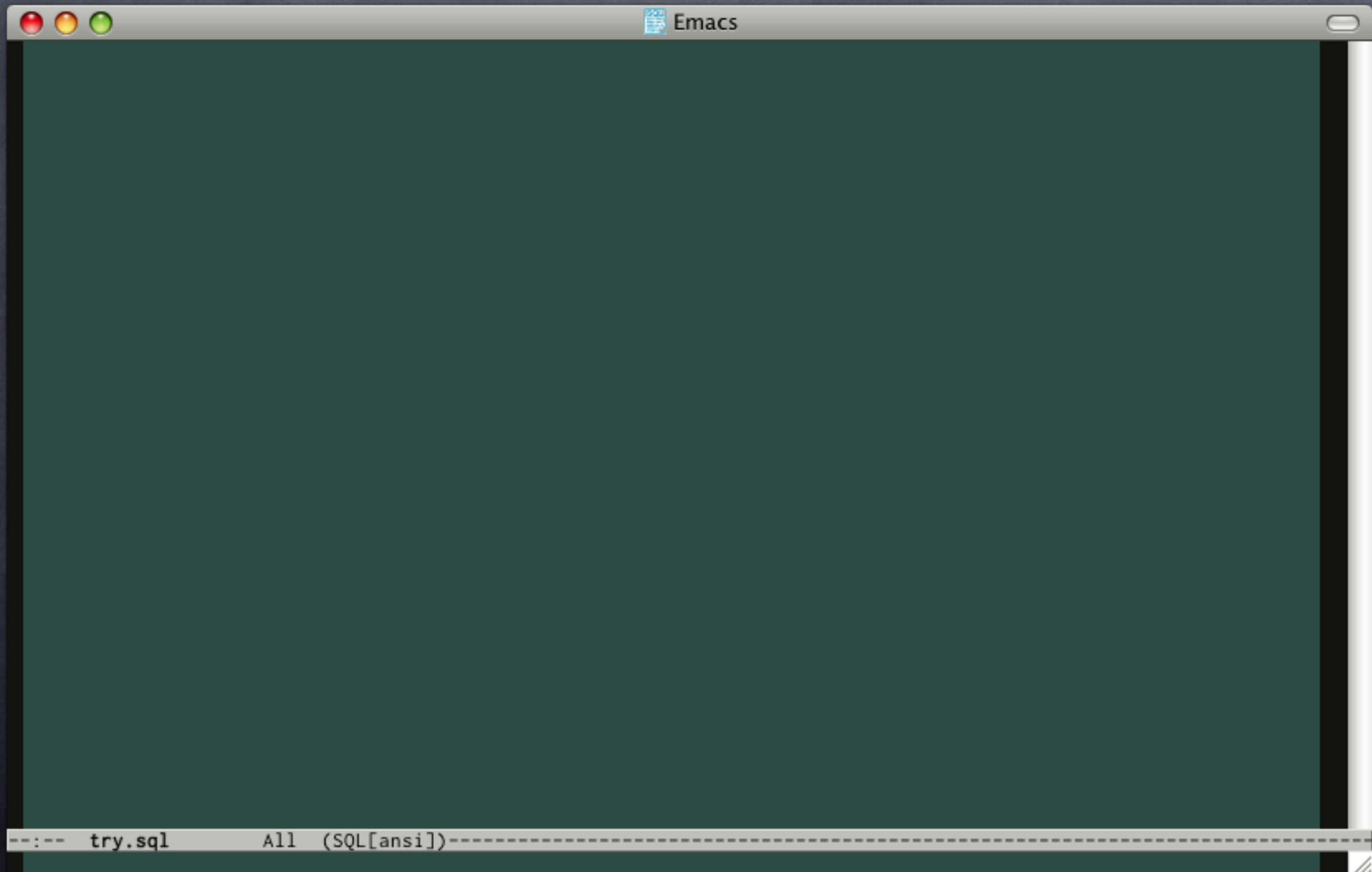
```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 - SKIP: Collation-specific test
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```


Skipping Tests

```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..1
ok 1 - SKIP: Collation-specific test
ok
All tests successful.
Files=1, Tests=1, 0 wallclock secs
( 0.02 usr + 0.00 sys = 0.02 CPU)
Result: PASS
```

! Darwin

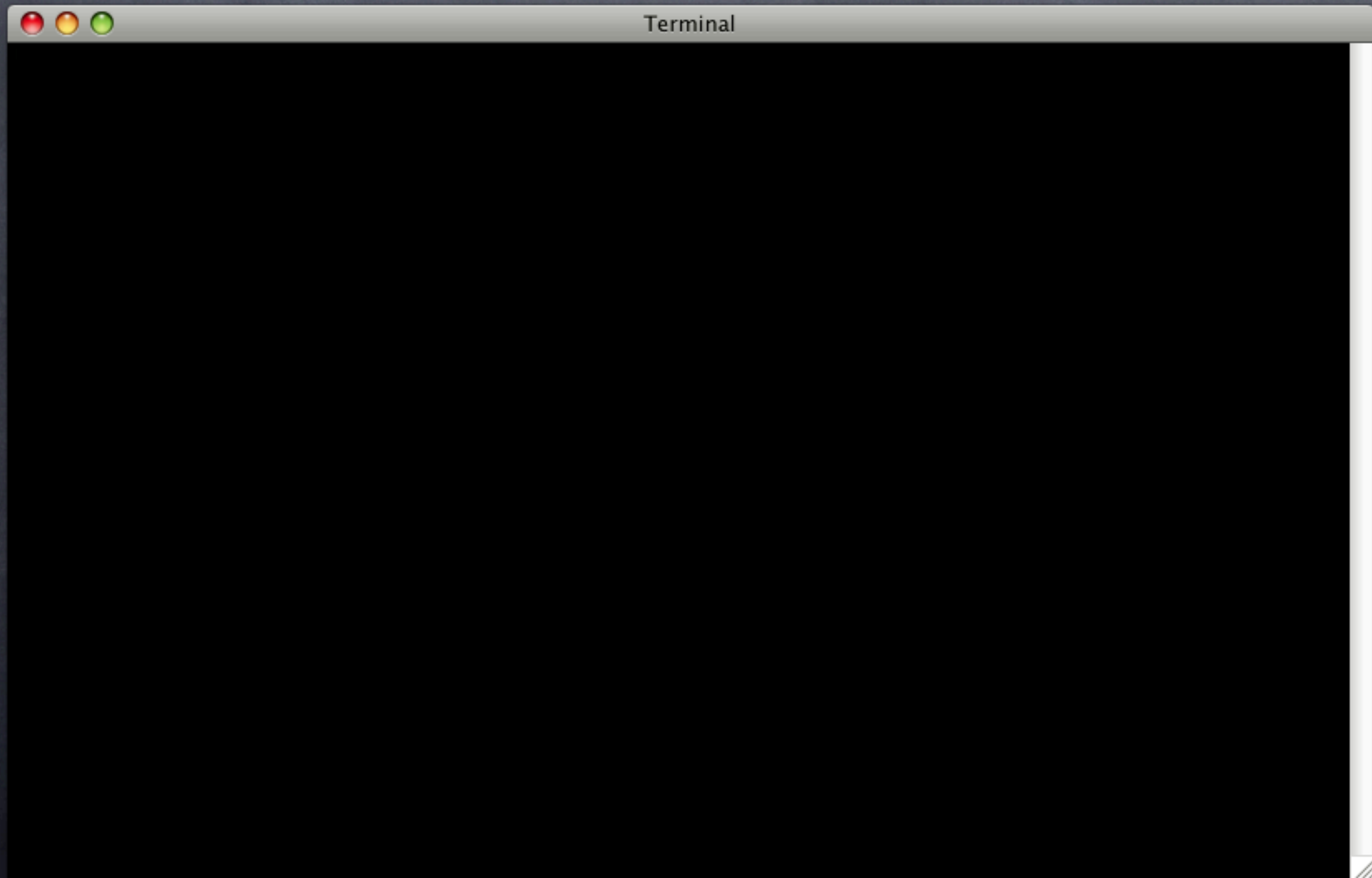
Skip More



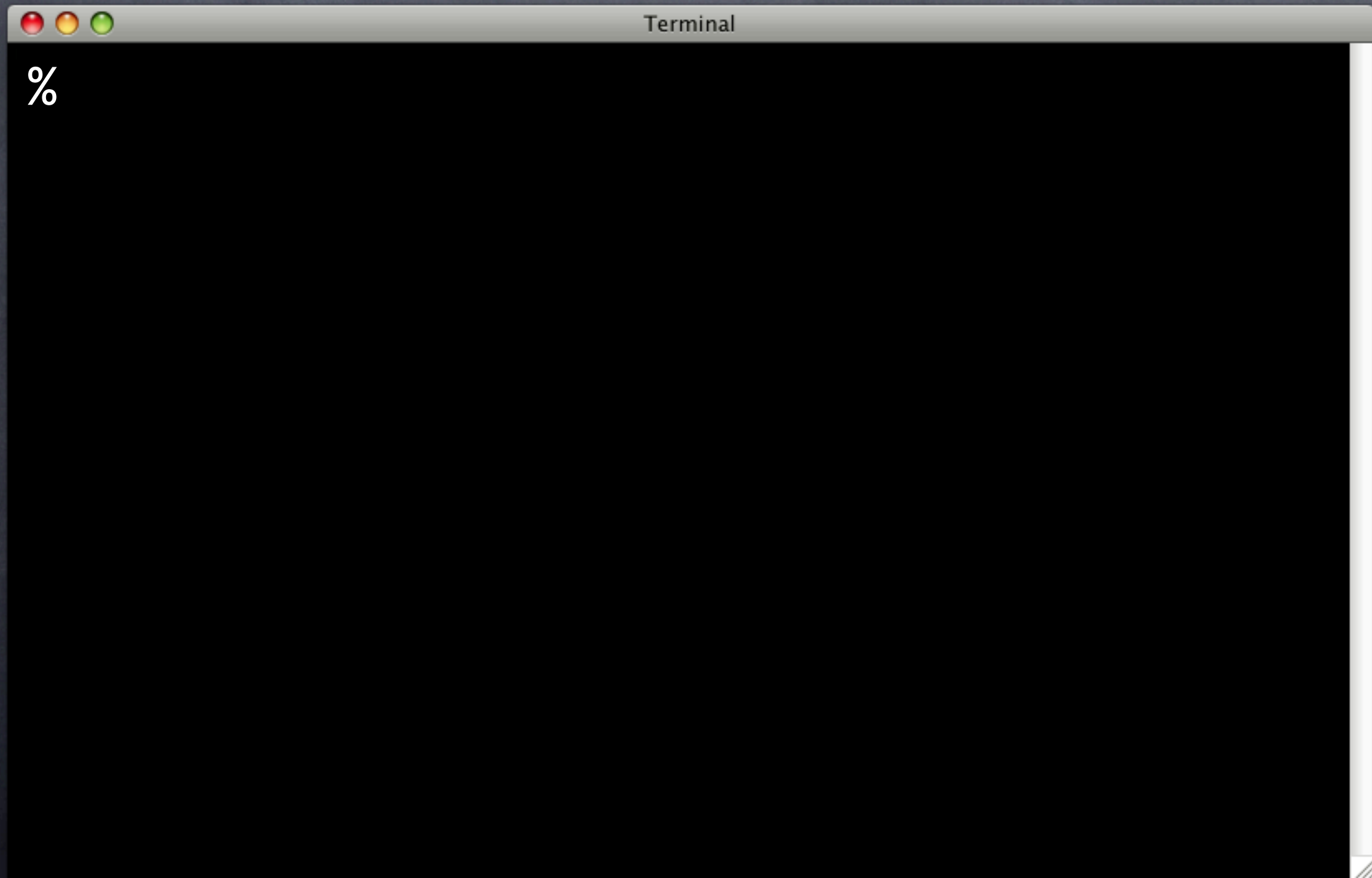
Skip More

```
SELECT CASE os_name() WHEN 'darwin' THEN
  array_to_string( ARRAY[
    cmp_ok( 'Bjørn'::text, '>', 'Bjorn', 'ø > o' ),
    cmp_ok( 'Pınar'::text, '>', 'Pinar', 'ı > i' ),
    cmp_ok( 'José'::text, '>', 'Jose', 'é > e' ),
    cmp_ok( 'Täp'::text, '>', 'Tap', 'ä > a' )
  ], E'\n' )
ELSE
  skip('Collation-specific test', 4)
END;
```


Skip More



Skip More



Skip More

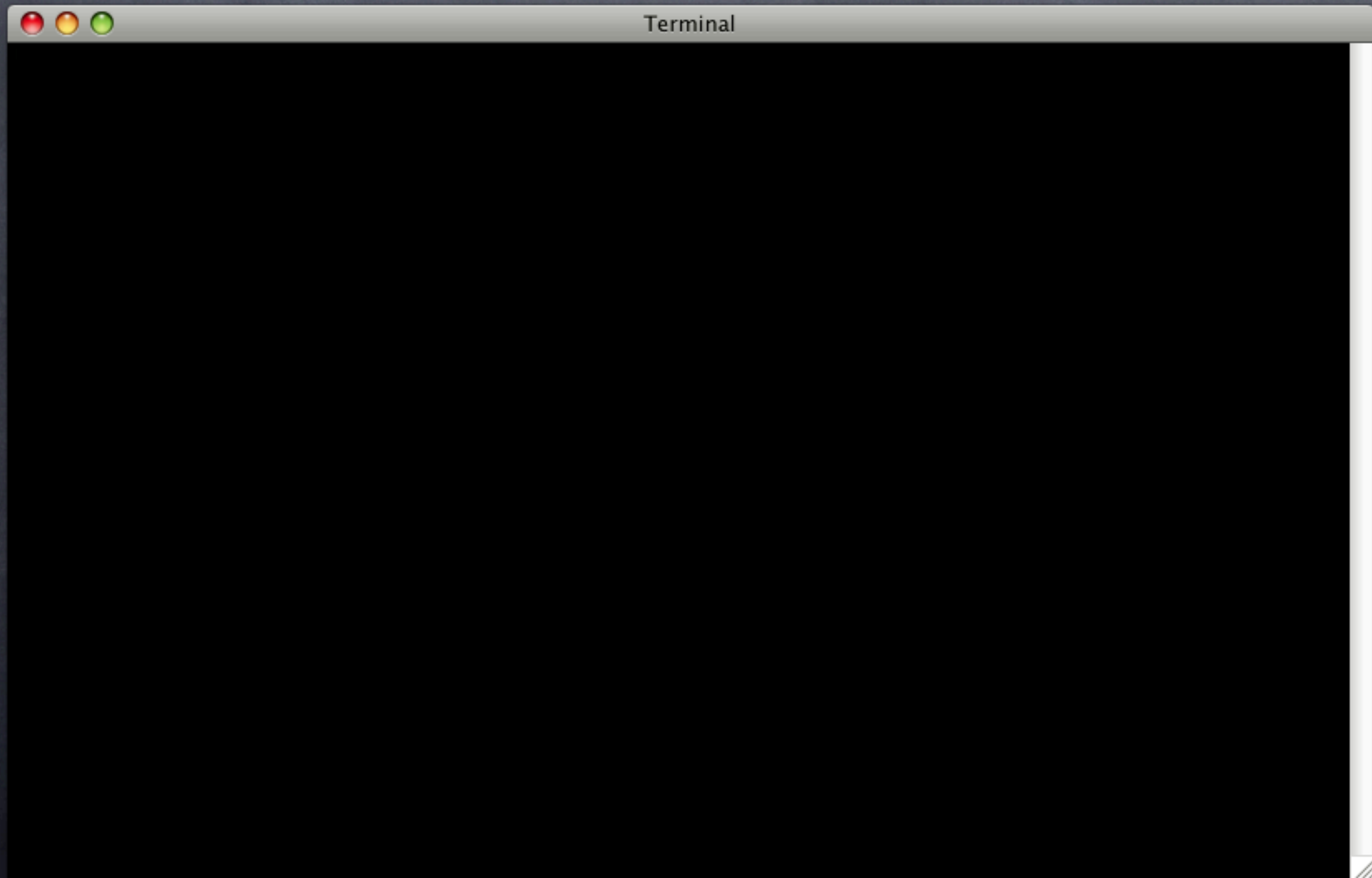
```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..4
ok 1 - ø > o
ok 2 - ı > i
ok 3 - é > e
ok 4 - ä > a
ok
All tests successful.
```


Skip More

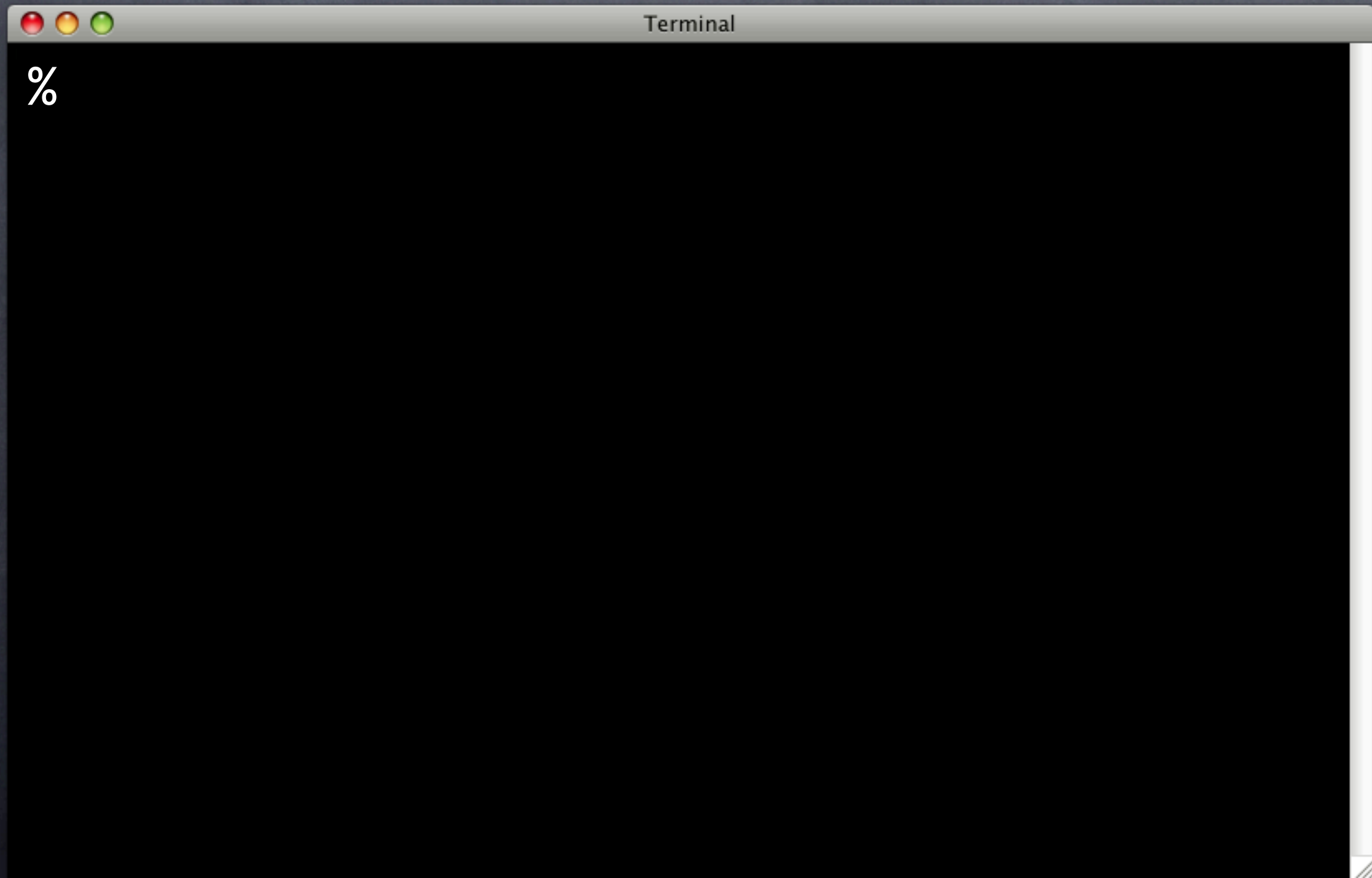
```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..4
ok 1 - ø > o
ok 2 - ı > i
ok 3 - é > e
ok 4 - ä > a
ok
All tests successful.
```

✓ Darwin

Skip More



Skip More



Skip More

```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..4
ok 1 - SKIP: Collation-specific test
ok 2 - SKIP: Collation-specific test
ok 3 - SKIP: Collation-specific test
ok 4 - SKIP: Collation-specific test
ok
All tests successful.
```


Skip More

```
Terminal
% pg_prove -v -d try collation.sql
collation.sql ..
1..4
ok 1 - SKIP: Collation-specific test
ok 2 - SKIP: Collation-specific test
ok 3 - SKIP: Collation-specific test
ok 4 - SKIP: Collation-specific test
ok
All tests successful.
```

! Darwin

Todo Tests

Todo Tests

- ◉ Sometimes need to ignore failures

Todo Tests

- Sometimes need to ignore failures
 - Features not implemented

Todo Tests

- Sometimes need to ignore failures
 - Features not implemented
 - Unfixed regression

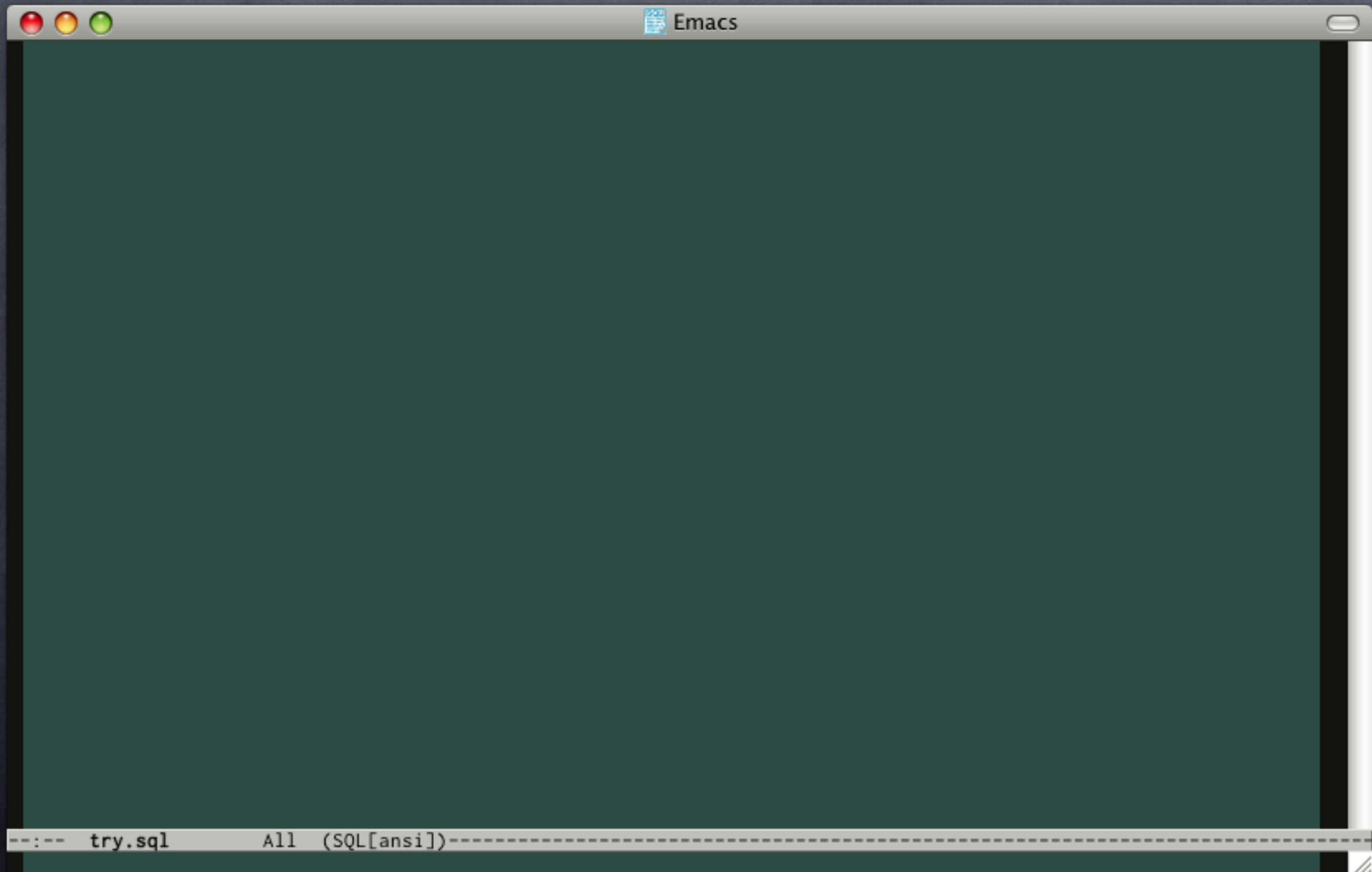
Todo Tests

- **Sometimes need to ignore failures**
 - **Features not implemented**
 - **Unfixed regression**
 - **Version dependences**

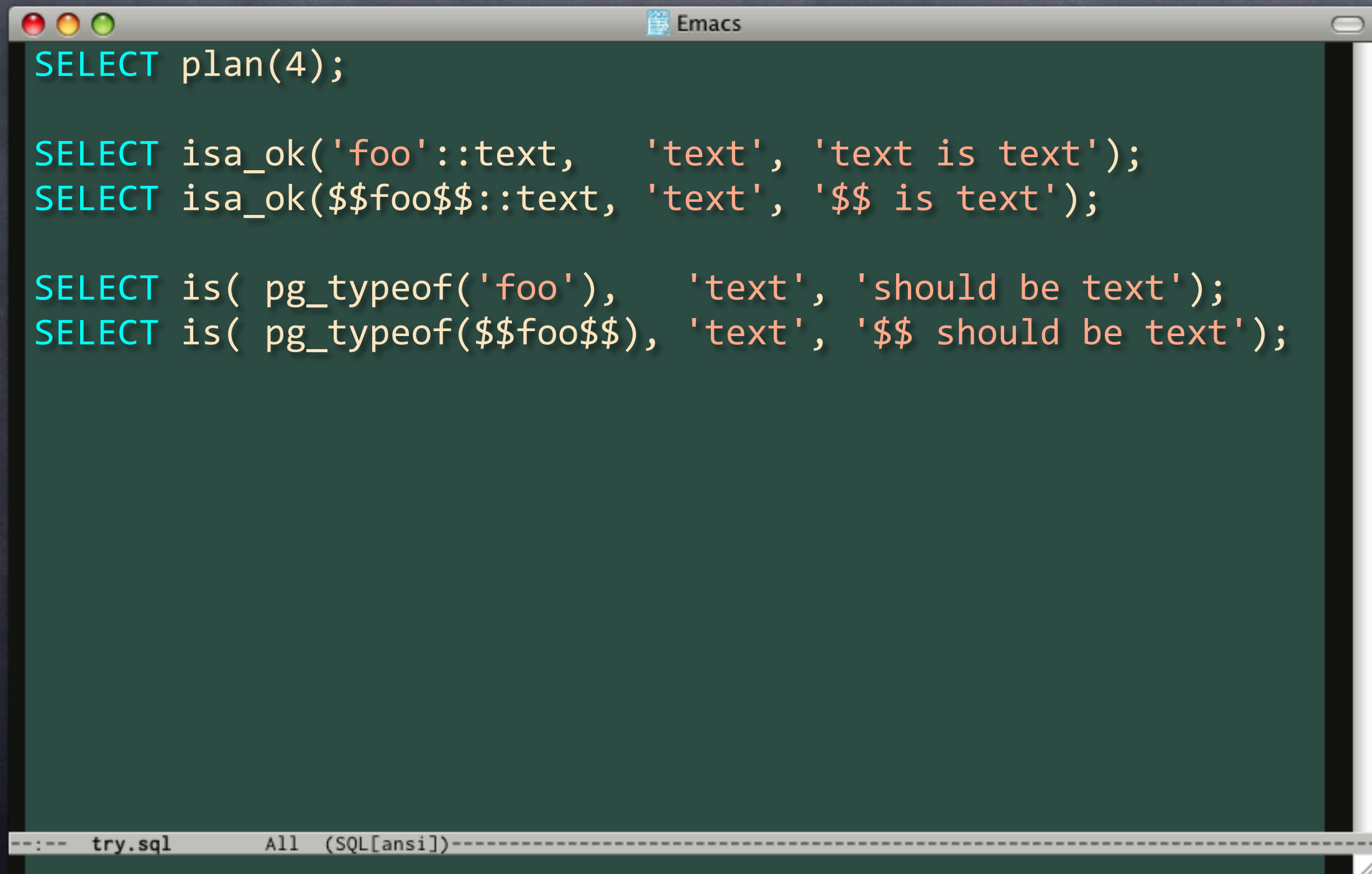
Todo Tests

- Sometimes need to ignore failures
 - Features not implemented
 - Unfixed regression
 - Version dependences
- Use `todo()`

Todo Tests



Todo Tests

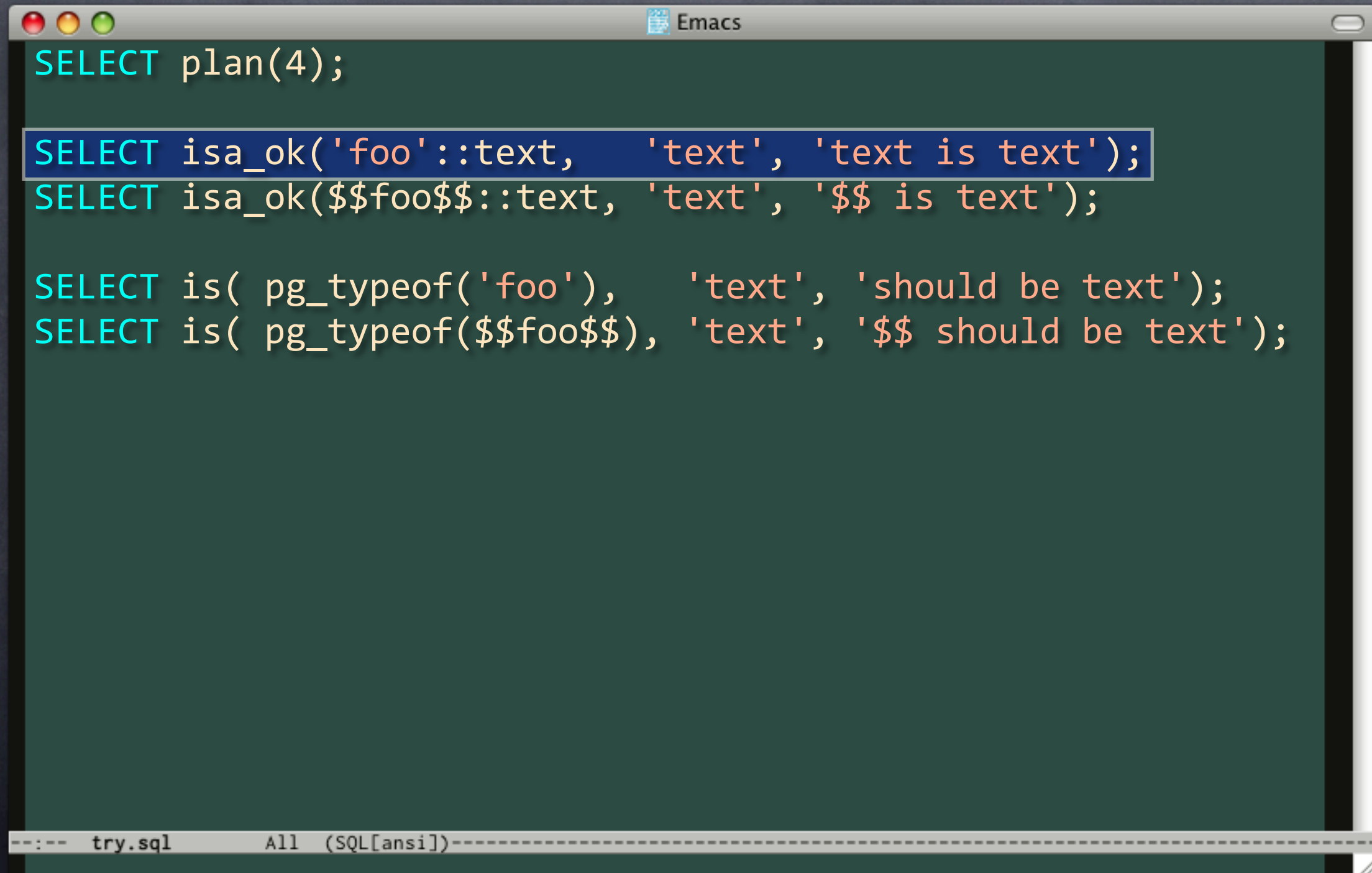
A screenshot of an Emacs window with a dark green background. The window title bar shows the Emacs logo and the text "Emacs". The code is displayed in a monospaced font with syntax highlighting: "SELECT" is cyan, "plan(4)", "isa_ok", and "is" are orange, and string literals are white. The code consists of eight lines of SQL test statements. At the bottom of the window, a status bar shows "--:-- try.sql All (SQL[ansi])-----".

```
SELECT plan(4);

SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

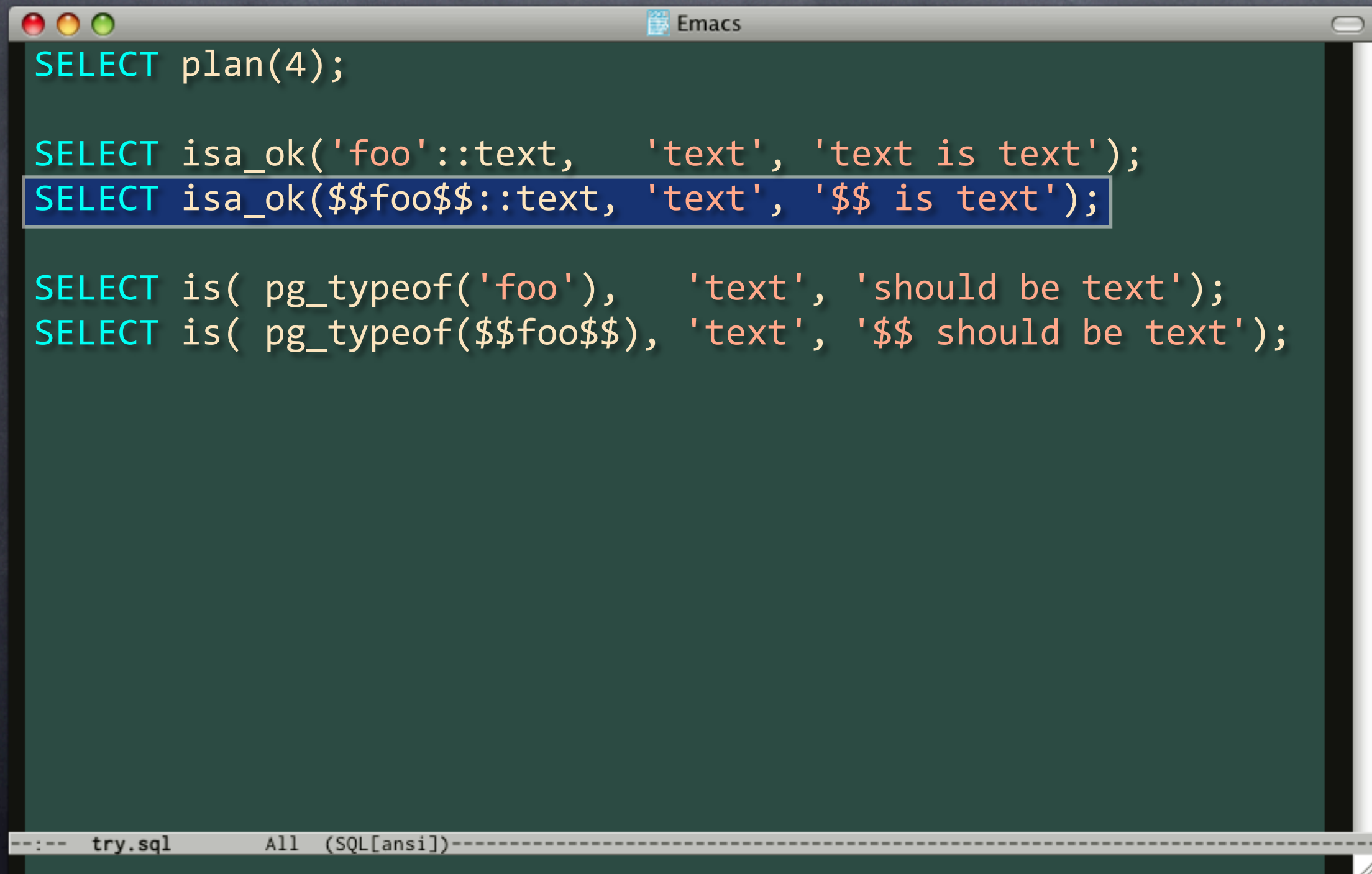

Todo Tests

A screenshot of an Emacs window titled "Emacs" with a dark green background. The window contains several lines of SQL code. The second line is highlighted with a blue selection box. The code includes a plan command, two is_a_ok tests, and two is tests. The status bar at the bottom shows "try.sql" and "All (SQL[ansi])".

```
SELECT plan(4);  
SELECT isa_ok('foo'::text, 'text', 'text is text');  
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');  
  
SELECT is( pg_typeof('foo'), 'text', 'should be text');  
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

try.sql All (SQL[ansi])

Todo Tests

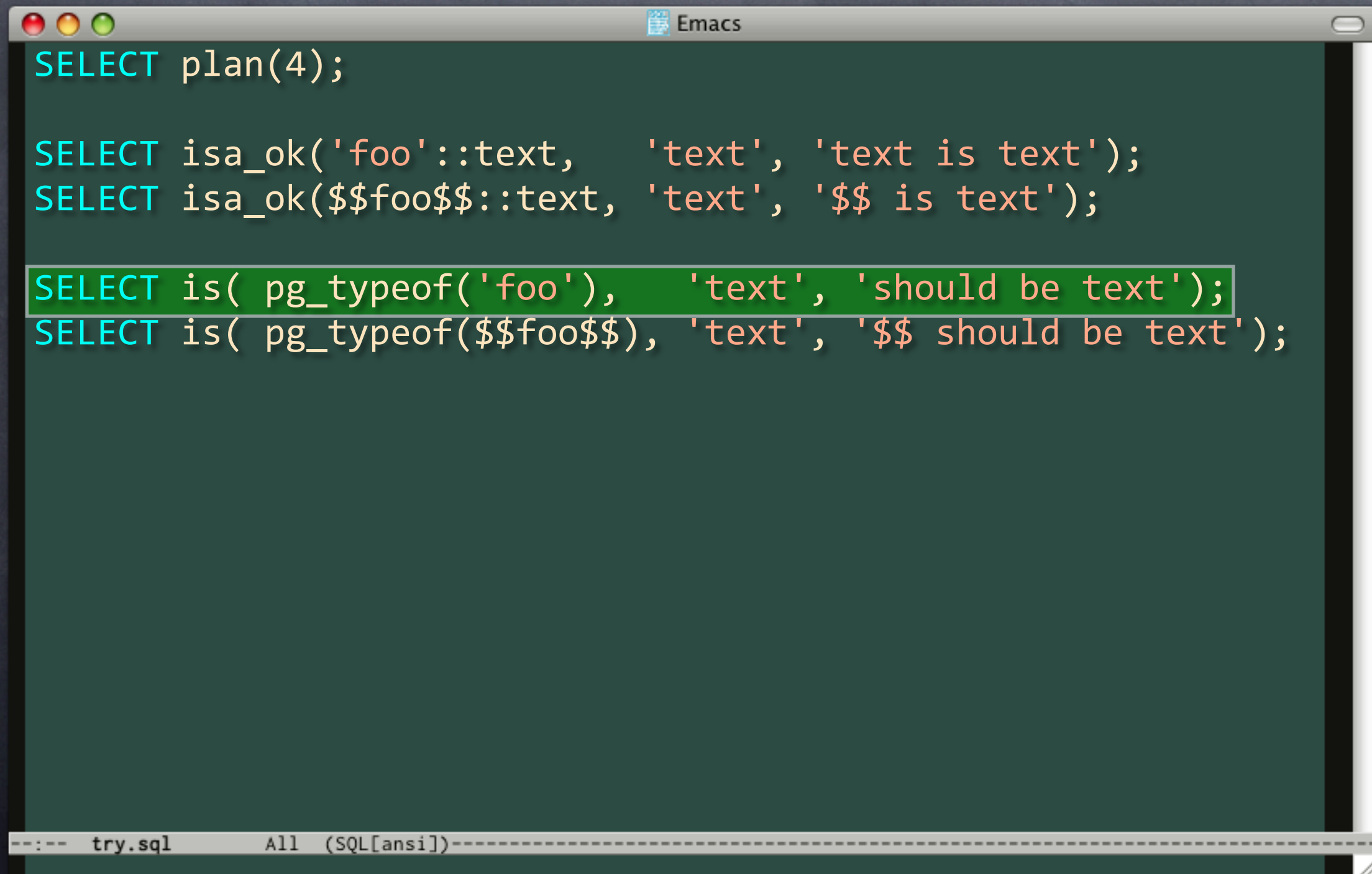
The image shows a screenshot of an Emacs window with a dark green background. The window title bar at the top contains the Emacs logo and the text "Emacs". The main area of the window displays several lines of SQL code. The third line, `SELECT isa_ok($$foo$$::text, 'text', '$$ is text');`, is highlighted with a blue selection box. At the bottom of the window, a status bar shows the file name "try.sql", the cursor position "All", and the mode "(SQL[ansi])".

```
SELECT plan(4);  
  
SELECT isa_ok('foo'::text, 'text', 'text is text');  
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');
```

```
SELECT is( pg_typeof('foo'), 'text', 'should be text');  
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

---:-- try.sql All (SQL[ansi])-----

Todo Tests

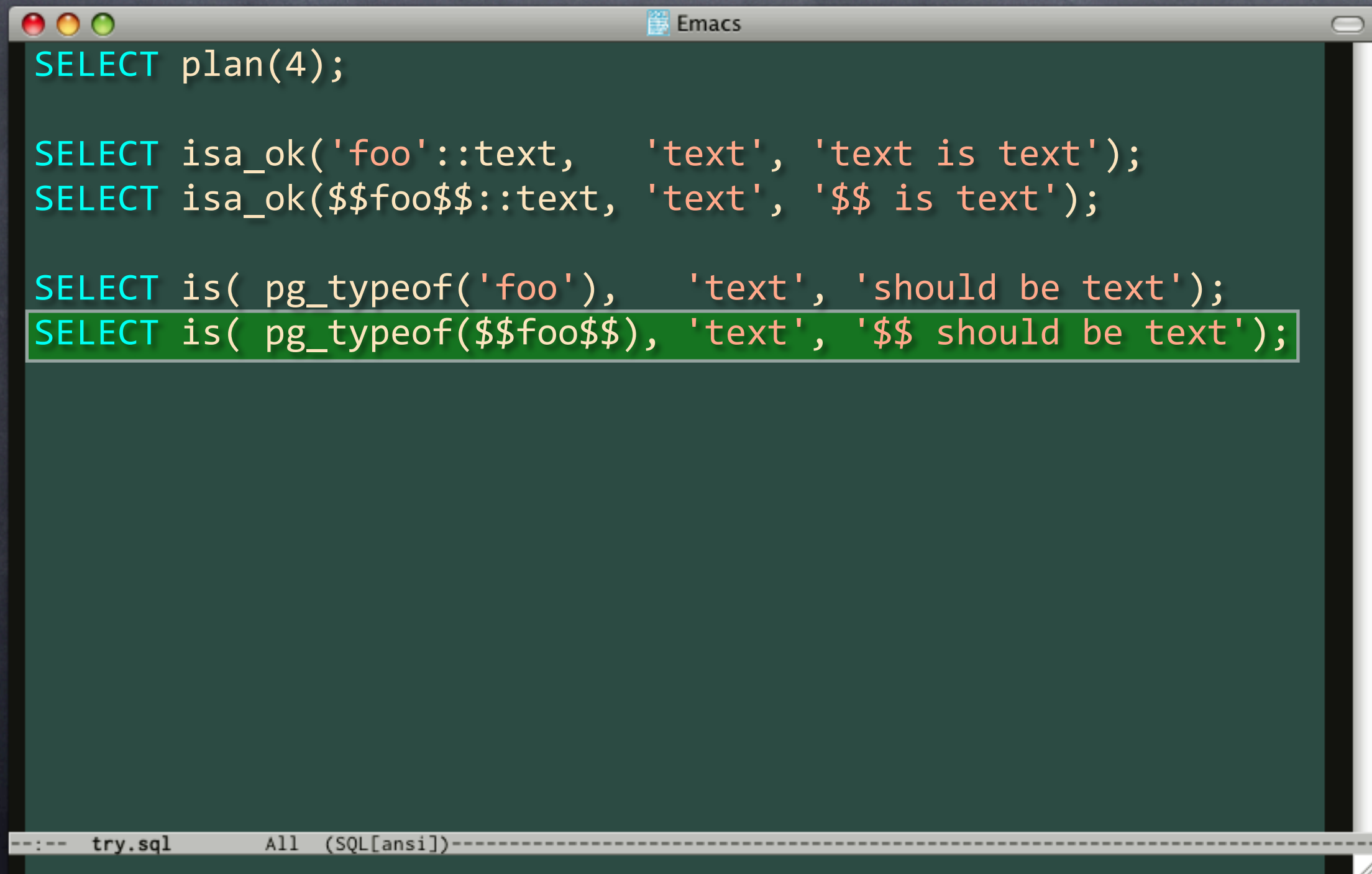
A screenshot of an Emacs window titled "Emacs" with a dark green background. The window contains several lines of SQL code. The first line is "SELECT plan(4);". The next two lines are "SELECT isa_ok('foo'::text, 'text', 'text is text');" and "SELECT isa_ok(\$\$foo\$\$::text, 'text', '\$\$ is text');". The fourth line, "SELECT is(pg_typeof('foo'), 'text', 'should be text');", is highlighted with a green background. The fifth line is "SELECT is(pg_typeof(\$\$foo\$\$), 'text', '\$\$ should be text');". At the bottom of the window, a status bar shows "--:-- try.sql All (SQL[ansi])-----".

```
SELECT plan(4);

SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```


Todo Tests

A screenshot of an Emacs window titled "Emacs" with a dark green background. The window contains several lines of SQL code. The first line is "SELECT plan(4);". The next two lines are "SELECT isa_ok('foo'::text, 'text', 'text is text');" and "SELECT isa_ok(\$\$foo\$\$::text, 'text', '\$\$ is text');". The following two lines are "SELECT is(pg_typeof('foo'), 'text', 'should be text');" and "SELECT is(pg_typeof(\$\$foo\$\$), 'text', '\$\$ should be text');". The last line is highlighted with a green background. At the bottom of the window, there is a status bar with the text "--:-- try.sql All (SQL[ansi])-----".

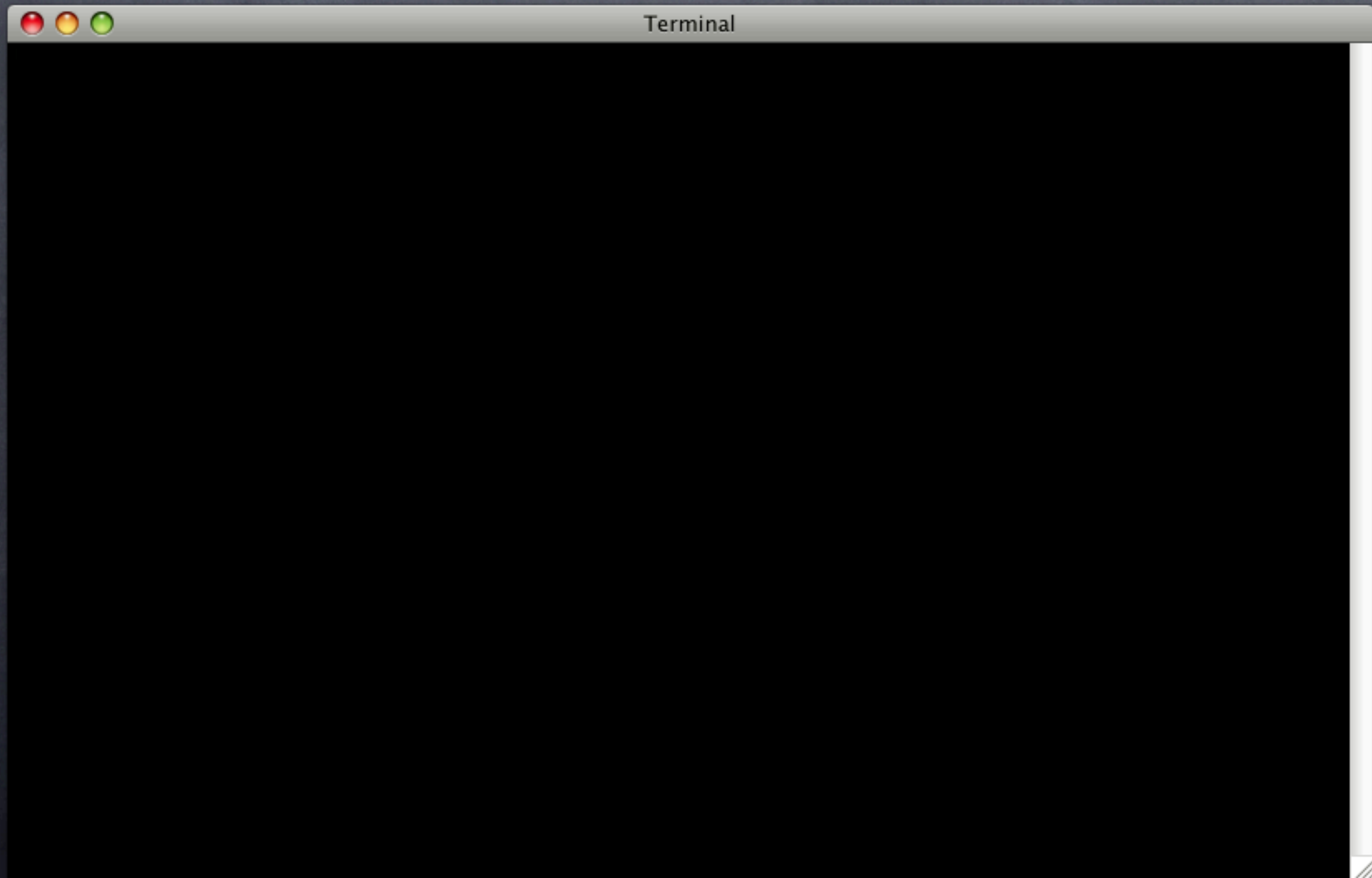
```
SELECT plan(4);

SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

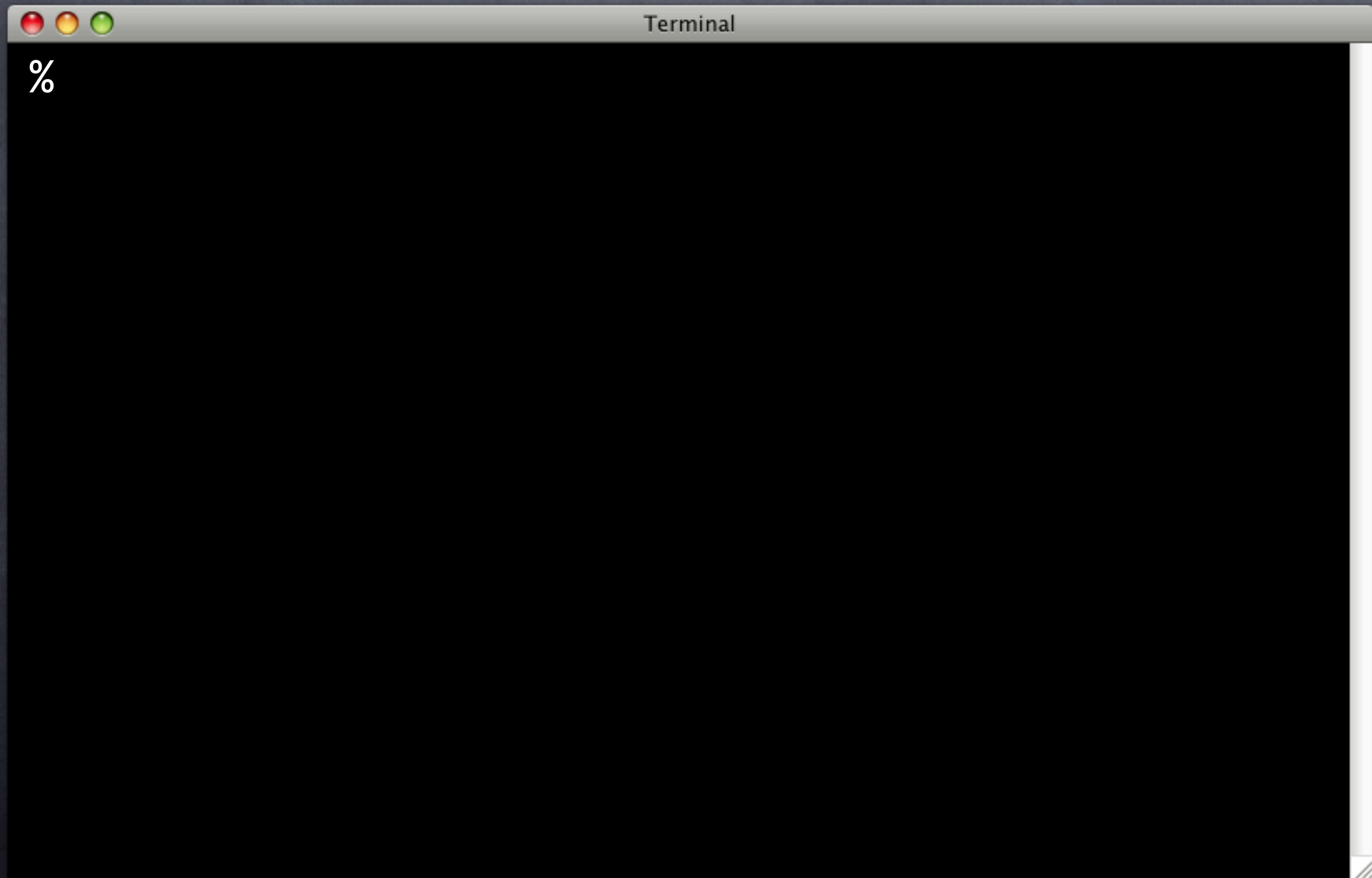
SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');

--:-- try.sql All (SQL[ansi])-----
```


Todo Tests



Todo Tests



Todo Tests

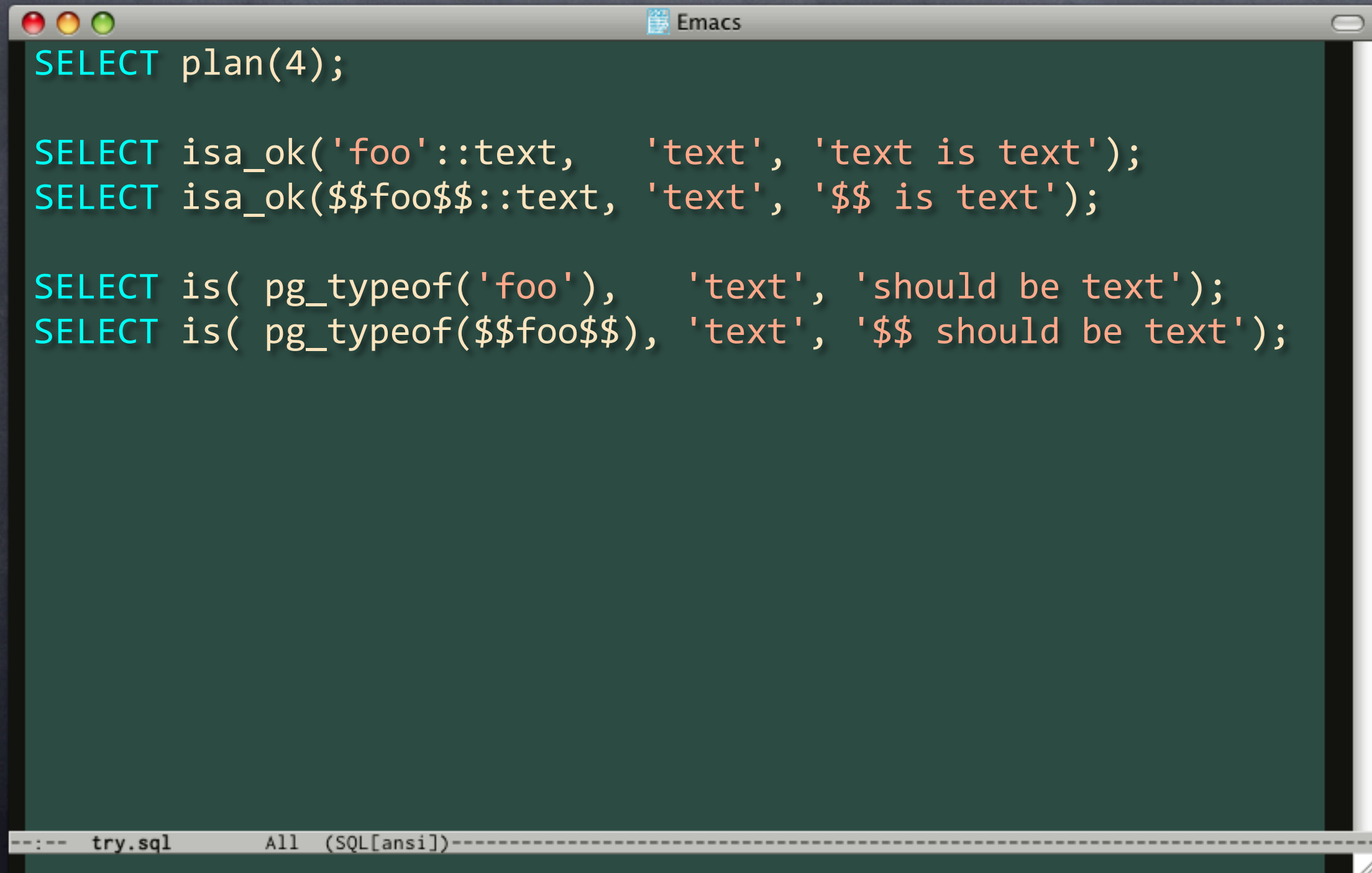
```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text
# Failed test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text
# Failed test 4: "$$ should be text"
#       have: unknown
#       want: text
# Looks like you failed 2 tests of 4
Failed 2/4 subtests
```


Todo Tests

```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text
# Failed test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text
# Failed test 4: "$$ should be text"
#       have: unknown
#       want: text
# Looks like you failed 2 tests of 4
Failed 2/4 subtests
```

Will get around to those...

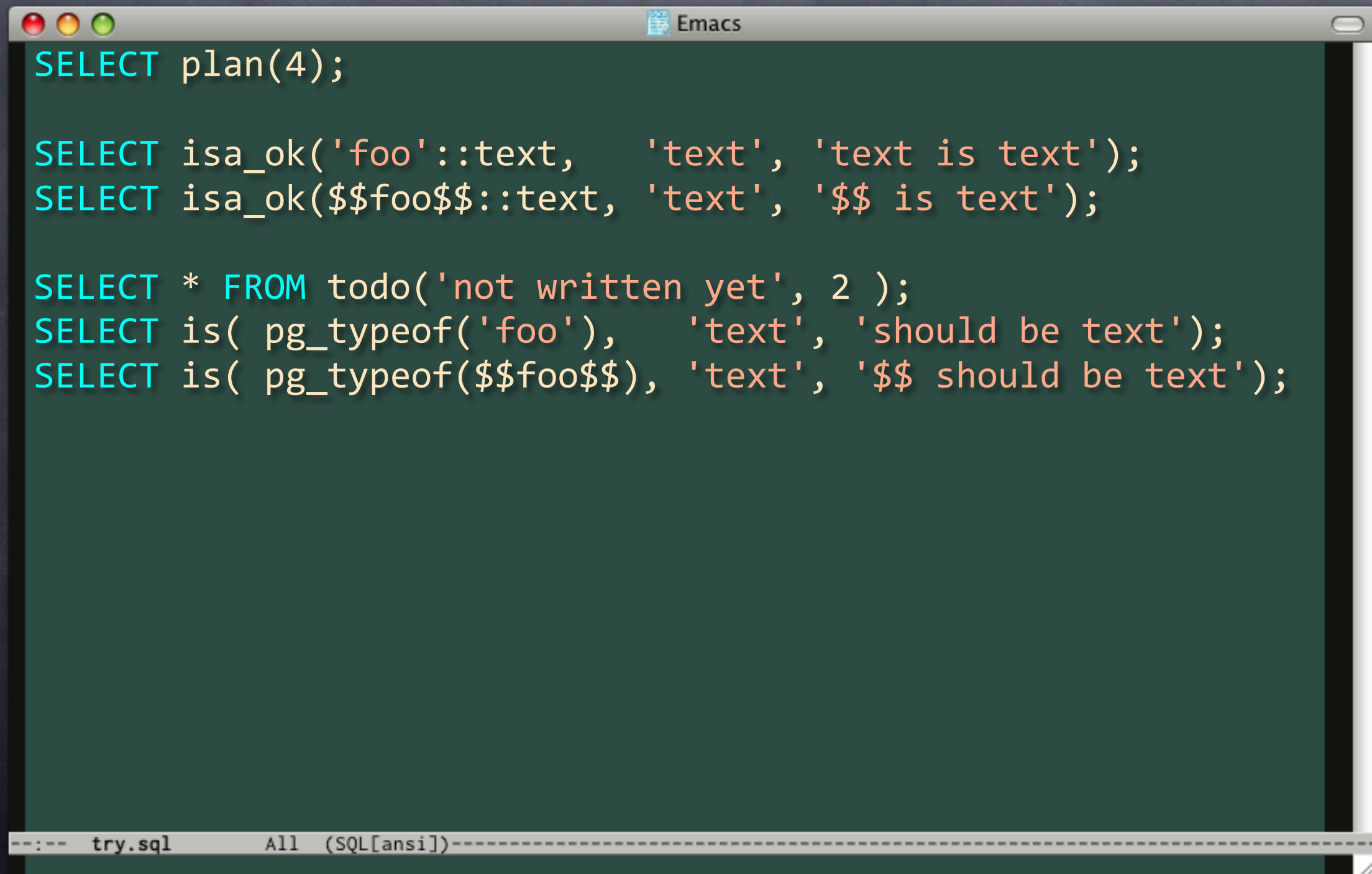
Todo Tests

A screenshot of an Emacs window with a dark green background. The window title bar shows the Emacs logo and the text "Emacs". The code is written in a light blue/cyan font for the "SELECT" keyword and a light orange/red font for the rest of the SQL statements. The code is as follows:

```
SELECT plan(4);  
  
SELECT isa_ok('foo'::text, 'text', 'text is text');  
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');  
  
SELECT is( pg_typeof('foo'), 'text', 'should be text');  
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

At the bottom of the window, there is a status bar with the text: "--:-- try.sql All (SQL[ansi])-----".

Todo Tests

The image shows a screenshot of an Emacs window with a dark green background. The window title bar at the top contains the Emacs logo and the text "Emacs". The main area of the window displays several lines of SQL code in a light-colored font. The code includes a plan statement, two is_ok tests, a FROM clause, and two is tests. The status bar at the bottom of the window shows the file name "try.sql", the cursor position "All", and the mode "(SQL[ansi])".

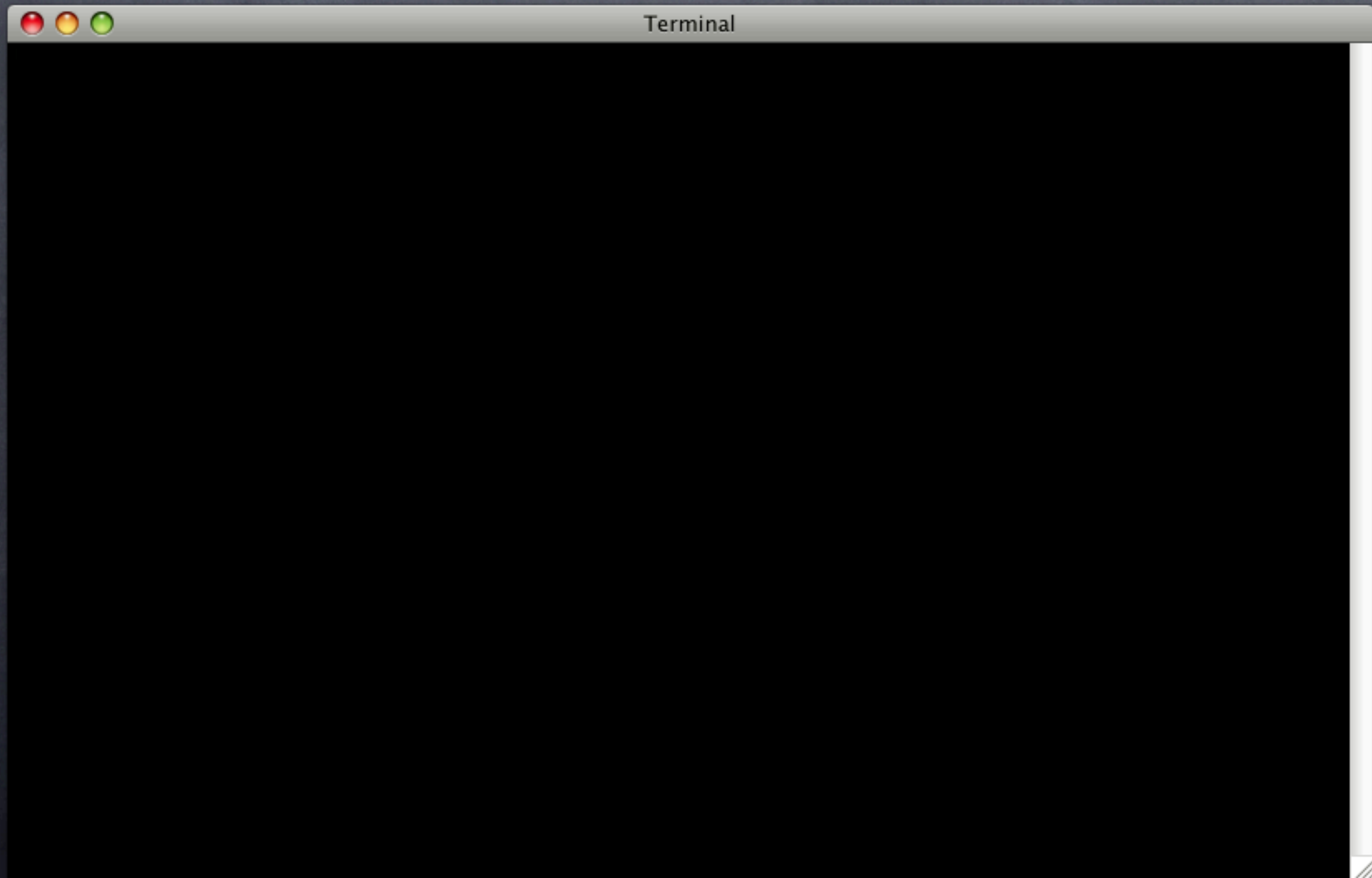
```
SELECT plan(4);

SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

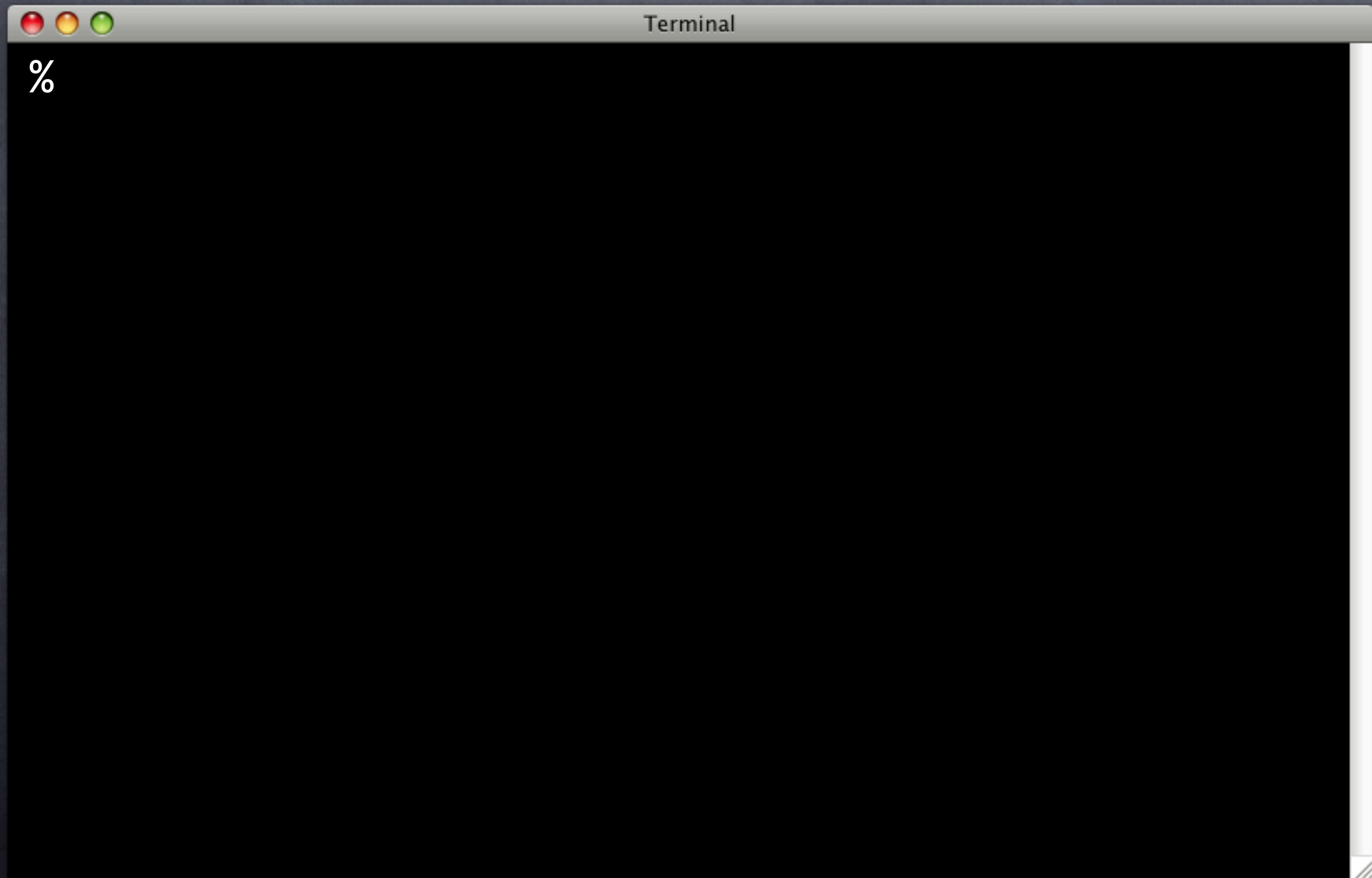
SELECT * FROM todo('not written yet', 2 );
SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

--:-- try.sql All (SQL[ansi])-----

Todo Tests



Todo Tests



Todo Tests

```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
1..4
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text # TODO not written yet
# Failed (TODO) test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text # TODO not written yet
# Failed (TODO) test 4: "$$ should be text"
#       have: unknown
#       want: text
ok
All tests successful.
```

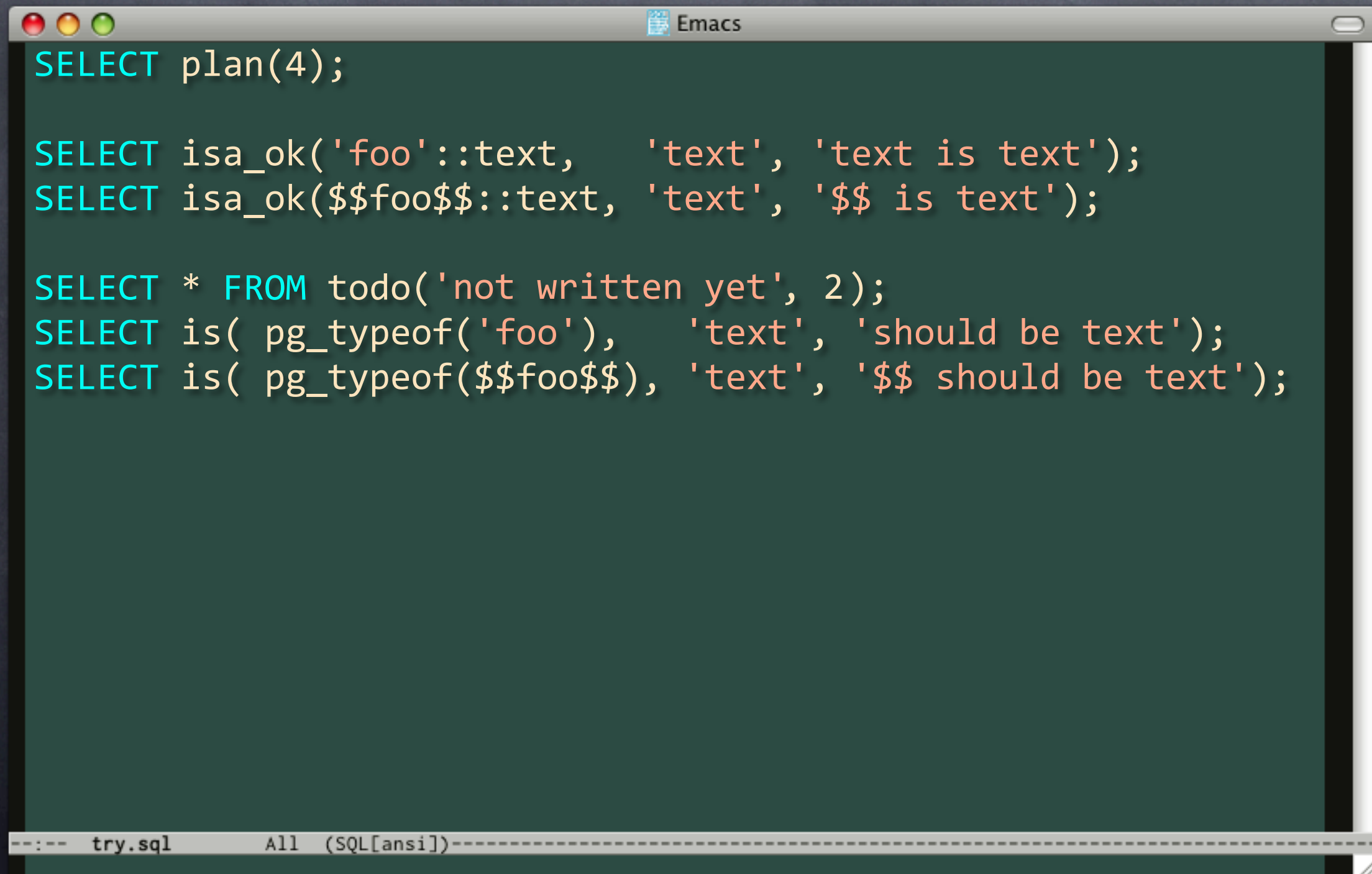

Todo Tests

```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
1..4
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text # TODO not written yet
# Failed (TODO) test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text # TODO not written yet
# Failed (TODO) test 4: "$$ should be text"
#       have: unknown
#       want: text
ok
All tests successful.
```


Todo Tests

```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
1..4
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text # TODO not written yet
# Failed (TODO) test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text # TODO not written yet
# Failed (TODO) test 4: "$$ should be text"
#       have: unknown
#       want: text
ok
All tests successful.
```


Todo Tests

A screenshot of an Emacs window titled "Emacs" with a dark green background. The window contains several lines of SQL code. The code is color-coded: "SELECT" is in cyan, "FROM" is in cyan, and the rest of the code is in orange. The code includes a plan statement, two "isa_ok" tests, a "FROM" statement, and two "is" tests. The status bar at the bottom shows "--:-- try.sql All (SQL[ansi])-----".

```
SELECT plan(4);

SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

SELECT * FROM todo('not written yet', 2);
SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
```

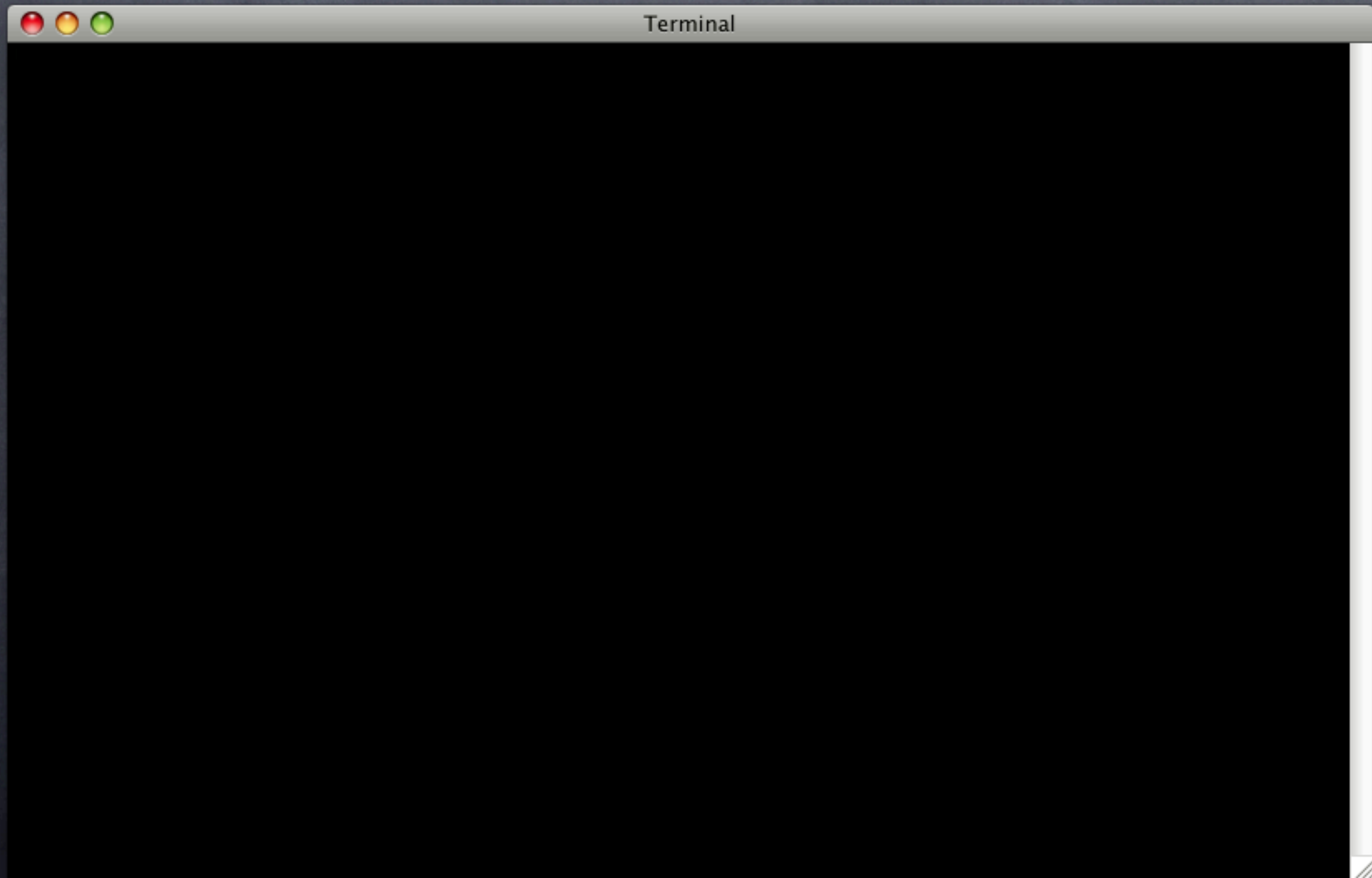

Todo Tests

```
SELECT plan(4);

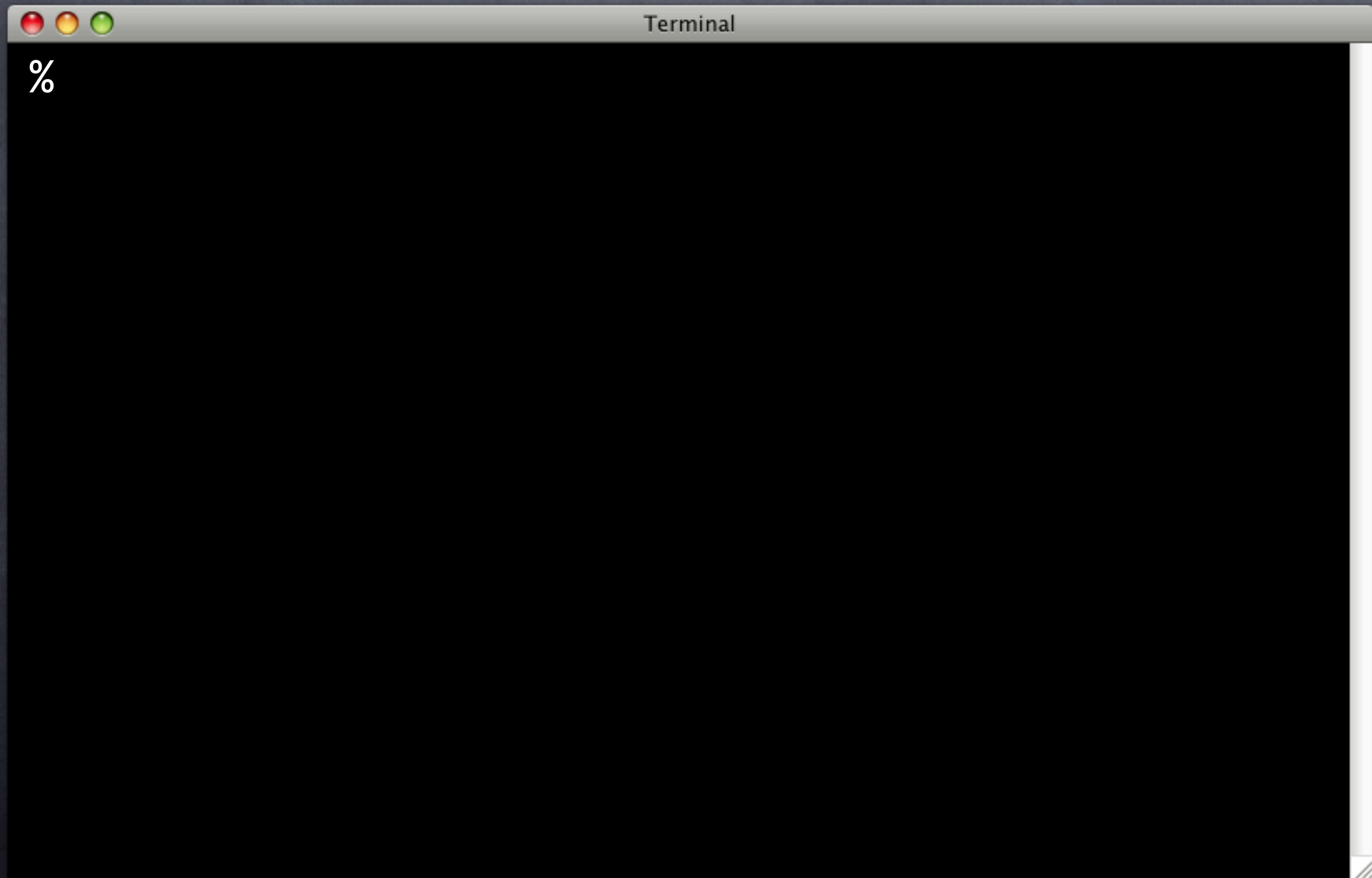
SELECT isa_ok('foo'::text, 'text', 'text is text');
SELECT isa_ok($$foo$$::text, 'text', '$$ is text');

SELECT * FROM todo_start('not written yet');
SELECT is( pg_typeof('foo'), 'text', 'should be text');
SELECT is( pg_typeof($$foo$$), 'text', '$$ should be text');
SELECT * FROM todo_end();
```


Todo Tests



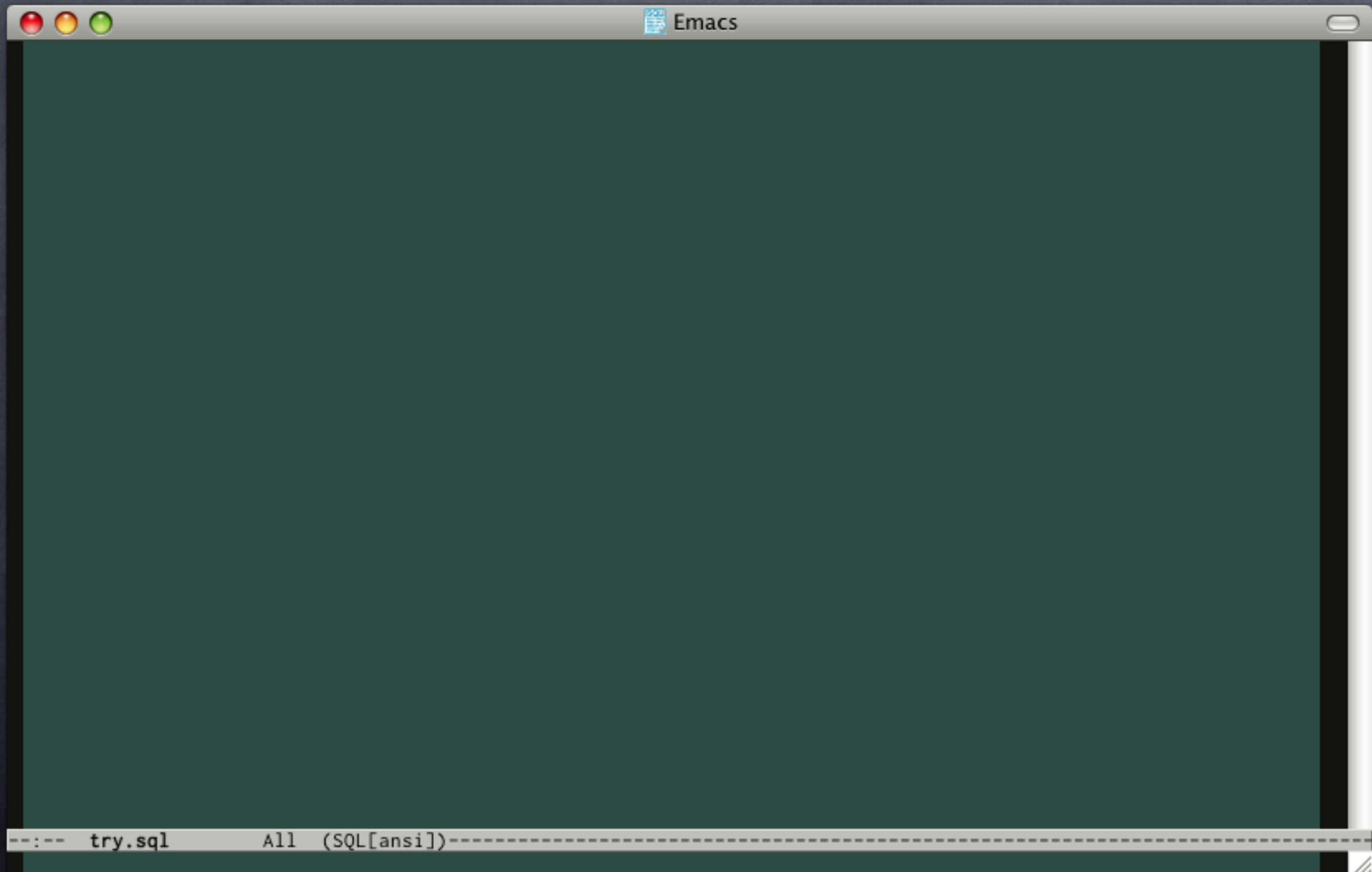
Todo Tests



Todo Tests

```
Terminal
% pg_prove -v -d try text.sql
text.sql ..
1..4
ok 1 - text is text isa text
ok 2 - $$ is text isa text
not ok 3 - should be text # TODO not written yet
# Failed (TODO) test 3: "should be text"
#       have: unknown
#       want: text
not ok 4 - $$ should be text # TODO not written yet
# Failed (TODO) test 4: "$$ should be text"
#       have: unknown
#       want: text
ok
All tests successful.
```


Other Goodies



Other Goodies

```
SELECT * FROM no_plan();

SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It''s alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```

--:-- try.sql All (SQL[ansi])-----

Other Goodies

```
SELECT * FROM no_plan();
```

```
SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
```

```
SELECT alike( :country, 'United %' );
```

```
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
```

```
SELECT throws_ok(  
    'SELECT lower(10.2)',  
    'function lower(numeric) does not exist'  
);
```

```
SELECT lives_ok( 'SELECT 2 + 2', 'It's alive!' );
```

```
SELECT diag( 'OMG TAP WTF?' );
```

```
SELECT performs_ok( 'SELECT fib(32)', 500 );
```

```
SELECT * FROM finish();
```


Other Goodies

```
SELECT * FROM no_plan();

SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It's alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```


Other Goodies

```
SELECT * FROM no_plan();

SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It's alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```


Other Goodies

```
SELECT * FROM no_plan();

SELECT matches( :zip_code, '^[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It's alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```


Other Goodies

```
SELECT * FROM no_plan();

SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It's alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```

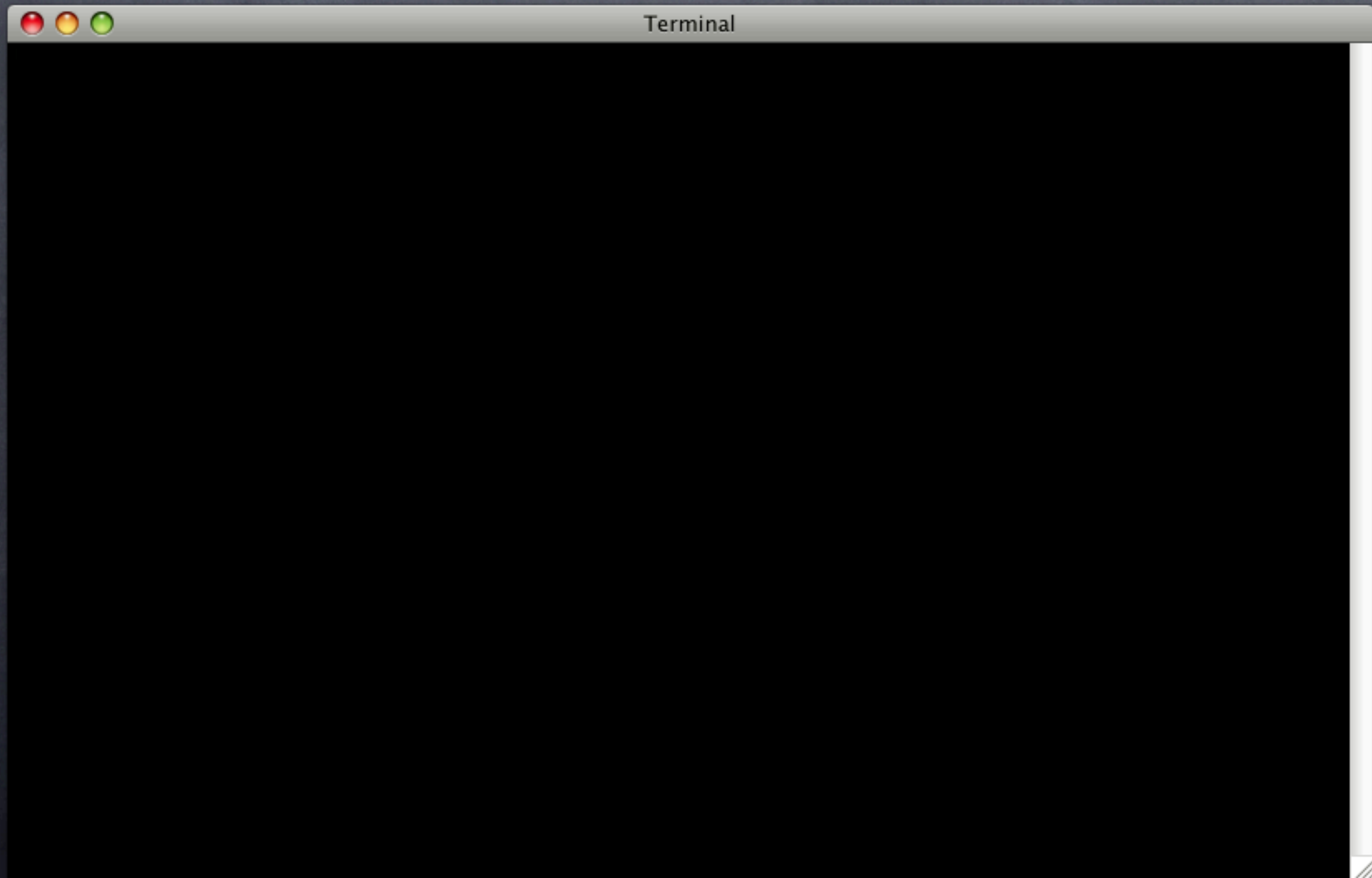

Other Goodies

```
SELECT * FROM no_plan();

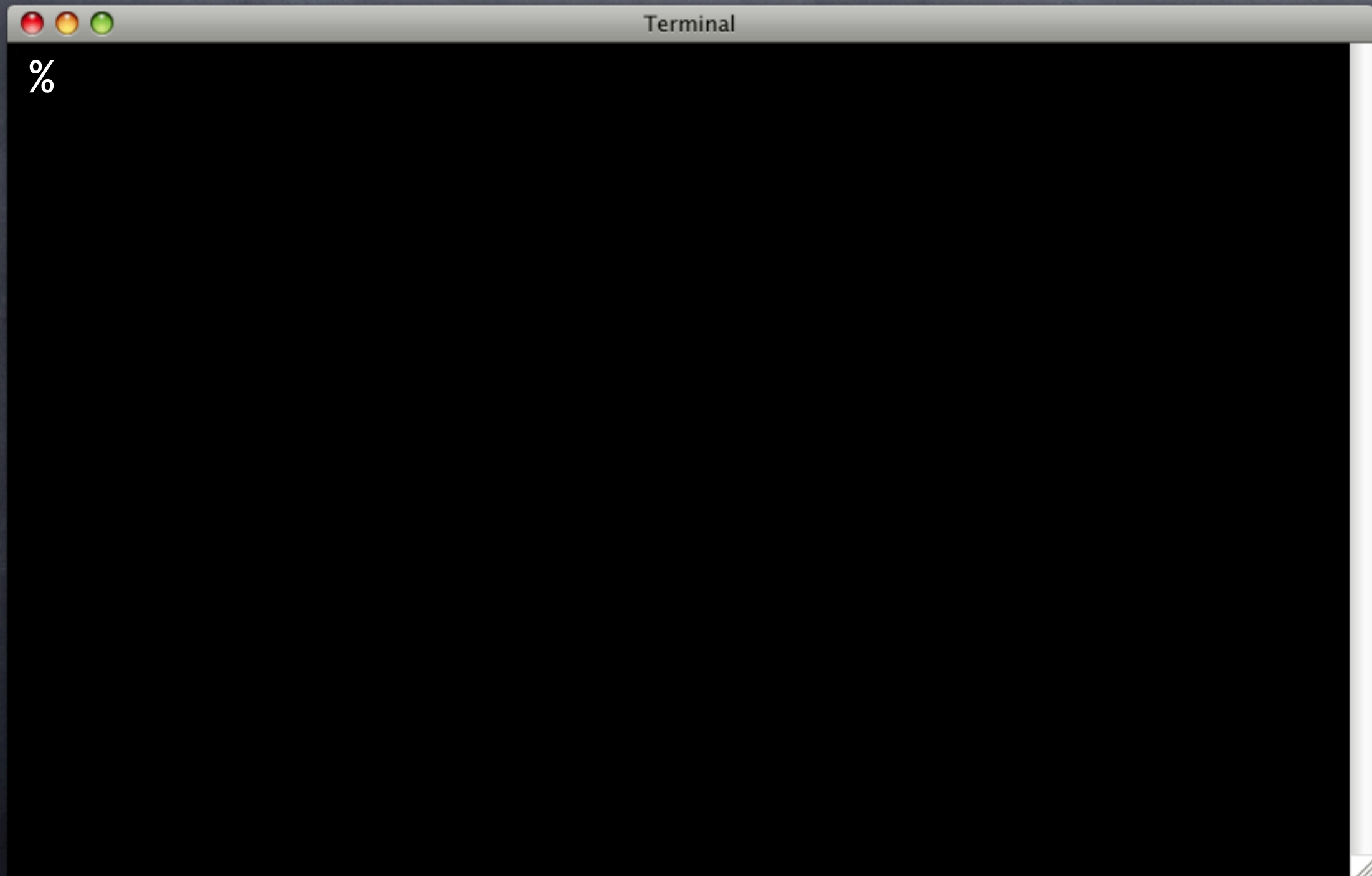
SELECT matches( :zip_code, '^[[[:digit:]]{5}$' );
SELECT alike( :country, 'United %' );
SELECT throws_ok( 'SELECT lower(10.2)', '42883' );
SELECT throws_ok(
    'SELECT lower(10.2)',
    'function lower(numeric) does not exist'
);
SELECT lives_ok( 'SELECT 2 + 2', 'It''s alive!' );
SELECT diag( 'OMG TAP WTF?' );
SELECT performs_ok( 'SELECT fib(32)', 500 );

SELECT * FROM finish();
```


Other Goodies



Other Goodies



Other Goodies

```
Terminal
% pg_prove -v -d try goodies.sql
goodies.sql ..
ok 1
ok 2
ok 3 - threw 42883
ok 4 - threw function lower(numeric) does not exist
ok 5 - It's alive!
# OMG TAP WTF?
ok 6 - Should run in less than 500 ms
1..6
ok
All tests successful.
```


Other Goodies

```
Terminal
% pg_prove -v -d try goodies.sql
goodies.sql ..
ok 1
ok 2
ok 3 - threw 42883
ok 4 - threw function lower(numeric) does not exist
ok 5 - It's alive!
# OMG TAP WTF?
ok 6 - Should run in less than 500 ms
1..6
ok
All tests successful.
```


Other Goodies

```
Terminal
% pg_prove -v -d try goodies.sql
goodies.sql ..
ok 1
ok 2
ok 3 - threw 42883
ok 4 - threw function lower(numeric) does not exist
ok 5 - It's alive!
# OMG TAP WTF?
ok 6 - Should run in less than 500 ms
1..6
ok
All tests successful.
```


Other Goodies

```
Terminal
% pg_prove -v -d try goodies.sql
goodies.sql ..
ok 1
ok 2
ok 3 - threw 42883
ok 4 - threw function lower(numeric) does not exist
ok 5 - It's alive!
# OMG TAP WTF?
ok 6 - Should run in less than 500 ms
1..6
ok
All tests successful.
```


Other Goodies

```
Terminal
% pg_prove -v -d try goodies.sql
goodies.sql ..
ok 1
ok 2
ok 3 - threw 42883
ok 4 - threw function lower(numeric) does not exist
ok 5 - It's alive!
# OMG TAP WTF?
ok 6 - Should run in less than 500 ms
1..6
ok
All tests successful.
```


Pursuing your Query

Pursuing your Query

- Several functions execute queries

Pursuing your Query

- Several functions execute queries
- Seen `throws_ok()`, `lives_ok()`, `performs_ok()`

Pursuing your Query

- Several functions execute queries
- Seen `throws_ok()`, `lives_ok()`, `performs_ok()`
- Take SQL statement argument

Pursuing your Query

- Several functions execute queries
- Seen throws_ok(), lives_ok(), performs_ok()
- Take SQL statement argument
- PITA for complicated queries

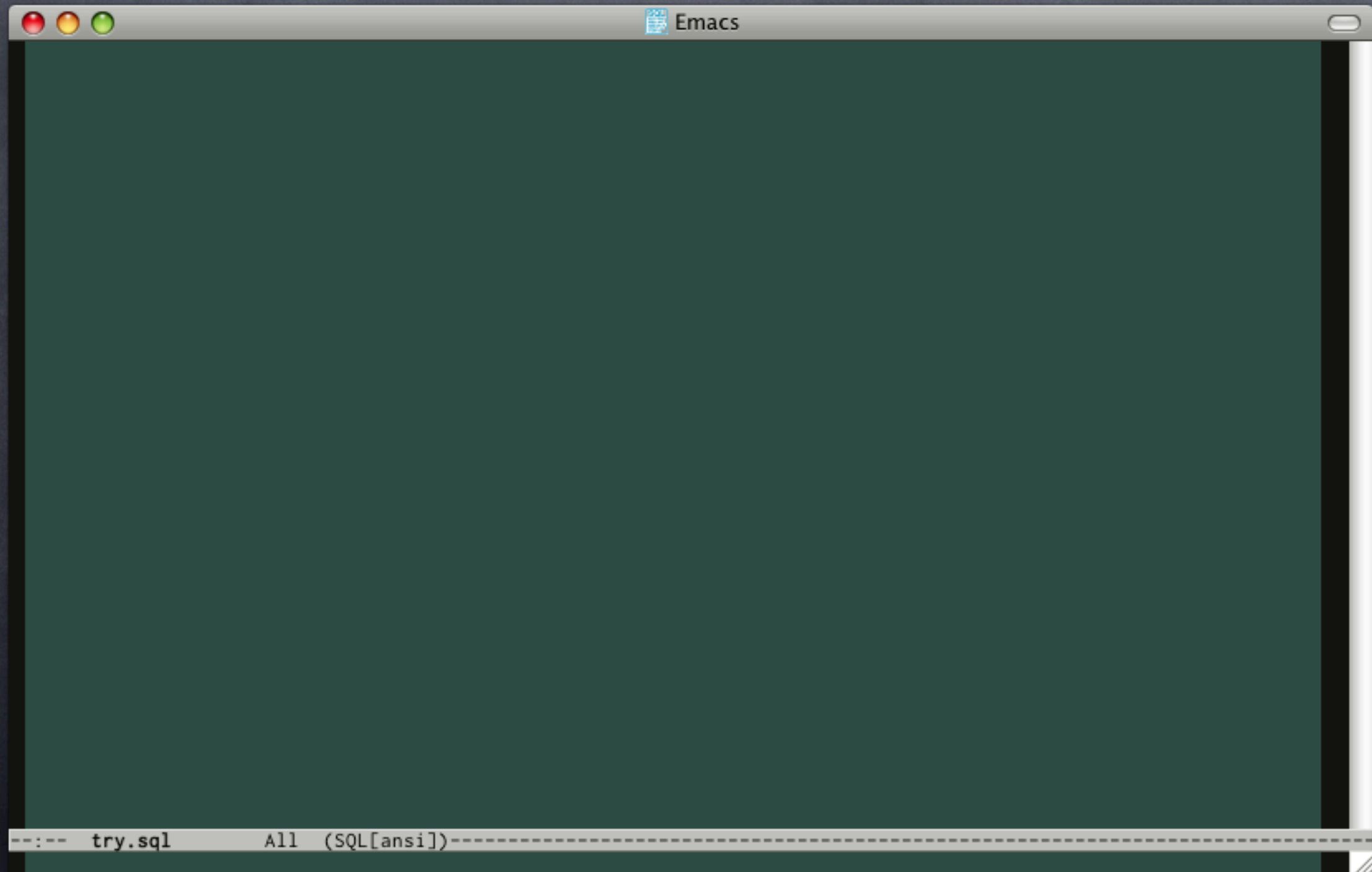
Pursuing your Query

- Several functions execute queries
- Seen `throws_ok()`, `lives_ok()`, `performs_ok()`
- Take SQL statement argument
- PITA for complicated queries
- Single quotes a particular PITA

Pursuing your Query

- Several functions execute queries
- Seen throws `_ok()`, `lives_ok()`, `performs_ok()`
- Take SQL statement argument
- PITA for complicated queries
- Single quotes a particular PITA
- Alternative: Prepared Statements

Pursuing your Query



Pursuing your Query

```
Emacs  
PREPARE new_user AS  
INSERT INTO users (nick, pass, name)  
VALUES ('theory', 's3kr1t', 'David Wheeler');  
  
EXECUTE new_user;  
SELECT throws_ok ('new_user');
```

```
--:-- try.sql All (SQL[ansi])-----
```


Pursuing your Query

```
Emacs  
PREPARE new_user AS  
INSERT INTO users (nick, pass, name)  
VALUES ('theory', 's3kr1t', 'David Wheeler');  
  
EXECUTE new_user;  
SELECT throws_ok ('new_user');
```

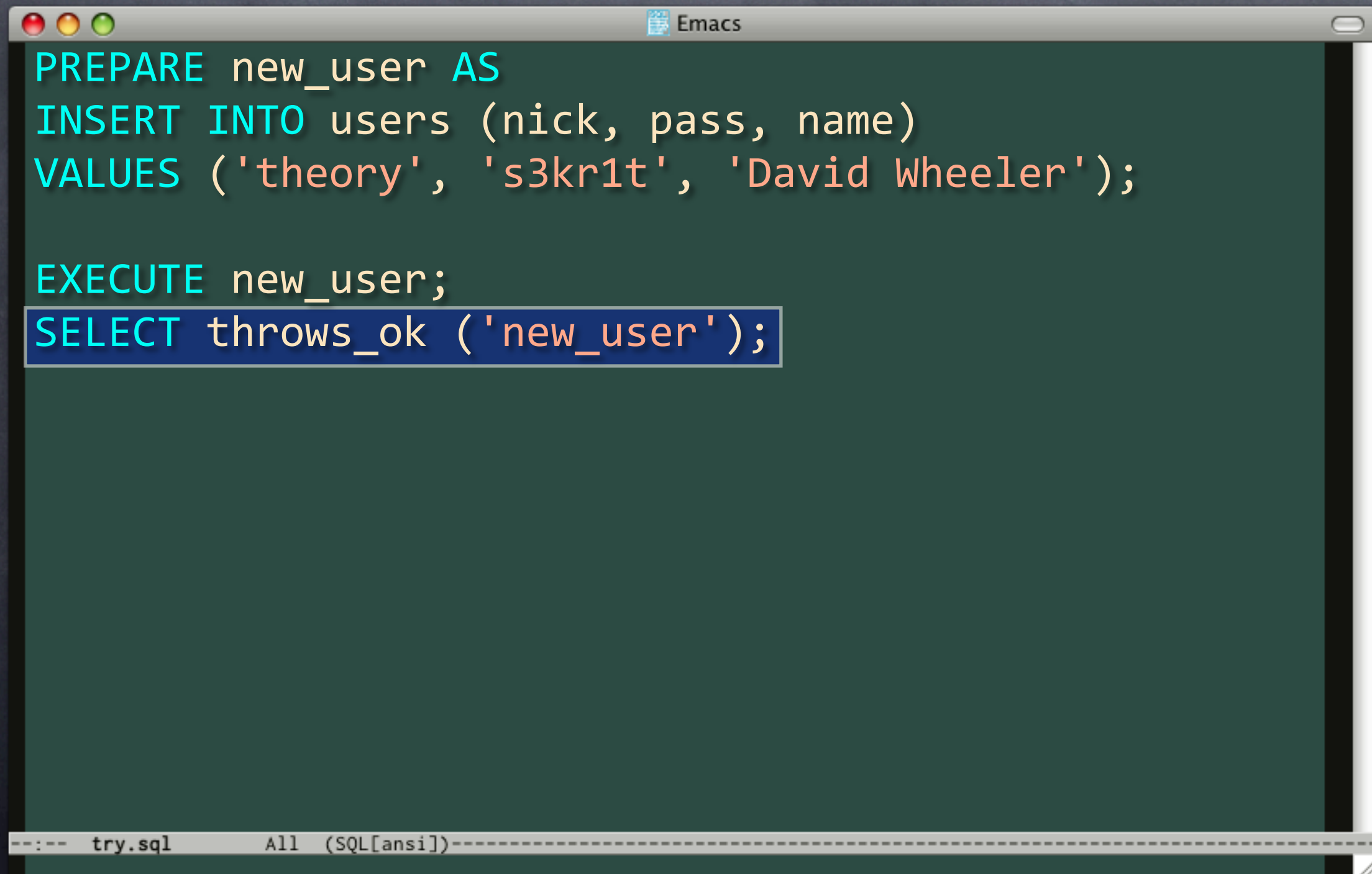
```
--:-- try.sql All (SQL[ansi])-----
```


Pursuing your Query

```
Emacs  
PREPARE new_user AS  
INSERT INTO users (nick, pass, name)  
VALUES ('theory', 's3kr1t', 'David Wheeler');  
EXECUTE new_user;  
SELECT throws_ok ('new_user');
```

```
--:-- try.sql All (SQL[ansi])-----
```


Pursuing your Query

A screenshot of an Emacs window with a dark green background. The window title bar shows 'Emacs' and standard window control buttons. The code is as follows:

```
PREPARE new_user AS  
INSERT INTO users (nick, pass, name)  
VALUES ('theory', 's3kr1t', 'David Wheeler');  
  
EXECUTE new_user;  
SELECT throws_ok ('new_user');
```

The last line is highlighted with a blue background. At the bottom of the window, a status bar shows: `--:-- try.sql All (SQL[ansi])-----`

Testing Relations

Testing Relations

- Not everything is a scalar

Testing Relations

- Not everything is a scalar
- It's a **RELATIONAL** database, after all

Testing Relations

- **Not everything is a scalar**
- **It's a RELATIONAL database, after all**
- **Only so much coercion to scalars to do**

Testing Relations

- **Not everything is a scalar**
- **It's a RELATIONAL database, after all**
- **Only so much coercion to scalars to do**
- **Need to test**

Testing Relations

- Not everything is a scalar
- It's a RELATIONAL database, after all
- Only so much coercion to scalars to do
- Need to test
 - results

Testing Relations

- Not everything is a scalar
- It's a **RELATIONAL** database, after all
- Only so much coercion to scalars to do
- Need to test
 - results
 - sets

Testing Relations

- Not everything is a scalar
- It's a RELATIONAL database, after all
- Only so much coercion to scalars to do
- Need to test
 - results
 - sets
 - bags

`results_eq()`

results_eq()

- Tests query results

results_eq()

- Tests query results
- Row-by-row comparison

results_eq()

- Tests query results
- Row-by-row comparison
- In order

results_eq()

- Tests query results
- Row-by-row comparison
- In order
- Test query and expected query may be:

results_eq()

- Tests query results
- Row-by-row comparison
- In order
- Test query and expected query may be:
 - SQL statements

results_eq()

- Tests query results
- Row-by-row comparison
- In order
- Test query and expected query may be:
 - SQL statements
 - Prepared statement names

results_eq()

- Tests query results
- Row-by-row comparison
- In order
- Test query and expected query may be:
 - SQL statements
 - Prepared statement names
 - Cursor names

Testing Results

Testing Results

- Example: `active_users()`

Testing Results

- Example: `active_users()`
- Returns set of active users

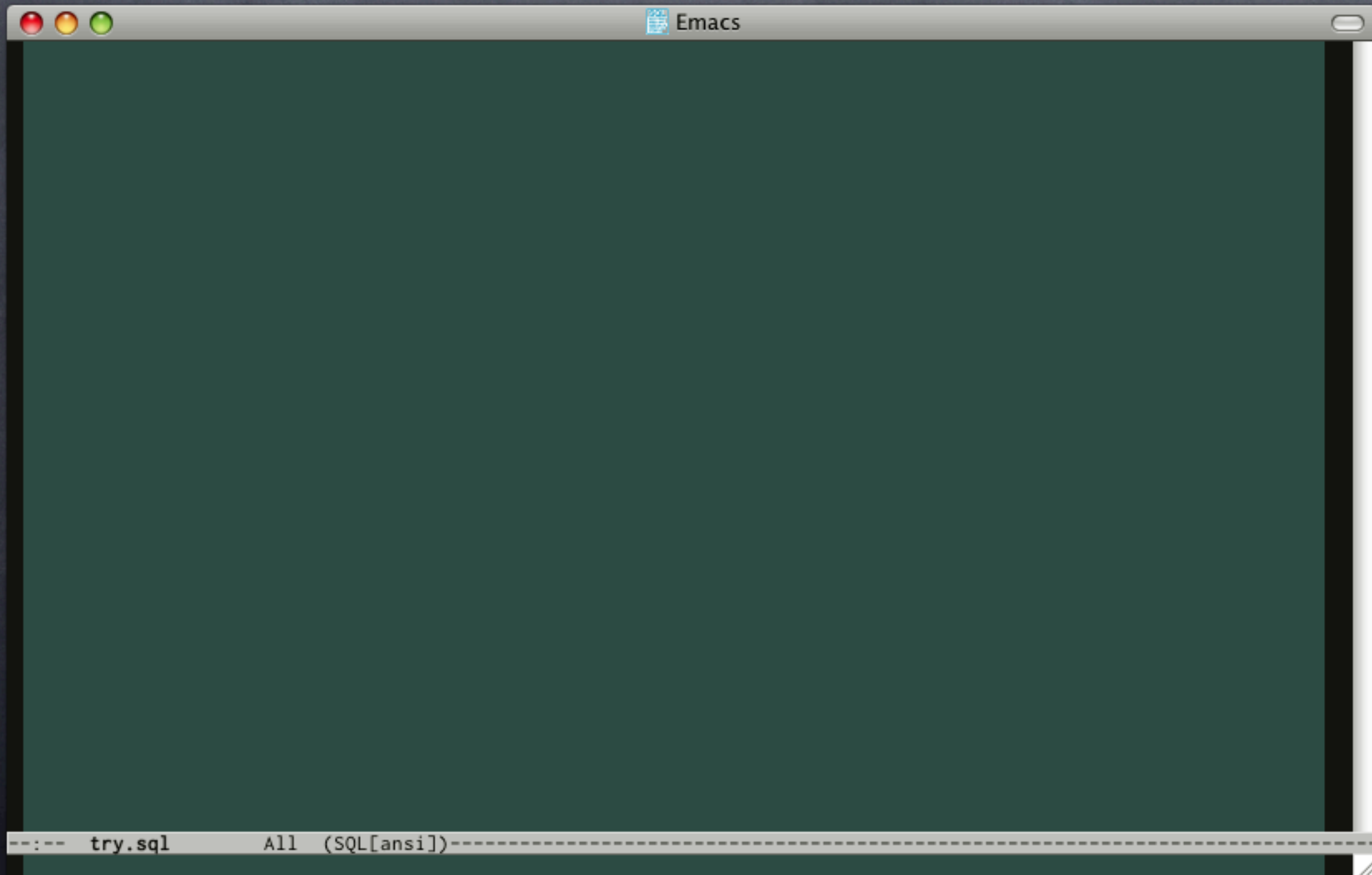
Testing Results

- **Example: `active_users()`**
- **Returns set of active users**
- **Test the function**

Testing Results

- **Example: `active_users()`**
- **Returns set of active users**
- **Test the function**
- **Compare results**

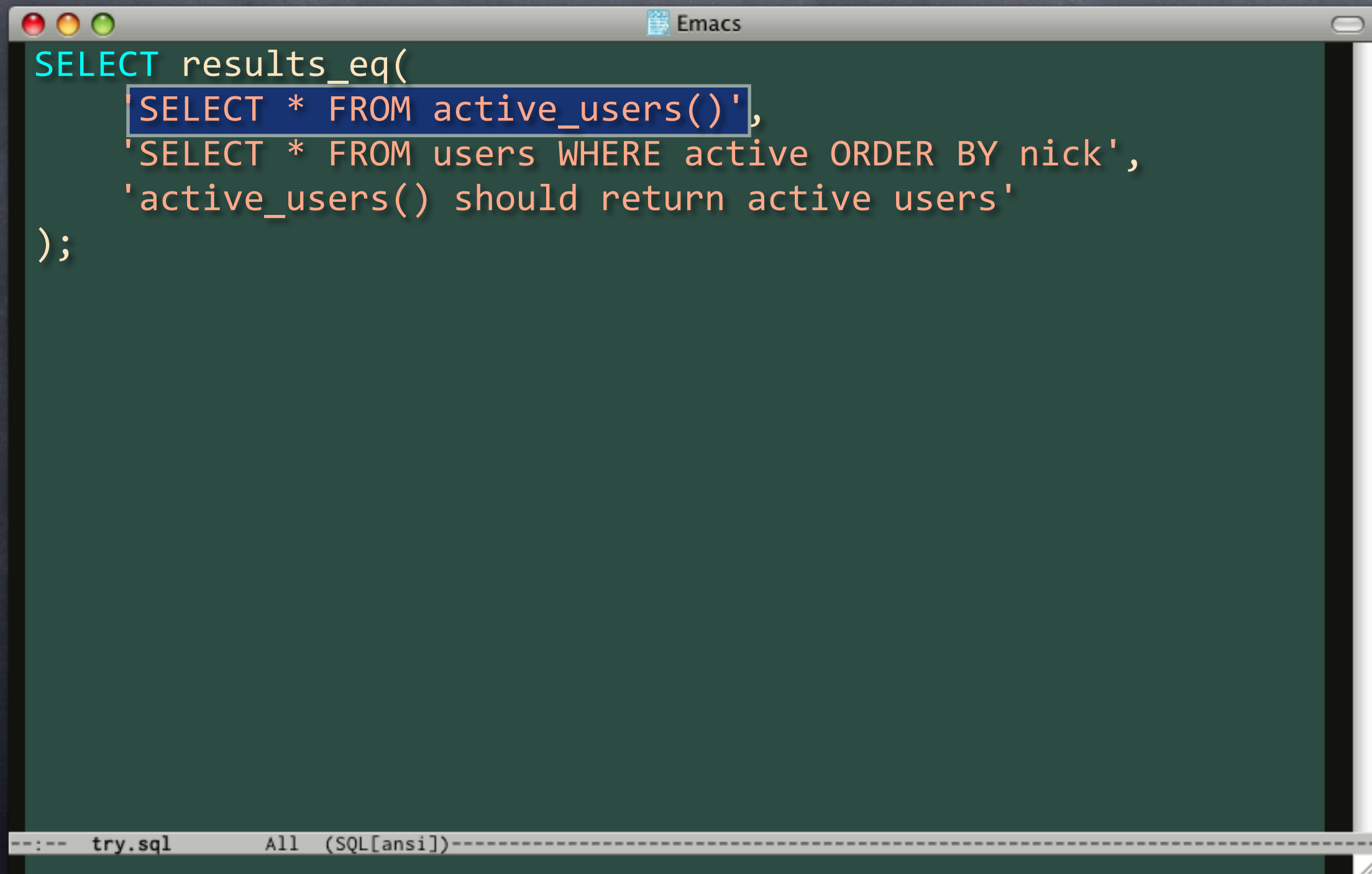
results_eq() Arguments



results_eq() Arguments

```
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active ORDER BY nick',  
  'active_users() should return active users'  
);
```


results_eq() Arguments



```
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active ORDER BY nick',  
  'active_users() should return active users'  
);
```

try.sql All (SQL[ansi])

results_eq() Arguments

```
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active ORDER BY nick',  
  'active_users() should return active users'  
);
```


results_eq() Arguments

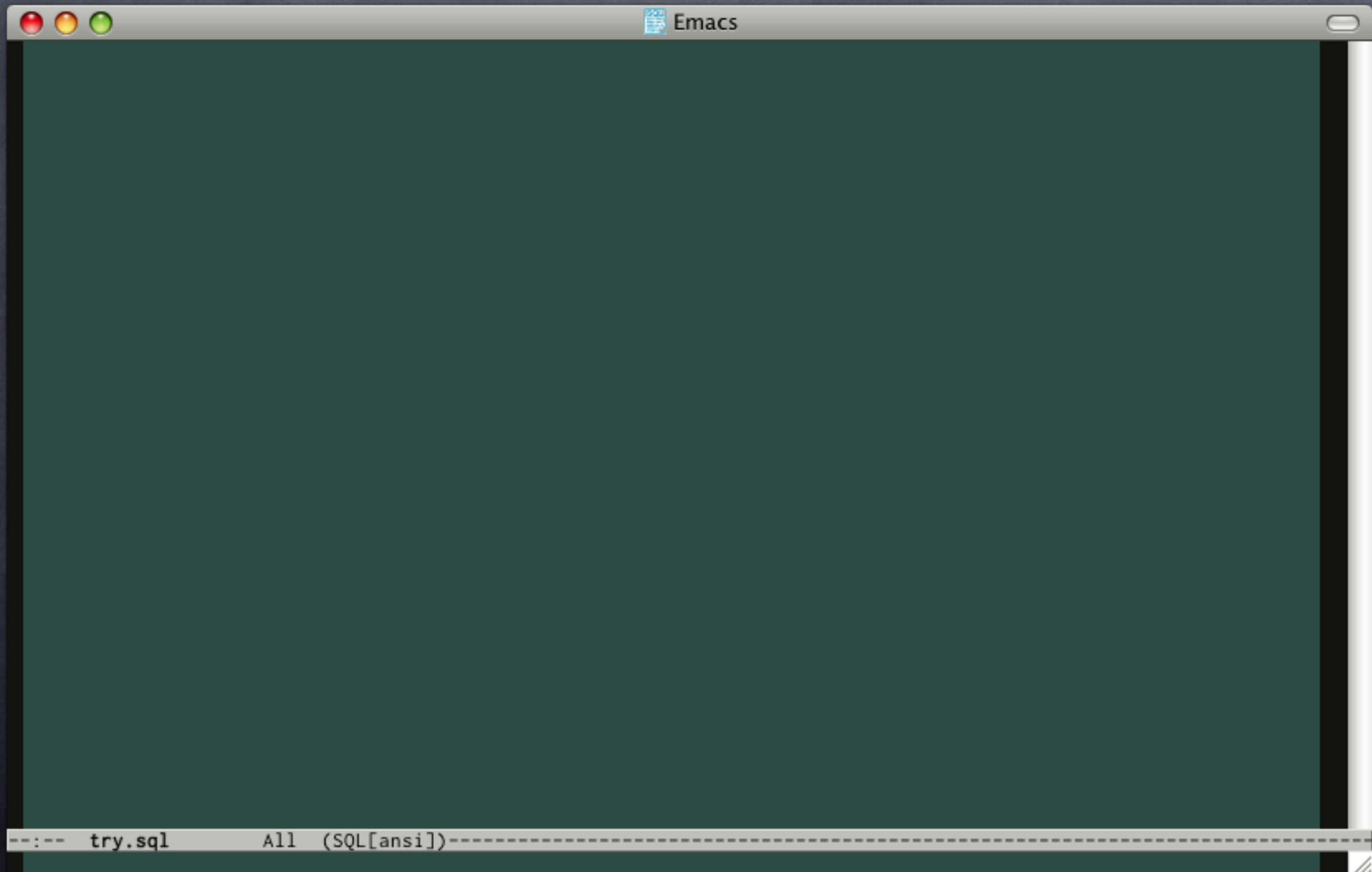
```
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active ORDER BY nick',  
  'active_users() should return active users'  
);  
  
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),  
           ('strongrr1', 'design', 'Julie Wheeler', true),  
           ('theory', 's3kr1t', 'David Wheeler', true)  
  $$,  
  'active_users() should return active users'  
);
```

--:-- try.sql All (SQL[ansi])-----

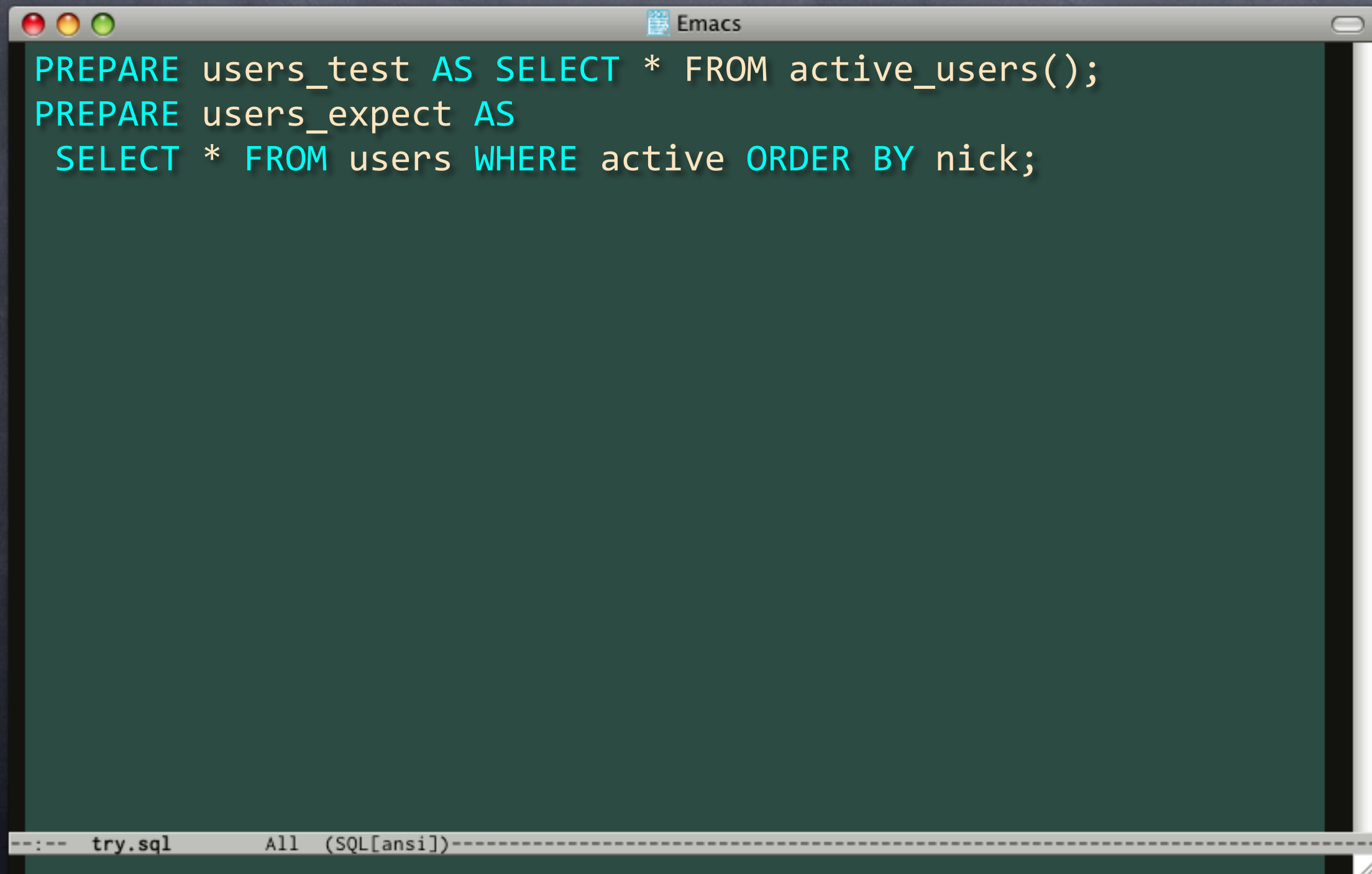
results_eq() Arguments

```
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active ORDER BY nick',  
  'active_users() should return active users'  
);  
  
SELECT results_eq(  
  'SELECT * FROM active_users()',  
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),  
           ('strongrr1', 'design', 'Julie Wheeler', true),  
           ('theory', 's3kr1t', 'David Wheeler', true)  
  $$,  
  'active_users() should return active users'  
);
```


results_eq() Arguments



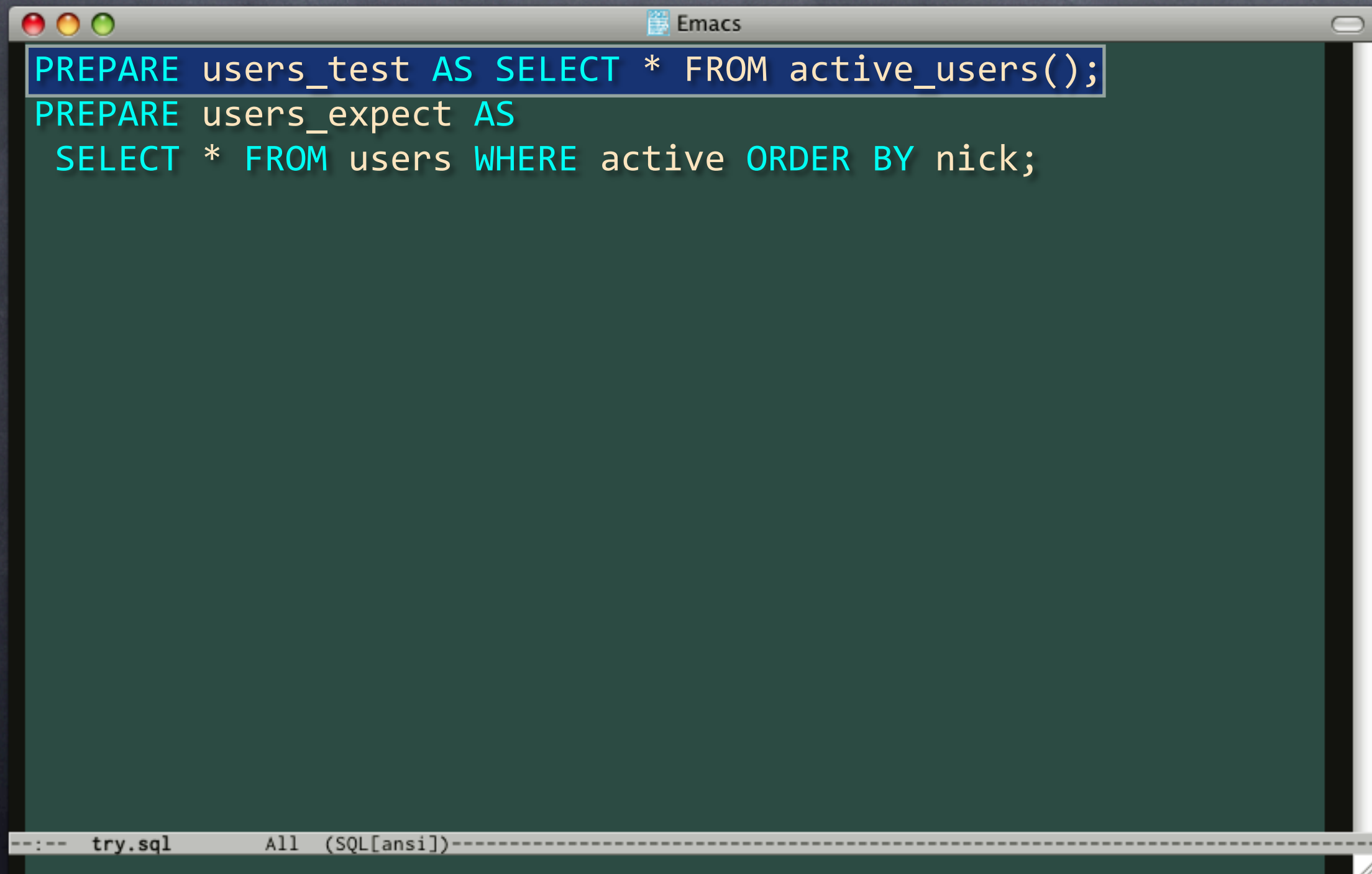
results_eq() Arguments

A screenshot of an Emacs window with a dark green background. The window title bar shows 'Emacs' and standard macOS window controls (red, yellow, green buttons). The main area contains SQL code in cyan text:

```
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
  SELECT * FROM users WHERE active ORDER BY nick;
```

At the bottom of the window, a status bar shows the file name 'try.sql', the cursor position 'All', and the mode '(SQL[ansi])'.

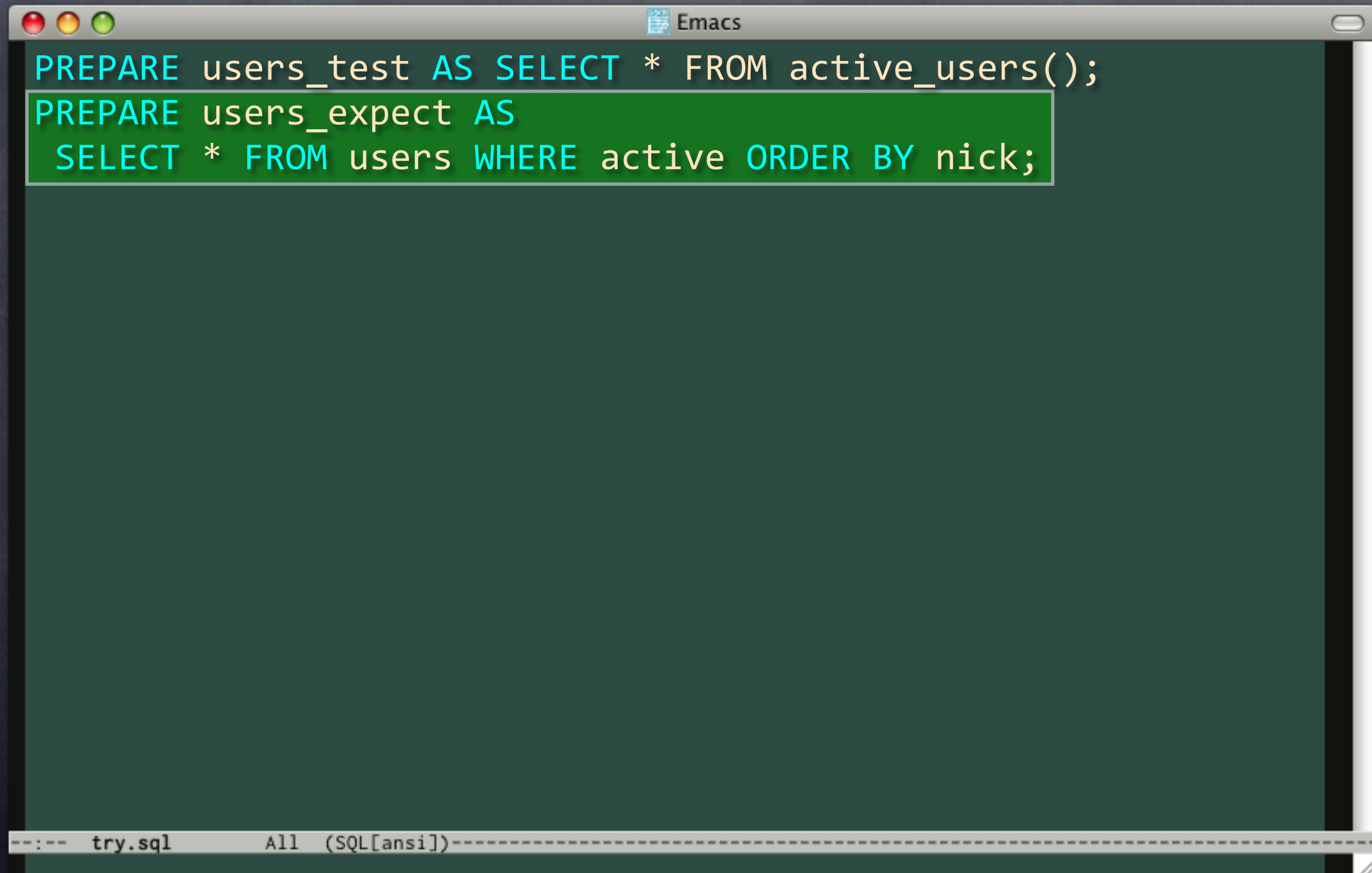
results_eq() Arguments

A screenshot of an Emacs editor window. The window title is "Emacs". The editor content shows two SQL statements. The first line is highlighted in blue: "PREPARE users_test AS SELECT * FROM active_users();". The second line is "PREPARE users_expect AS". The third line is "SELECT * FROM users WHERE active ORDER BY nick;". At the bottom of the window, there is a status bar with the text "--:-- try.sql All (SQL[ansi])-----".

```
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
SELECT * FROM users WHERE active ORDER BY nick;
```

--:-- try.sql All (SQL[ansi])-----

results_eq() Arguments

A screenshot of an Emacs editor window. The window title is "Emacs". The editor contains two lines of SQL code. The first line is "PREPARE users_test AS SELECT * FROM active_users();". The second line is "PREPARE users_expect AS SELECT * FROM users WHERE active ORDER BY nick;". The second line is highlighted with a green background. At the bottom of the window, there is a status bar that reads "--:-- try.sql All (SQL[ansi])-----".

```
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
SELECT * FROM users WHERE active ORDER BY nick;
```


results_eq() Arguments

```
Emacs
PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active ORDER BY nick;

SELECT results_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```

```
--:-- try.sql All (SQL[ansi])-----
```


results_eq() Arguments

```
Emacs
PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active ORDER BY nick;

SELECT results_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```




--:-- try.sql All (SQL[ansi])-----


```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;
```



```
PREPARE get_users_test(boolean) AS  
SELECT * FROM get_users($1);
```

```
PREPARE get_users_expect(boolean) AS  
SELECT * FROM users  
WHERE active = $1  
ORDER BY nick;
```



```
PREPARE get_users_test(boolean) AS  
SELECT * FROM get_users($1);
```

```
PREPARE get_users_expect(boolean) AS  
SELECT * FROM users  
WHERE active = $1  
ORDER BY nick;
```



```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;

SELECT results_eq(
    'EXECUTE get_users_test(true)',
    'EXECUTE get_users_expect(true)',
    'We should have active users'
);
```



```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;

SELECT results_eq(
    'EXECUTE get_users_test(true)',
    'EXECUTE get_users_expect(true)',
    'We should have active users'
);
```



```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;

SELECT results_eq(
    'EXECUTE get_users_test(true)',
    'EXECUTE get_users_expect(true)',
    'We should have active users'
);
```



```
Emacs

PREPARE get_users_test(boolean) AS
  SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
  SELECT * FROM users
  WHERE active = $1
  ORDER BY nick;

SELECT results_eq(
  'EXECUTE get_users_test(true)',
  'EXECUTE get_users_expect(true)',
  'We should have active users'
);

SELECT results_eq(
  'EXECUTE get_users_test(false)',
  'EXECUTE get_users_expect(false)',
  'We should have inactive users'
);
```



```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;

SELECT results_eq(
    'EXECUTE get_users_test(true)',
    'EXECUTE get_users_expect(true)',
    'We should have active users'
);

SELECT results_eq(
    'EXECUTE get_users_test(false)',
    'EXECUTE get_users_expect(false)',
    'We should have inactive users'
);
```



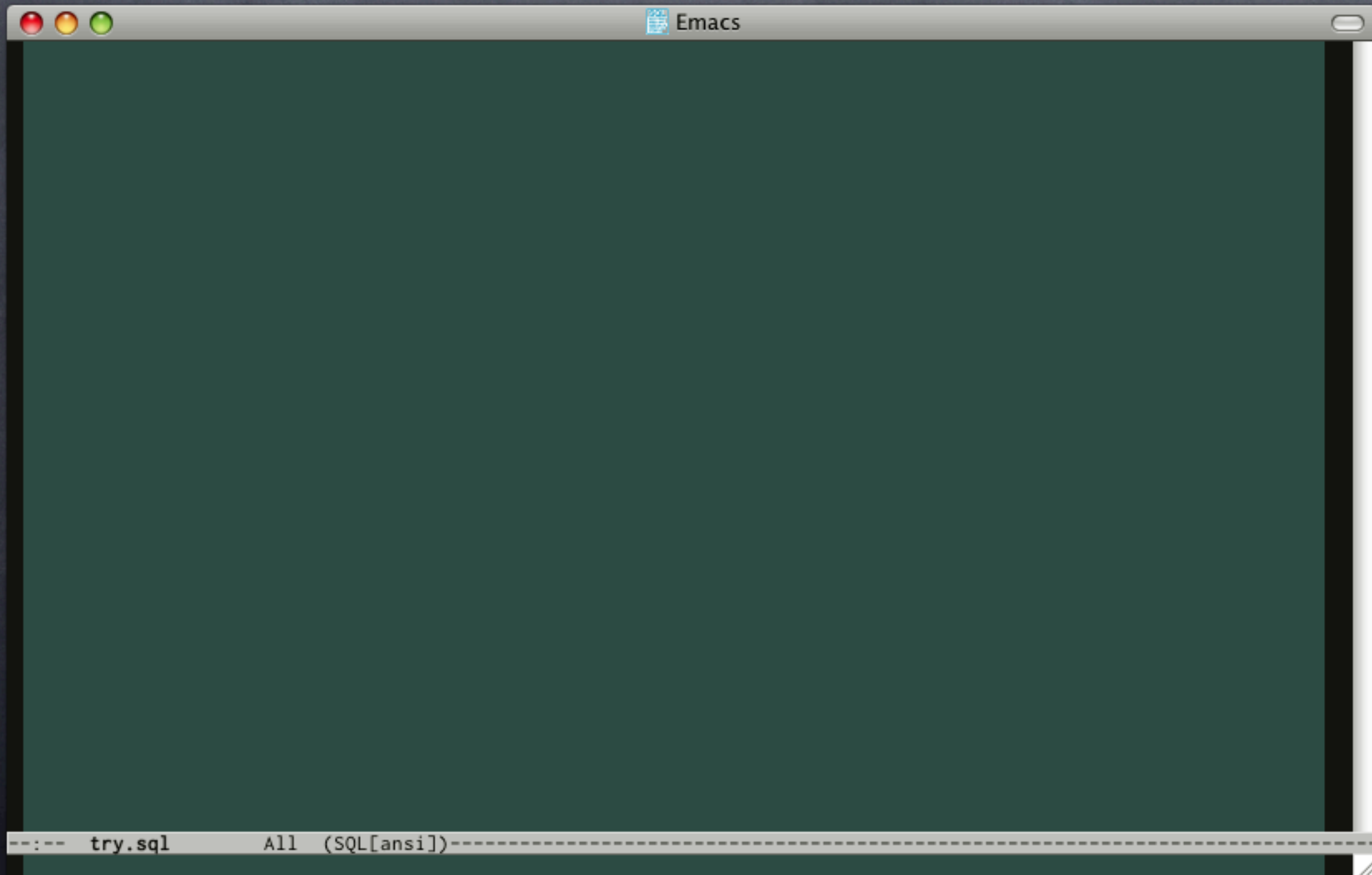
```
PREPARE get_users_test(boolean) AS
SELECT * FROM get_users($1);

PREPARE get_users_expect(boolean) AS
SELECT * FROM users
WHERE active = $1
ORDER BY nick;

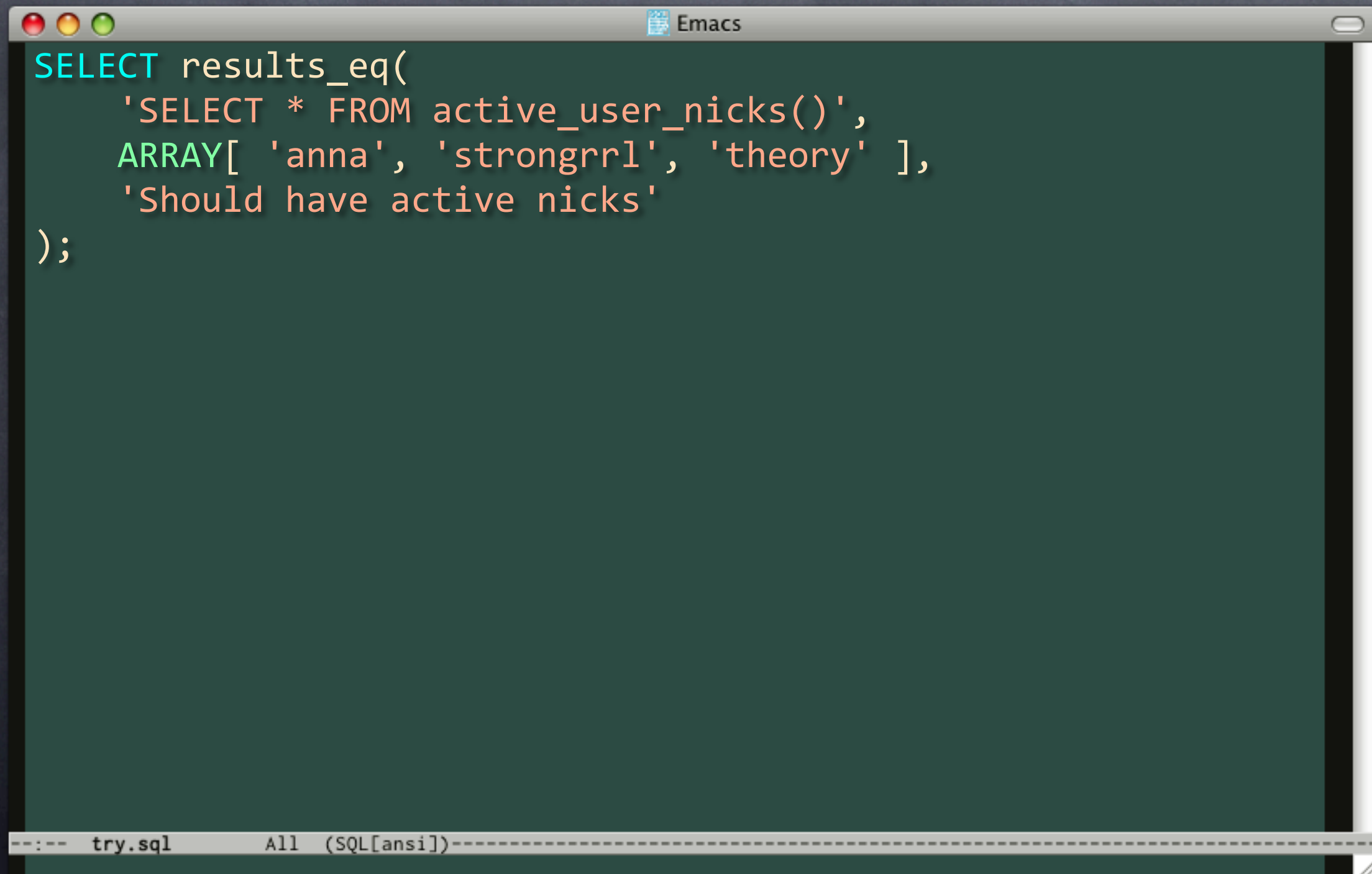
SELECT results_eq(
    'EXECUTE get_users_test(true)',
    'EXECUTE get_users_expect(true)',
    'We should have active users'
);

SELECT results_eq(
    'EXECUTE get_users_test(false)',
    'EXECUTE get_users_expect(false)',
    'We should have inactive users'
);
```


results_eq() Single Column & Array



results_eq() Single Column & Array

The image shows a screenshot of an Emacs editor window. The window title is "Emacs". The editor content is a SQL query using the results_eq() function. The code is as follows:

```
SELECT results_eq(  
  'SELECT * FROM active_user_nicks()',  
  ARRAY[ 'anna', 'strongrrl', 'theory' ],  
  'Should have active nicks'  
);
```

The status bar at the bottom of the window shows "--:-- try.sql All (SQL[ansi])-----".

results_eq() Single Column & Array

```
Emacs
SELECT results_eq(
  'SELECT * FROM active_user_nicks()',
  ARRAY[ 'anna', 'strongrrl', 'theory' ],
  'Should have active nicks'
);

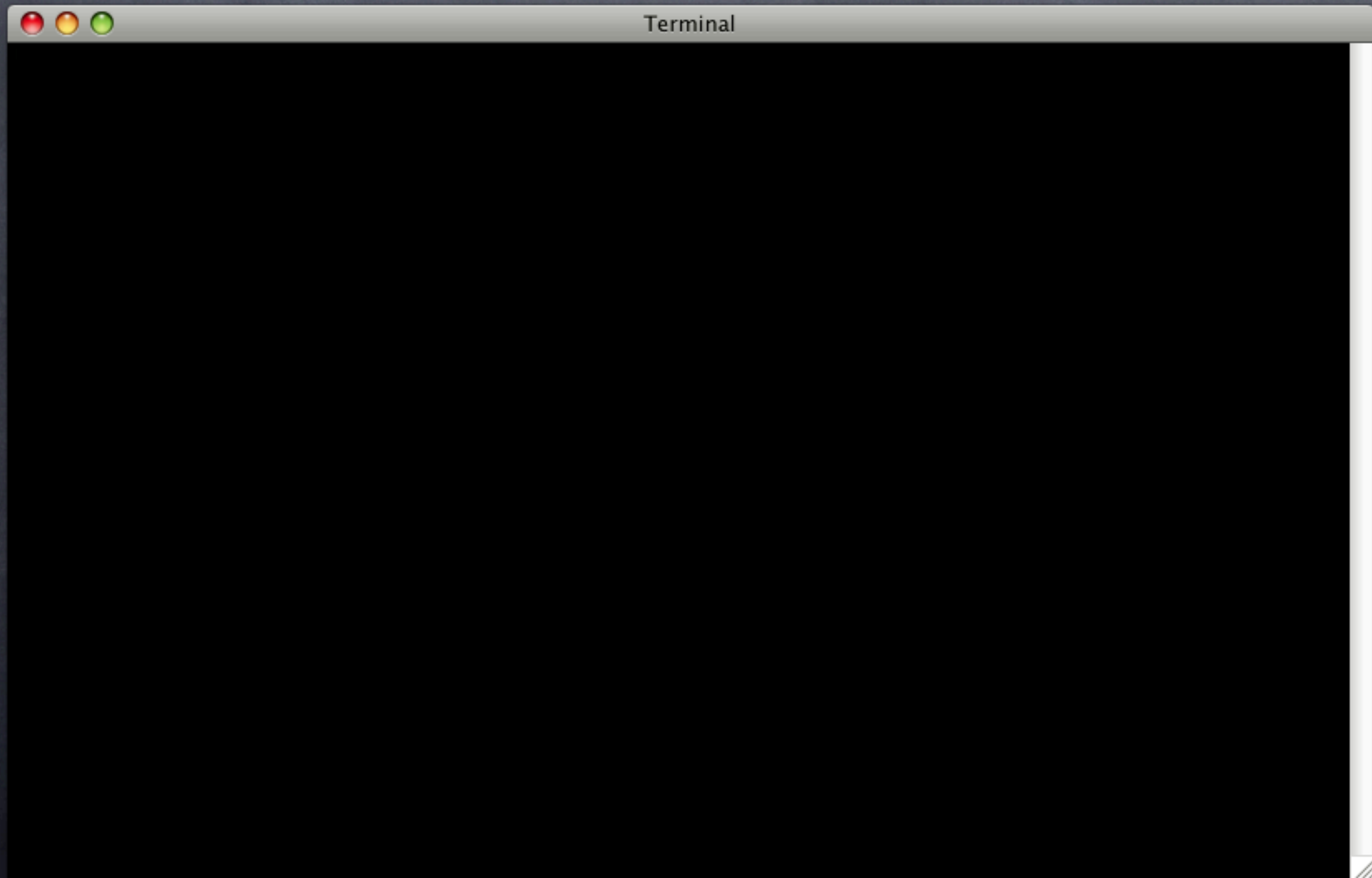
---:-- try.sql All (SQL[ansi])-----
```


results_eq() Single Column & Array

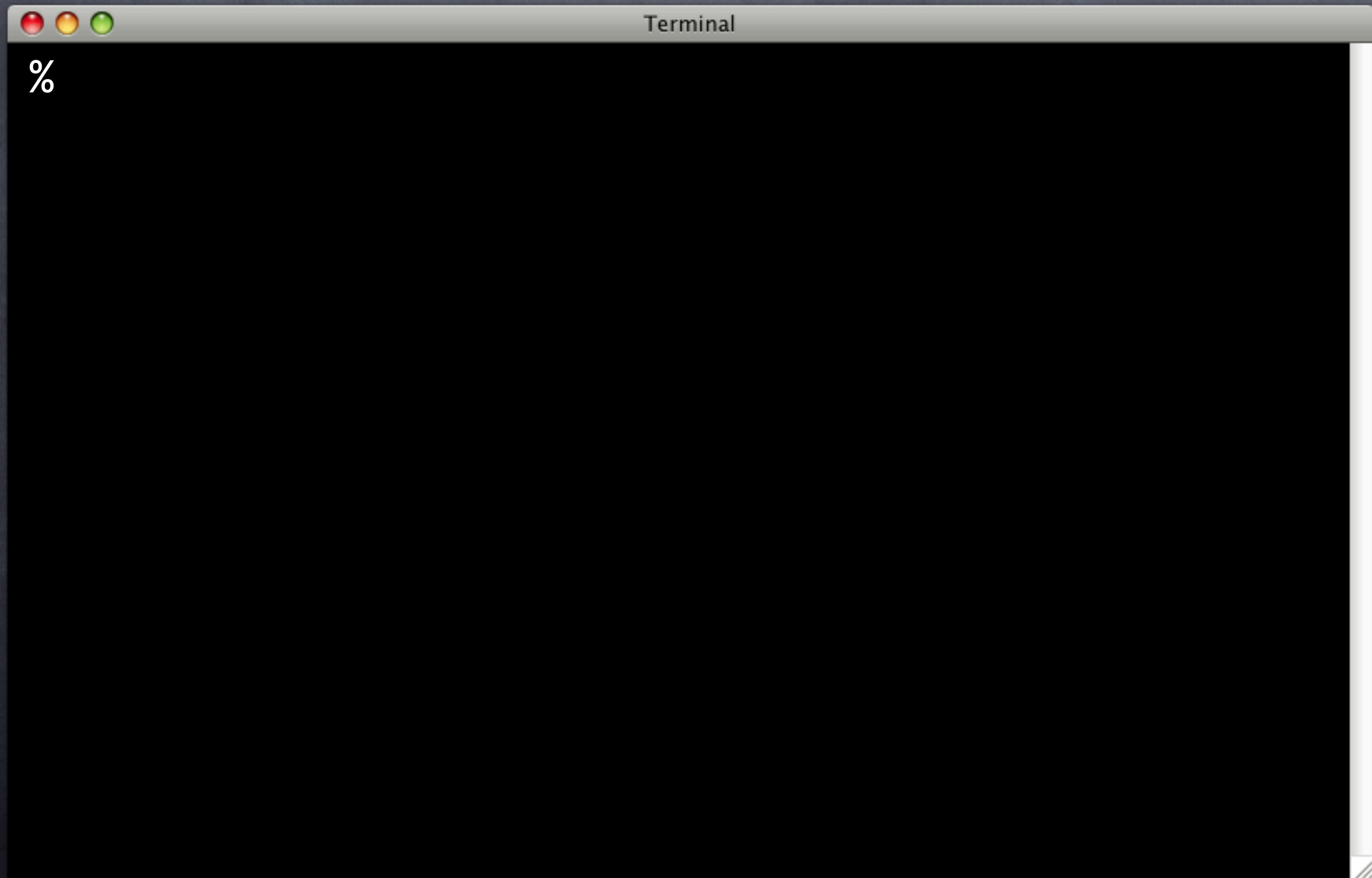
```
Emacs
SELECT results_eq(
  'SELECT * FROM active_user_nicks()',
  ARRAY[ 'anna', 'strongrr1', 'theory' ],
  'Should have active nicks'
);

--- try.sql All (SQL[ansi])
```


Testing Results



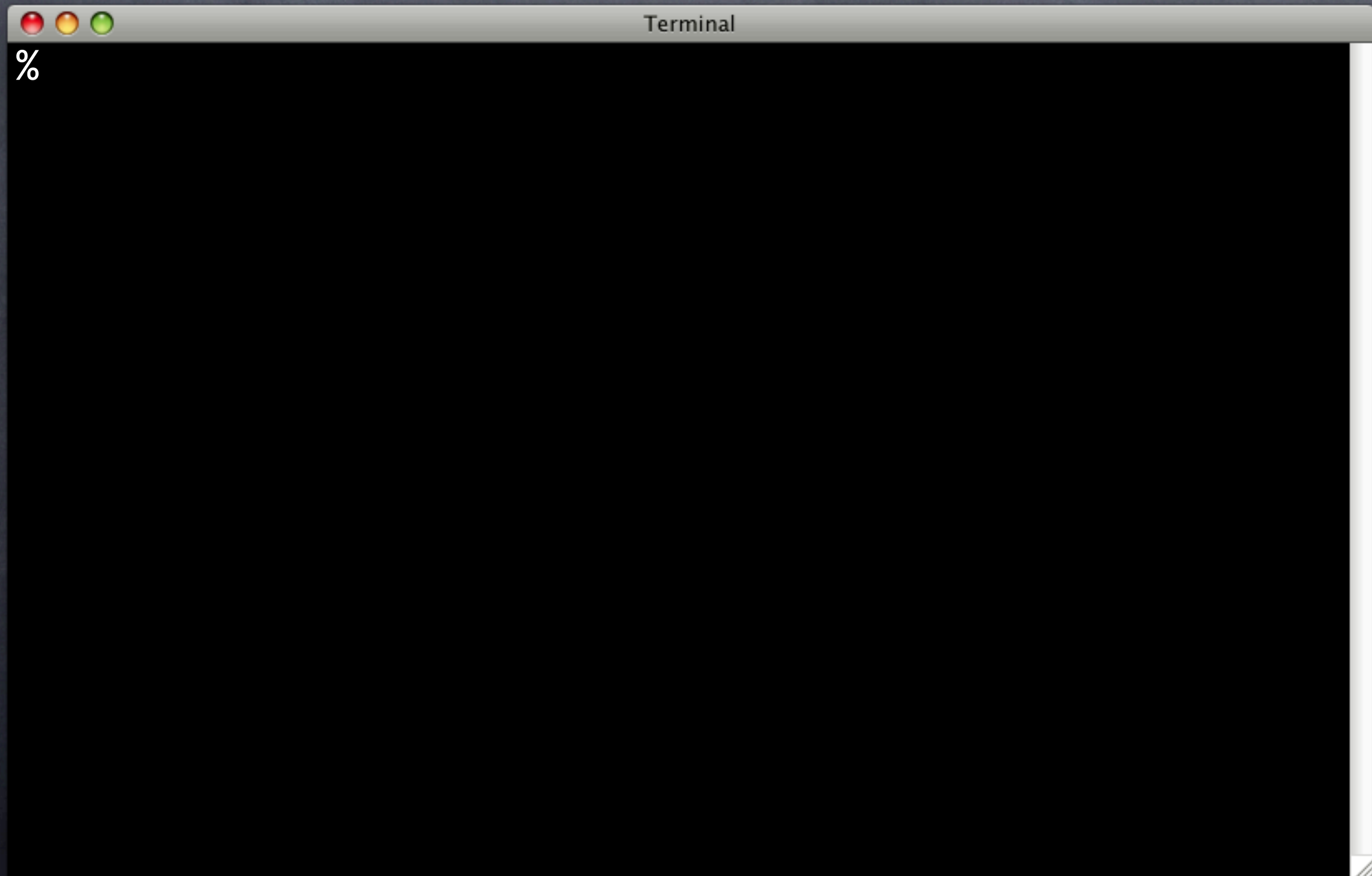
Testing Results



Testing Results

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() should return active users
ok 2 - active_users() should return active users
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
ok
All tests successful.
```


Differing Rows Diagnostics



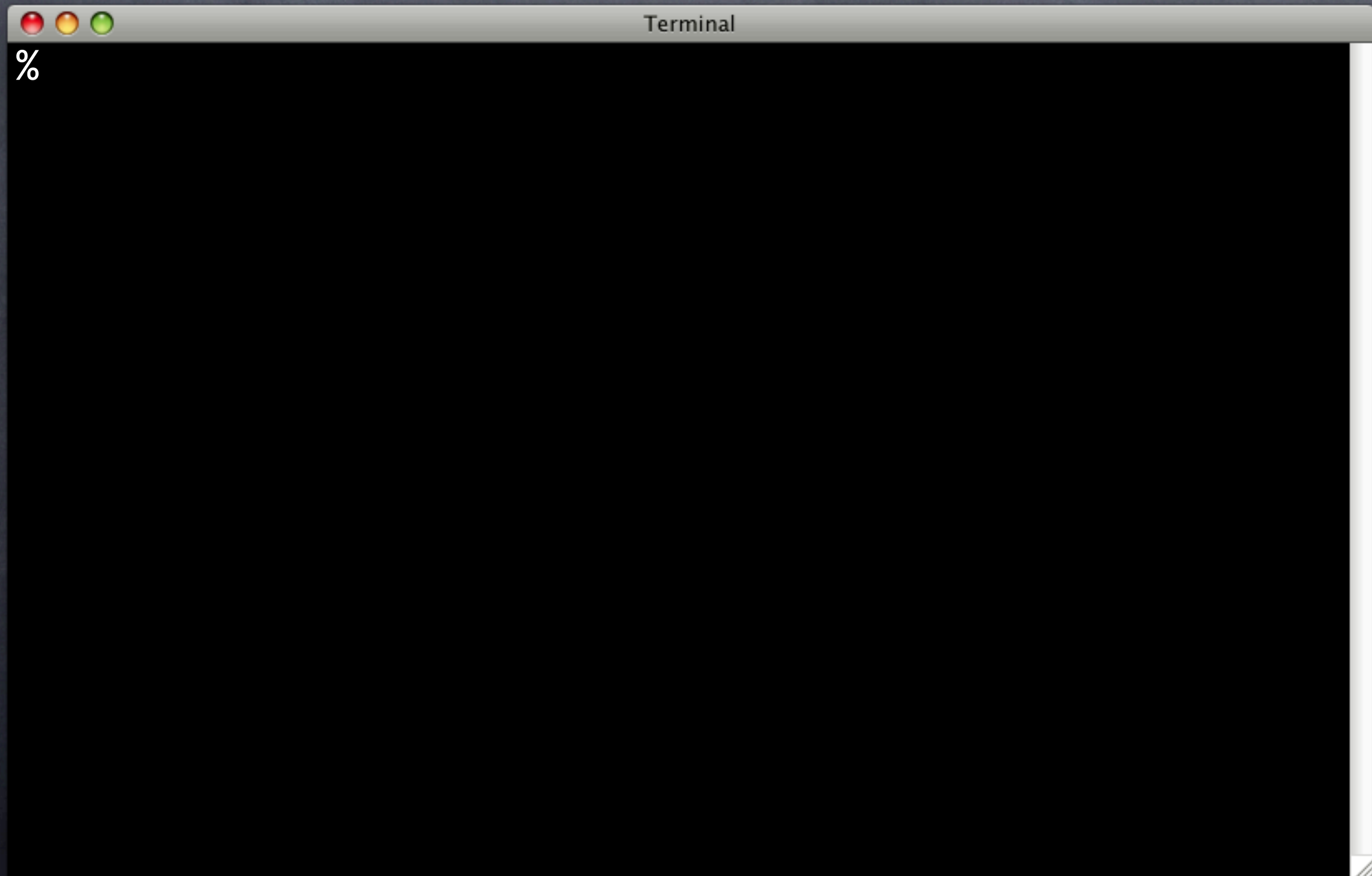
Differing Rows Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() should return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Results differ beginning at row 2:
#       have: (strongr1,portland,"Julie Wheeler",t)
#       want: (strongr1,design,"Julie Wheeler",t)
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Differing Rows Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() should return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Results differ beginning at row 2:
#       have: (strongr1,portland,"Julie Wheeler",t)
#       want: (strongr1,design,"Julie Wheeler",t)
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Differing Columns Diagnostics



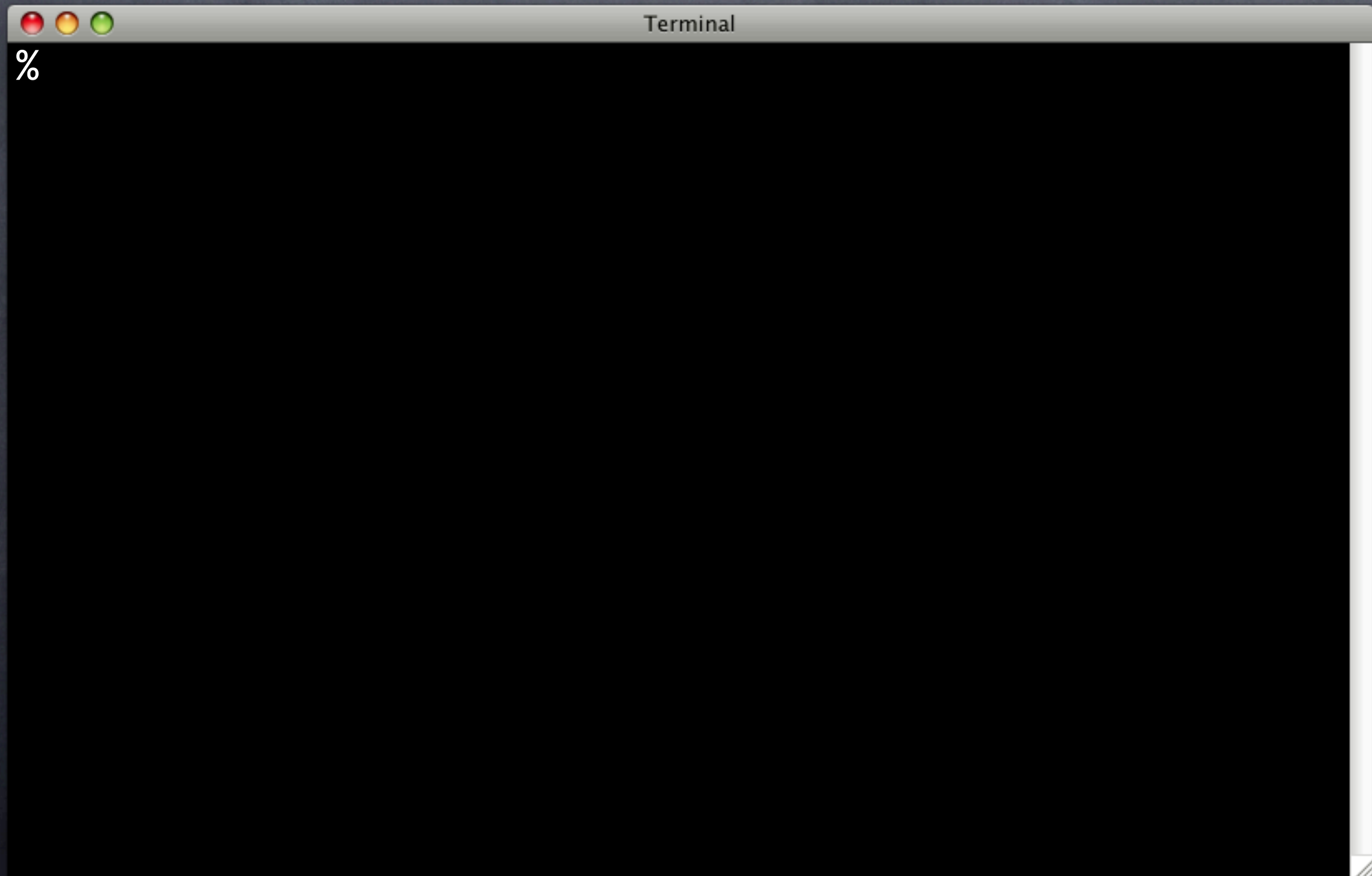
Differing Columns Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Columns differ between queries:
#         have: (anna,yddad,"Anna Wheeler",t)
#         want: (anna,yddad,"Anna Wheeler")
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Differing Columns Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Columns differ between queries:
#       have: (anna,yddad,"Anna Wheeler",t)
#       want: (anna,yddad,"Anna Wheeler")
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Missing Row Diagnostics



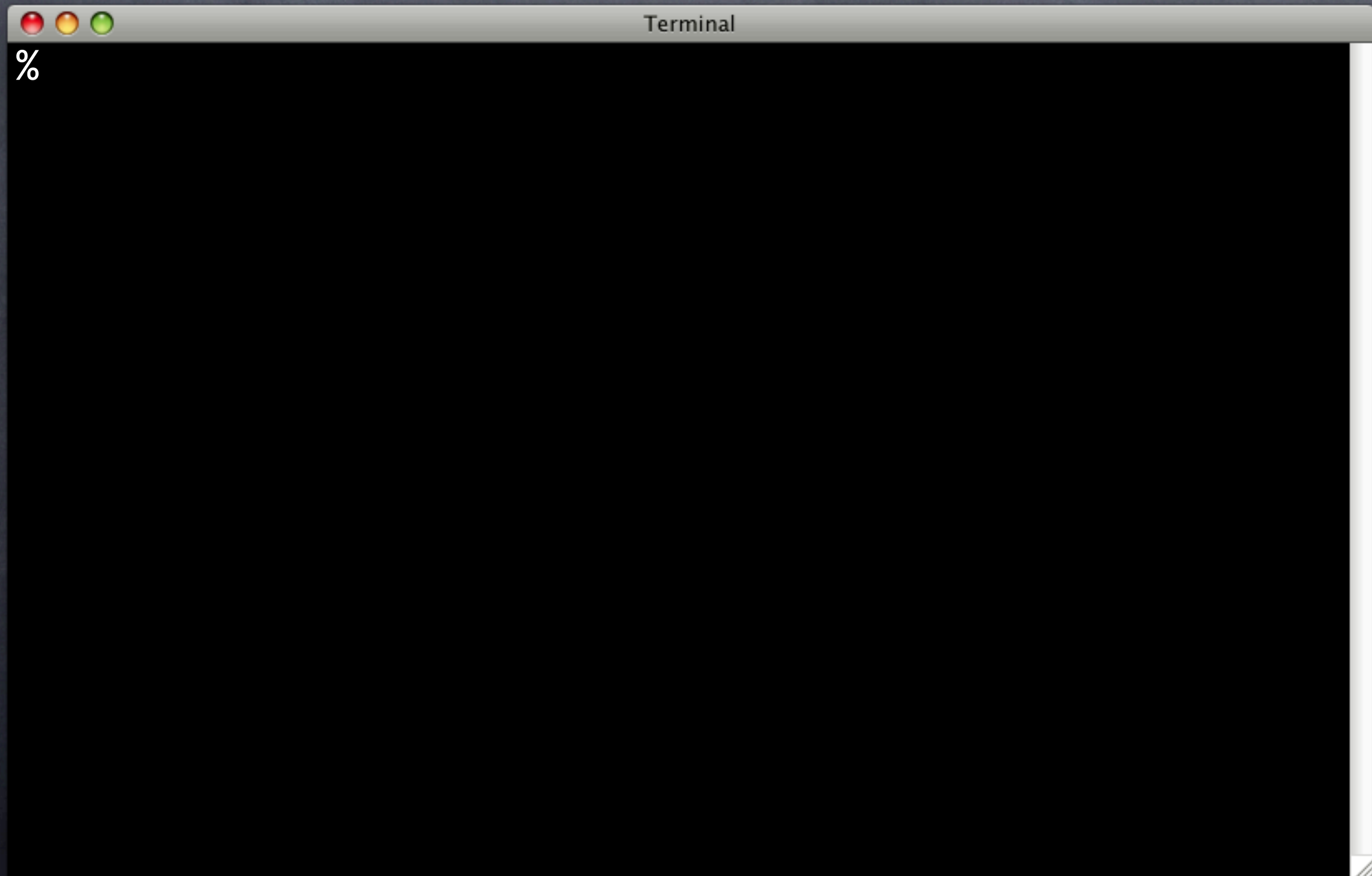
Missing Row Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Results differ beginning at row 3:
#         have: (theory,s3kr1t,"David Wheeler",t)
#         want: NULL
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Missing Row Diagnostics

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Results differ beginning at row 3:
#       have: (theory,s3kr1t,"David Wheeler",t)
#       want: NULL
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Diagnostics for Differing Data Types



Diagnostics for Differing Data Types

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Columns differ between queries:
#         have: (anna,yddad,"Anna Wheeler",t)
#         want: (anna,yddad,"Anna Wheeler",t)
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Diagnostics for Differing Data Types

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Columns differ between queries:
#       have: (anna,yddad,"Anna Wheeler",t)
#       want: (anna,yddad,"Anna Wheeler",t)
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```


Diagnostics for Differing Data Types

```
Terminal
% pg_prove -v -d try results_eq.sql
results_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Columns differ between queries:
#       have: (anna,yddad,"Anna Wheeler",t)
#       want: (anna,yddad,"Anna Wheeler",t)
ok 3 - We should have users
ok 4 - We should have active users
ok 5 - We should have inactive users
ok 6 - Should have active nicks
1..6
# Looks like you failed 1 test of 6
Failed 1/6 subtests
```

Say
what?

Testing Sets

Testing Sets

- **When order does not matter**

Testing Sets

- **When order does not matter**
- **When duplicate tuples do not matter**

Testing Sets

- **When order does not matter**
- **When duplicate tuples do not matter**
- **Use queries**

Testing Sets

- **When order does not matter**
- **When duplicate tuples do not matter**
- **Use queries**
- **Or prepared statement names**

Testing Bags

Testing Bags

- **When order does not matter**

Testing Bags

- **When order does not matter**
- **Duplicates matter**

Testing Bags

- **When order does not matter**
- **Duplicates matter**
- **Use queries**

Testing Bags

- **When order does not matter**
- **Duplicates matter**
- **Use queries**
- **Or prepared statement names**



--:-- try.sql All (SQL[ansi])-----


```
SELECT set_eq(
  'SELECT * FROM active_users()',
  'SELECT * FROM users WHERE active',
  'active_users() should return active users'
);

SELECT set_eq(
  'SELECT * FROM active_users()',
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),
           ('strongrrl', 'design', 'Julie Wheeler', true),
           ('theory', 's3kr1t', 'David Wheeler', true)
  $$,
  'active_users() should return active users'
);

PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active;

SELECT bag_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```



```
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active',  
  'active_users() should return active users'  
);  
  
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),  
           ('strongrrl', 'design', 'Julie Wheeler', true),  
           ('theory', 's3kr1t', 'David Wheeler', true)  
  $$,  
  'active_users() should return active users'  
);  
  
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
  SELECT * FROM users WHERE active;  
  
SELECT bag_eq(  
  'users_test',  
  'users_expect',  
  'We should have users'  
);
```



```
SELECT set_eq(
  'SELECT * FROM active_users()',
  'SELECT * FROM users WHERE active',
  'active_users() should return active users'
);

SELECT set_eq(
  'SELECT * FROM active_users()',
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),
           ('strongrrl', 'design', 'Julie Wheeler', true),
           ('theory', 's3kr1t', 'David Wheeler', true)
  $$,
  'active_users() should return active users'
);

PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active;

SELECT bag_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```



```
SELECT set_eq(
  'SELECT * FROM active_users()',
  'SELECT * FROM users WHERE active',
  'active_users() should return active users'
);

SELECT set_eq(
  'SELECT * FROM active_users()',
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),
           ('strongrrl', 'design', 'Julie Wheeler', true),
           ('theory', 's3kr1t', 'David Wheeler', true)
  $$,
  'active_users() should return active users'
);

PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active;

SELECT bag_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```



```
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active',  
  'active_users() should return active users'  
);
```

```
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),  
           ('strongrrl', 'design', 'Julie Wheeler', true),  
           ('theory', 's3kr1t', 'David Wheeler', true)  
  $$,  
  'active_users() should return active users'  
);
```

```
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
  SELECT * FROM users WHERE active;
```

```
SELECT bag_eq(  
  'users_test',  
  'users_expect',  
  'We should have users'  
);
```



```
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  'SELECT * FROM users WHERE active',  
  'active_users() should return active users'  
);
```

```
SELECT set_eq(  
  'SELECT * FROM active_users()',  
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),  
           ('strongrrl', 'design', 'Julie Wheeler', true),  
           ('theory', 's3kr1t', 'David Wheeler', true)  
  $$,  
  'active_users() should return active users'  
);
```

```
PREPARE users_test AS SELECT * FROM active_users();  
PREPARE users_expect AS  
  SELECT * FROM users WHERE active;
```

```
SELECT bag_eq(  
  'users_test',  
  'users_expect',  
  'We should have users'  
);
```



```
SELECT set_eq(
  'SELECT * FROM active_users()',
  'SELECT * FROM users WHERE active',
  'active_users() should return active users'
);

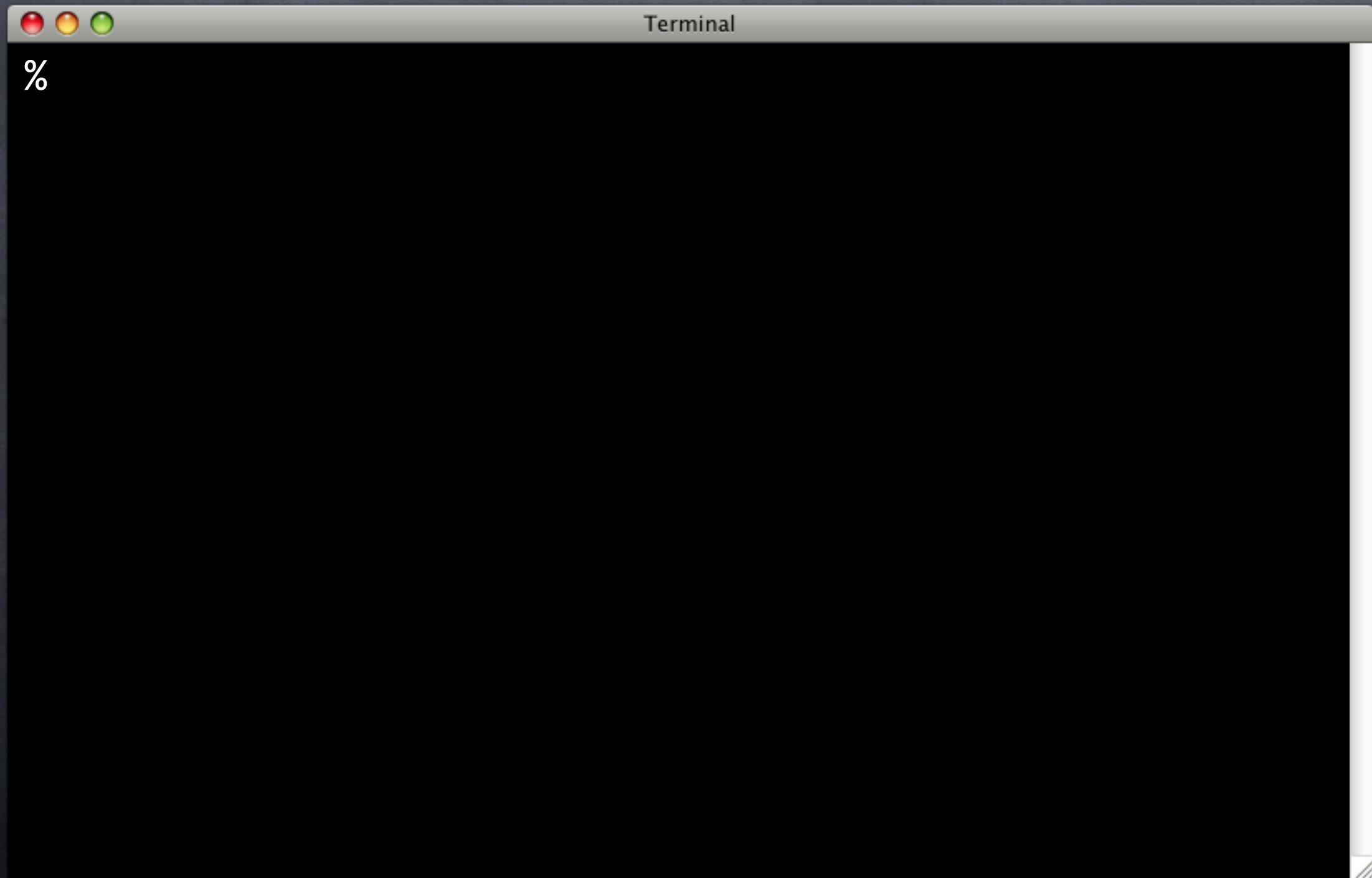
SELECT set_eq(
  'SELECT * FROM active_users()',
  $$VALUES ('anna', 'yddad', 'Anna Wheeler', true),
           ('strongrrl', 'design', 'Julie Wheeler', true),
           ('theory', 's3kr1t', 'David Wheeler', true)
  $$,
  'active_users() should return active users'
);

PREPARE users_test AS SELECT * FROM active_users();
PREPARE users_expect AS
  SELECT * FROM users WHERE active;

SELECT bag_eq(
  'users_test',
  'users_expect',
  'We should have users'
);
```

No ORDER BY

Differing Rows Diagnostics



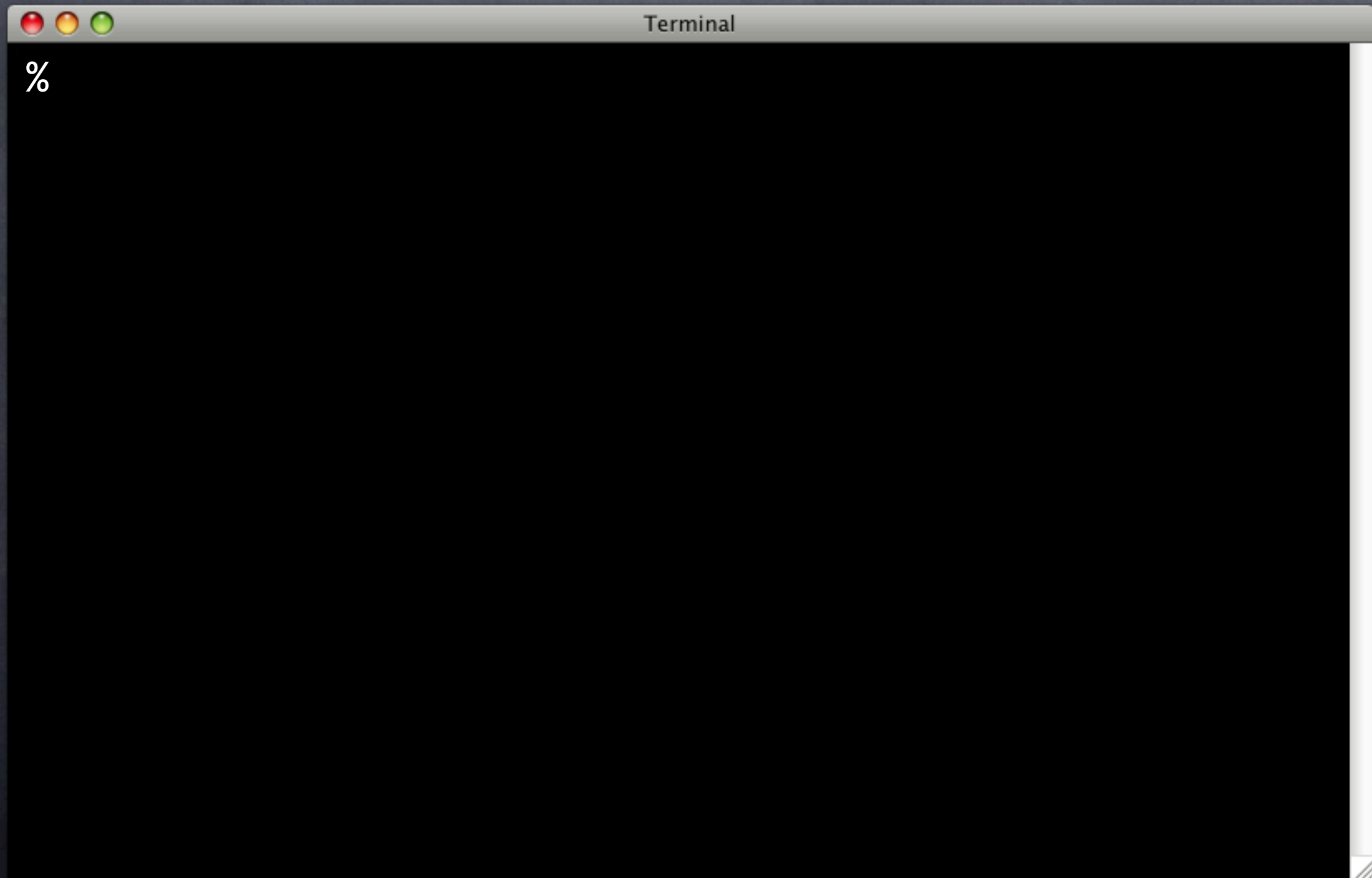
Differing Rows Diagnostics

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2 "active_users() return active users"
#   Extra records:
#     (87, Jackson)
#     (1, Jacob)
#   Missing records:
#     (44, Anna)
#     (86, Angelina)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Differing Rows Diagnostics

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2 "active_users() return active users"
#      Extra records:
#      (87, Jackson)
#      (1, Jacob)
#      Missing records:
#      (44, Anna)
#      (86, Angelina)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Diagnostics for Differing Data Types



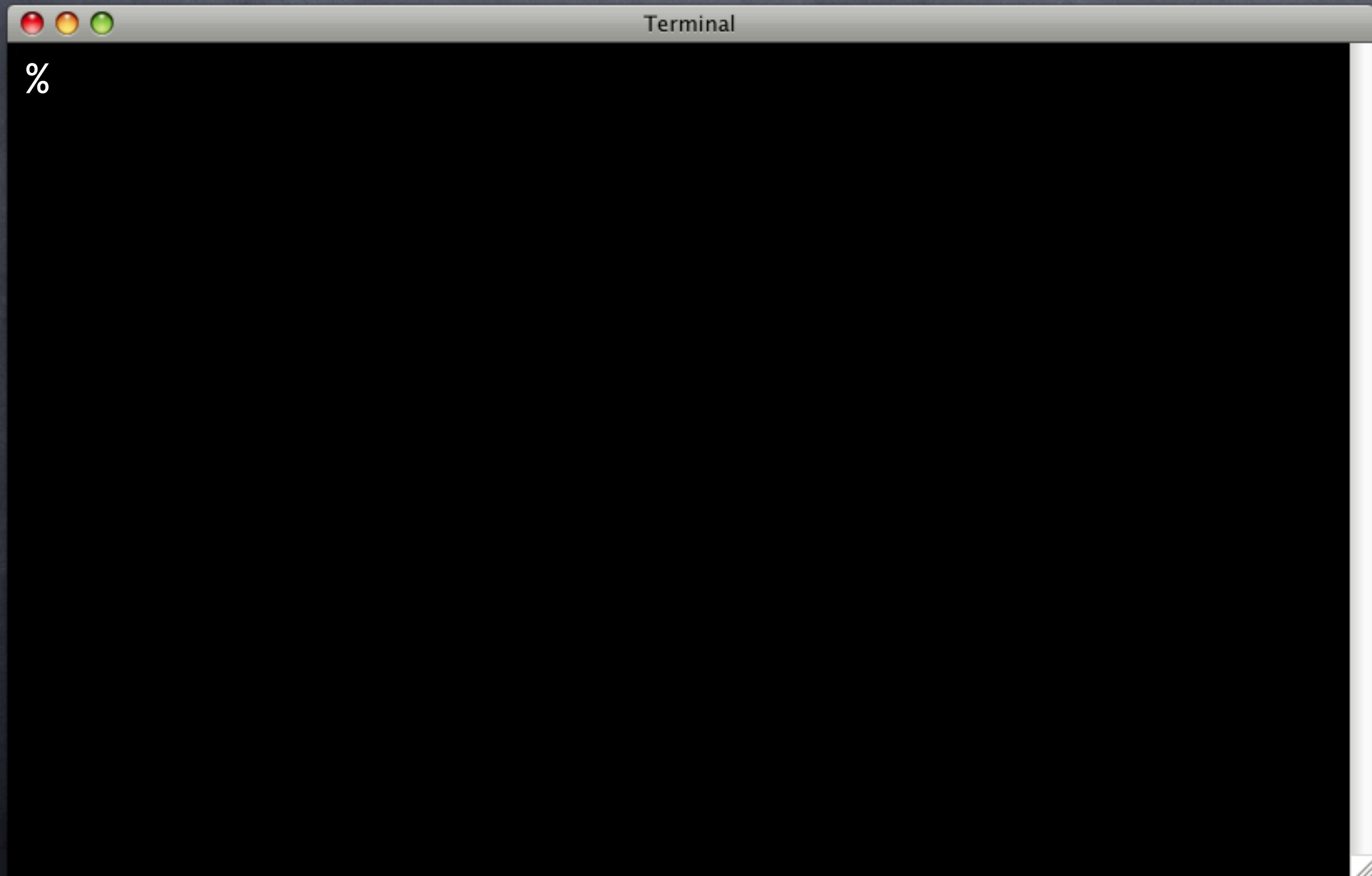
Diagnostics for Differing Data Types

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Columns differ between queries:
#         have: (integer,text)
#         want: (text,integer)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Diagnostics for Differing Data Types

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#       Columns differ between queries:
#       have: (integer,text)
#       want: (text,integer)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Diagnostics for Different Numbers of Columns



Diagnostics for Different Numbers of Columns

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Columns differ between queries:
#     have: (integer)
#     want: (text,integer)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Diagnostics for Different Numbers of Columns

```
Terminal
% pg_prove -v -d try set_eq.sql
set_eq.sql ..
ok 1 - active_users() return active users
not ok 2 - active_users() return active users
# Failed test 2: "active_users() return active users"
#     Columns differ between queries:
#     have: (integer)
#     want: (text,integer)
ok 3 - We should have users
1..3
# Looks like you failed 1 test of 3
Failed 1/3 subtests
```


Structural Testing

Structural Testing

- **Validate presence/absence of objects**

Structural Testing

- Validate presence/absence of objects
- *_are()

Structural Testing

- **Validate presence/absence of objects**
- ***_are()**
- **Specify complete lists of objects**

Structural Testing

- **Validate presence/absence of objects**
- ***_are()**
- **Specify complete lists of objects**
- **Useful failure diagnostics**



--:-- try.sql All (SQL[ansi])-----


```
SELECT tablespaces_are(  
    ARRAY[ 'dbspace', 'indexspace' ]  
);
```



```
SELECT tablespaces_are(  
    ARRAY[ 'dbspace', 'indexspace' ]  
);  
  
SELECT schemas_are(  
    ARRAY[ 'public', 'contrib', 'biz' ]  
);
```



```
SELECT tablespaces_are(  
    ARRAY[ 'dbspace', 'indexspace' ]  
);  
  
SELECT schemas_are(  
    ARRAY[ 'public', 'contrib', 'biz' ]  
);  
  
SELECT tables_are(  
    'biz',  
    ARRAY[ 'users', 'widgets' ]  
);
```



```
SELECT tablespaces_are(  
    ARRAY[ 'dbspace', 'indexspace' ]  
);  
  
SELECT schemas_are(  
    ARRAY[ 'public', 'contrib', 'biz' ]  
);  
  
SELECT tables_are(  
    'biz',  
    ARRAY[ 'users', 'widgets' ]  
);  
  
SELECT views_are(  
    'biz',  
    ARRAY[ 'user_list', 'widget_list' ]  
);
```



```
SELECT tablespaces_are(  
    ARRAY[ 'dbspace', 'indexspace' ]  
);  
  
SELECT schemas_are(  
    ARRAY[ 'public', 'contrib', 'biz' ]  
);  
  
SELECT tables_are(  
    'biz',  
    ARRAY[ 'users', 'widgets' ]  
);  
  
SELECT views_are(  
    'biz',  
    ARRAY[ 'user_list', 'widget_list' ]  
);  
  
SELECT sequences_are(  
    'biz',  
    ARRAY[ 'users_id_seq', 'widgets_id_seq' ]  
);
```




--:-- try.sql All (SQL[ansi])-----


```
SELECT indexes_are(  
    'biz', 'users',  
    ARRAY[ 'users_pkey' ]  
);
```



```
SELECT indexes_are(  
    'biz', 'users',  
    ARRAY[ 'users_pkey' ]  
);  
  
SELECT functions_are(  
    'biz',  
    ARRAY[ 'get_users', 'get_widgets' ]  
);
```

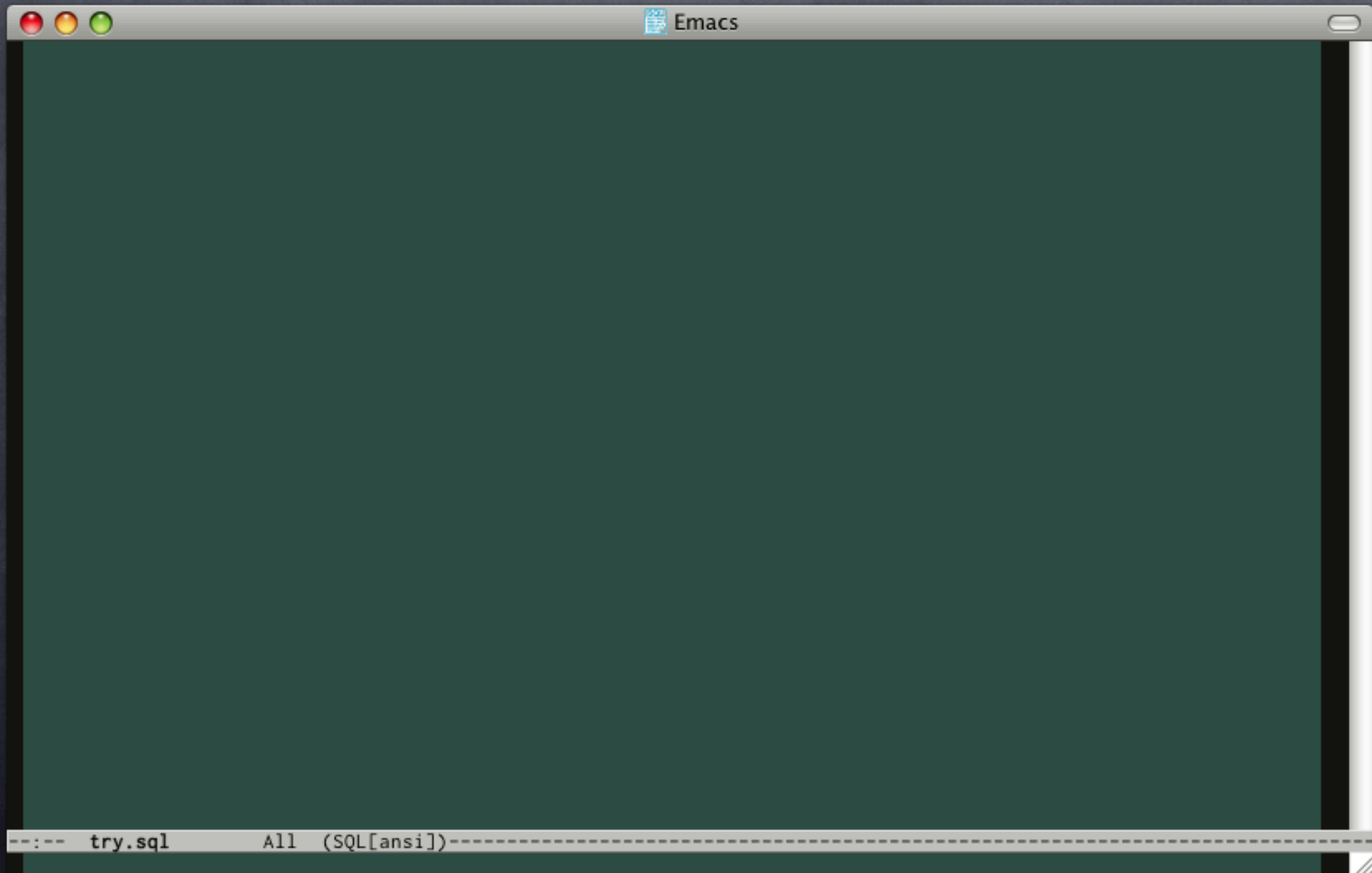


```
SELECT indexes_are(  
    'biz', 'users',  
    ARRAY[ 'users_pkey' ]  
);  
  
SELECT functions_are(  
    'biz',  
    ARRAY[ 'get_users', 'get_widgets' ]  
);  
  
SELECT users_are(  
    ARRAY[ 'postgres', 'david', 'bric' ]  
);
```

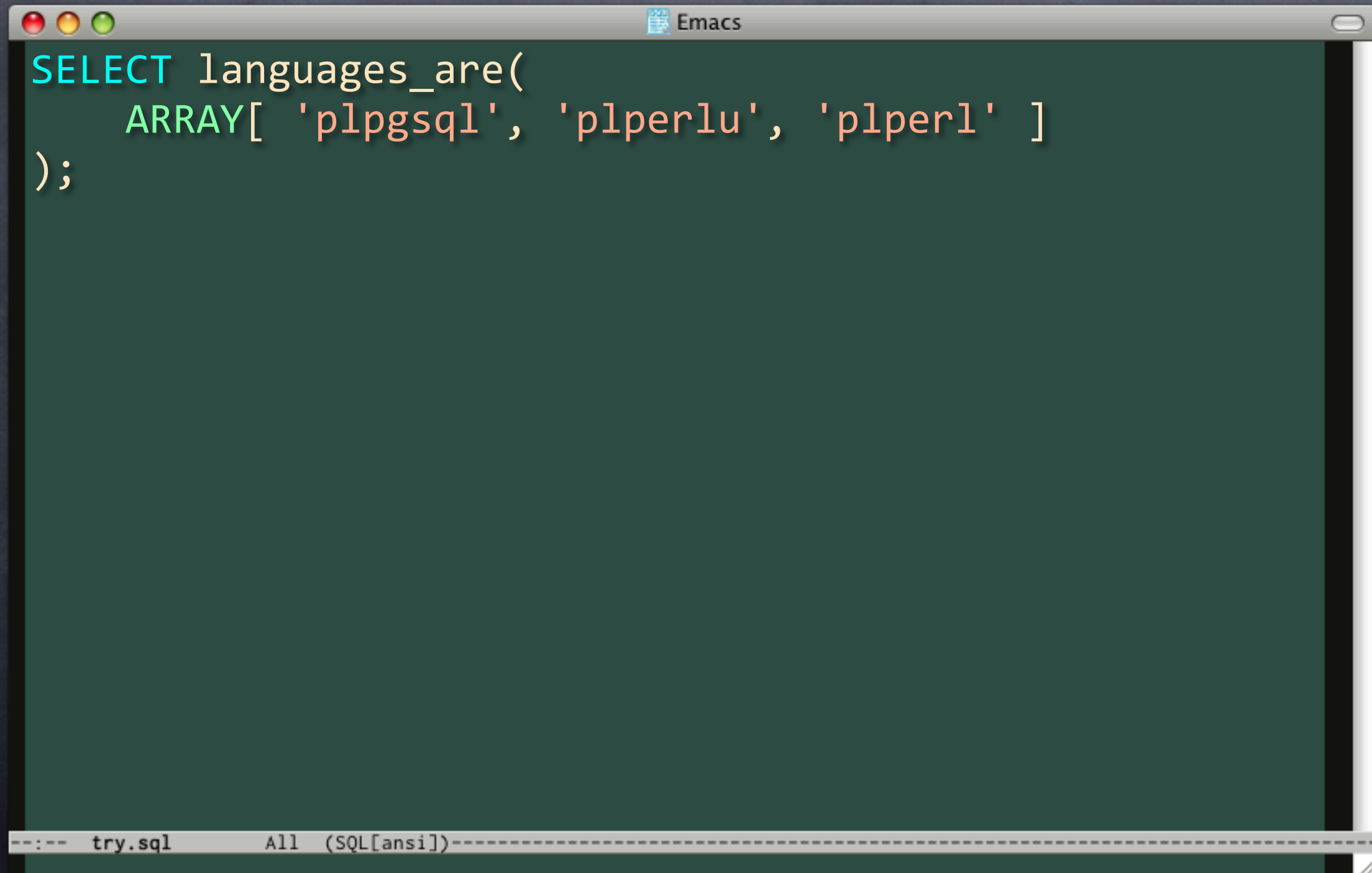


```
SELECT indexes_are(  
    'biz', 'users',  
    ARRAY[ 'users_pkey' ]  
);  
  
SELECT functions_are(  
    'biz',  
    ARRAY[ 'get_users', 'get_widgets' ]  
);  
  
SELECT users_are(  
    ARRAY[ 'postgres', 'david', 'bric' ]  
);  
  
SELECT groups_are(  
    ARRAY[ 'admins', 'devs' ]  
);
```


Structural Testing



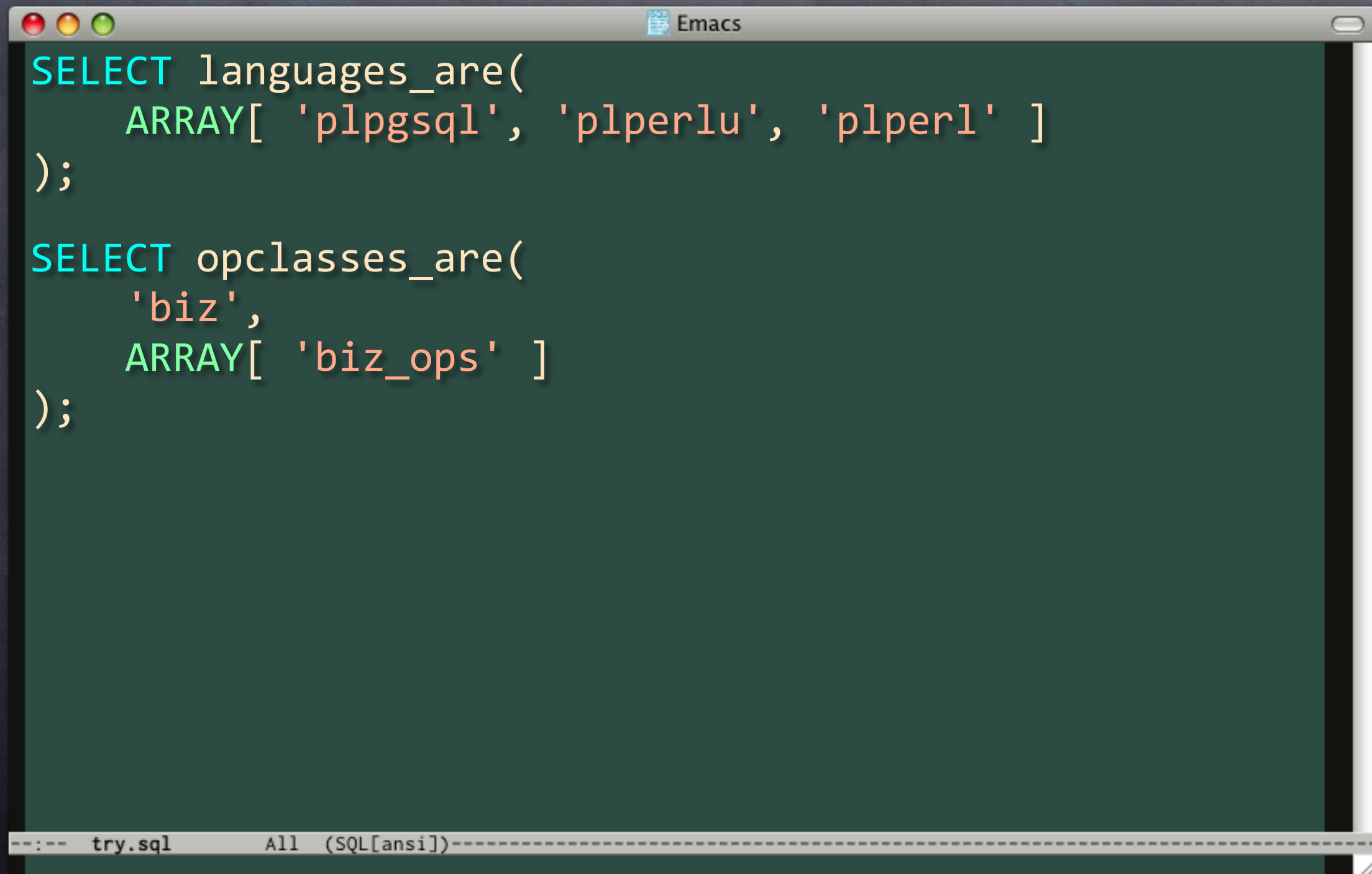
Structural Testing

An Emacs editor window with a dark green background. The window title bar shows the Emacs logo and the text "Emacs". The code is as follows:

```
SELECT languages_are(  
    ARRAY[ 'plpgsql', 'plperlu', 'plperl' ]  
);
```

The status bar at the bottom of the window displays "--:-- try.sql All (SQL[ansi])-----".

Structural Testing



```
SELECT languages_are(  
    ARRAY[ 'plpgsql', 'plperlu', 'plperl' ]  
);  
  
SELECT opclasses_are(  
    'biz',  
    ARRAY[ 'biz_ops' ]  
);
```

try.sql All (SQL[ansi])

Structural Testing

```
Emacs
SELECT languages_are(
  ARRAY[ 'plpgsql', 'plperlu', 'plperl' ]
);

SELECT opclasses_are(
  'biz',
  ARRAY[ 'biz_ops' ]
);

SELECT rules_are(
  'biz', 'users',
  ARRAY[ 'ins_me', 'upd_me' ]
);

---:-- try.sql      All (SQL[ansi])-----
```




%


```
% pg_prove -v -d try schema.sql
schema.sql ..
ok 1 - There should be the correct schemas
not ok 2 - Schema biz should have the correct tables
# Failed test 2: "Schema biz should have the correct
tables"
#      Extra tables:
#          mallots
#          __test_table
#      Missing tables:
#          users
#          widgets
ok 3 - Schema biz should have the correct views
ok 4 - Schema biz should have the correct sequences
ok 5 - Table biz.users should have the correct indexes
ok 6 - Schema biz should have the correct functions
ok 7 - There should be the correct users
ok 8 - There should be the correct groups
ok 9 - There should be the correct procedural languages
ok 10 - Schema biz should have the correct operator classes
ok 11 - Relation biz.users should have the correct rules
1..11
# Looks like you failed 1 test of 11
Failed 1/11 subtests
```



```
% pg_prove -v -d try schema.sql
schema.sql ..
ok 1 - There should be the correct schemas
not ok 2 - Schema biz should have the correct tables
# Failed test 2: "Schema biz should have the correct
tables"
#      Extra tables:
#          mallots
#          __test_table
#      Missing tables:
#          users
#          widgets
ok 3 - Schema biz should have the correct views
ok 4 - Schema biz should have the correct sequences
ok 5 - Table biz.users should have the correct indexes
ok 6 - Schema biz should have the correct functions
ok 7 - There should be the correct users
ok 8 - There should be the correct groups
ok 9 - There should be the correct procedural languages
ok 10 - Schema biz should have the correct operator classes
ok 11 - Relation biz.users should have the correct rules
1..11
# Looks like you failed 1 test of 11
Failed 1/11 subtests
```


Schema Testing

Schema Testing

- **Validate presence of objects**

Schema Testing

- **Validate presence of objects**
- **Validate object interfaces**

Schema Testing

- **Validate presence of objects**
- **Validate object interfaces**
- **Prevent others from changing schema**

Schema Testing

- **Validate presence of objects**
- **Validate object interfaces**
- **Prevent others from changing schema**
- **Maintain a consistent interface**



--:-- try.sql All (SQL[ansi])-----


```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(    'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column(    'users', 'user_id' );
SELECT col_type_is(   'users', 'user_id', 'integer' );
SELECT col_not_null(  'users', 'user_id' );
SELECT col_is_pk(     'users', 'user_id' );

SELECT has_column(    'users', 'birthdate' );
SELECT col_type_is(   'users', 'birthdate', 'date' );
SELECT col_is_null(   'users', 'birthdate' );

SELECT has_column(    'users', 'state' );
SELECT col_type_is(   'users', 'state', 'text' );
SELECT col_not_null(  'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk( 'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk( 'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(    'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(    'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(    'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(   'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```



```
BEGIN;
SET search_path TO public, tap;
SELECT plan(13);

SELECT has_table( 'users' );
SELECT has_pk(    'users' );

SELECT has_column( 'users', 'user_id' );
SELECT col_type_is( 'users', 'user_id', 'integer' );
SELECT col_not_null( 'users', 'user_id' );
SELECT col_is_pk(    'users', 'user_id' );

SELECT has_column( 'users', 'birthdate' );
SELECT col_type_is( 'users', 'birthdate', 'date' );
SELECT col_is_null( 'users', 'birthdate' );

SELECT has_column( 'users', 'state' );
SELECT col_type_is( 'users', 'state', 'text' );
SELECT col_not_null( 'users', 'state' );
SELECT col_default_is( 'users', 'state', 'active' );

SELECT * FROM finish();
ROLLBACK;
```

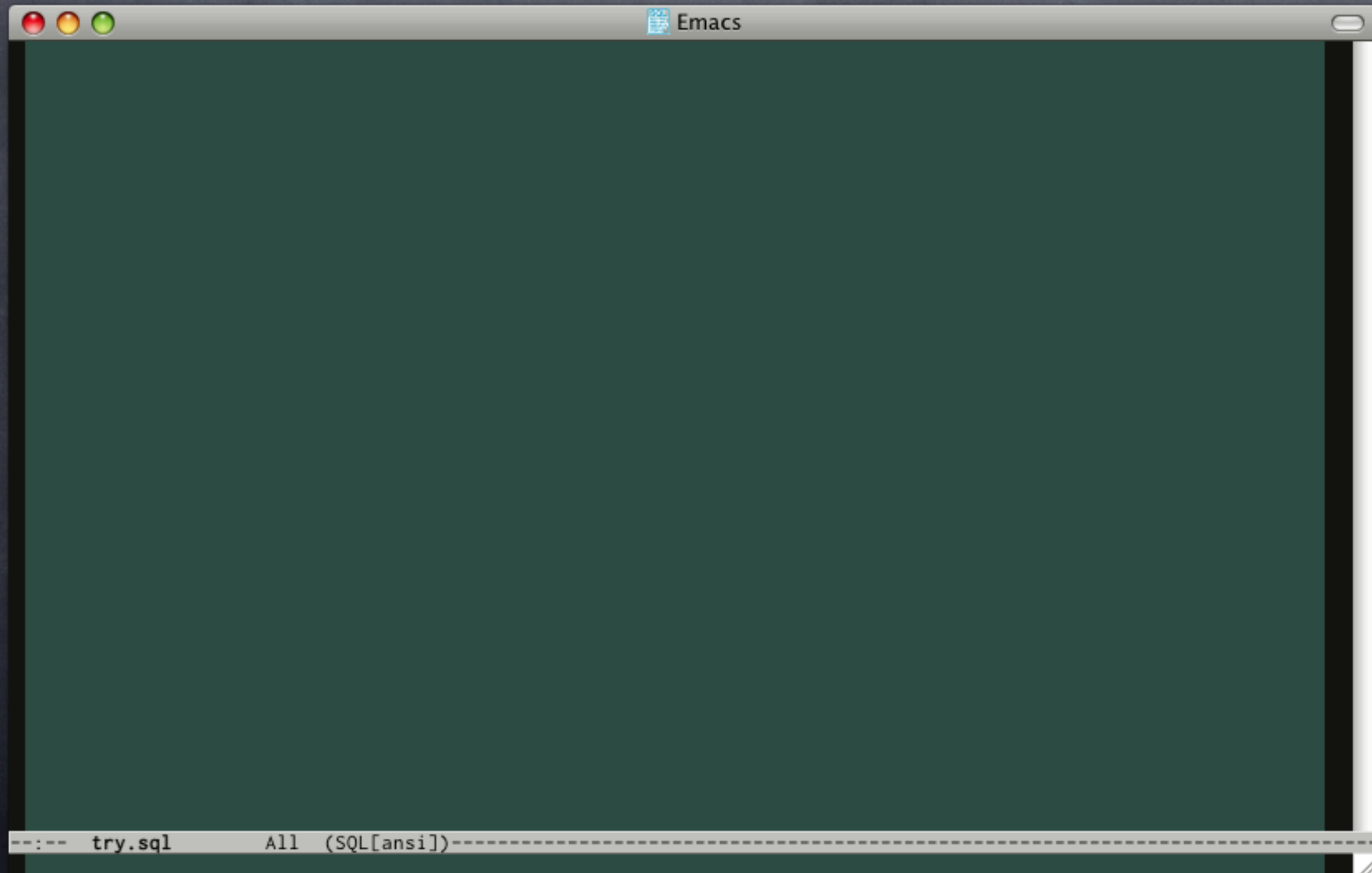



--:-- try.sql All (SQL[ansi])-----

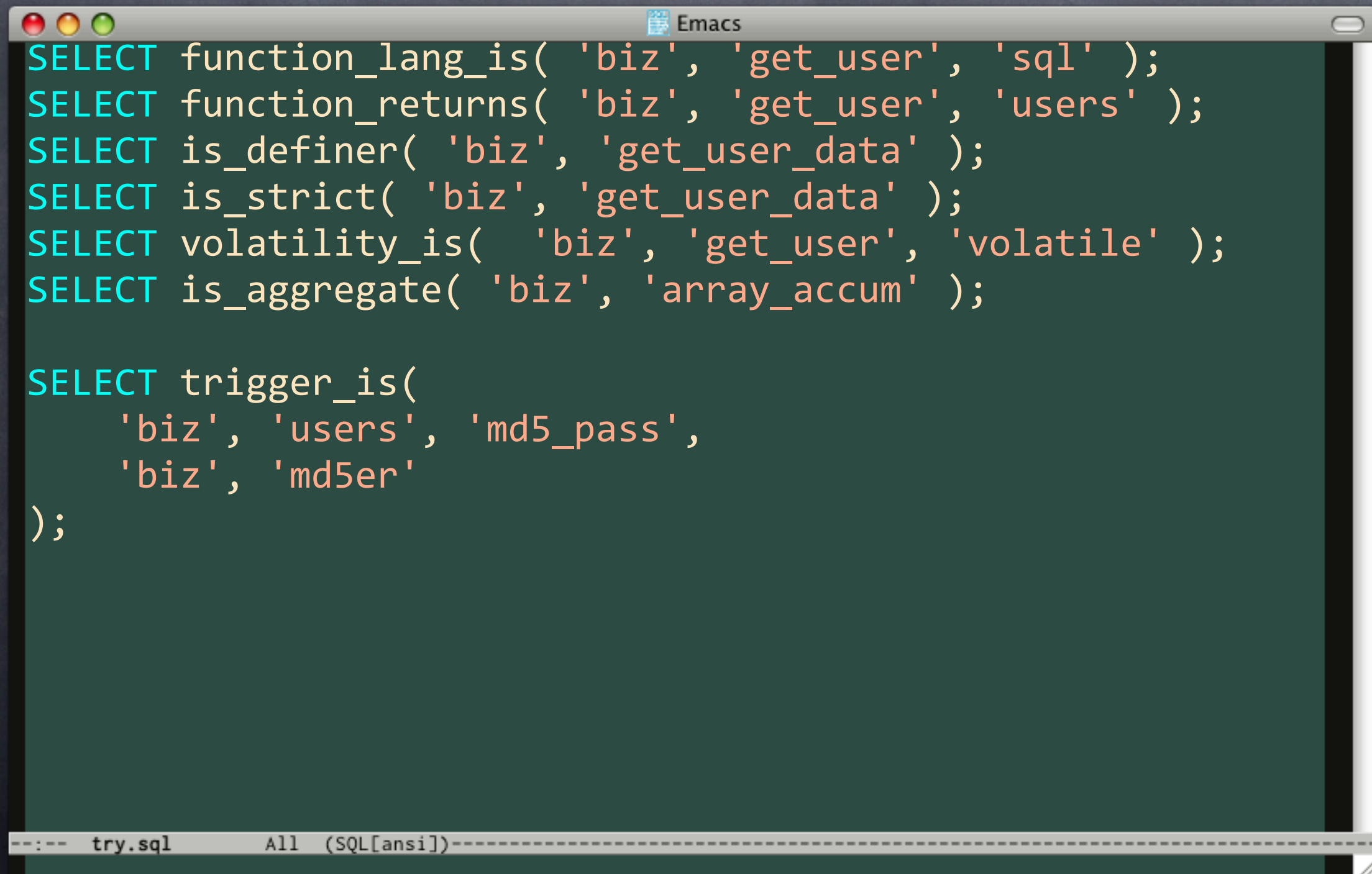

```
SELECT has_tablespace( 'indexspace', '/data/idxs' );
SELECT has_schema( 'biz' );
SELECT has_view( 'biz', 'list_users' );
SELECT has_sequence( 'biz', 'users_id_seq' );
SELECT has_type( 'biz', 'timezone' );
SELECT has_domain( 'biz', 'timezone' );
SELECT has_enum( 'biz', 'escalation' );
SELECT has_index( 'biz', 'users', 'idx_nick' );
SELECT has_trigger( 'biz', 'users', 'md5_pass' );
SELECT has_rule( 'biz', 'list_users', 'user_ins' );
SELECT has_function( 'biz', 'get_user_data' );
SELECT has_role( 'postgres' );
SELECT has_user( 'postgres' );
SELECT has_group( 'postgres' );
SELECT has_language( 'plpgsql' );

SELECT fk_ok(
    'biz', 'widgets', 'user_id',
    'biz', 'users', 'id'
);
```


Function Testing Functions



Function Testing Functions

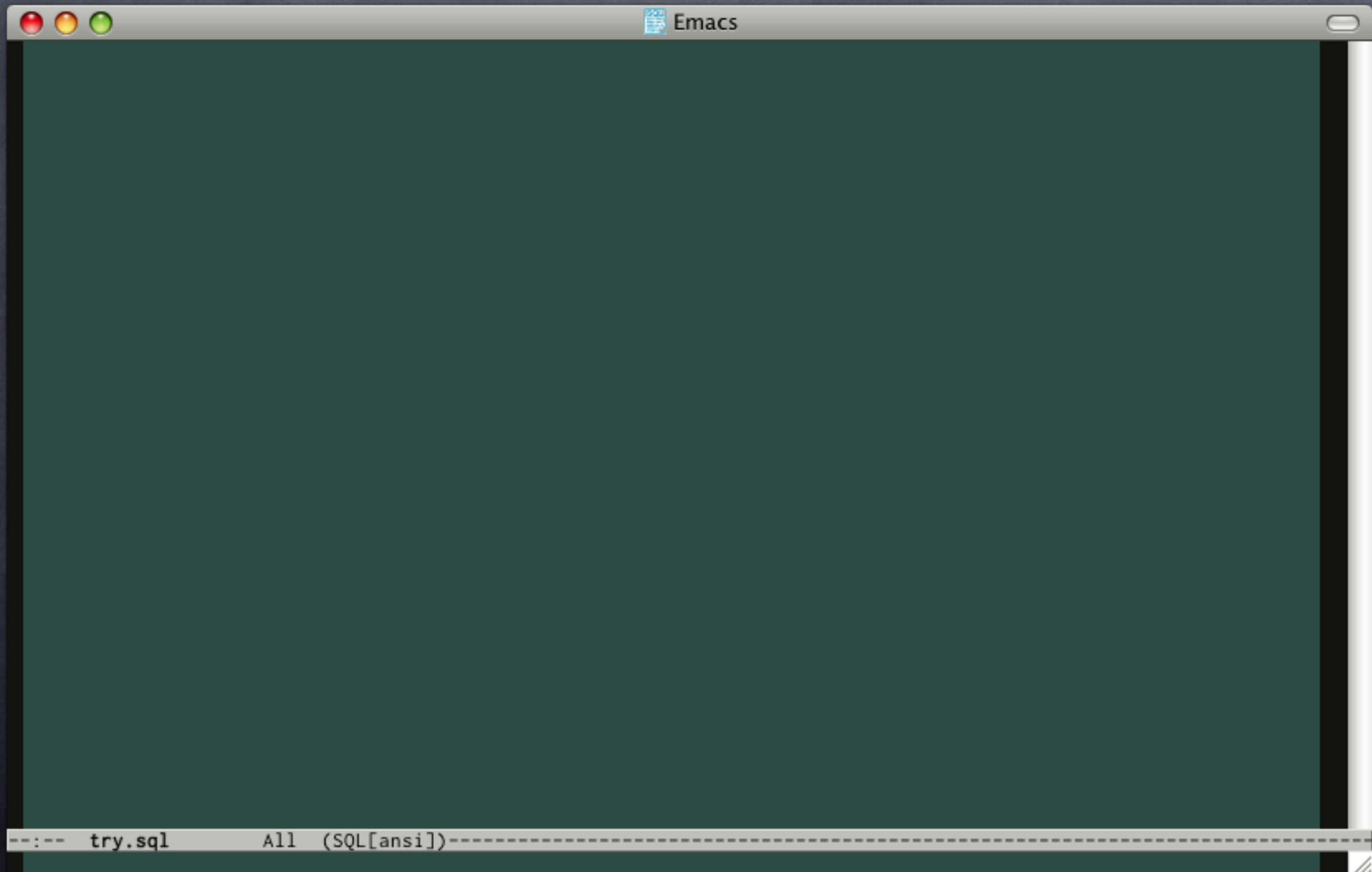
A screenshot of an Emacs window titled "Emacs" showing a SQL script. The script contains several SELECT statements for testing database functions. The text is color-coded: "SELECT" is cyan, and the rest of the queries is orange. The window has a standard macOS-style title bar with red, yellow, and green window control buttons on the left and a scroll bar on the right. At the bottom of the window, a status bar shows the file name "try.sql", the cursor position "All", and the encoding "(SQL[ansi])".

```
SELECT function_lang_is( 'biz', 'get_user', 'sql' );
SELECT function_returns( 'biz', 'get_user', 'users' );
SELECT is_definer( 'biz', 'get_user_data' );
SELECT is_strict( 'biz', 'get_user_data' );
SELECT volatility_is( 'biz', 'get_user', 'volatile' );
SELECT is_aggregate( 'biz', 'array_accum' );

SELECT trigger_is(
    'biz', 'users', 'md5_pass',
    'biz', 'md5er'
);
```

---:-- try.sql All (SQL[ansi])-----

Utilities



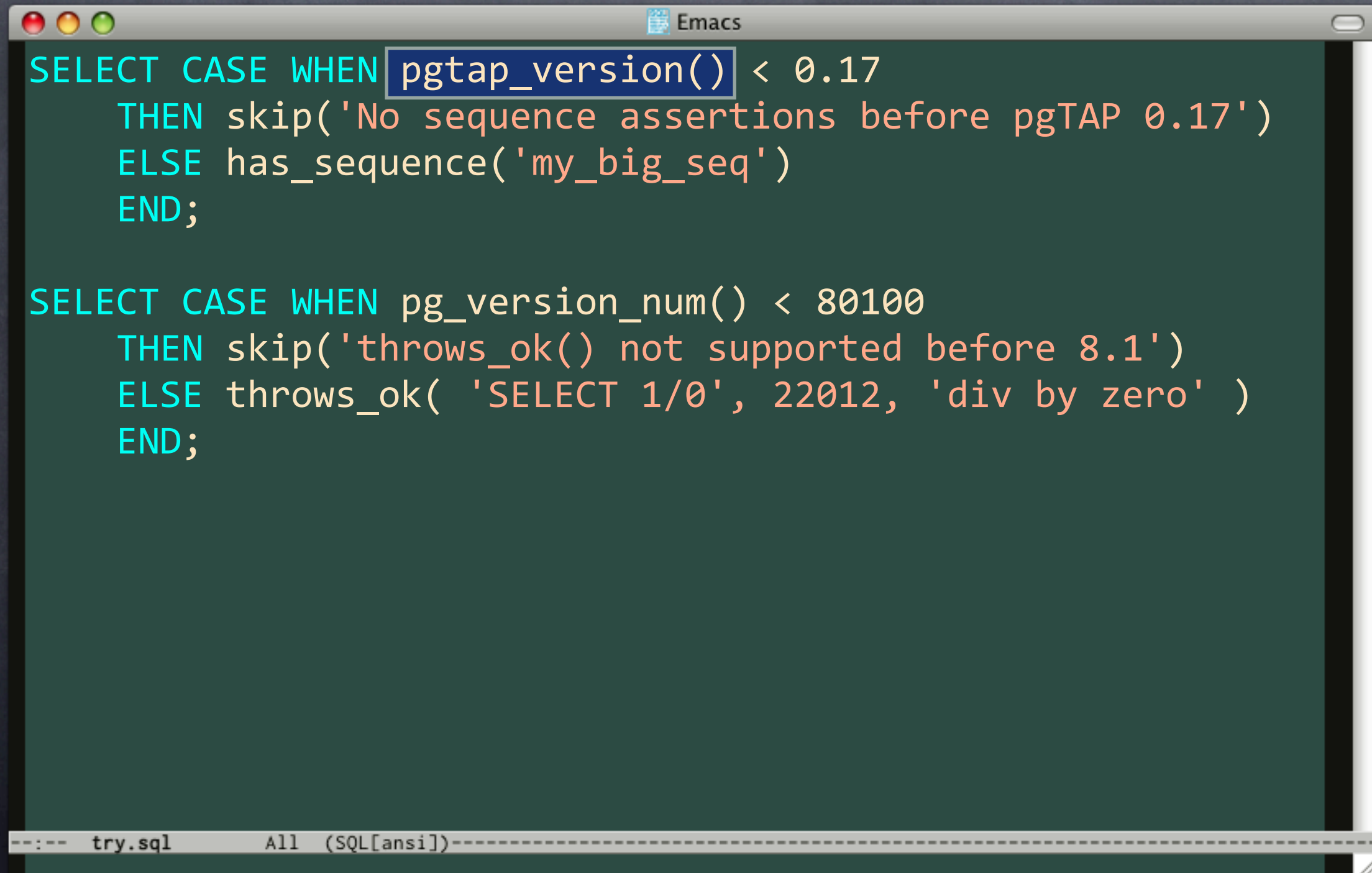
Utilities

```
SELECT CASE WHEN pgtap_version() < 0.17
  THEN skip('No sequence assertions before pgTAP 0.17')
  ELSE has_sequence('my_big_seq')
  END;

SELECT CASE WHEN pg_version_num() < 80100
  THEN skip('throws_ok() not supported before 8.1')
  ELSE throws_ok( 'SELECT 1/0', 22012, 'div by zero' )
  END;
```

--:-- try.sql All (SQL[ansi])-----

Utilities



```
Emacs
SELECT CASE WHEN pgtap_version() < 0.17
  THEN skip('No sequence assertions before pgTAP 0.17')
  ELSE has_sequence('my_big_seq')
  END;

SELECT CASE WHEN pg_version_num() < 80100
  THEN skip('throws_ok() not supported before 8.1')
  ELSE throws_ok( 'SELECT 1/0', 22012, 'div by zero' )
  END;

---:-- try.sql All (SQL[ansi])-----
```


Utilities

```
SELECT CASE WHEN pgtap_version() < 0.17
  THEN skip('No sequence assertions before pgTAP 0.17')
  ELSE has_sequence('my_big_seq')
  END;

SELECT CASE WHEN pg_version_num() < 80100
  THEN skip('throws_ok() not supported before 8.1')
  ELSE throws_ok( 'SELECT 1/0', 22012, 'div by zero' )
  END;
```

--:-- try.sql All (SQL[ansi])-----

Other Features and Topics

Other Features and Topics

- xUnit-Style testing

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**
- **Integration with pg_regress**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**
- **Integration with pg_regress**
- **Negative assertions**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**
- **Integration with pg_regress**
- **Negative assertions**
- **Constraint assertions**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**
- **Integration with pg_regress**
- **Negative assertions**
- **Constraint assertions**
- **Roles and permissions assertions**

Other Features and Topics

- **xUnit-Style testing**
- **Test-Driven development**
- **Integration with Perl unit tests**
- **Integration with pg_regress**
- **Negative assertions**
- **Constraint assertions**
- **Roles and permissions assertions**
- **Lots more!**

Thank You

pgTAP Best Practices

David E. Wheeler

Kineticcode, Inc.
PostgreSQL Experts, Inc.

PostgreSQL Conference West 2009

