

# The Postgres Application Server

Copyright 2008  
David Fetter  
All Rights Reserved.

# Large Applications In Postgres

- PostGIS
- Bucardo
- DBI-Link
- etc., etc., etc.

# Getting Started with Postgres



# Start Your First Postgres Application

```
CREATE TABLE fibonacci_memoize(  
    n numeric PRIMARY KEY,  
    fib_n numeric NOT NULL  
);
```

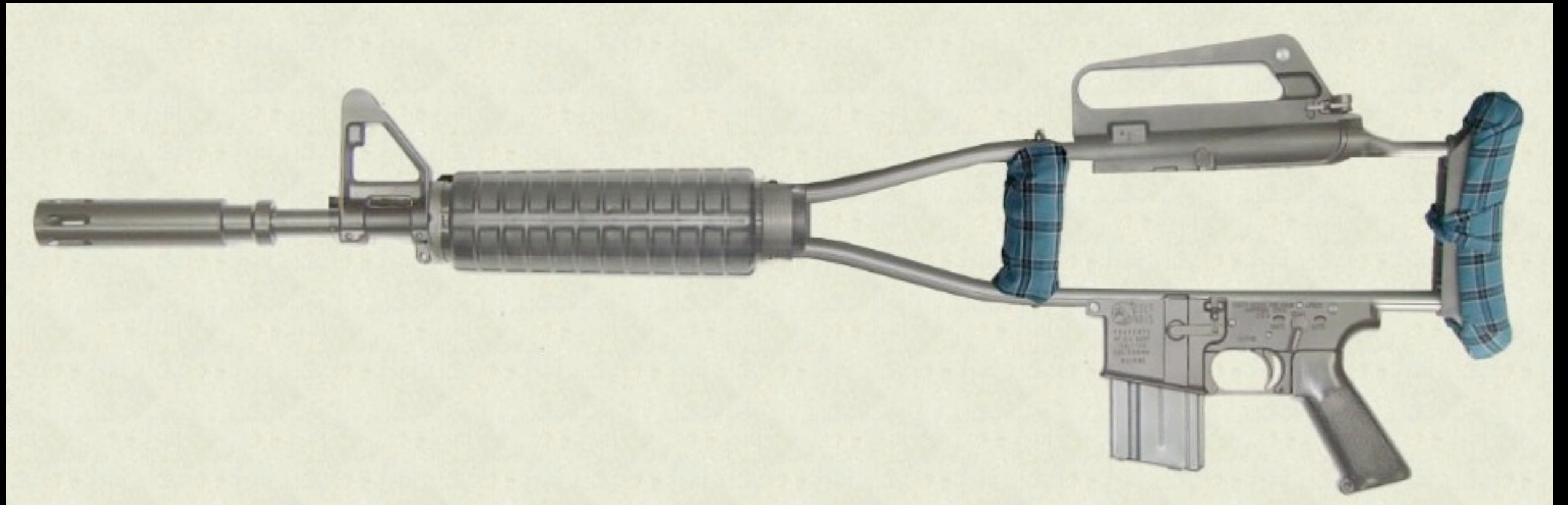
# More...

```
CREATE OR REPLACE FUNCTION memoize_fib
(n numeric, fib_n numeric)
RETURNS numeric
LANGUAGE SQL
AS $$
    INSERT INTO fibonacci_memoize
        VALUES ($1, $2);
    SELECT $2;
$$;
```

# Done!

```
CREATE OR REPLACE FUNCTION fibonacci(numeric)
RETURNS numeric
LANGUAGE SQL
AS $$
    SELECT COALESCE(
        (SELECT fib_n FROM fibonacci_memoize WHERE n=$1),
        memoize_fib(
            $1,
            CASE WHEN $1 < 2
                THEN $1
            ELSE
                fibonacci($1-2) + fibonacci($1-1)
            END
        )
    );
$$;
```

# Postgres Tools



# Programming Languages

- SQL
- PL/PgSQL
- PL/Lua(U)
- PL/Perl(U)
- PL/PHP(U)
- PL/Parrot(U)?
- etc., etc., etc.

# Trust

(Security Backwards)

- Functions in untrusted languages can:
  - Open Filehandles
  - Open Pipes
  - Do Dangerous Things™.
- Functions in trusted languages
  - Cannot

# Lua!

Você pode usar a linguagem Lua dentro Postgres!



<http://pgfoundry.org/projects/pllua/>

# Perl



# Ruby



<http://rubygems.org/>

# Debuggers

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

**Kernighan**

# A New Data Type

```
CREATE DOMAIN email AS TEXT  
CHECK(is_email(VALUE));
```

# The Function

```
CREATE OR REPLACE  
FUNCTION is_email  
(text)  
RETURNS BOOLEAN  
IMMUTABLE  
LANGUAGE plperlu  
AS $$  
...  
$$;
```



Every Perl Program Starts with:

**use strict;**

**use warnings;**

# More of the Program

```
use Email::Valid;
my $address = shift;
my $checks = {
    -address => $address,
    -mxcheck => 1,
    -tldcheck => 1,
    -rfc822  => 1,
};
```

# Done!

```
if (defined Email::Valid->address( %$checks )) {  
    return 'true'  
}
```

```
warn "address failed $Email::Valid::Details check."  
return 'false';
```

# The Email Domain in Action

```
FISL9=> SELECT 'david@fetter.org'::email;  
email
```

-----

```
david@fetter.org  
(1 row)
```

```
FISL9=> SELECT 'david@fetter.con'::email;  
ERROR: value for domain email violates check constraint  
"email_check"
```

# A Domain Maker

```
use Proc::ProcessTable;
my $t = new Proc::ProcessTable;
my $p = shift ( @{ $t->table } );
my @fields = ( );
```

# More Domain Maker

```
foreach my $f ($t->fields) {
    if ($p->{$f} =~ m/-?\d*\.\d+/) {
        push @fields, "$f FLOAT8";
    }
    elsif ($p->{$f} =~ m/^-?\d+$/) {
        push @fields, "$f INT8";
    }
    else {
        push @fields, "$f TEXT";
    }
}
```

# Done!

```
print <<CREATE_TYPE;
CREATE TYPE process_table_type AS (
    @ {[ join(",\n", @fields) ] }
);
CREATE_TYPE
```

# Dynamic Custom Type?!

```
$ psql FISL9
Welcome to psql 8.3.1, the PostgreSQL interactive terminal.
...
FISL9=> \set mptt `./make_process_table_type.pl`
FISL9=> :mptt
CREATE TYPE
```

# The PL/PerlU Function

```
CREATE OR REPLACE FUNCTION get_ps ()  
RETURNS SETOF process_table_type  
LANGUAGE plperl  
AS $$  
    use Proc::ProcessTable;  
    my $proctab = new Proc::ProcessTable;  
    return $proctab->table;  
$$;
```

# A Query

```
SELECT *  
FROM  
    pg_class "c1",  
    pg_class "c2",  
    pg_class "c3";
```

# What is happening?!?

```
SELECT
    a.current_query,
    MAX(p.size) AS "size"
FROM
    pg_stat_activity AS "a"
JOIN
    get_ps() AS "p"
    ON (a.procpid = p.pid)
GROUP BY a.current_query;
```

# Aha!

current_query	size
SELECT *	
FROM	
pg_class "c1",	
pg_class "c2",	
pg_class "c3";	195290000

(1 row)

Perguntas?

Comentários?

Joguem as pedras?

# Muito Obrigado!

Copyright David Fetter 2008

All Rights Reserved

<http://fetter.org/>