

# Writing an “Enterprise Scheduler” using Perl, MySQL, SSH, and POE

Albert P. Tobey  
Pittsburgh Perl Workshop  
September 23, 2006

<http://www.tobert.org>  
tobert@gmail.com

Thank you to:



Developers, Advisors, and Testers:

Aaron Nienhuis  
Keith Sederholm  
Rick Siner  
Joel Meulenberg  
Keith Brunsting  
Jeff Klein

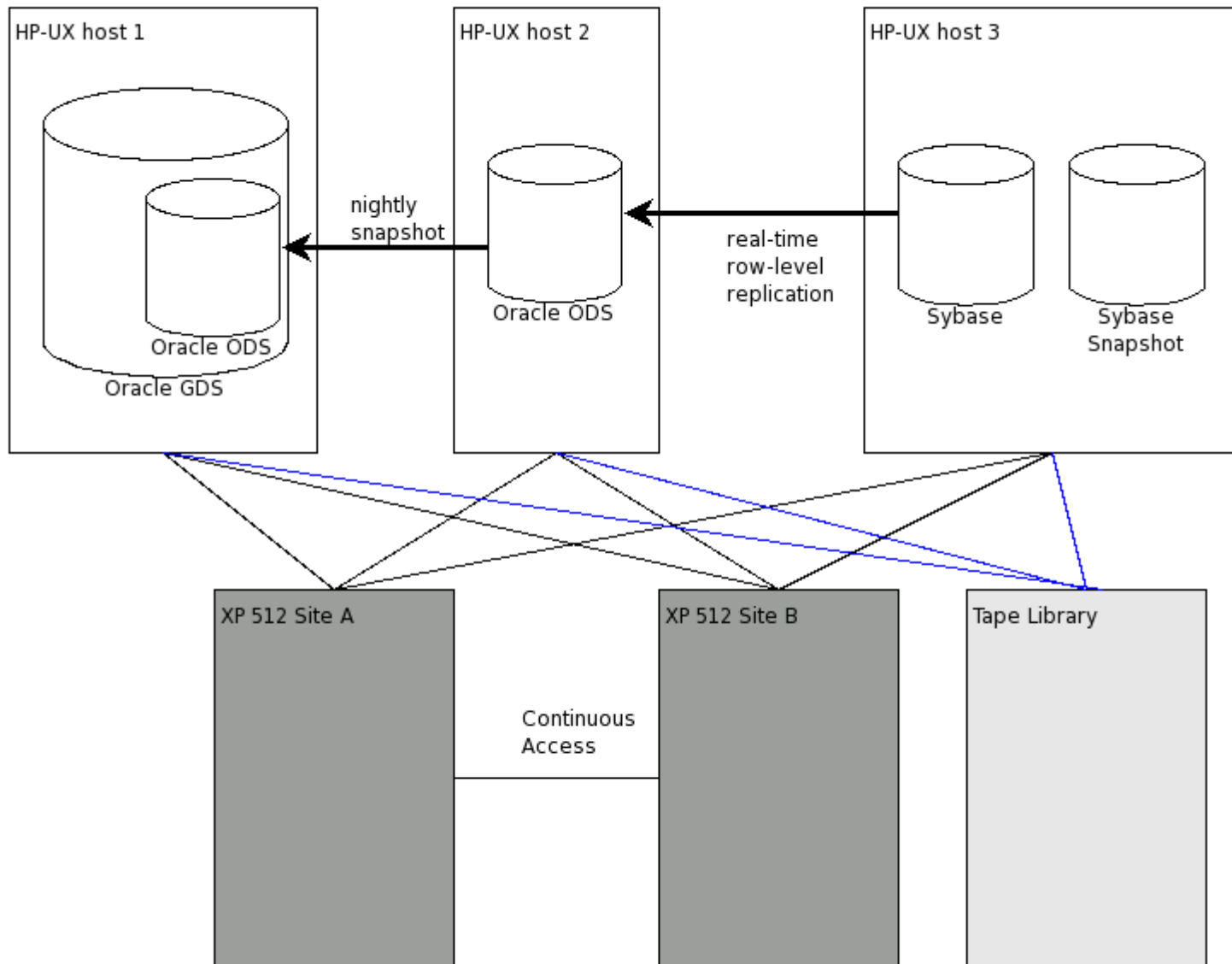
IS Leadership:

Jim Slubowski  
Mike Trus  
Krischa Winright  
Rick Siner

# The Problem



- We upgraded our ERP-like system in 2002.
- It runs Sybase.
- That's fine, but we're an Oracle shop.
- Replication!
- Sybase -> Datamirror -> Oracle
- Now back it up, smart guy.

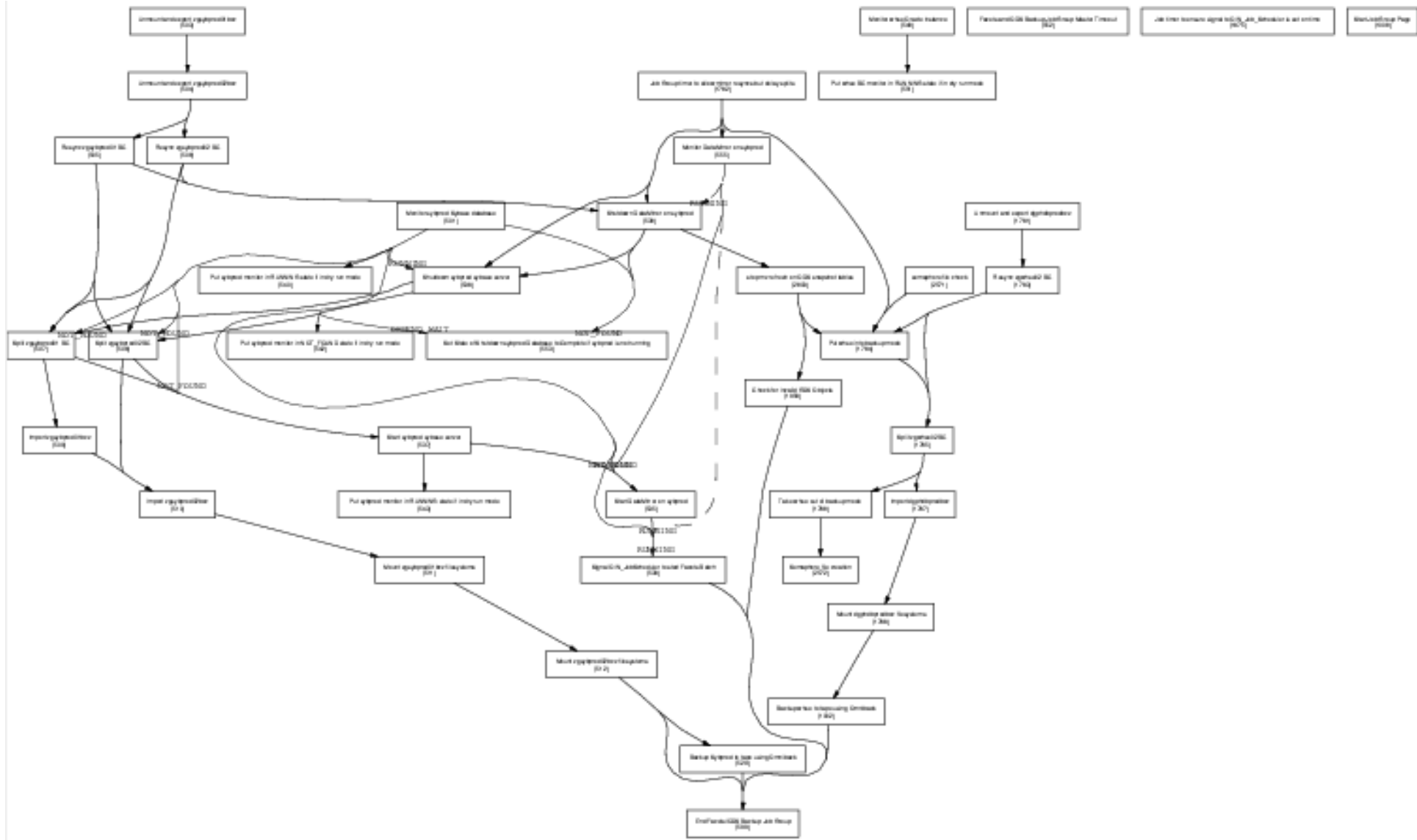


ODS: Operational Data Store  
GDS: Global Data Store

# Process Overview

- kick all users out of FACETS
- quiesce all transactions in Sybase
- stop replication
- shut down Sybase
- put the Oracle ODS in hot backup mode
- start BCV copy operation on the Sybase volumes (disk snapshot)
- export the ODS snapshot from the GDS
- export the ODS volume group from the GDS host
- start BCV copy operation to update the ODS snapshot
- put the GDS into hot backup mode
- start BCV copy operation on the GDS
- start backup of each BCV after copy is complete
- after ODS backup completes:
  - import the volume group on the GDS host
  - rename the oracle instance
  - start Oracle
  - export relevant tables
  - import tables into the GDS read-only
- start Sybase
- start the ODS
- start replication
- once replication is complete, make Sybase available to users





# Solution Requirements

- sane dependency management
- parallelize as widely as possible
- run arbitrary code on multiple hosts
- centralized management
- configuration separate from program
- extensive logging
- ability to notify administrators of status and incidents



# Tools

- Perl
  - rapid prototype/development
  - also great for production
- SSH
  - well-understood and available
- MySQL
  - I knew it better than Oracle at the time
- POE (Perl Object Environment)
  - Handy for multiplexing lots of things in a single thread



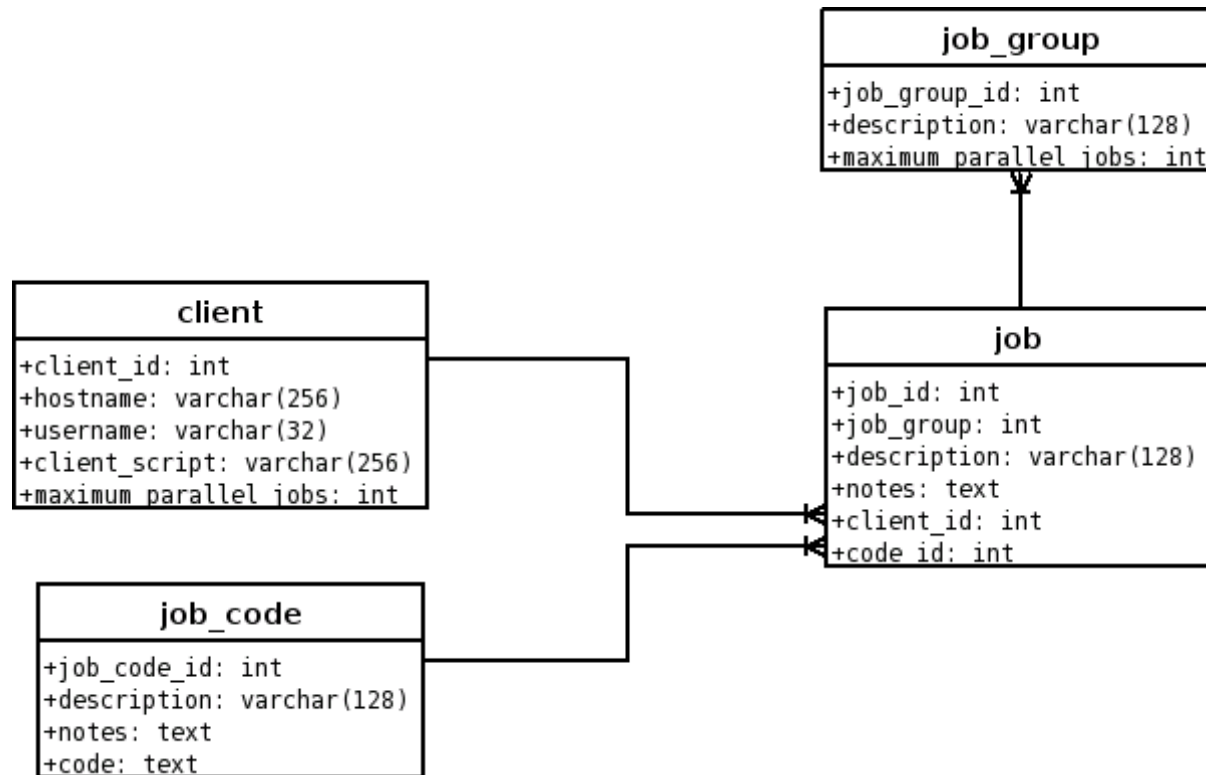


# Dependency Graphs

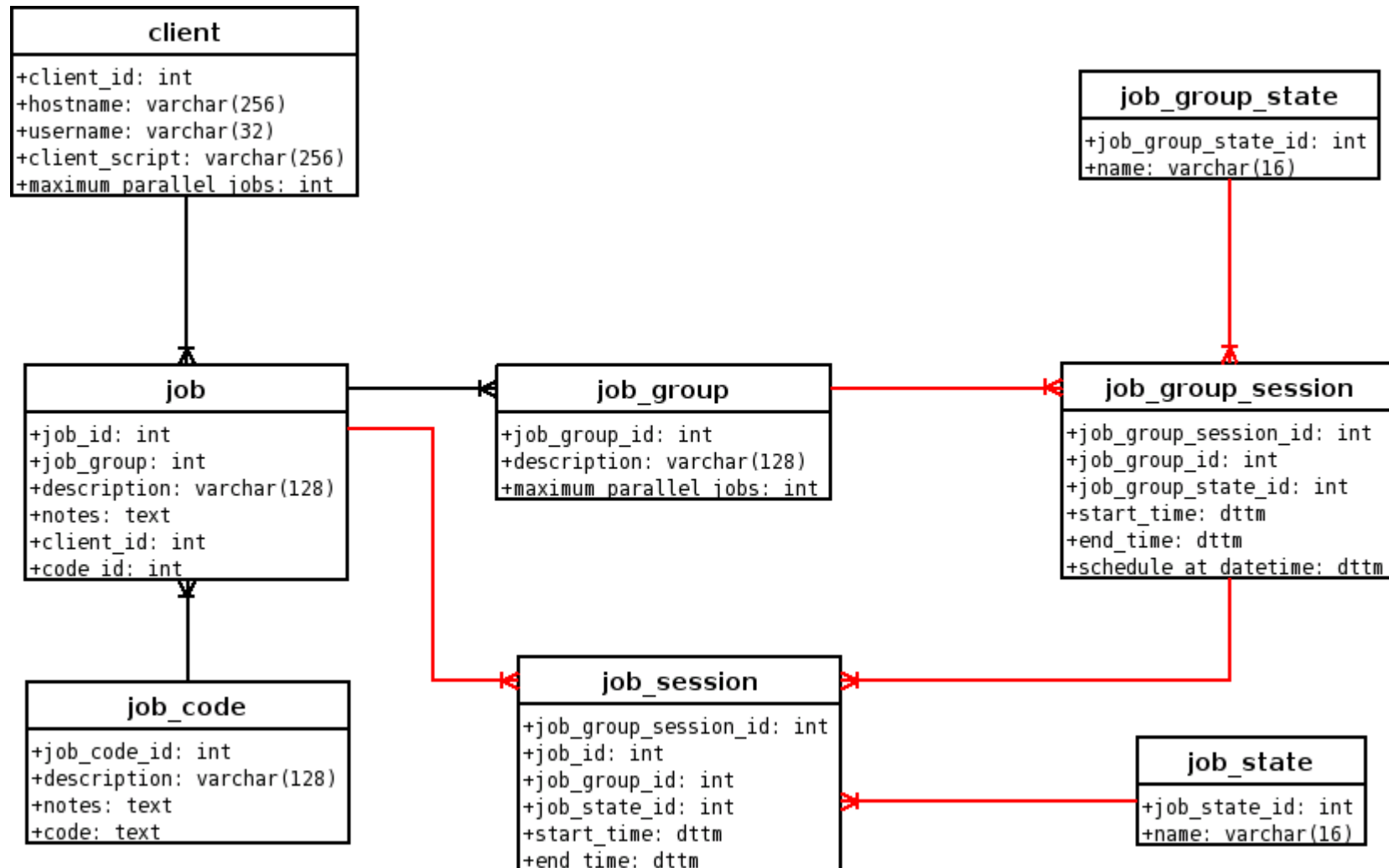


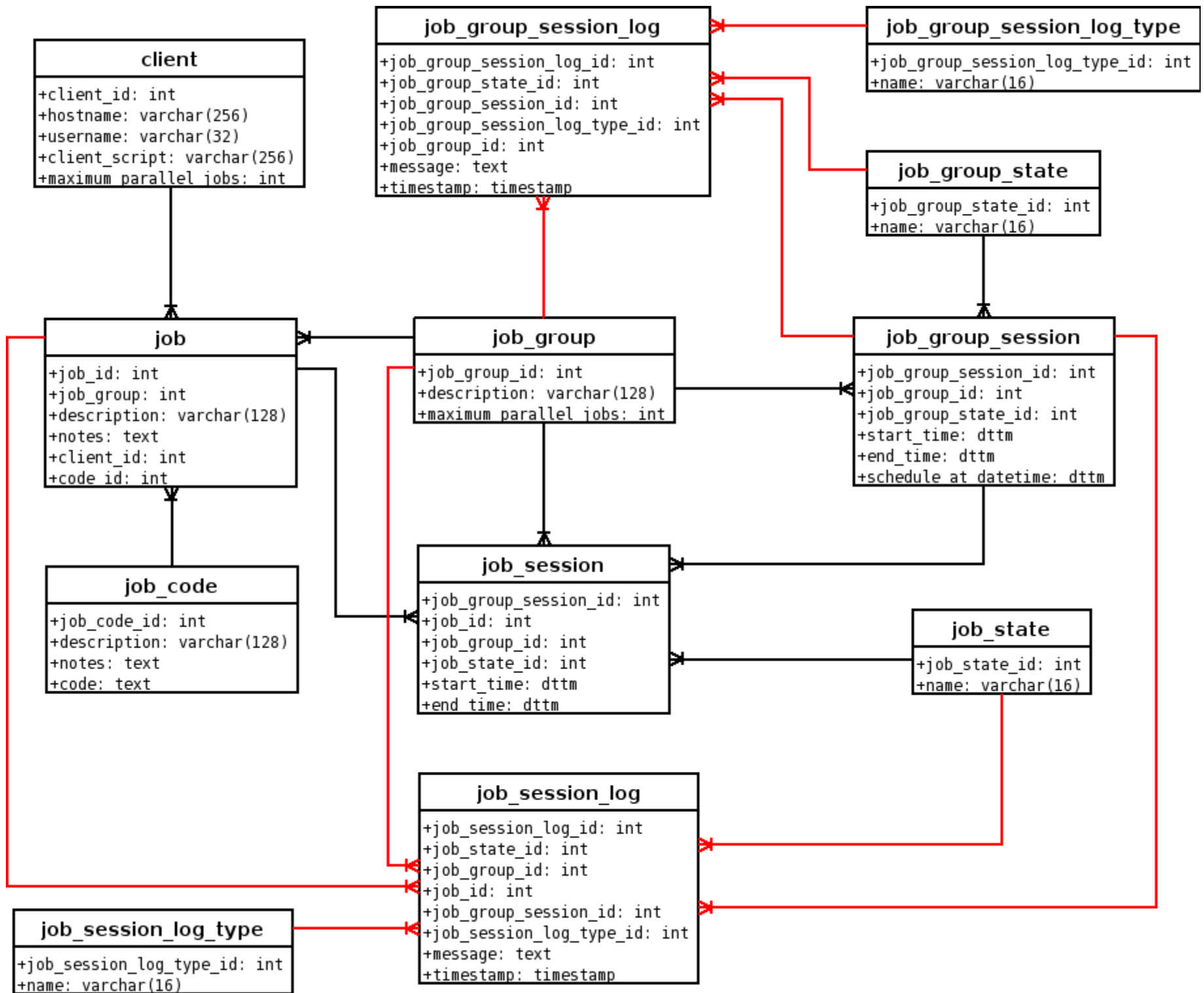
- Most of it was written before somebody pointed out it was executing graphs.
- Each job group can be thought of as either a graph or a state machine, although it fits neither well formally.
- Jobs can depend on an arbitrary number of jobs.
- They actually depend on a combination of job + job state.

# Basic Data Model



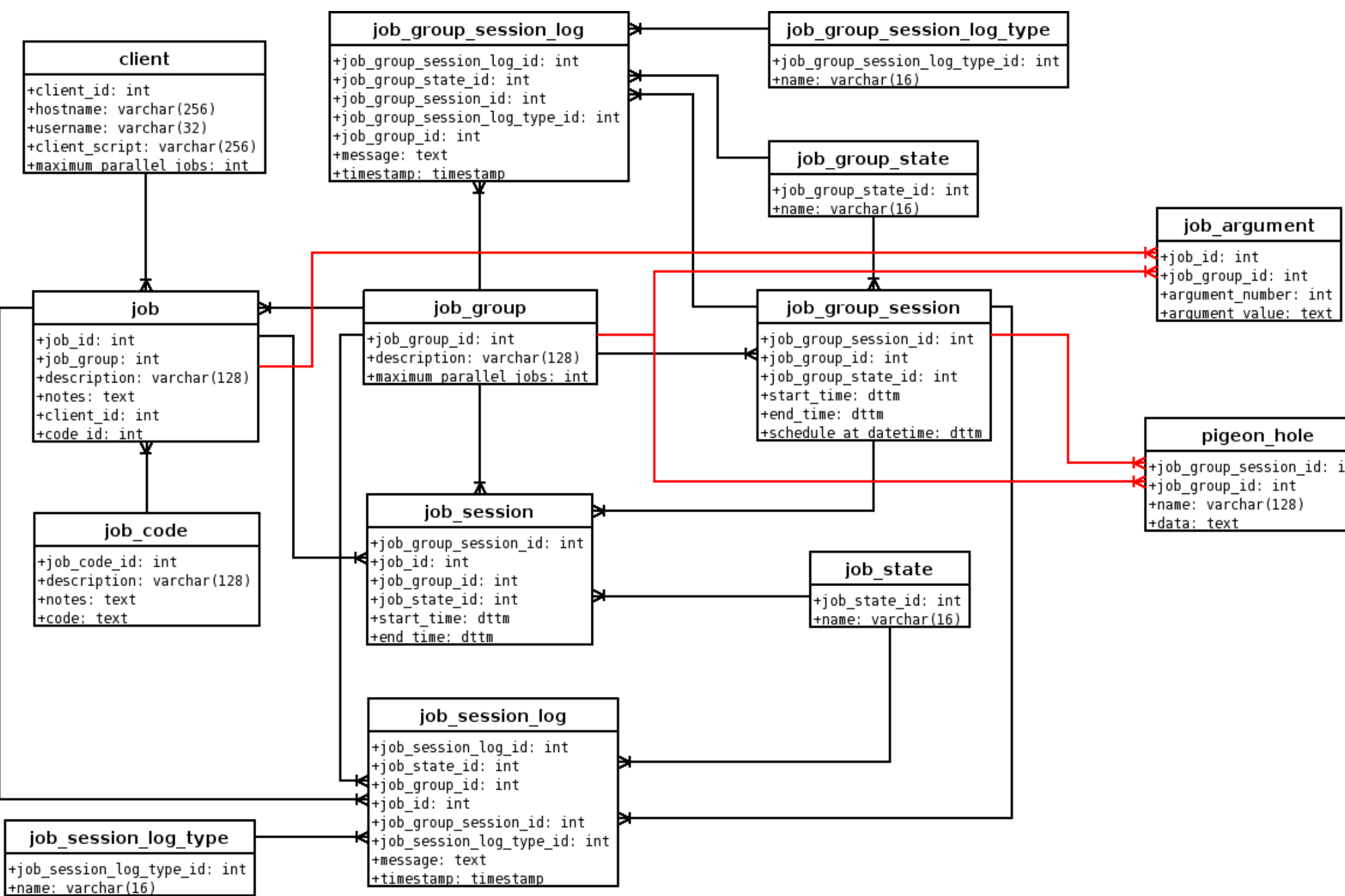
# Sessions











## Job Arguments

```
my $code = <<EOCODE;
    my $sth = $dbh->prepare("[ARG0]");
    $sth->execute( '[ARG1]', [ARG2] );
    $sth->finish;
EOCODE

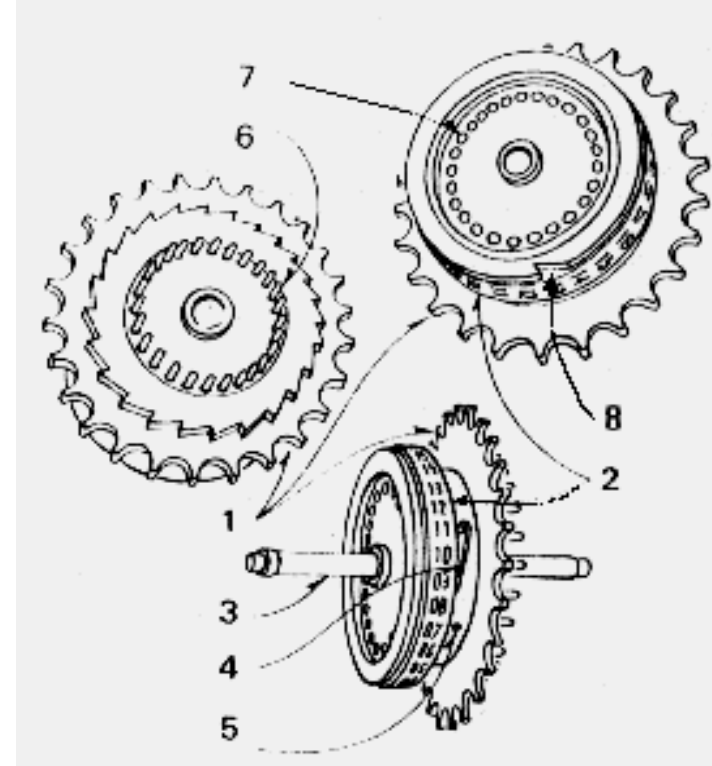
my $code = <<EOCODE;
    my $sth = $dbh->prepare("
UPDATE huge_table SET foo=? WHERE bar=?
");
    $sth->execute( 'w00t', 42 );
    $sth->finish;
EOCODE
```

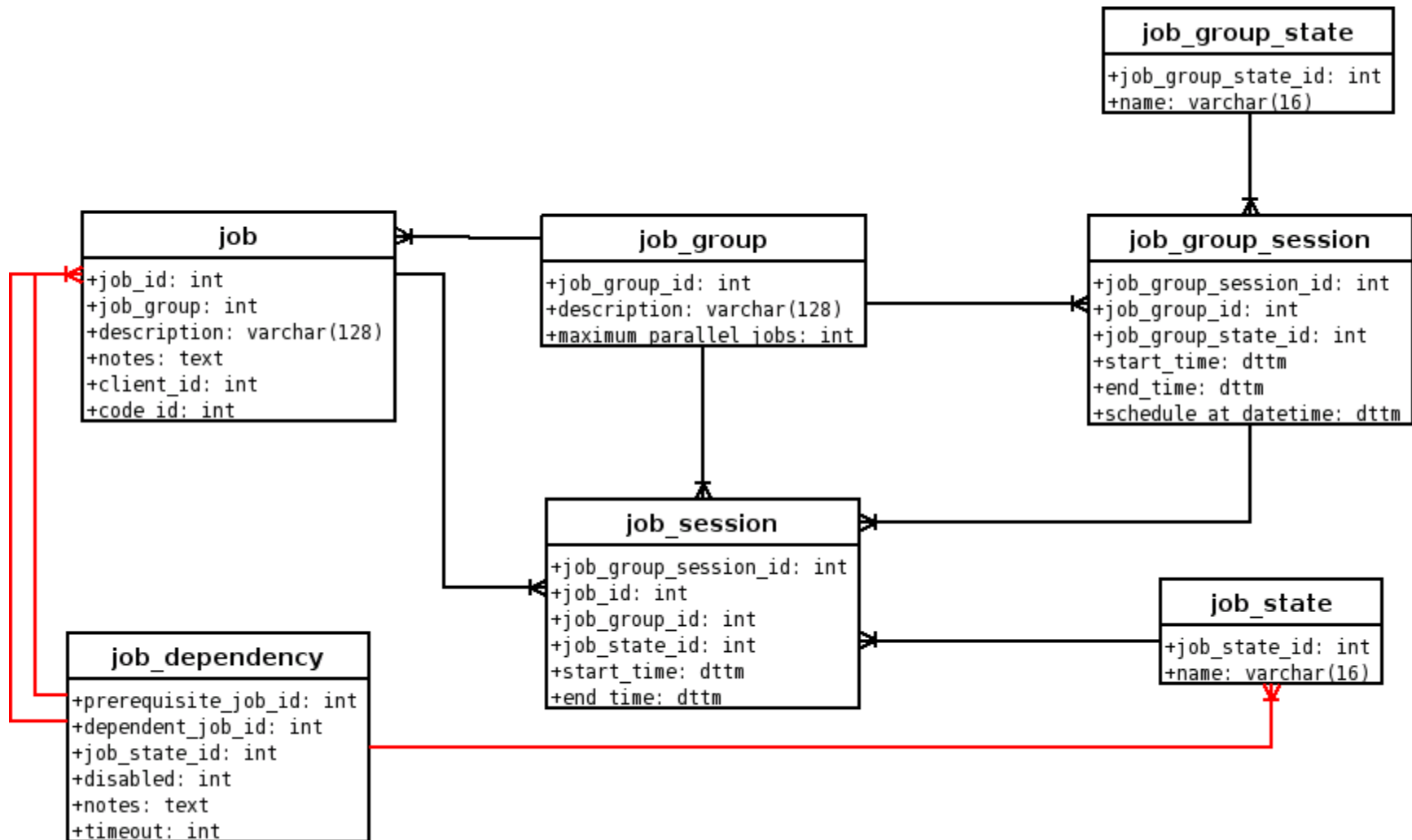
## Pigeon Hole

```
pigeon_hole_store( "index_ddl", $dropped_index_ddl );

...

my $ddl = pigeon_hole_fetch( "index_ddl" );
$dbh->do( $ddl );
```







# Perl Object Environment



- Event model
- Makes maintaining many loops easy
- Makes managing multiple ssh connections easier
- Handles all of the traditional I/O problems
  - non-blocking stdin/stdout/stderr
  - forking processes and controlling their stdio
- <http://poe.perl.org>, freenode: #POE

# POE::Wheel::Run

```
POE::Session->create( ... );
$poe_kernel->run;

my $cmd    = "/home/page/client.pl";
my $prog   = "/usr/bin/ssh -l $user $hostname $cmd";

my $wheel = POE::Wheel::Run->new(
    Program => $prog,
    StdioFilter => POE::Filter::Line->new,
    StdoutEvent => 'handle_ssh_output',
    StderrEvent => 'handle_ssh_error'
);

$wheel->put( "Foo the bar over there" );

sub handle_ssh_output {
    my $buffer = $_[ARG0];
    print "Got \"$buffer\" from client!\n";
}

sub handle_ssh_error {
    my $buffer = $_[ARG0];
    print STDERR "Got error \"$buffer\" from client!\n";
}
```



# Cheesy Client Protocol

- Very simple
- All plain text
- States are prepended to messages
- Possible to support multiple job groups on a single ssh connection

```
EVAL_CODE(500,128): system( "rm -rf /*" );
```

```
STATUS_REPORT(500,128): RUNNING
```

```
RUNNING(500,128): blah blah blah blah blah
```



# client.pl

- Uses POE
- forks before evaling code
  - protects other jobs from segfaults and other things that can crash through eval()
- mini-API for job logging, control, and utility
- completely driven by the controller



# controller.pl



- All state is in MySQL and is never cached.
- Handles all I/O from client.pl
- Has an embedded version of client.pl for jobs that need to access its guts.
- Actually, there is some POE runtime state that is necessary for a job group to exist that is kept in core only.
  - can be problematic if controller.pl crashes or must be restarted
- Difficult/tedious to test

# Open Source

- Priority Health gave me permission to release the code.
- Dubbed PAGE v0, it is available at:
  - <http://www.tobert.org/scheduler/index.html>
- Not really easy to install/use yet
- v1.0 will be a near 100% rewrite, but reusing any code it can and much of the design
  - OO, testable, tested, auto-generated code
- Help!

`mailto:tobert@gmail.com`

Questions?