Phoenix in 15 minutes or less

What is this new Phoenix thing I've been hearing about?
Phoenix is an open source SQL skin for HBase. You use the standard JDBC APIs instead of the regular HBase client APIs to create tables, insert data, and query your HBase data.

Doesn't putting an extra layer between my application and HBase just slow things down?
Actually, no. Phoenix achieves as good or likely better performance than if you hand-coded it yourself (not to mention with a heck of a lot less code) by:

- compiling your SQL queries to native HBase scans
- determining the optimal start and stop for your scan key
- orchestrating the parallel execution of your scans
- bringing the computation to the data by
- pushing the predicates in your where clause to a server-side filter
- executing aggregate queries through server-side hooks (called co-processors)

In addition to these items, we've got some interesting enhancements in the works to further optimize performance:

- secondary indexes to improve performance for queries on non row key columns
- stats gathering to improve parallelization and guide choices between optimizations
- skip scan filter to optimize IN, LIKE, and OR queries
- optional salting of row keys to evenly distribute write load

Ok, so it's fast. But why SQL? It's so 1970s
Well, that's kind of the point: give folks something with which they're already familiar. What better way to spur the adoption of HBase? On top of that, using JDBC and SQL:

- Reduces the amount of code users need to write
- Allows for performance optimizations transparent to the user
- Opens the door for leveraging and integrating lots of existing tooling

But how can SQL support my favorite HBase technique of x,y,z
Didn't make it to the last HBase Meetup did you? SQL is just a way of expressing ***what you want to get*** not ***how you want to get it***. Check out my presentation for various existing and to-be-done Phoenix features to support your favorite HBase trick. Have ideas of your own? We'd love to hear about them: file an issue for us and/or join our mailing list.

Blah, blah, blah - I just want to get started!
Ok, great! Just follow our install instructions:

- download and expand our installation tar
- copy the phoenix server jar that is compatible with your HBase installation into the lib directory of every region server
- restart the region servers
- add the phoenix client jar to the classpath of your HBase client
- download and setup SQuirrel as your SQL client so you can issue adhoc SQL against your HBase cluster

I don't want to download and setup anything else!
Ok, fair enough - you can create your own SQL scripts and execute them using our command line tool instead. Let's walk through an example now. Begin by navigating to the `bin/` directory of your Phoenix install location.

- First, let's create a `us_population.sql` file, containing a table definition:

```
CREATE TABLE IF NOT EXISTS us_population (
      state CHAR(2) NOT NULL,
      city VARCHAR NOT NULL,
      population BIGINT
      CONSTRAINT my_pk PRIMARY KEY (state, city));
```

- Now let's create a `us_population.csv` file containing some data to put in that table:

```
NY,New York,8143197
CA,Los Angeles,3844829
IL,Chicago,2842518
TX,Houston,2016582
PA,Philadelphia,1463281
AZ,Phoenix,1461575
TX,San Antonio,1256509
CA,San Diego,1255540
TX,Dallas,1213825
CA,San Jose,912332
```

- And finally, let's create a `us_population_queries.sql` file containing a query we'd like to run on that data.

```
SELECT state as "State",count(city) as "City Count",sum(population) as "Population Sum"
FROM us_population
GROUP BY state
ORDER BY sum(population) DESC;
```

- Execute the following command from a command terminal

```
./psql.py <your_zookeeper_quorum> us_population.sql us_population.csv
us_population_queries.sql
```

Congratulations! You've just created your first Phoenix table, inserted data into it, and executed an aggregate query with just a few lines of code in 15 minutes or less!

Big deal - 10 rows! What else you got?
Ok, ok - tough crowd. Check out our `bin/performance.py` script to create as many rows as you want, for any schema you come up with, and run timed queries against it.

Why is it called Phoenix anyway? Did some other project crash and burn and this is the next generation?
I'm sorry, but we're out of time and space, so we'll have to answer that next time!