Flock demo

This demo will walk through setting up a local development flockdb instance and interacting with it via the ruby client. To play along, you need:

Constraint x4

- java 1.6
- sbt 0.7.4
- ruby 1.8
- mysql 5.0
- gizzmo

Newer versions should work for all of the above. Make sure to put the gizzmo binary on your path.

Building it

If you haven't built flockdb yet, do that first:

```
$ sbt update package-dist
```

RootFile

You may need to set `DB_USERNAME` and `DB_PASSWORD` for tests to complete (see below).

Setting up shards

To create a set of shards for development mode, a script called `setup-env.sh` is included. Make sure mysql is running, and set these env vars so the script can talk to mysql:

```
$ export DB_USERNAME="root"
$ export DB_PASSWORD="password"
```

ChangeFileRegex with DeclareVariables

These are also used by `config/development.conf` in flockdb.

Now run `setup-env.sh` :

```
$ ./dist/flockdb/scripts/setup-env.sh
```

RootFile

It kills and restarts flockdb, creates the `flockdb_development` database if necessary, and runs `gizzmo` to create shard configurations for graphs 1-15.

You can tell flockdb is running because it will create a `flock.log` file in the current folder, and it will respond to `server_info` queries:

```
$ curl localhost:9990/server_info.txt
build: 20100713-165811
```

RootFile

```
build_revision: 4b2443968d131b7967885b0b0cb62dde04ab5455
name: flockdb
start_time: Tue Jul 13 17:01:33 PDT 2010
uptime: 440837
version: 1.0.4
```

You should also be able to see that `gizzmo` created a forward and backward shard for each of 15 made-up graphs, by asking it to show you the forwarding table. First, set up a default host & port in your `.gizzmorc` to make the rest of the demo easier:

RootFile

```
$ cat ~/.gizzmorc
host: localhost
port: 7920
```

Then, let's see what tables we have in our system:

RootFile

```
$ gizzmo tables
-15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Notice that we create graphs as a pair of tables, one in the positive (forward) direction and one in the negative (backward) direction. This way we can query for relationships in either direction, such as "Who does Bob follow?" and "Who follows Alice?"

Now let's query the forwardings for those tables:

RootFile

```
$ gizzmo -T -15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 topology --for
[-10] 0 = localhost/backward_10 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-11] 0 = localhost/backward_11 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-12] 0 = localhost/backward_12 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-13] 0 = localhost/backward_13 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-14] 0 = localhost/backward_14 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-15] 0 = localhost/backward_15 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-1] 0 = localhost/backward_1   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-2] 0 = localhost/backward_2   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-3] 0 = localhost/backward_3   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-4] 0 = localhost/backward_4   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-5] 0 = localhost/backward_5   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-6] 0 = localhost/backward_6   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-7] 0 = localhost/backward_7   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-8] 0 = localhost/backward_8   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[-9] 0 = localhost/backward_9   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[10] 0 = localhost/forward_10   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[11] 0 = localhost/forward_11   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[12] 0 = localhost/forward_12   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[13] 0 = localhost/forward_13   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[14] 0 = localhost/forward_14   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[15] 0 = localhost/forward_15   com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[1] 0 = localhost/forward_1 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[2] 0 = localhost/forward_2 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[3] 0 = localhost/forward_3 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[4] 0 = localhost/forward_4 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[5] 0 = localhost/forward_5 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[6] 0 = localhost/forward_6 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[7] 0 = localhost/forward_7 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[8] 0 = localhost/forward_8 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
[9] 0 = localhost/forward_9 com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIGNED)
```

The shard config is necessary so that flockdb knows where to write edges for a graph. If no forwarding info is provided for a graph, any operation on that graph will throw an exception.

Talking to flockdb

Now install the ruby flockdb client:

```
$ sudo gem install flockdb
```

RootFile

The flockdb interface is thrift, so you can talk to it in many different languages, but the raw thrift interface isn't as expressive as the one in the ruby client, which adds some nice syntactic sugar for creating queries.

If flockdb is running, you should be able to connect with it from an `irb` ruby prompt:

```
>> require "flockdb"
=> true
>> flock = Flock.new "localhost:7915", :graphs => { :follows => 1, :blocks => 2 }
=> #<Flock::Client:0x101505aa8 @service=..., @graphs={:follows=>1, :blocks=>2}>
```

RootFile executed by ruby

Okay, in an empty database, how many people are following user 1?

```
>> flock.select(nil, :follows, 1).to_a
=> []
```

ContinueFile section 2

Let's make user 1 a bit more popular, then.

```
>> flock.add(20, :follows, 1)
=> nil
>> flock.add(21, :follows, 1)
=> nil
>> flock.add(22, :follows, 1)
=> nil
```

Continue section 3

Did that help?

```
>> flock.select(nil, :follows, 1).to_a
=> [22, 21, 20]
```

Continue section 4

Notice that the results are given in recency order, most recent first.

Under the hood

You can ask `gizzmo` where a shard is stored:

```
$ gizzmo lookup 1 1
localhost/forward_1
$ gizzmo lookup -- -1 1
localhost/backward_1
```

RootFile

In development mode, all forward edges from graph 1 are stored in a single table, so we didn't really need to ask, but it can be useful when you have a lot of shards for a graph.

```
mysql> use edges_development;
mysql> select * from backward_1_metadata where source_id=1;
+-----------+-------+-------+------------+
| source_id | count | state | updated_at |
+-----------+-------+-------+------------+
|         1 |     3 |     0 |          0 |
+-----------+-------+-------+------------+
1 row in set (0.00 sec)
```

RootFile with CodeReference using another command to call mysql

So, in the backward direction, user 1 is being followed by 3 people.

```
mysql> select * from backward_1_edges where source_id=1;
+-----------+---------------------+------------+----------------+-------+-------+
| source_id | position            | updated_at | destination_id | count | state |
+-----------+---------------------+------------+----------------+-------+-------+
```

Continue section 2

```
|         1 | 1334224838599527719 | 1272415960 |              20 |     1 |     0 |
|         1 | 1334224842163537338 | 1272415964 |              21 |     1 |     0 |
|         1 | 1334224846422671757 | 1272415968 |              22 |     1 |     0 |
+-----------+---------------------+------------+-----------------+-------+-------+
3 rows in set (0.01 sec)
```

And there they are. You can look up user 20 in the forward direction (`forward_1_edges`) to see the same edge in the forward table.

Bundling up modifications

You can bundle up modify operations in a "transaction":

```
>> flock.transaction do |t|
?>   t.add(1, :follows, 20)
>>   t.add(1, :follows, 30)
>> end
=> nil
```

ContinueFile section 3

It's not a transaction in the database sense, but just a way to bundle multiple modifications into a single RPC call. Flockdb accepts the collection of modifications with a single "okay" and promises to take care of all of them eventually.

FlockDB can also perform a "mass-action" on all edges going to (or from) a vertex:

```
>> flock.remove(229, :follows, nil)
```

ContinueFile

which can be useful when removing a vertex from the system.

Once an edge has been added to the system, it's never deleted. Instead, the state of an edge can be changed to "removed" or "archived". Removing an edge is similar to deleting it, except that the row isn't deleted from mysql for performance reasons.

Archiving an edge changes its state to "archived", which hides it from normal queries, but allows it to be restored to a normal state by "un-archiving" it.

```
>> flock.archive(229, :follows, nil)
>> flock.unarchive(229, :follows, nil)
```

ContinueFile

Compound queries

To find out who's reciprocally following user 1, we can ask for the intersection of "who is following user 1" and "who is user 1 following":

```
>> flock.select(1, :follows, nil).intersect(nil, :follows, 1).to_a
=> [20]
```

ContinueFile

Oh, just user 20. Well, how about the union then?

```
>> flock.select(1, :follows, nil).union(nil, :follows, 1).to_a
=> [30, 22, 21, 20]
```

ContinueFile

Cool. So wait, who's following user 1 that user 1 is *not* following back?

```
>> flock.select(nil, :follows, 1).difference(1, :follows, nil).to_a
=> [22, 21]
```

ContinueFile

Ahh okay.

Paging through results

If the result set is really long, you may want to page through them.

```
>> pager = flock.select(1, :follows, nil).union(nil, :follows, 1).paginate(2)
=> #<Flock::Operation:0x10157a538 ...>
>> pager.next_page
=> [30, 22]
>> pager.next_page
=> [21, 20]
```

ContinueFile

For stateless interaction (like websites), you can manually retrieve the next and previous cursor:

```
>> query = flock.select(1, :follows, nil).union(nil, :follows, 1)
>> page, next_cursor, prev_cursor = query.paginate(2).unapply
=> [[30, 22], 1334246632933954956, 0]
>> page, next_cursor, prev_cursor = query.paginate(2, next_cursor).unapply
=> [[21, 20], 0, -1334246630353519131]
```

ContinueFile

The client library can also handle pagination automatically for you:

```
>> flock.select(1, :follows, nil).union(nil, :follows, 1).paginate(2).each { |n| puts n }
30
22
21
20
```

ContinueFile

(The client is fetching a page of 2 results at a time, and querying for the next page every time it runs out.)

Migrations

As a last demo, let's create a few shards for a new graph "99", add some data, and then migrate it to a new database.

Remember, we create graphs in pairs of positive and negative tables. To create 10 shards for the new graph:

```
$ gizzmo -T -99,99 create-table --base-name=edges --shards=10 1 "com.twitter.gizzard.shards.ReplicatingShard(1) -
Create tables -99, 99:
  com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNS
  for 10 base ids:
    1152921504606846976
    8070450532247928797
    2305843009213693952
    1037629354146162273
    9223372036854775776
    3458764513820540917
    0
    4611686018427387888
    5764607523034234885
    6917529027641081826

Continue? (y/n)
y
```

RootFile

User interaction

RootFile

And to verify that they were created:

```
$ gizzmo -T 99 topology --forwardings
[99] 0 = localhost/edges_99_0003_replicating    com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flc
[99] 1999999999999999 = localhost/edges_99_0009_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] 3333333333333332 = localhost/edges_99_0007_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] 4ccccccccccccccb = localhost/edges_99_0004_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] 6666666666666664 = localhost/edges_99_0002_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] 7ffffffffffffffd = localhost/edges_99_0001_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] 9999999999999996 = localhost/edges_99_0000_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] b33333333333332f = localhost/edges_99_0008_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
```

```
[99] cccccccccccccc8 = localhost/edges_99_0005_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
[99] e66666666666661 = localhost/edges_99_0006_replicating  com.twitter.gizzard.shards.ReplicatingShard(1) -> com
```

Make sure the local flockdb instance reloads the forwarding tables:

RootFile

```
$ gizzmo reload
Are you sure? Reloading will affect production services immediately! (Type 'yes')
yes
```

Make a client with our new graph, and add some edges:

```
>> flock = Flock.new "localhost:7915", :graphs => { :loves => 99 }
>> flock.add(300, :loves, 400)
>> flock.add(600, :loves, 800)
>> flock.add(123456, :loves, 800)
```

RootFile

What shard is user 123456 on?

RootFile

```
$ gizzmo --subtree lookup 99 123456
localhost/edges_99_0005_replicating
  localhost/edges_99_0005
```

Hm, but localhost has been behaving strangely lately. Let's move that shard to 127.0.0.1, which is really lightly loaded. To move individual shards we'll take a look at the current topology and then run a `transform-tree` operation to move things where we want them.

To see the current overall topology of graph 99:

RootFile

```
$ gizzmo -T 99 topology
10 com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UN
```

As we already knew, graph 99 is made up of 10 shards, all of which are replicating to just one server, which is also `localhost` for all them.

Now let's change that shard to replicate only to the new host we want, 127.0.0.1. We'll specify the new topology template we want, and which shard that should apply to:

```
$ gizzmo transform-tree "com.twitter.gizzard.shards.ReplicatingShard(1) -> (com.twitter.flockdb.SqlShard(127.0.0.

com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT UNSIG
  PREPARE
    create_shard(com.twitter.flockdb.SqlShard/127.0.0.1)
    create_shard(WriteOnlyShard)
    add_link(WriteOnlyShard -> com.twitter.flockdb.SqlShard/127.0.0.1)
    add_link(com.twitter.gizzard.shards.ReplicatingShard -> WriteOnlyShard)
  COPY
    copy_shard(com.twitter.flockdb.SqlShard/127.0.0.1)
  CLEANUP
    add_link(com.twitter.gizzard.shards.ReplicatingShard -> com.twitter.flockdb.SqlShard/127.0.0.1)
    remove_link(com.twitter.gizzard.shards.ReplicatingShard -> com.twitter.flockdb.SqlShard/localhost)
    remove_link(WriteOnlyShard -> com.twitter.flockdb.SqlShard/127.0.0.1)
    remove_link(com.twitter.gizzard.shards.ReplicatingShard -> WriteOnlyShard)
    delete_shard(com.twitter.flockdb.SqlShard/localhost)
    delete_shard(WriteOnlyShard)

Continue? (y/n)
```

RootFile

User Interaction

gizzmo gives us the plan for this transformation so we can approve it before it makes any changes. This looks good so put in `y` and let it run. The destination shard will be marked "busy" during the copy, if you're quick you might be able to see it marked such by checking with:

```
$ gizzmo busy
```

RootFile

After the operation has finished, you can check to see that it did what we expect:

RootFile

```
$ gizzmo subtree localhost/edges_99_0005_replicating
localhost/edges_99_0008_forward_replicating
  127.0.0.1/edges_99_0005
$ gizzmo -T 99 topology
   9 com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flockdb.SqlShard(localhost,1,INT UNSIGNED,INT
   1 com.twitter.gizzard.shards.ReplicatingShard(1) -> com.twitter.flockdb.SqlShard(127.0.0.1,1,INT UNSIGNED,INT
```

Sweet! Reload to make sure flockdb knows about the right topology.

RootFile

```
$ gizzmo reload
```

And make sure the data is still there.

Constraint

```
>> flock.select(123456, :loves, nil).to_a
=> [800]
```

The end