Dec 27 2012, Gregory Trubetskoy

What is Thredis?

Thredis is Redis + SQL + Threads. Or perhaps it's pure lunacy resulting from some mad winter hacking mixed with eggnog. Or perhaps it's the first hybrid SQL/NoSQL server. You be the judge.

Thredis embeds an in-memory SQLite database within Redis to enable a sophisticated level of SQL (joins, sub-selects, etc, all supported), as well as introduces multi-threaded processing to take advantage of SMP systems.

Thredis is Work In Progress - currently available at github.com/grisha/thredis
This is ALPHA quality code, tests and docs have not been written (yet).

Now the FUN part, best shown by example:

I'm assuming you're familiar with Redis and its CLI. If not, you should probably spend some time on redis.io first. I'm also assuming you're no stranger to SQL.

Thredis adds a new command 'SQL' which takes one argument, a string of SQL:

```
redis 127.0.0.1:6379> sql "select 'hello from sql' as greeting"
1) 1) "greeting"
2) 1) "hello from sql"
```

We can create a virtual table using the redis module. The name of the table maps to a Redis object (which doesn't have to exist).

```
redis 127.0.0.1:6379> sql "create virtual table foo using redis"
OK
```

Let's create a simple string object.

```
redis 127.0.0.1:6379> set foo bar
OK
```

And let's see if we can use SQL to read it:

```
redis 127.0.0.1:6379> sql "select * from foo"
1) 1) "key"
   2) "val"
2) 1) "1"
   2) "bar"
```

Et voila!

Notice how a SELECT from a redis-mapped table always has two columns, key and val. val is always the value, and key is sometimes just a sequential number (as above) or the actual key or the score (see below).

Let's get rid of the string object. Notice that this doesn't break the virtual table. When the correposnding Redis object does not exist, the table will simply appear empty.

```
redis 127.0.0.1:6379> del foo
(integer) 1
```

How about a list?

```
redis 127.0.0.1:6379> lpush foo a b c
(integer) 3
redis 127.0.0.1:6379> sql "select * from foo"
1) 1) "key"
   2) "val"
2) 1) "1"
   2) "c"
3) 1) "2"
   2) "b"
4) 1) "3"
   2) "a"
```

Or a hash?

```
redis 127.0.0.1:6379> hmset foo_hash a b c d e f
OK
redis 127.0.0.1:6379> sql "select * from foo_hash"
(error) ERR SQL error: no such table: foo_hash
```

Oh, we forgot to create the virtual table so that SQLite can see the Redis object, let's do that:

```
redis 127.0.0.1:6379> sql "create virtual table hash using redis (foo_hash)"
OK
```

Notice how this time we named the table 'hash' and gave the name of the Redis object as an argument at the end.

```
redis 127.0.0.1:6379> sql "select * from hash"
1) 1) "key"
   2) "val"
2) 1) "a"
   2) "b"
3) 1) "c"
   2) "d"
4) 1) "e"
   2) "f"
```

Let's try this with a sorted set?

```
redis 127.0.0.1:6379> zadd foo_zset 1 a 2 b 3 c
(integer) 3
redis 127.0.0.1:6379> sql "create virtual table zset using redis (foo_zset)"
OK
redis 127.0.0.1:6379> sql "select * from zset"
1) 1) "key"
   2) "val"
2) 1) "1.0"
   2) "a"
3) 1) "2.0"
   2) "b"
4) 1) "3.0"
   2) "c"
```

Now let's try joining a hash with a sorted set:

```
redis 127.0.0.1:6379> sql "select * from zset join hash on zset.val = hash.val"
1) 1) "key"
   2) "val"
   3) "key"
   4) "val"
2) 1) "2.0"
   2) "b"
   3) "a"
   4) "b"
```

What else can you do with it?

```
redis 127.0.0.1:6379> sql "select max(key) from zset"
1) 1) "max(key)"
2) 1) "3.0"


redis 127.0.0.1:6379> sql "select sum(key) from zset"
1) 1) "sum(key)"
2) 1) "6.0"
```

We can also create a real SQLite table (it's all in-memory and very fast):

```
redis 127.0.0.1:6379> sql "create table bar (one int, two text)"
OK
redis 127.0.0.1:6379> sql "insert into bar values (101, 'blah')"
OK
redis 127.0.0.1:6379> sql "insert into bar values (202, 'bleh')"
OK
```

This table is NOT a redis object, it exists in SQLite memory space:

```
redis 127.0.0.1:6379> debug object bar
(error) ERR no such key
```

Copy this table into a hash, like so (here we're introducing the
redis() SQL function, which allows executing Redis commands from
within SQL):

```
redis 127.0.0.1:6379> sql "select redis('hset', 'bar_copy', one, two) from bar"
1) 1) "redis('hset', 'bar_copy', one, two)"
2) 1) "1"
3) 1) "1"
```

Just to make sure it worked:

```
redis 127.0.0.1:6379> hgetall bar_copy
1) "101"
2) "blah"
3) "202"
4) "bleh"
```

And let's not forget you can also use Lua in conjunction with this.

```
eval "return redis.call('sql', 'select * from foo')" 0
1) 1) "key"
   2) "val"
2) 1) "1.0"
   2) "a"
3) 1) "2.0"
```

```
   2) "b"
4) 1) "3.0"
   2) "c"
```

Triggers are supported as well. Let's say we have a SQL table called blah which we created like this:

```
redis 127.0.0.1:6379> sql "create table blah (one int, two text);"
OK
```

And we want to count all updates of column text in a Redis key called 'blah_update_count' using a trigger. We create the trigger like so:

```
redis 127.0.0.1:6379> sql "CREATE TRIGGER blah_update_count UPDATE OF two ON blah BEGIN SELECT redis('incr', 'blah_update_count'); END"
OK
```

What happens if we insert a record into blah?

```
redis 127.0.0.1:6379> sql "insert into blah values (2012, 'new year')"
OK
redis 127.0.0.1:6379> get blah_update_count
(nil)
```

Nothing. This is because that was an INSERT, not an UPDATE. Let's try an UPDATE:

```
redis 127.0.0.1:6379> sql "update blah set two = 'old year' where one = 2012"
OK
redis 127.0.0.1:6379> get blah_update_count
"1"
```

Hopefully by now you're getting the idea.

Your SQL data will get saved for you automatically in the background according to your Redis SAVE policy and loaded on startup. It is saved in a separate file by default called dump.sqlite, and it is simply a SQLite database, which can be opened using the sqlite3 command. Accessing your Redis virtual tables from a sqlite3 command line will give you errors, of course.

You can also force a save manually:

```
redis 127.0.0.1:6379> sqlsave
OK
```

What do threads have to do with any of this? Thredis started out as simply threaded Redis. The idea of SQLite integration came later. Why and how threads are used is explained in README-THREDIS.

6 comments Sort by **Oldest**

Add a comment...

Chase Sechrist

awesome

Like · Reply · 9y

Brett Wilkins

Zed Shaw created MulletDB (http://mulletdb.com) a while back as the first SQL/NoSQL hybrid database. Postgres has also had the option of the hstore extension for a bit now too. Sorry if I'm being "that guy."

Like · Reply · 9y

Chase Sechrist

There was this as well http://code.google.com/p/alchemydatabase/

Like · Reply · 9y

Josh Varty

very cool

Like · Reply · 9y

Areek Zillur

was just looking into that

Like · Reply · 9y

Jeffrey Kao

the point of redis is to use it like simple data structures, so that the way you write application code will be similar to how you make redis calls... I think a SQL interface just uglifies it tenfold, no?

Like · Reply · 9y

Rudraksh Mukta Kulshreshtha

This a very cool idea! Maybe we could fork of some of our production code, and use it with thredis, see how it goes..

Like · Reply · 9y

Yunfan Jiang

I noticed that sqlite3 has an internal VM inside, so why not do that from the other side? from sqlite3 interface access redis datastruct.

Like · Reply · 9y

Grisha Trubetskoy

That's what the SQL redis() function does? Look for "select redis('hset', 'bar_copy', one, two) from bar" above.

Like · Reply · 9y

Huitao Yu

very cool ti's a good project i like it ..

Like · Reply · 8y

Facebook Comments Plugin