Craig Kerstiens

Sharding Your Database

I’m increasingly encountering users on Heroku that are encountering the need to shard their data. For most users this is something you delay as long as possible as you can generally go for sometime before you have to worry about it. Additionally scaling up your database is often a reasonable approach early on and something I encourage as a starting point as scaling up is easy to do with regards to databases. However, for the 1% of users that do need to shard when the time comes many are left wondering where to start, hence the following guide.

What and Why

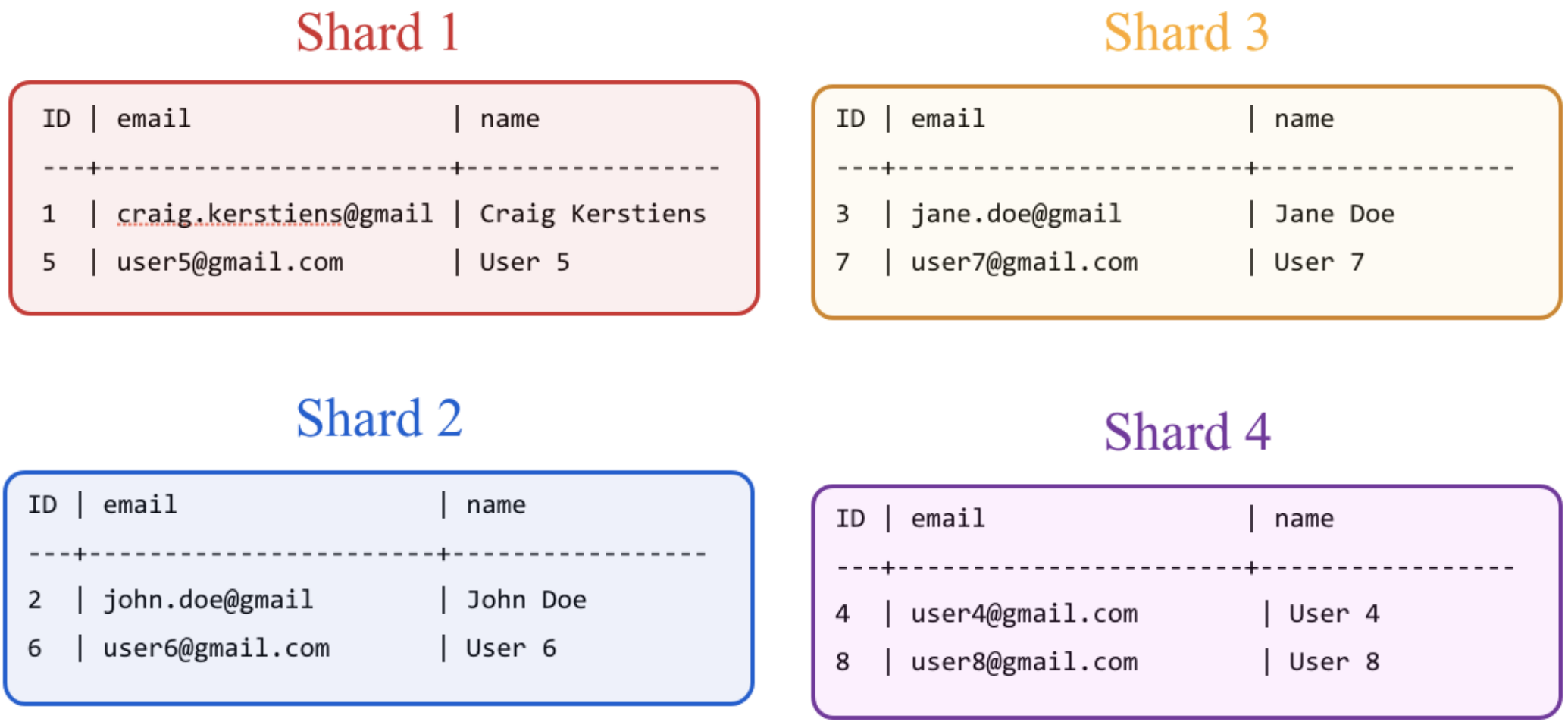
Sharding is the process of splitting up your data so it resides in different tables or often different physical databases. Sharding is helpful when you have some specific set of data that outgrows either storage or reasonable performance within a single database.

Logical Shards

First when initially implementing sharding you’ll want to create an arbitrary number of logical shards. This will allow you to change less code later when it comes to adding more shards. You’ll also want to define your shards to the power of 2. Generally I’d recommend for most services 1024 can be a good number, I believe Instagram actually used 4096, either can really be an appropriate number. For simplicity sake lets start with an example of using 4 logical shards. First lets look at an example set of users:

id	email	name
1	craig.kerstiens@gmail.com	Craig Kerstiens
2	john.doe@gmail.com	John Doe
3	jane.doe@gmail.com	Jane Doe
4	user4@gmail.com	User 4
5	user5@gmail.com	User 5
6	user6@gmail.com	User 6
7	user7@gmail.com	User 7
8	user8@gmail.com	User 8

Dividing these up into logical shards we’re going to have something that looks roughly like this:



sharding layout

Its important when sharding that you find a mechanism that requires you to not hit the database. As the above example shows its using the ID of the row inside the database instead we’re likely going to want to determine the shard based on a hash of some value such as the email:

```
logical_shard = hash(User.email) % 4
```

Physical Shards

From here we'll then take the logical shards and create actual physical shards. If you have a single physical shard you're using a single database, but the rest of your application code is ready to handle additional shards. For now lets use an example of two physical shards, the end result would be dividing our data up somehow like this:

Shard 1

ID	email	name
1	craig.kerstiens@gmail	Craig Kerstiens
5	user5@gmail.com	User 5

ID	email	name
3	jane.doe@gmail	Jane Doe
7	user7@gmail.com	User 7

Shard 2

ID	email	name
2	john.doe@gmail	John Doe
6	user6@gmail.com	User 6

ID	email	name
4	user4@gmail.com	User 4
8	user8@gmail.com	User 8

sharding layout

The physical shard to access can easily be counted by taking the modulus of the logical shard its on by the physical shards that exist:

```
total_physical_shards = 2
total_logical_shards = 4
logical_shard = hash(User.email) % total_logical_shards
physical_shard = logical_shard % total_physical_shards
```

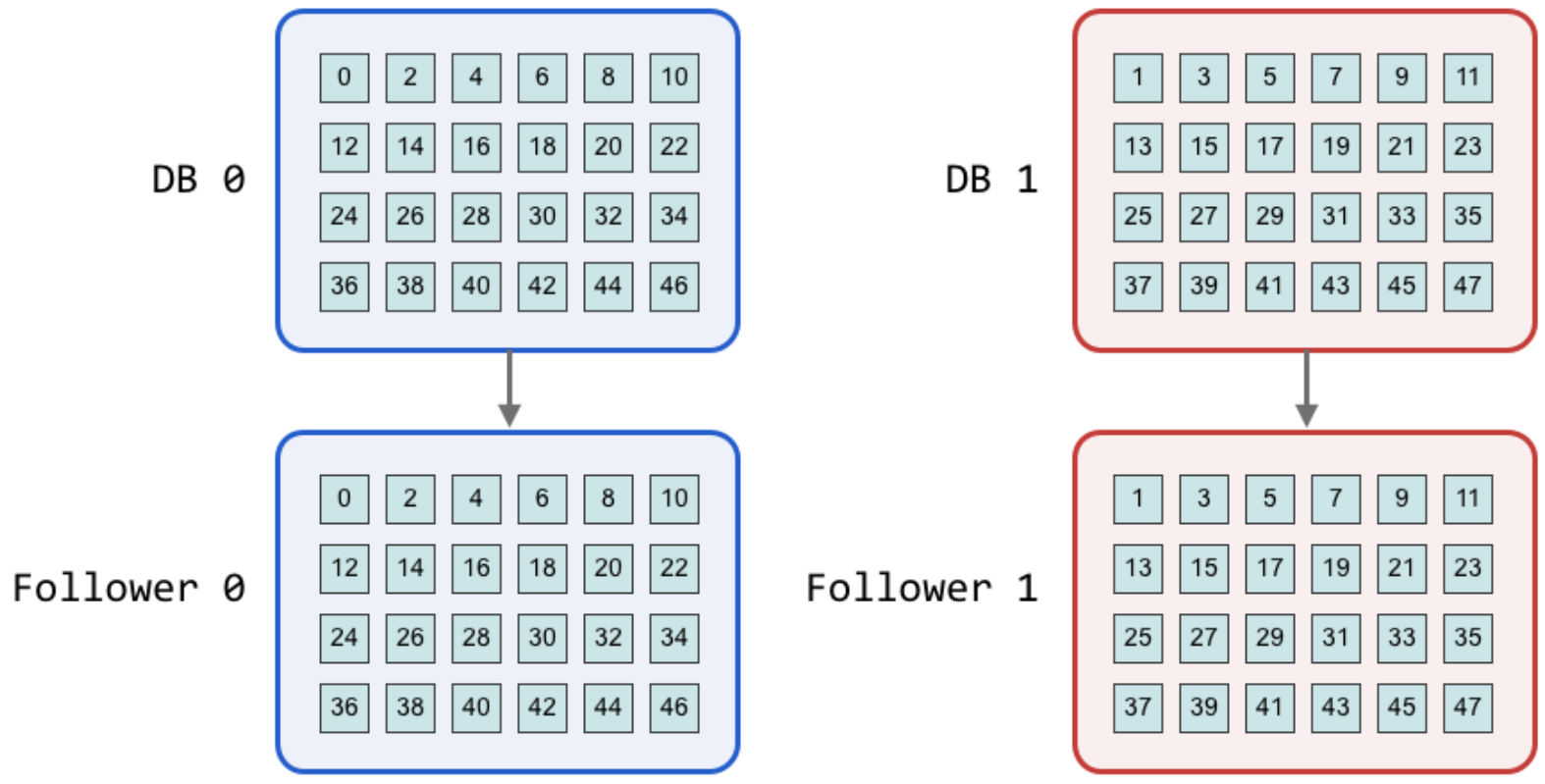
Generating IDs (Primary Keys)

As you're distributing data across multiple databases you'll want to avoid using an integer as your primary key. This would cause for keys to be duplicated within your database and make for a headache when attempting to report against your data. Instead the ideal is to use a UUID as the primary key. By using a UUID and generating this in either your application code or within your database you ensure each User ID is actually unique.

Adding Capacity

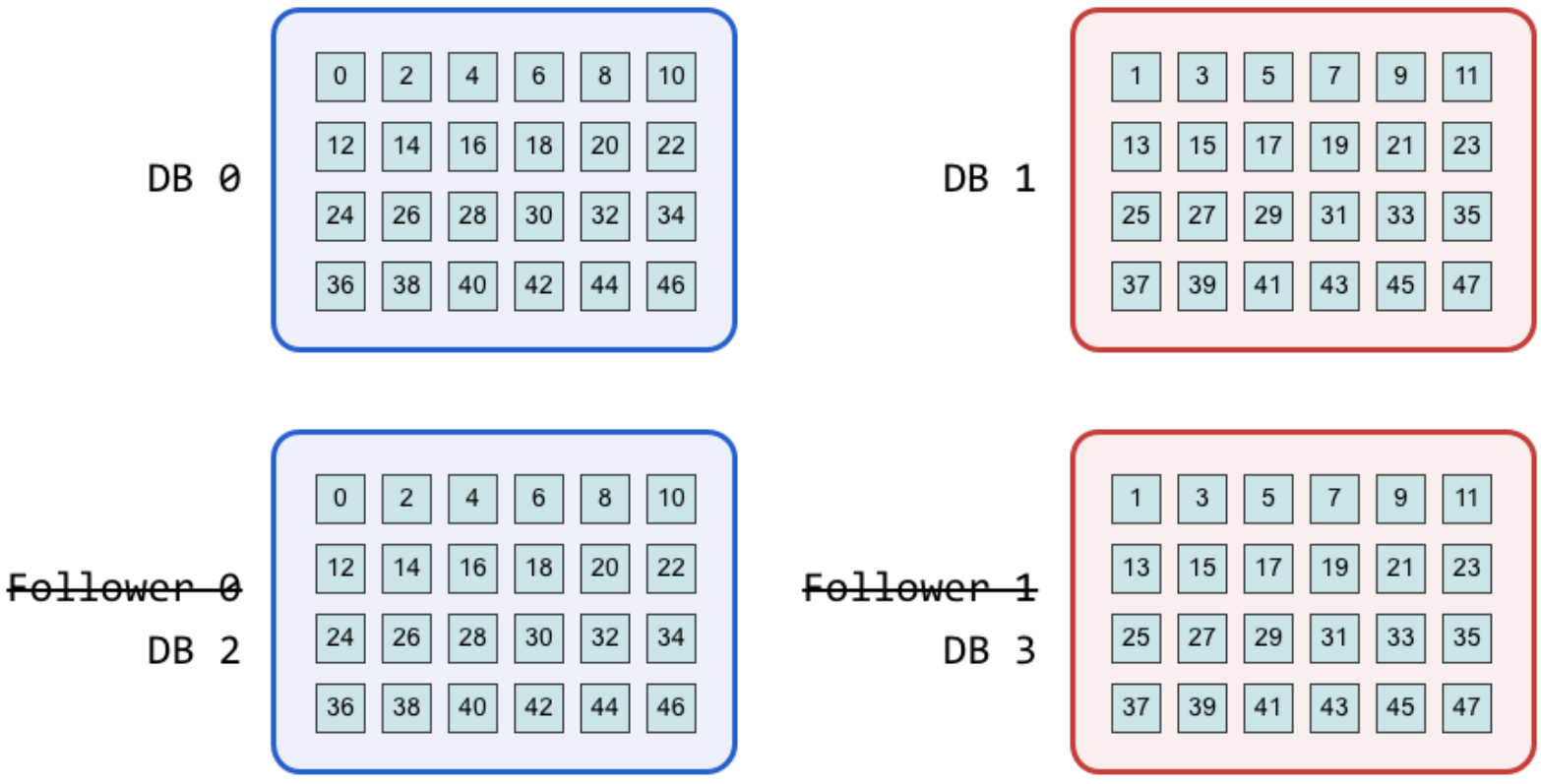
The best case scenario for most web applications, such is the case for Instagram, is to have to scale beyond their initial capacity, in order to do this you'll simple expand the number of physical shards. In order to do this you'll want to move data from one physical shard to another, then remove data from the old physical shard. Its also generally a good practice to grow your physical shards in powers of 2 the same way you would your logical shards. Lets take a look at a clearer example of how we would do this using the Heroku Postgres Service…

First we're going to add a follower for each shard we have:



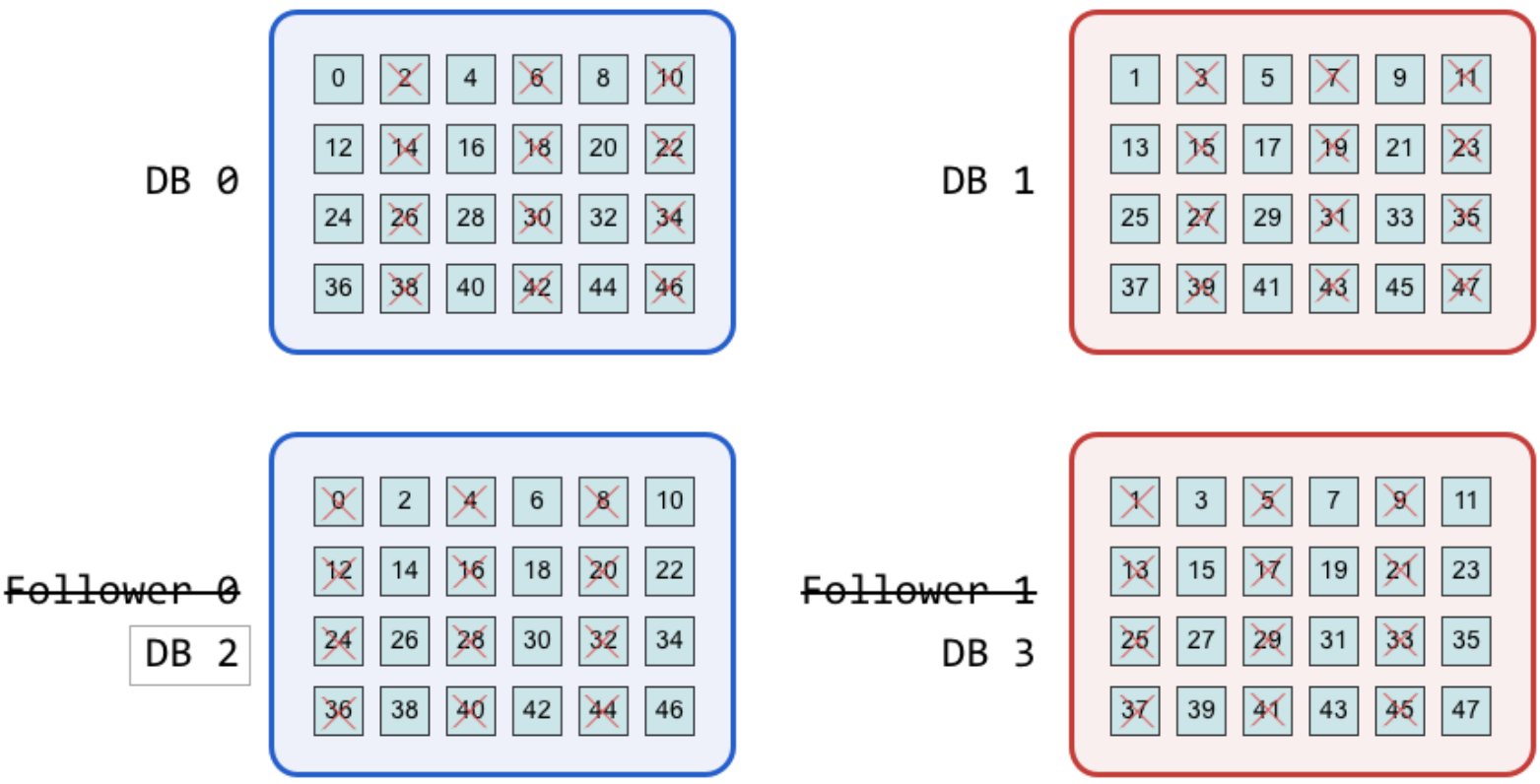
Followers

We're then going to promote our followers to be their own independent databases which can accept writes. This means we'll have two copies that can be written to with the same data:



Promotion

At this point you can update your application code to have the new number of physical shards and it should begin writing data to the appropriate place. Of course you do still want to clean up some of that extra data. To do this you'll want to remove the data that doesn't belong in the appropriate shard. For example, id of 3 wouldn't belong in physical shard 1 any more. Now we can run a background process to clean up such data:



Cleanup

Conclusion

While many applications may never need to scale out their database, when they do, sharding can be both straight forward and effective. While I would encourage many to scale up first as it is an easy option, hopefully this provides further guidance to how to scale out. For those that do anticipate this needed planning for it early with key things such as using UUID's can make the process less painful.

This article of course only grazes the surface, if there's interest from readers there will be more specifics to follow with actual code examples.

Development