# Tiny URL

**How does your system ensure that 2 URLs never map to the same shortened URL?**

IIntroduce MySql unique key constraint on the tiny url field.

**How will you ensure the system is very low latency?**

Introduce a cache between stateless instances and the MySql Serving Layer. So whenever a fetch request comes we can check whether the tiny url exists in the cache or not. If it exists, serve from the cache. If it doesn’t query from mysql and then store it in the caching and then serve the response.

**What will happen if the machine storing the URL mapping dies? (Power outage / Hard Disk gone bad)**

We should have one Master MySql DB and 1 Slave DB and 1 Standby DB with replication. In case the master mysql dies, we can make the Slave instance as the new Master. And stand by as the new slave.

**How do you make sure your system is consistent? This is to say, if I have shortened a URL, given the shortened URL, my system should always return the original URL no matter when I call your system.**

Unique key constraint for the tiny URL will ensure only single url is mapped to an url and the same will be returned maintaining the consistency.

**How do you make sure that your service never goes down? (No outage). You have to assume that your machines will die. How do you make sure your service is unaffected by those incidents (unless all of your machines die at once - Lets assume that never happens).**

We can use ELB which will have multiple stateless instances behind it. Each stateless instance in turn will use Redis cache for fetch requests which will be highly available. Similarly in case of cache miss and new url shortening request the serving layer will query to MySQL which is again setup into a master slave setup, so that we could promote the Slave as master in case the current master dies.

tate

Using the above approach we have handled outages at the serving layer, cache layer and the DB layer.