1. Code design

```python
#!/usr/bin/python3
# -*-coding:utf8-*-
import requests
import urllib
import pymysql
from django.shortcuts import render

#Get the request URL. If it is a short URL, forward the request directly.
#If it is not a short URL, create a short URL and then forward it.
def reponse_url(url):
    #url = 'http://www.google.com'
    reques_url=url
    if reques_url:
        headers ={
        "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36",
        }
        request = urllib.request.Request(url,headers=headers)
        search_short_url_sql = F''' select real_url t_short_url where short_url = {reques_url} '''

        try:
            conn = get_conn()
            cur = conn.cursor()
            len = cur.execute(search_short_url_sql)
            if len:
                real_url = cur.fetchone()
                if real_url:
                    request_url(request, real_url)
            else:
                short_url=get_short_url(reques_url)
                real_url=reques_url
                short_sql = F'''insert into t_short_url(short_url,real_url) values({short_url},{reques_url})'''
                excute_mysql_save_short_url(short_sql)
                request_url(request, real_url)

```

```
            except Exception as e:  # catch all exception
                # If an exception occurs, roll back
                print("exception", Exception)
                print("open exception: %s: %s" % (e.errno, e.strerror))
                conn.rollback()
            finally:
                # Finally close the database connection
                cur.close()
                conn.close()

#Forwarding module
def request_url(request,real_url):
    return render(request, real_url)

#Generate short URL
def get_short_url(reques_url):
    #short url Generation method.................
    return short_url

#Get database connection
def get_conn():
    conn = pymysql.connect(host="192.168.1.1", user="db_user",password="xxxxxxx", database="db_name", charset="utf8")
    return conn

#Save short URL
def excute_mysql_save_short_url(short_sql):
    try:
        conn = get_conn()
        cur = conn.cursor()
        cur.execute(short_sql)

    except Exception as e:  # catch all exception
        # If an exception occurs, roll back
        print("exception", Exception)
        print("open exception: %s: %s" % (e.errno, e.strerror))
        conn.rollback()
    finally:
```

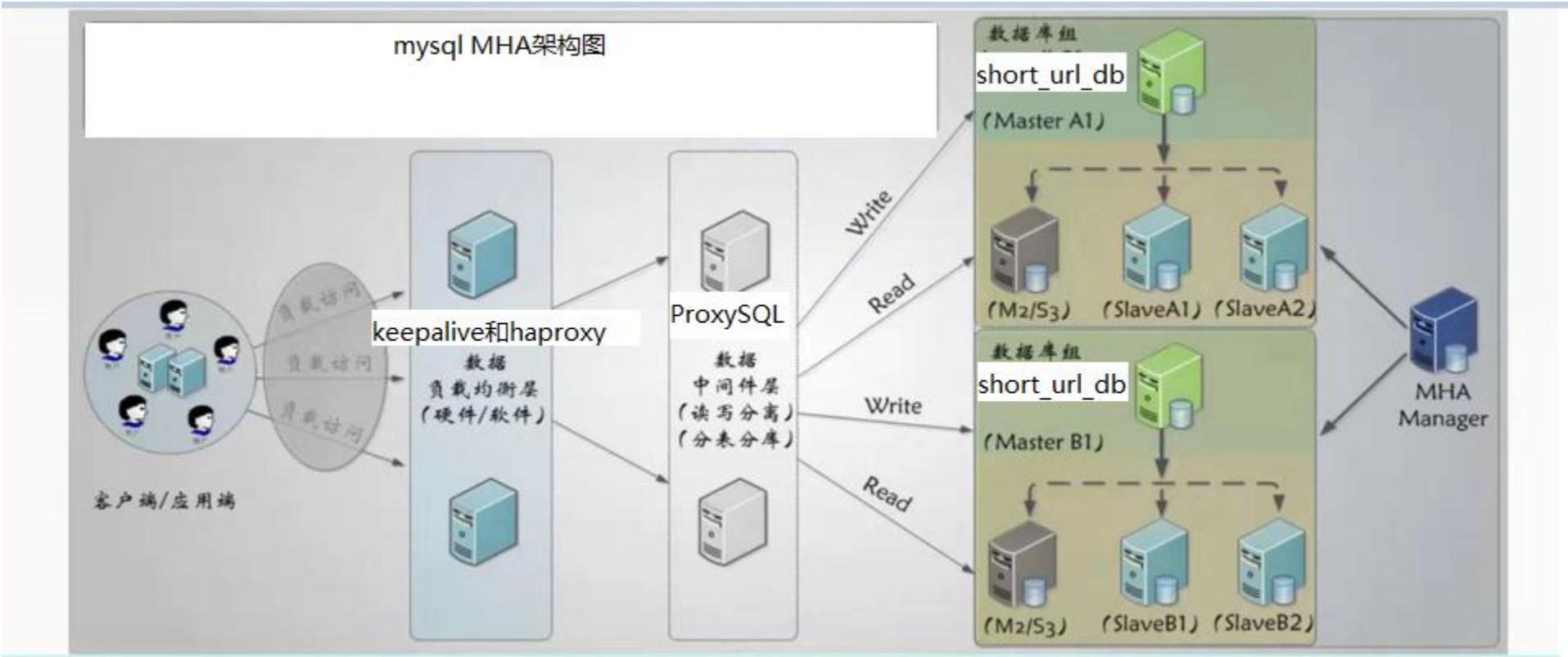
2. DATABASE design

```
CREATE DATABASE `short_url_db` /*!40100 DEFAULT CHARACTER SET utf8 */
```

3. Table design

```
CREATE TABLE `t_short_url` (
  `short_url_id` bigint(22) NOT NULL AUTO_INCREMENT,
  `short_url` varchar(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `real_url` varchar(1024) NOT NULL,
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`short_url_id`),
  UNIQUE KEY `idx_short_url` (`short_url`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

4. Database architecture diagram



5. Design description:

1)I choose a data architecture which is commonly used at present.It can automatically switch to the standby primary node when the current primary database fails.
2)Middleware can use proxysql or MYCAT
3)Load balancing middleware can use keepalive and haproxy
4)Data disaster recovery database can be implemented by message middleware
5)If the concurrency is high, redis cache can be used. The application program increases the code of operation cache.
6)When the amount of data is large, we can consider the logical partition and physical partition of the table, and we can also consider the use of distributed database