1.分区

mysql数据库中的数据是以文件的形势存在磁盘上的，默认放在/mysql/data下面（可以通过my.cnf中的datadir来查看），一张表主要对应着三个文件，一个是**frm存放表结构**的，一个是**myd存放表数据**的，一个是**myi存表索引**的。如果一张表的数据量太大的话，那么myd,myi就会变的很大，查找数据就会变的很慢，这个时候我们可以利用mysql的分区功能，

在物理上将这一张表对应的三个文件，分割成许多个小块，这样呢，我们查找一条数据时，就不用全部查找了，只要知道这条数据在哪一块，然后在那一块找就行了。

如果表的数据太大，可能一个磁盘放不下，这个时候，我们可以把数据分配到不同的磁盘里面去

分区的二种方式

a，横向分区

什么是横向分区呢？就是横着来分区了，举例来说明一下，假如有100W条数据，分成十份，前10W条数据放到第一个分区，第二个10W条数据放到第二个分区，依此类推。也就是把表分成了十分，根用merge来分表，有点像哦。取出一条数据的时候，这条数据包含了表结构中的所有字段，也就是说横向分区，并没有改变表的结构。

b，纵向分区

什么是纵向分区呢？就是竖来分区了，举例来说明，在设计用户表的时候，开始的时候没有考虑好，而把个人的所有信息都放到了一张表里面去，这样这个表里面就会有比较大的字段，如个人简介，而这些简介呢，也许不会有好多人去看，所以等到有人要看的时候，在去查找，分表的时候，可以把这样的大字段，分开来。

mysql提供的分区属于第一种，横向分区，并且细分成很多种方式：

1.1 MySQL5.1及以上支持分区功能

查看是否支持分区

```
mysql> show variables like "%part%";
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| have_partitioning | YES   |
+-------------------+-------+
1 row in set (0.00 sec)
```

1.2 range 分区

这种模式允许将数据划分不同范围。例如可以将一个表通过年份划分成若干个分区

```
 create table t_range(
      id int(11),
      money int(11) unsigned not null,
      date datetime
   )partition by range(year(date))(
   partition p2007 values less than (2008),
   partition p2008 values less than (2009),
   partition p2009 values less than (2010)
   partition p2010 values less than maxvalue  #MAXVALUE 表示最大的可能的整数值
   );
```

RANGE分区在如下场合特别有用：

1）、当需要删除一个分区上的“旧的”数据时，只删除分区即可。如果你使用上面最近的那个例子给出的分区方案，你只需简单地使用”ALTER TABLE employees DROP PARTITION p0;”

来删除所有在1991年前就已经停止工作的雇员相对应的所有行。对于有大量行的表，这比运行一个如”DELETE FROM employees WHERE YEAR (separated) <= 1990;”

这样的一个DELETE查询要有效得多。

2）、想要使用一个包含有日期或时间值，或包含有从一些其他级数开始增长的值的列。

3）、经常运行直接依赖于用于分割表的列的查询。

例如，当执行一个如”SELECT COUNT(*) FROM employees WHERE YEAR(separated) = 2000 GROUP BY store_id;”这样的查询时，

MySQL可以很迅速地确定只有分区p2需要扫描，这是因为余下的分区不可能包含有符合该WHERE子句的任何记录

1.3 list分区

这种模式允许系统通过预定义的列表的值来对数据进行分割。

```
create table t_list(
   a int(11),
   b int(11)
   )(partition by list (b)
   partition p0 values in (1,3,5,7,9),
   partition p1 values in (2,4,6,8,0)
);
```

LIST分区没有类似如“VALUES LESS THAN MAXVALUE”这样的包含其他值在内的定义。将要匹配的任何值都必须在值列表中找到。

1.4 hash分区

这中模式允许通过对表的一个或多个列的Hash Key进行计算，最后通过这个Hash码不同数值对应的数据区域进行分区。例如可以建立一个对表主键进行分区的表。

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

1.5 key分区

上面Hash模式的一种延伸，这里的Hash Key是MySQL系统产生的。

```
CREATE TABLE tk (
    col1 INT NOT NULL,
    col2 CHAR(5),
    col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

1.6 子分区

子分区是分区表中每个分区的再次分割，子分区既可以使用HASH希分区，也可以使用KEY分区。这 也被称为复合分区（*composite partitioning*）。

1，如果一个分区中创建了子分区，其他分区也要有子分区

2，如果创建了了分区，每个分区中的子分区数必有相同

3，同一分区内的子分区，名字不相同，不同分区内的子分区名子可以相同（5.1.50不适用

```
mysql> CREATE TABLE IF NOT EXISTS `sub_part` (
 ->    `news_id` int(11) NOT NULL  COMMENT '新闻ID',
 ->    `content` varchar(1000) NOT NULL DEFAULT '' COMMENT '新闻内容',
 ->    `u_id`  int(11) NOT NULL DEFAULT 0s COMMENT '来源IP',
 ->    `create_time` DATE NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT '时间'
 -> ) ENGINE=INNODB  DEFAULT CHARSET=utf8
 -> PARTITION BY RANGE(YEAR(create_time))
 -> SUBPARTITION BY HASH(TO_DAYS(create_time))(
 -> PARTITION p0 VALUES LESS THAN (1990)(SUBPARTITION s0,SUBPARTITION
s1,SUBPARTITION s2),
 -> PARTITION p1 VALUES LESS THAN (2000)(SUBPARTITION s3,SUBPARTITION
s4,SUBPARTITION good),
 -> PARTITION p2 VALUES LESS THAN MAXVALUE(SUBPARTITION tank0,SUBPARTITION
tank1,SUBPARTITION tank3)
 -> );
Query OK, 0 rows affected (0.07 sec)
```


分区的优点

1，分区可以分在多个磁盘，存储更大一点

2，根据查找条件，也就是where后面的条件，查找只查找相应的分区不用全部查找了

3，进行大数据搜索时可以进行并行处理。

4，跨多个磁盘来分散数据查询，来获得更大的查询吞吐量

1.7 分区管理

a.删除分区

alter table user drop partion p4

b.新增分区

```
alter table user add partition(partition p4 values less than MAXVALUE); #新增
range分区
alter table list_part add partition(partition p4 values in(25,26,27))    #新增list
分区
alter table hash_part add partition partitions 4; # hash重新分区
```

```
alter table key_part add partition partitions 4; #key 重新分区
//子分区添加新分区，虽然我没有指定子分区，但是系统会给子分区命名的
alter table sub1_part add partition(partition p3 values less than MAXVALUE);
//range重新分区
ALTER TABLE user REORGANIZE PARTITION p0,p1,p2,p3,p4 INTO (PARTITION p0 VALUES
LESS THAN MAXVALUE);
//list重新分区
 ALTER TABLE list_part REORGANIZE PARTITION p0,p1,p2,p3,p4 INTO (PARTITION p0
VALUES in (1,2,3,4,5));
 #hash和key分区不能用REORGANIZE，官方网站说的很清楚
```

参考文献：http://blog.csdn.net/yongchao940/article/details/55266603
http://www.cnblogs.com/mliudong/p/3625522.html

2.分表管理

2.1 MySQL集群

利用mysql cluster ， mysql proxy，mysql replication，drdb等等

有人会问mysql集群，根分表有什么关系吗？虽然它不是实际意义上的分表，但是它启到了分表的作用，做集群的意义是什么呢？为一个数据库减轻负担，说白了就是减少sql排队队列中的sql的数量，
举个例子：有10个sql请求，如果放在一个数据库服务器的排队队列中，他要等很长时间，如果把这10个sql请求，分配到5个数据库服务器的排队队列中，一个数据库服务器的队列中只有2个，
这样等待时间是不是大大的缩短了呢？这已经很明显了。所以我把它列到了分表的范围以内；集群我们在第三部分详情说明；

优点：扩展性好，没有多个分表后的复杂操作（php代码）
缺点：单个表的数据量还是没有变，一次操作所花的时间还是那么多，硬件开销大。

2.2 预先估计会出现的大数据并且访问频繁的表，将其分为若干个表
我事先建100个这样的表，
message_00,message_01,message_02.....message_98,message_99.然后根据用户的ID来判断这个用户的聊天信息放到哪张表里面，
你可以用hash的方式来获得，可以用求余的方式来获得，方法很多，各人想各人的吧。下面用hash的方法来获得表名：

```
<?php
function get_hash_table($table,$userid) {
```

```
 $str = crc32($userid);
 if($str<0){
 $hash = "0".substr(abs($str), 0, 1);
 }else{
 $hash = substr($str, 0, 2);
 }

 return $table."_".$hash;
}

echo get_hash_table('message','user18991');      //结果为message_10
echo get_hash_table('message','user34523');     //结果为message_13
```

优点：避免一张表出现几百万条数据，缩短了一条sql的执行时间
缺点：当一种规则确定时，打破这条规则会很麻烦，上面的例子中我用的hash算法是crc32，如果我现在不想用这个算法了，改用md5后，会使同一个用户的消息被存储到不同的表中，
这样数据乱套了。扩展性很差。

2.3 利用merge存储引擎来实现分表

merge分表，分为主表和子表，主表类似于一个壳子，逻辑上封装了子表，实际上数据都是存储在子表中的。

```
mysql> CREATE TABLE IF NOT EXISTS `user1` (
 ->    `id` int(11) NOT NULL AUTO_INCREMENT,
 ->    `name` varchar(50) DEFAULT NULL,
 ->    `sex` int(1) NOT NULL DEFAULT '0',
 ->    PRIMARY KEY (`id`)
 -> ) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE IF NOT EXISTS `user2` (
 ->    `id` int(11) NOT NULL AUTO_INCREMENT,
 ->    `name` varchar(50) DEFAULT NULL,
 ->    `sex` int(1) NOT NULL DEFAULT '0',
 ->    PRIMARY KEY (`id`)
 -> ) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO `user1` (`name`, `sex`) VALUES('张映', 0);
```

```
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO `user2` (`name`, `sex`) VALUES('tank', 1);
Query OK, 1 row affected (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS `alluser` (
 ->   `id` int(11) NOT NULL AUTO_INCREMENT,
 ->   `name` varchar(50) DEFAULT NULL,
 ->   `sex` int(1) NOT NULL DEFAULT '0',
 ->   INDEX(id)
 -> ) TYPE=MERGE UNION=(user1,user2) INSERT_METHOD=LAST
AUTO_INCREMENT=1 ;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

创建主表的时候有个INSERT_METHOD，指明插入方式，取值可以是：0 不允许插入；FIRST 插入到UNION中的第一个表； LAST 插入到UNION中的最后一个表。

通过主表查询的时候，相当于将所有子表合在一起查询。这样并不能体现分表的优势，建议还是查询子表。

优点：扩展性好，并且程序代码改动的不是很大

缺点：这种方法的效果比第二种要差一点，查询性能不高

参考资料：http://blog.51yip.com/mysql/949.html

3.集群

MySQL Proxy就是这么一个中间层代理，简单的说，MySQL Proxy就是一个连接池，负责将前台应用的连接请求转发给后台的数据库，并且通过使用lua脚本，可以实现复杂的连接控制和过滤，从而实现读写分离和负载

平衡。对于应用来说，MySQL Proxy是完全透明的，应用则只需要连接到MySQL Proxy的监听端口即可。当然，这样proxy机器可能成为单点失效，但完全可以使用多个proxy机器做为冗余，在应用服务器的连接池配置

中配置到多 个proxy的连接参数即可。