哈尔滨工业大学

**计算机科学与技术学院/国家示范性软件学院**

**2019年春季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 3：数据层软件架构实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| **刘思琦** | **1163710228** | **hit1163710228@163.com/18846183728** |
| **石昊** | **1163710226** | **2670041246@qq.com/15645077257** |

目 录

[1 实验概述 1](#_Toc9010651)

[1.1 实验目的 1](#_Toc9010652)

[1.2 实验要求 1](#_Toc9010653)

[2 实验内容与过程 1](#_Toc9010654)

[2.1 Mycat数据库分库分表实验 1](#_Toc9010655)

[2.2 Sharding-JDBC数据库分库分表实验 3](#_Toc9010656)

[2.3 Redis数据缓存实验 7](#_Toc9010657)

[3 结对开发过程记录 9](#_Toc9010658)

[4 实验总结 10](#_Toc9010659)

[文档全部完成之后，请在上述 区域点击右键，选择“更新域”，在打开的对话框中选择“更新整个目录”]

# 实验概述

## 实验目的

1）学习使用Mycat和Sharding-JDBC实现数据分库分表

2）学习使用Redis数据库实现数据缓存

3）能够灵活应用Mycat或Sharding-JDBC实现分库分表架构到实际系统

4）能够灵活应用Redis实现数据缓存架构到实际系统

## 实验要求

1）2人结对成组

2）实验2.1、2.2、2.3均为必做

3）结合《软件过程与工具》课程中进销存系统(或其他实际软件系统)进行数据层架构重构，实现根据业务垂直划分的数据库分库分表；面向海量数据带来的数据检索慢问题，实现数据库水平分片，达到数据检索的性能提升；利用缓存架构实现数据读取的性能提升。

4）应给出关键过程的细节

# 实验内容与过程

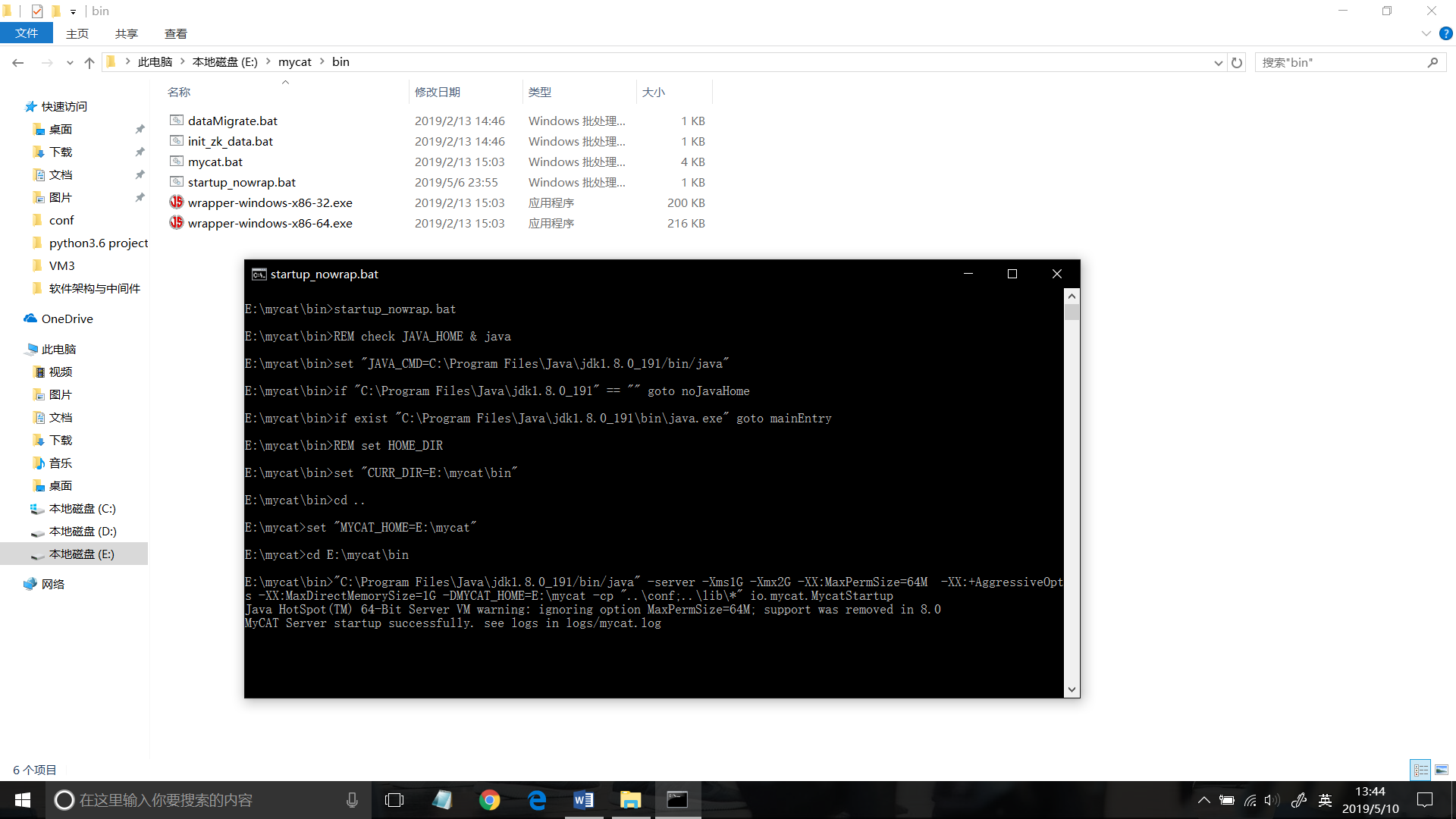
根据给出的“实验指导书-Mycat-Sharding-JDBC”、 “实验指导书-Redis”完成实验2.1-2.3，并回答相关问题。

## Mycat数据库分库分表实验

**1）请给出Mycat配置安装过程中遇到的问题和解决方案。**

在Mycat的安装过程中没有遇到问题，去官网下载与操作系统对应的压缩包到本地然后解压缩即可。惯例是存放在以全英文命名的目录下。

配置Mycat时，在windows10下双击startup\_nowrap.bat文件启动Mycat的时候遇到了闪退的情况。解决方案是以管理员身份打开控制台，在控制台下启动startup\_nowrap.bat。



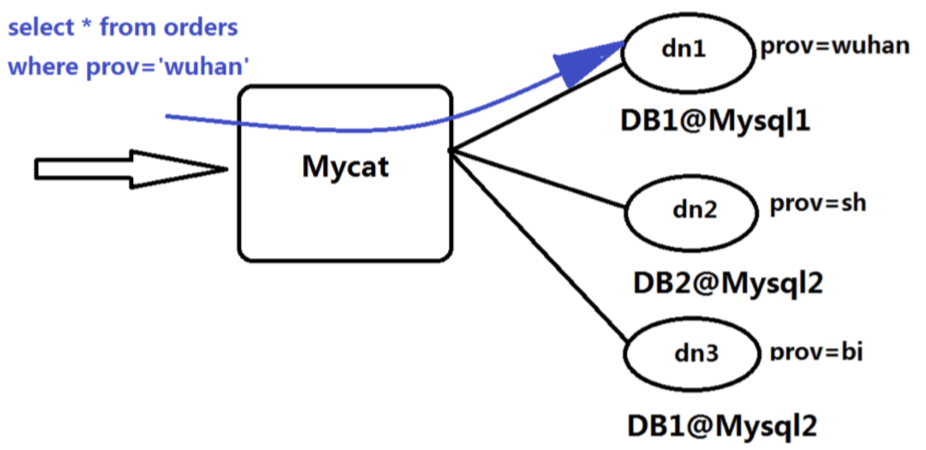
*（更正：不以管理员身份启动控制台也可以）*

**2）请详析Mycat的分库分表原理和操作方法。**

**1.Mycat的分库分表原理**

Mycat的原理中最重要的一个动词是“拦截”，它拦截了用户发送过来的SQL

语句，首先对SQL语句做了一些特定的分析：如分片分析、路由分析、读写分离分析、缓存分析等，然后将此SQL发往后端的真实数据库，并将返回的结果做适当的处理，最终再返回给用户。



以上图为例，orders表被分为三个分片datanode（简称dn)，这三个分片是分

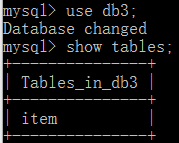
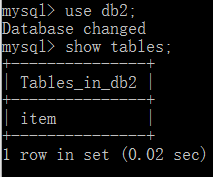
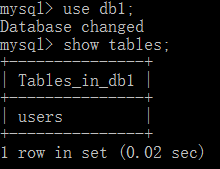
布在两台MySQL Server上(DataHost)，即 datanode=database@datahost方式，因此可以用一台到N台服务器来分片，分片规则为（sharding rule)典型的字符串枚举分片规则，一个规则的定义是分片字段（sharding column)+分片函数(rulefunction)，这里的分片字段为prov而分片函数为字符串枚举方式。

当Mycat收到一个SQL时，会先解析这个SQL，查找涉及到的表，然后看此表的定义，如果有分片规则，则获取到SQL里分片字段的值，并匹配分片函数，得到该SQL对应的分片列表，然后将SQL发往这些分片去执行，最后收集和处理所有分片返回的结果数据，并输出到客户端。以select \* from orders where prov=?语句为例，查到prov=wuhan，按照分片函数，wuhan返回dn1，于是SQL就发给了MySQL1，去取DB1上的查询结果，并返回给用户。

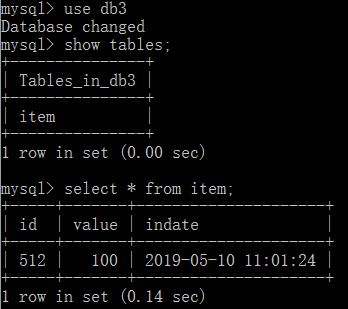
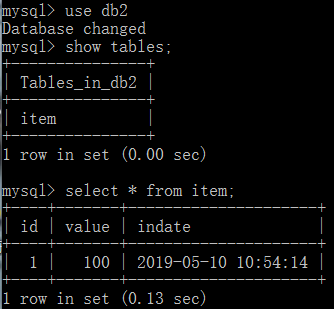
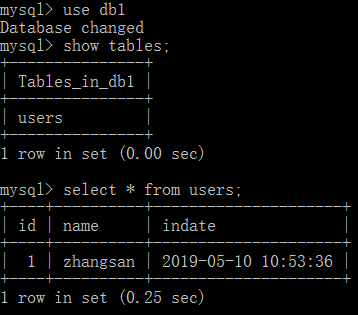
**2.Mycat的分库分表操作方法**

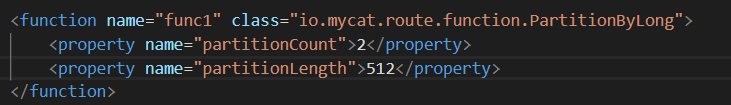
①参考实验指导书配置Mycat目录下的schema.xml，rule.xml和server.xml

②登录物理数据库mysql，使用实验指导书中的sql语句创建数据库和表，具体过程很简单，不在此赘述了。创建之后通过命令行或者数据库可视化软件来查看创建的结果：



③启动Mycat服务，插入数据进行分库分表的测试，然后登陆到物理数据库查看结果。可以看到，插入的user表中的数据全部在db1中，而item表中的数据分布在db2和db3中，这样就根据实际的路由策略完成了分表。

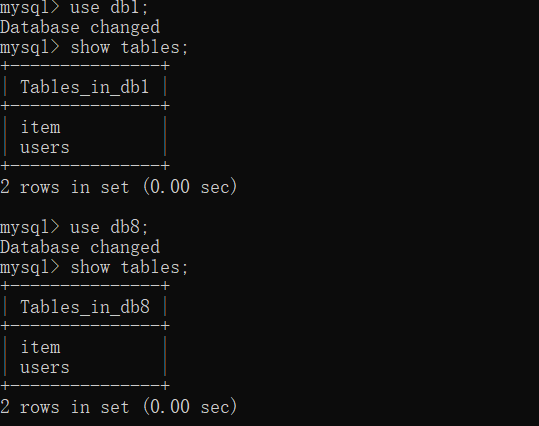


**（对结果的解释：在Mycat的配置目录conf中查看分表配置rule.xml，其中有一行：，其中配置了partitionLength，即分表的长度依据）**

**3）请模拟具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。**

①在mysql数据库中创建数据库

创建八个数据库db1-db8,每个数据库里面有item和users;



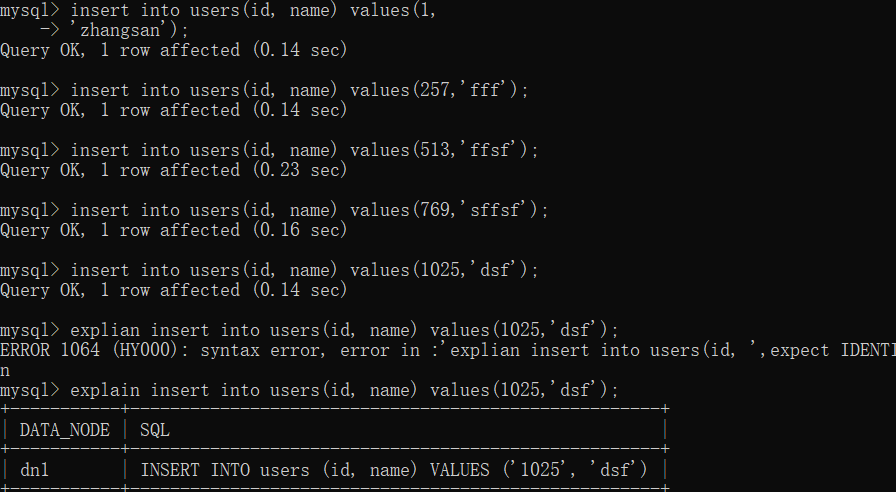
②配置Mycat目录下的schema.xml，rule.xml和server.xml



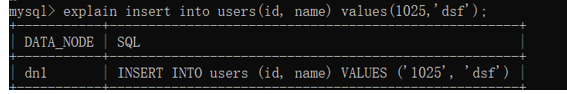


数据库1235的user分片，数据库4678的item分片，分片大小为64

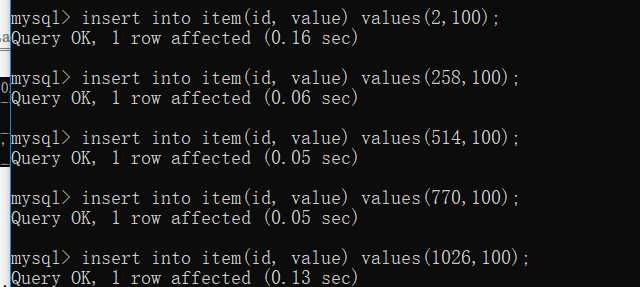
③插入数据，查看分片结果



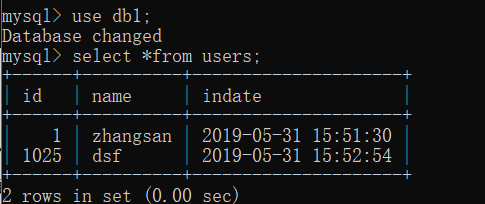
特别的查询

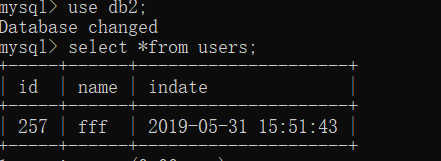


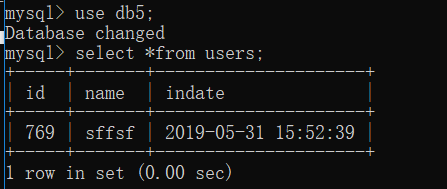
知道四个分片是以256为分片大小一直循环下去。

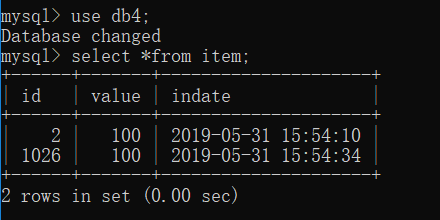


进入物理数据库查看结果









从结果可知，将数据库1235的user按256大小分片，将数据库4678的item按256大小分片

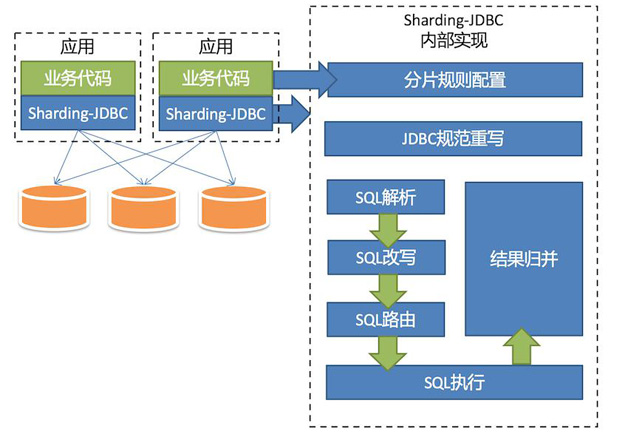
Sharding-JDBC数据库分库分表实验

**1）请给出****Sharding-JDBC配置安装过程中遇到的问题和解决方案。**

按照实验指导书中的步骤一步一步走来，没有出现任何问题。

**2）请详析Sharding-JDBC的分库分表原理和操作方法。**

**1. Sharding-JDBC的分库分表原理**



上图为Sharding-JDBC的整体架构图。

**分片规则配置**

Sharding-JDBC的分片逻辑非常灵活，支持分片策略自定义、复数分片键、多运算符分片等功能。如：根据用户ID分库，根据订单ID分表这种分库分表结合的分片策略；或根据年分库，月份+用户区域ID分表这样的多片键分片。

Sharding-JDBC除了支持等号运算符进行分片，还支持in/between运算符分片，提供了更加强大的分片功能。

Sharding-JDBC提供了spring命名空间用于简化配置，以及规则引擎用于简化策略编写。由于目前刚开源分片核心逻辑，这两个模块暂未开源，待核心稳定后将会开源其他模块。

**JDBC规范重写**

Sharding-JDBC对JDBC规范的重写思路是针对DataSource、Connection、Statement、PreparedStatement和ResultSet五个核心接口封装，将多个真实JDBC实现类集合（如：MySQL JDBC实现/DBCP JDBC实现等）纳入Sharding-JDBC实现类管理。

Sharding-JDBC尽量最大化实现JDBC协议，包括addBatch这种在JPA中会使用的批量更新功能。但分片JDBC毕竟与原生JDBC不同，所以目前仍有未实现的接口，包括Connection游标，存储过程和savePoint相关、ResultSet向前遍历和修改等不太常用的功能。此外，为了保证兼容性，并未实现JDBC 4.1及其后发布的接口（如：DBCP 1.x版本不支持JDBC 4.1）。

**SQL解析**

SQL解析作为分库分表类产品的核心，性能和兼容性是最重要的衡量指标。目前常见的SQL解析器主要有fdb/jsqlparser和Druid。Sharding-JDBC使用Druid作为SQL解析器，经实际测试，Druid解析速度是另外两个解析器的几十倍。

目前Sharding-JDBC支持join、aggregation（包括avg）、order by、 group by、limit、甚至or查询等复杂SQL的解析。目前不支持union、部分子查询、函数内分片等不太应在分片场景中出现的SQL解析。

**SQL改写**

SQL改写分为两部分，一部分是将分表的逻辑表名称替换为真实表名称。另一部分是根据SQL解析结果替换一些在分片环境中不正确的功能。这里具两个例子：

第1个例子是avg计算。在分片的环境中，以avg1 +avg2+avg3/3计算平均值并不正确，需要改写为（sum1+sum2+sum3）/（count1+count2+ count3）。这就需要将包含avg的SQL改写为sum和count，然后再结果归并时重新计算平均值。

第2个例子是分页。假设每10条数据为一页，取第2页数据。在分片环境下获取limit 10, 10，归并之后再根据排序条件取出前10条数据是不正确的结果。正确的做法是将分条件改写为limit 0, 20，取出所有前2页数据，再结合排序条件算出正确的数据。可以看到越是靠后的Limit分页效率就会越低，也越浪费内存。有很多方法可避免使用limit进行分页，比如构建记录行记录数和行偏移量的二级索引，或使用上次分页数据结尾ID作为下次查询条件的分页方式。

**SQL路由**

SQL路由是根据分片规则配置，将SQL定位至真正的数据源。主要分为单表路由、Binding表路由和笛卡尔积路由。

单表路由最为简单，但路由结果不一定落入唯一库（表），因为支持根据between和in这样的操作符进行分片，所以最终结果仍然可能落入多个库（表）。

Binding表可理解为分库分表规则完全一致的主从表。举例说明：订单表和订单详情表都根据订单ID作为分片键，任意时刻分片逻辑均相同。这样的关联查询和单表查询难度和性能相当。

笛卡尔积查询最为复杂，因为无法根据Binding关系定位分片规则的一致性，所以非Binding表的关联查询需要拆解为笛卡尔积组合执行。查询性能较低，而且数据库连接数较高，需谨慎使用。

**SQL执行**

路由至真实数据源后，Sharding-JDBC将采用多线程并发执行SQL，并完成对addBatch等批量方法的处理。

**结果归并**

结果归并包括4类：普通遍历类、排序类、聚合类和分组类。每种类型都会先根据分页结果跳过不需要的数据。

普通遍历类最为简单，只需按顺序遍历ResultSet的集合即可。

排序类结果将结果先排序再输出，因为各分片结果均按照各自条件完成排序，所以采用归并排序算法整合最终结果。

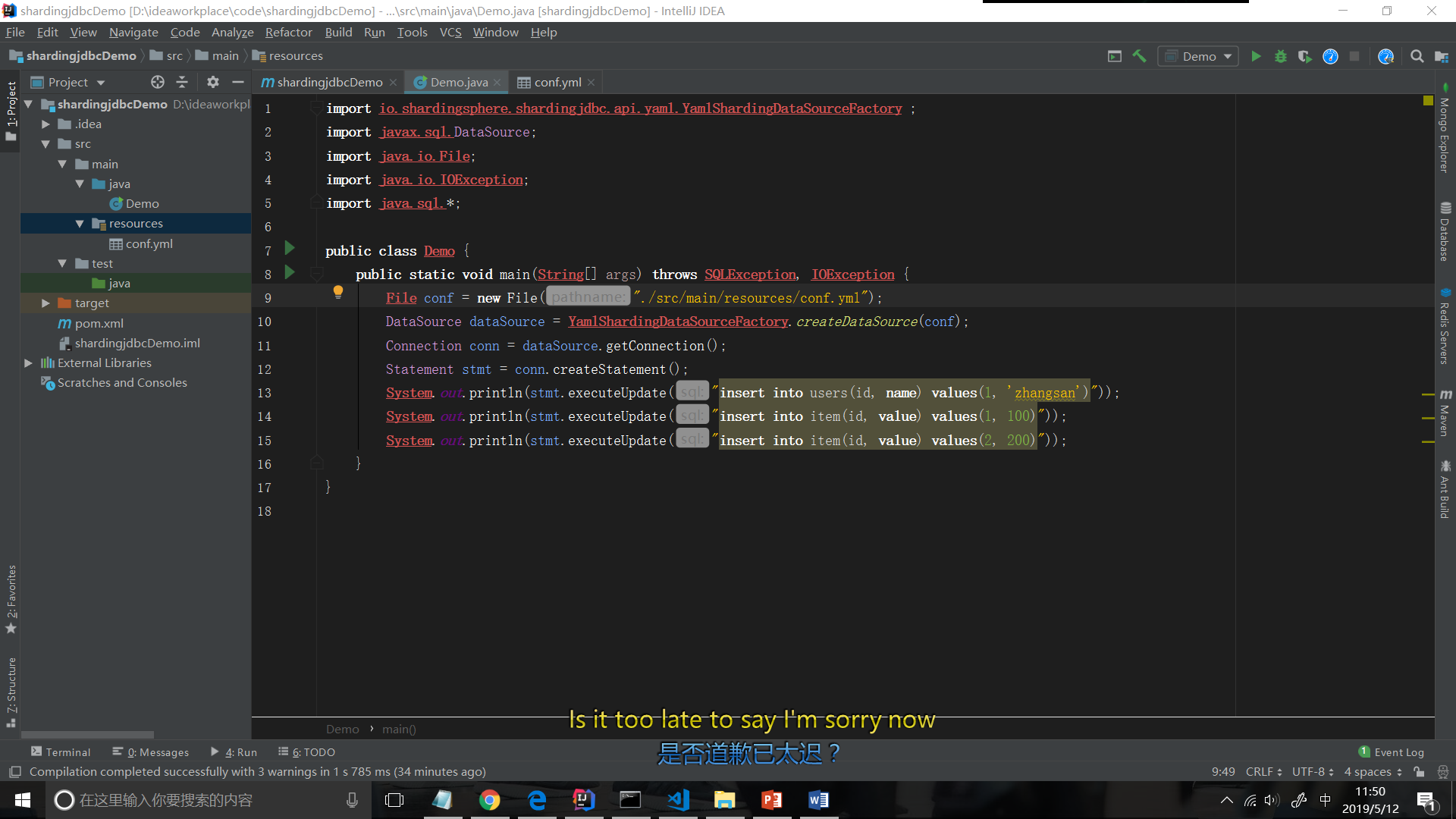
聚合类分为3种类型，比较型、累加型和平均值型。比较型包括max和min，只返回最大（小）结果。累加型包括sum和count，需要将结果累加后返回。平均值则是通过SQL改写的sum和count计算，相关内容已在SQL改写涵盖，不再赘述。

分组类最为复杂，需要将所有的ResultSet结果放入内存，使用map-reduce算法分组，最后根据排序和聚合条件做相关处理。最消耗内存，最损失性能的部分即是此，可以考虑使用limit合理的限制分组数据大小。

结果归并部分目前并未采用管道解析的方式，之后会针对这里做更多改进。

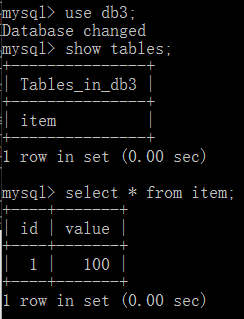
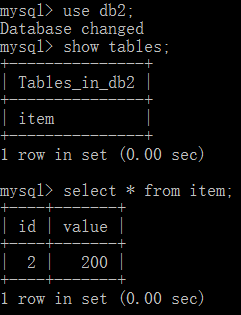
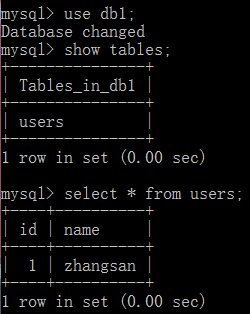
**2. Sharding-JDBC的操作方法**

①使用IDEA创建Maven项目，并配置pom.xml，等待Maven自动导入依赖。代码文件和配置文件参考实验指导书即可，配置完成后的项目目录如图所示：



②登录物理数据库MySQL，按实验指导书的sql语句编写SQL文件并导入。

③运行Demo.java，然后查看数据库中是否成功插入数据。结果如图所示：



**对结果图片的说明：**从结果中可以看到，成功实现了对item的分库分表，其中id为1的数据项被分配到db3中的item表，id为2的数据项被分配到db2中的item表。

**结果分析：**查看conf.yml中对item的分库分表规则：

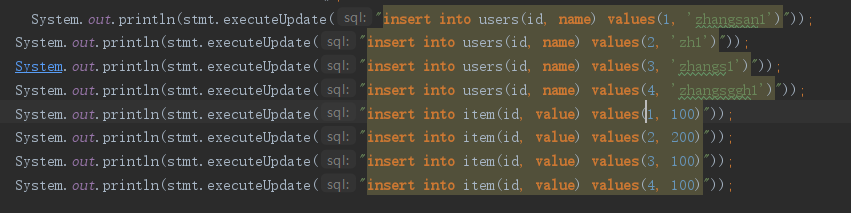


即，按id对2取余然后加2后的值进行分库，id=1时结果是3，所以分到db3中；id=2是结果是2，所以分到db2中，与图片中的结果吻合。

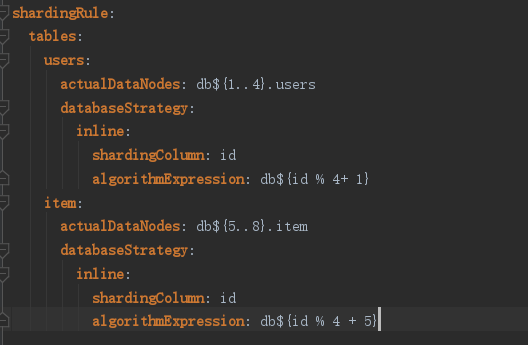
**3）请模拟具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。**

1.创建数据库1-8，每个数据库有两个表item,users.将数据库1-4中的users分片，5-8中的item分片

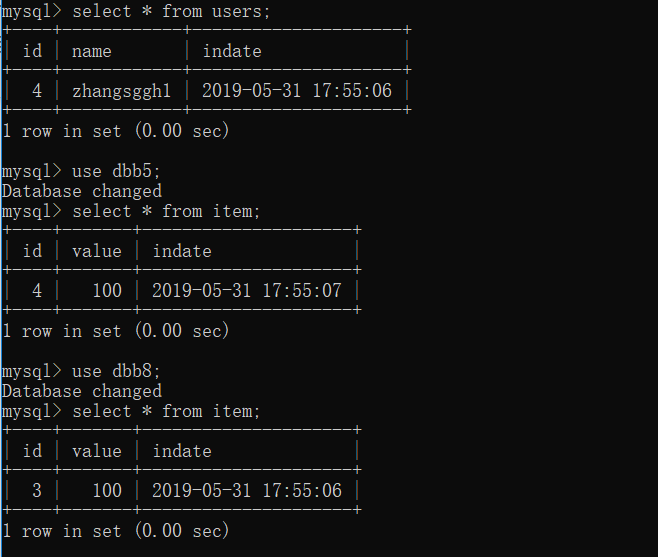
插入数据如下



分片规则如下



在数据库中查询



## Redis数据缓存实验

**1）请给出Redis配置安装过程中遇到的问题和解决方案。**

由于我在Lab1中间件的设计中已经应用了Redis，对其有所了解，所以在安装过程中没有遇到任何的问题。

**2）请详析Redis的缓存清洗策略，数据迁移及扩容策略，面向缓存雪崩、穿透等问题的策略。**

**①Redis缓存清洗策略**

Redis可以看作是一个内存数据库，可以通过Maxmemory指令配置Redis的数据集使用指定量的内存。当内存使用达到maxmemory极限时，需要使用某种淘汰算法来决定清理掉哪些数据，以保证新数据的存入。

常用的淘汰算法有三种：

FIFO：First In First Out，先进先出。判断被存储的时间，离目前最远的数据优先被淘汰。

LRU：Least Recently Used，最近最少使用。判断最近被使用的时间，目前最远的数据优先被淘汰。

LFU：Least Frequently Used，最不经常使用。在一段时间内，数据被使用次数最少的，优先被淘汰。

Redis提供了六种淘汰策略：

noeviction：达到内存限额后返回错误，客户尝试可以导致更多内存使用的命令（大部分写命令，但DEL和一些例外）

allkeys-lru：为了给新增加的数据腾出空间，驱逐键先试图移除一部分最近使用较少的（LRC）。

volatile-lru：为了给新增加的数据腾出空间，驱逐键先试图移除一部分最近使用较少的（LRC），但只限于过期设置键。

allkeys-random: 为了给新增加的数据腾出空间，驱逐任意键

volatile-random: 为了给新增加的数据腾出空间，驱逐任意键，但只限于有过期设置的驱逐键。

volatile-ttl: 为了给新增加的数据腾出空间，驱逐键只有秘钥过期设置，并且首先尝试缩短存活时间的驱逐键

**②Redis数据迁移及扩容策略**

**数据迁移**主要有三种方式：



**扩容集群**是分布式存储最常见的需求，Redis集群扩容可以分为如下步骤：

准备新节点→加入集群→迁移槽和数据

**③Redis面向缓存雪崩、穿透等问题的策略**

**缓存雪崩**是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。

**解决策略：**

方案1：加锁排队

方案2、建立备份缓存，缓存A和缓存B，A设置超时时间，B不设值超时时间，先从A读缓存，A没有读B，并且更新A缓存和B缓存;

方案3、设置缓存超时时间的时候加上一个随机的时间长度，比如这个缓存key的超时时间是固定的5分钟加上随机的2分钟，可从一定程度上避免雪崩问题

**缓存穿透**是指查询一个一定不存在的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。在流量大时，可能DB就挂掉了，要是有人利用不存在的key频繁攻击我们的应用，这就是漏洞。

**解决策略：**有很多种方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法，如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟

**3）请模拟一个简单场景，实现缓存读写操作，缓存更新操作，给出缓存的效果，分析2问题中相关策略的效果。**

# 结对开发过程记录

**（1）角色切换与任务分工**

表1-1结对开发角色与任务分工

| 日期 | 时间(HH:MM - HH:MM) | 驾驶员角色 | 领航员角色 | 本段时间的任务 |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

【注意】该表格可自行增加更多的行。

**（2）工作日志**

由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。

表1-2 结对开发工作日志

| 日期/时间 | 问题描述 | 最终解决方法 | 交流过程 |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

【注意】该表格可自行增加更多的行

**（3）结对开发工作现场照片**

请其他同学帮助拍摄结对开发现场照片至少2张。

结对开发现场照片1 结对开发现场照片2

# 实验总结

【本实验的收获与不足，对计算层软件架构中各种问题的理解和认识】