数据库架构师做什么？

58同城数据库架构设计思路

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

大数据技术探索和价值发现

技术中心-沈剑

shenjian@58.com

关于我-@58沈剑

- 前百度高级工程师
- 58同城技术委员会主席，高级架构师
- 58同城优秀讲师
- @58沈剑

目录

- 基本概念
- 58同城数据库架构设计思路
- 总结

一、基本概念

基本概念

- **分片**：sharding
- **复制**：replication
- **分组**：group
- **路由规则**：router rule
- 常用路由方法

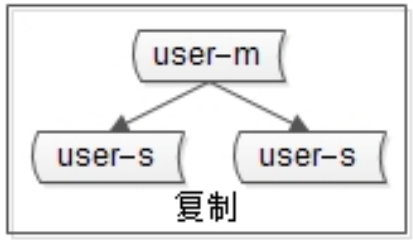
（1）范围：range

（2）哈希：hash

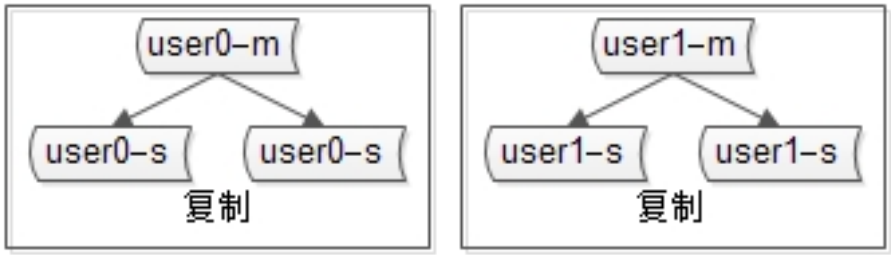
（3）路由服务：router-config-server

单库

分片



分组



实际应用：分组+分片

二、数据库架构设计思路

数据库架构师做什么？

数据库架构设计点

- 可用性
- 读性能
- 一致性
- 扩展性

（1）数据扩容

（2）增加字段

（3）水平拆分

- SQL玩法

（１）如何保证数据的可用性？

保证可用性的思路：冗余

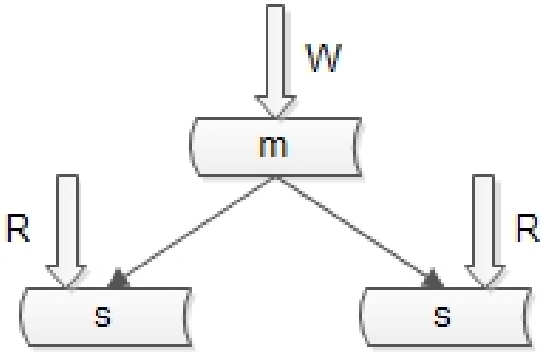
可用性：复制+冗余

- 如何保证**站点**的可用性？复制站点，多机冗余站点
- 如何保证**服务**的可用性？复制服务，多机冗余服务
- 如何保证**数据**的可用性？复制数据，多机冗余数据
- 数据冗余带来的问题？

数据冗余会引发一致性问题

数据可用性：数据冗余

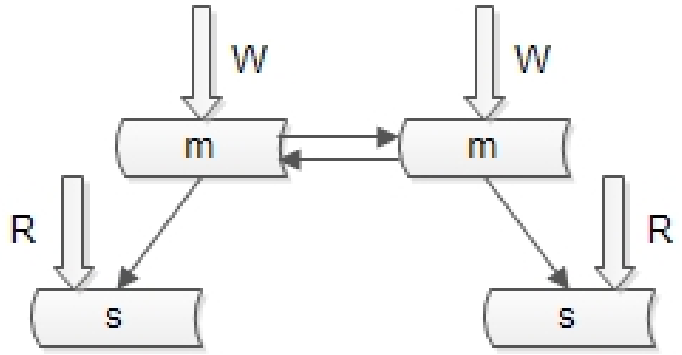
- 如何保证数据库“读”高可用？
- 存在什么问题？
- 怎么解决？



“读”高可用

数据可用性：数据冗余

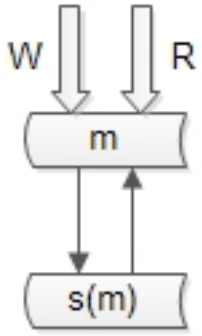
- 如何保证数据库“写”高可用？
- 存在什么问题？
- 怎么解决？



“写”高可用

数据可用性：58的玩法

- 58同城怎么玩数据可用性：“双主”当“主从”用
- 如何解决读写一致性问题？
- 如何解决读写可用性问题？
- 还存在什么问题？



“读写”高可用

（2）如何扩展读性能？

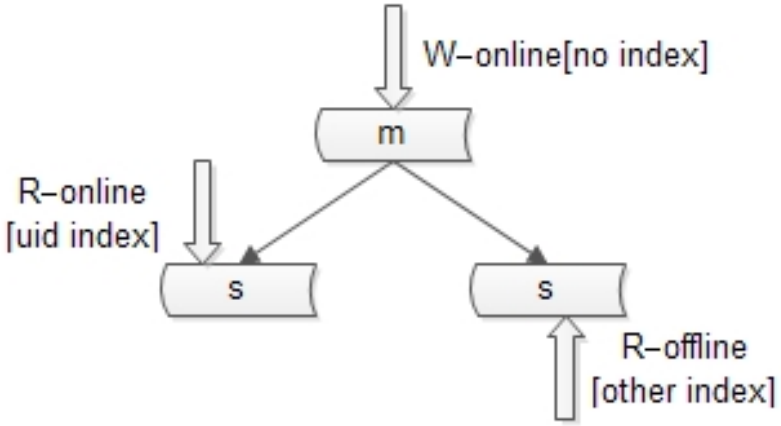
如何提高读性能：增加索引

- 存在什么问题？

（1）写性能降低

（2）索引占用内存大，数据命中率降低

- 有什么优化方案？



如何提高读性能：增加从库

- 存在什么问题？

（1）从库越多，同步越慢

（2）数据不一致

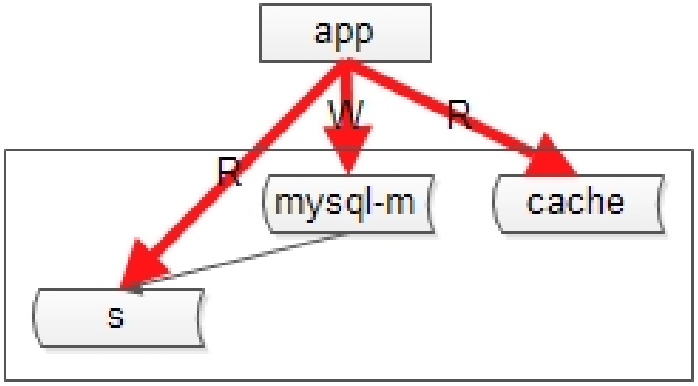
- 有什么优化方案？【见后文】
- 58同城没有采用这种玩法扩充读性能

如何提高读性能：增加缓存

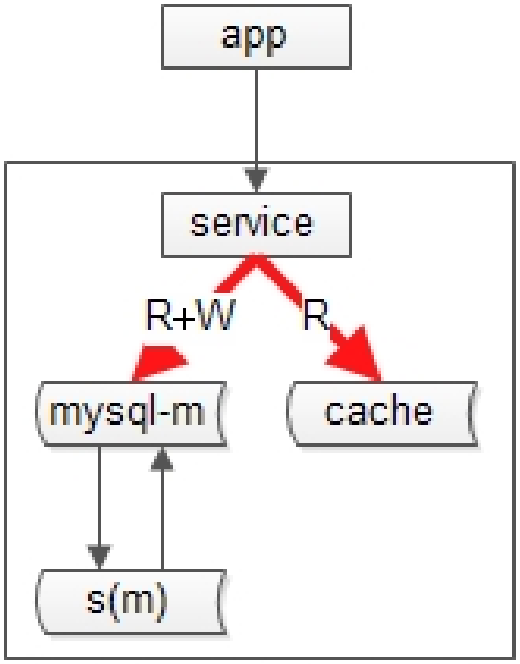
- 常见的缓存玩法存在什么问题？

（1）冗余会引发一致性问题

- 有什么优化方案？【见后文】
- 58同城怎么玩缓存



常见玩法：缓存+数据



58玩法：服务+缓存+数据

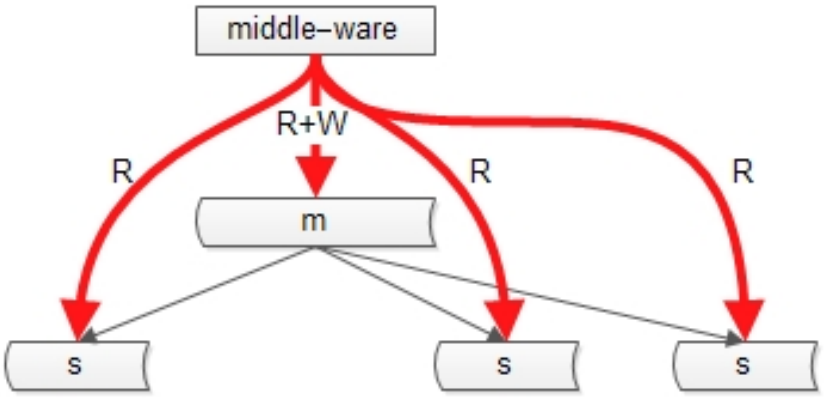
（３）如何保证一致性？

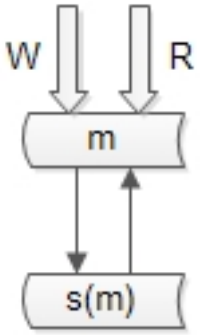
一致性：主从不一致

- 为什么会不一致？
- 主从不一致如何优化？

（1）引入中间件

（2）强制读主



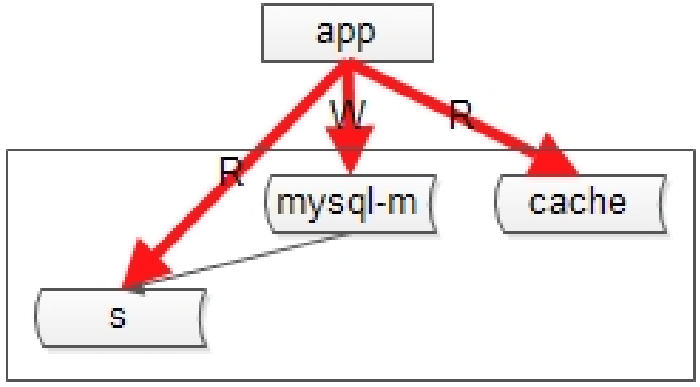


一致性：缓存不一致

- 为什么会不一致？
- 缓存不一致如何优化？

（1）缓存双淘汰

（2）设定过期时间

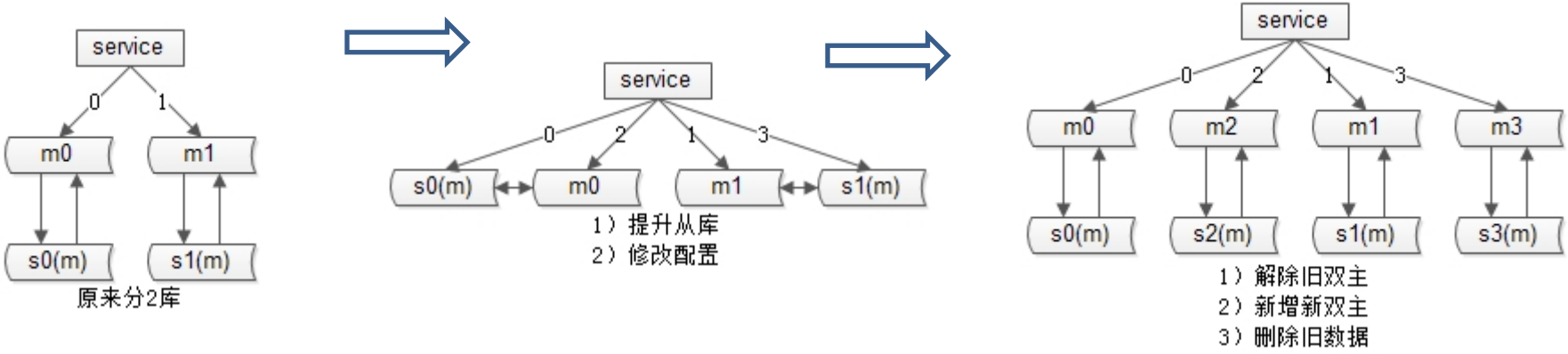


常见玩法：缓存+数据

（４）如何保证扩展性？

扩展性：数据扩容

- 原来水平切分成n个库，要扩容成2n个库，如何实现？
- 58同城秒级数据扩容
- 存在什么缺点，如何解决？【见后文】



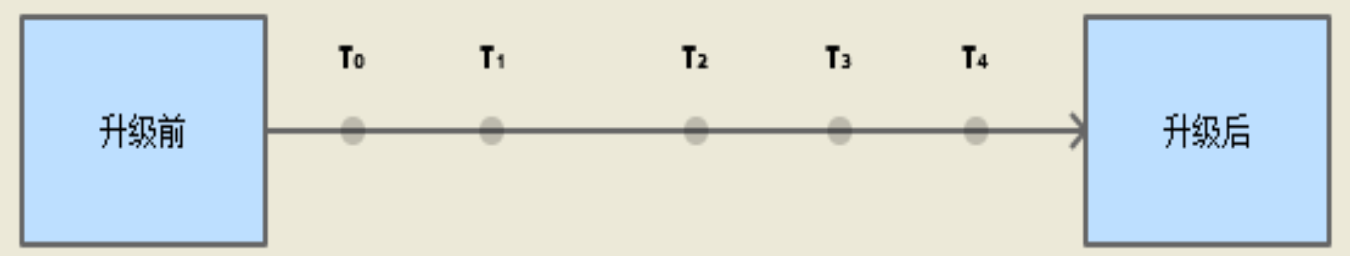
扩展性：扩展字段

- 原来3个字段，新增到5个字段，如何实现？（明显不能alter table）
- 3个库扩容到4个库，如何实现？
- 目标：平滑迁移，不停服务
- 扩展方法：倒库

（1）追日志法

（2）双写法【见后文】

1）记录写日志
2）倒库
3）倒库完毕
4）追日志
5）追日志完毕+数据校验
6）切库

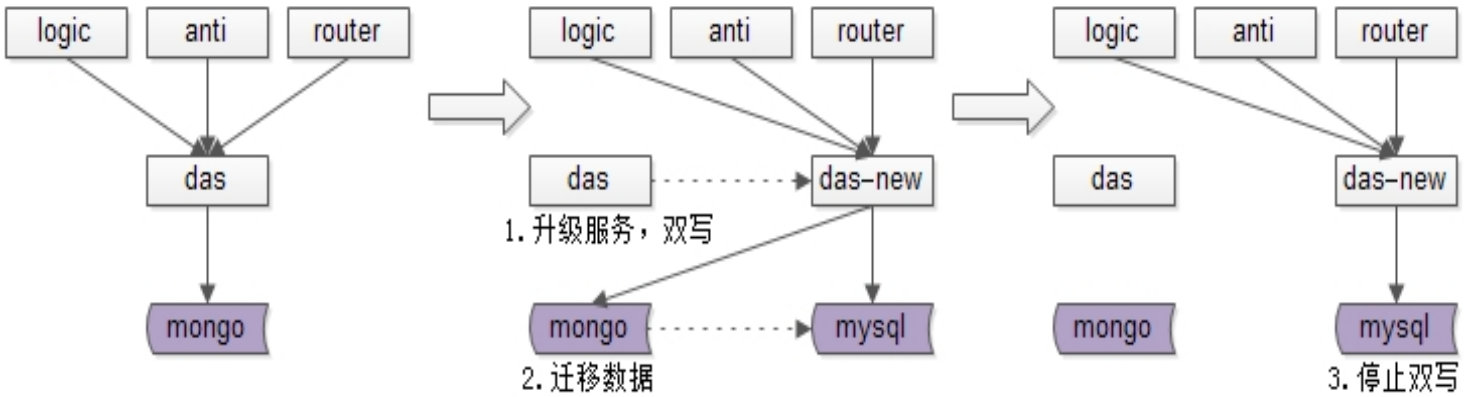


扩展性：扩展字段

- 原来3个字段，新增到5个字段，如何实现？（明显不能alter table）
- 3个库扩容到4个库，如何实现？
- 目标：平滑迁移，不停服务
- 扩展方法：倒库

（1）追日志法

（2）双写法



扩展性：水平拆分

如何拆？

四类典型场景

- 几乎涵盖互联网90%业务场景

（单key）用户库如何拆分： user(**uid**, XXOO)

（1对多）帖子库如何拆分： tiezi(**tid, uid**, XXOO)

（多对多）好友库如何拆分： friend(**uid, friend_uid**, XXOO)

（多key）订单库如何拆分：order(**oid, buyer_id, seller_id**, XXOO)

实战-用户库拆分？

- 用户库，10亿数据量

user(**uid**, uname, passwd, age, sex, create_time);

- 业务需求如下

（1）1%登录请求 => where uname=XXX and passwd=XXX

（2）99%查询请求 => where uid=XXX

- 问题？那uname的查询怎么办？【见后文】

实战-帖子库拆分？

- 帖子库，15亿数据量

tiezi(**tid**, **uid**, title, content, time);

- 业务需求如下

（1）查询帖子详情（90%请求）

SELECT * FROM tiezi WHERE tid=$tid

（2）查询用户所有发帖（10%请求）

SELECT * FROM tiezi WHERE uid=$uid

实战-好友库拆分？

- 好友库，1亿数据量

friend(**uid**, **friend_uid**, nick, memo, XXOO);

- 业务需求如下

（1）查询我的好友（50%请求） => 用于界面展示

SELECT friend_uid FROM friend WHERE uid=$my_uid

（2）查询加我为好友的用户（50%请求） => 用户反向通知

SELECT uid FROM friend WHERE friend_uid=$my_uid

实战-订单库如何拆分？

- 订单库，10亿数据量

order(**oid**, **buyer_id**, **seller_id**, order_info, XXOO);

- 业务需求如下

（1）查询订单信息（80%请求）

SELECT * FROM order WHERE oid=$oid

（2）查询我买的东东（19%请求）

SELECT * FROM order WHERE buyer_id=$my_uid

（3）查询我卖出的东东（1%请求）

SELECT * FROM order WHERE seller_id=$my_uid

拆分后带来什么问题？

SQL功能支持不了了！

（5）拆分后SQL怎么玩？

海量数据下SQL怎么玩

- 海量数据下，58同城不这么玩：

1）各种连接

2）子查询

3）触发器

4）用户自定义函数

5）“事务”都用的很少

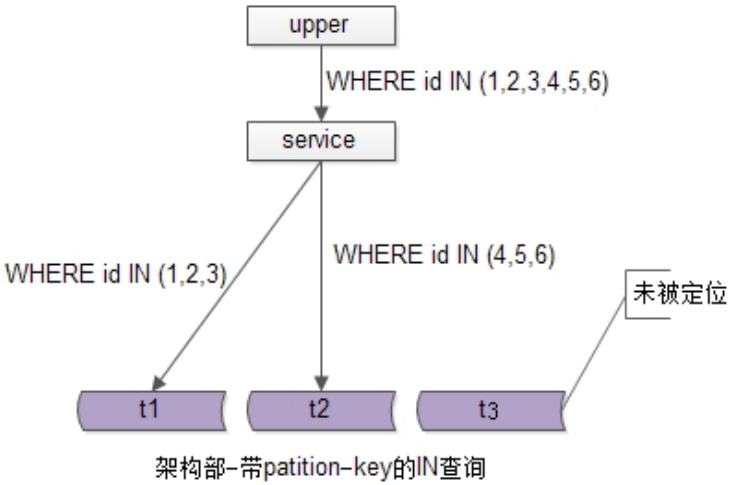
- 为什么？

实战-IN查询

- 需求：partition key 上的IN查询，WHERE uid IN(1,2,3,4,5,6)
- 解决方案：服务做MR

（1）直接分发

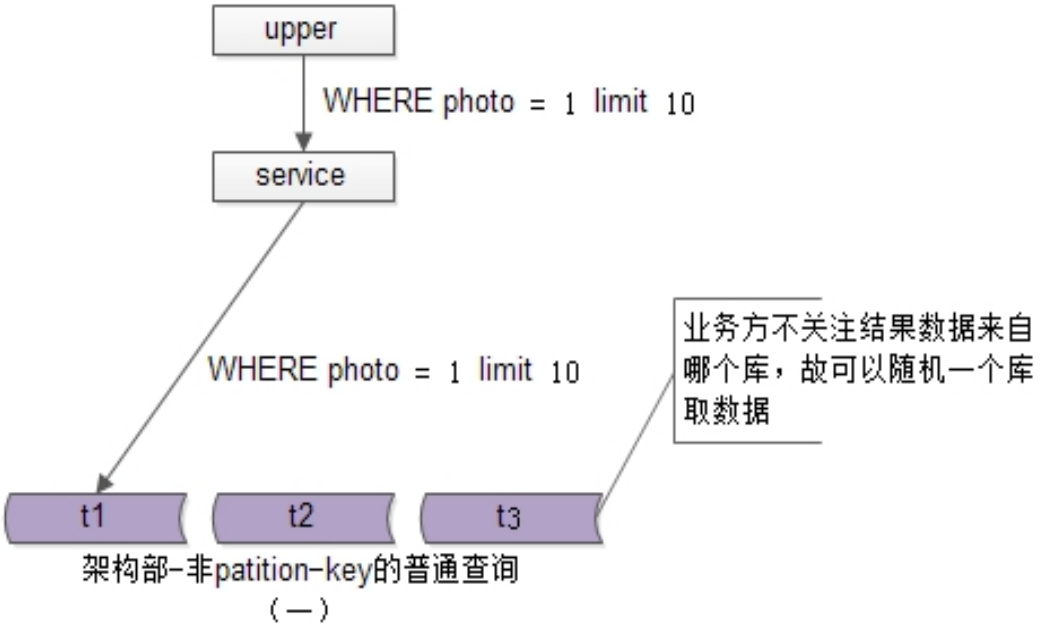
（2）拼装成不同SQL



架构部-带patition-key的IN查询

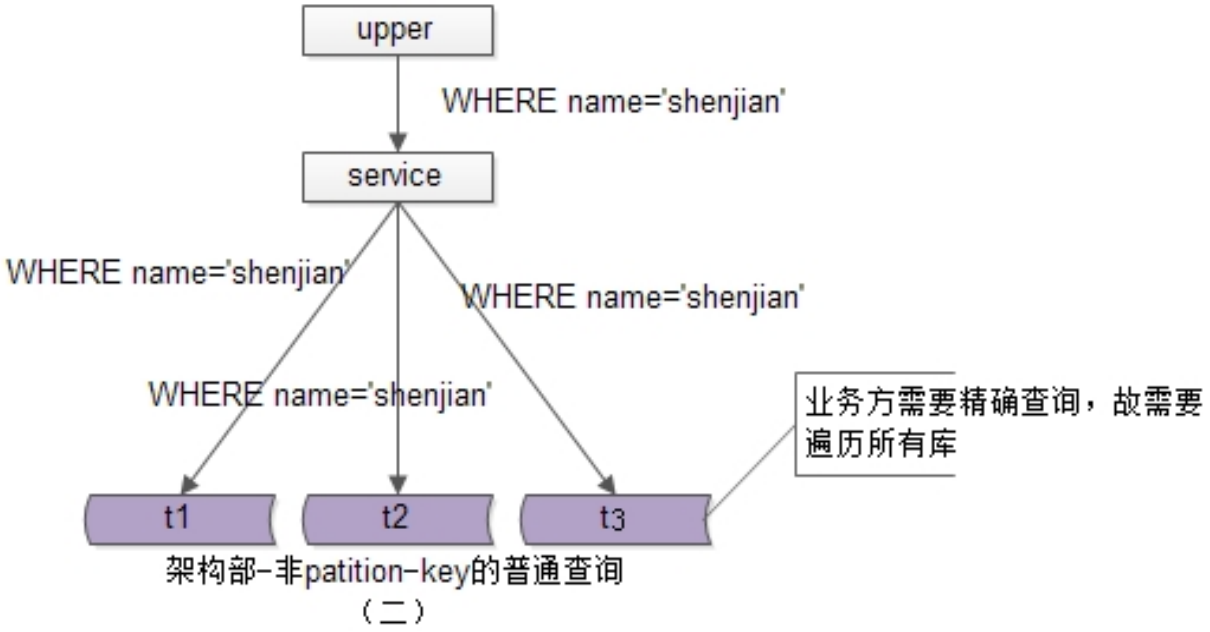
实战-非partition key查询

- 需求：头像查询
- 解决方案：只定位一个库



实战-非partition key查询

- 需求：登录查询
- 解决方案：服务做MR，一条数据返回则返回



实战-跨库分页

- 需求：ORDER BY xxx OFFSET xxx LIMIT xxx

（1）按时间排序；

（2）每页100条记录；

（3）取第100页的记录；

- 单机方案

ORDER BY time OFFSET 10000 LIMIT 100

- 分库后如何实现？

实战-跨库分页

- 分库后难点：如何全局排序？
- 传统方案：SQL改写 + 自己排序

（1）ORDER BY time OFFSET 0 LIMIT 10000+100

（2）对20200条记录进行排序

（3）返回第10000至10100条记录

实战-跨库分页

- 方案一：

（1）技术上，引入特殊id，作为查询条件（或者带入上一页的排序条件）

（2）业务上，尽量禁止跨页查询

- 单机情况

（1）第一页，直接查

（2）得到第一页的max(id)=123（一般是最后一条记录）

（3）第二页，带上id>123查询：**WHERE id>123 LIMIT 100**

=>

这样每次只要查100条，那分库情况呢？

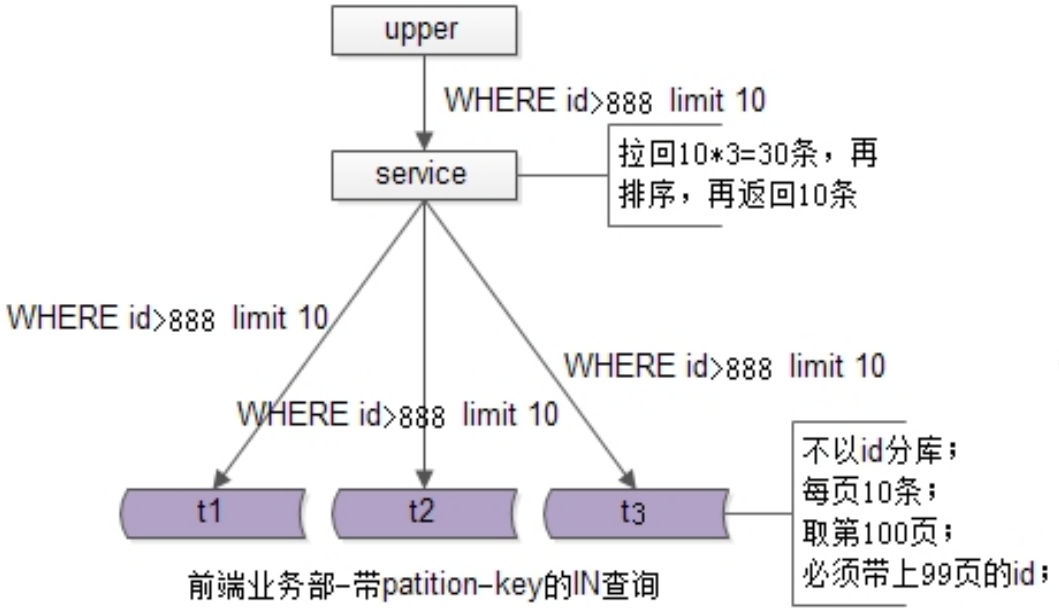
实战-跨库分页

- 分库情况（假设3个库）

（1）将WHERE id>xxx LIMIT 100分发

（2）将300条结果排序

（3）返回前100条



实战-跨库分页

- 方案二：

（1）业务上：禁止查询XX页之后的数据

（2）业务上：允许模糊返回 => 第100页数据的精确性真这么重要么？

三、总结

总结-数据库架构设计思路

- 基本概念：分片，复制，分组，路由规则（范围，哈希，路由服务）
- 数据库架构设计思路

1）可用性，解决思路是冗余（复制）

1.1）读可用性：多个从库

1.2）写可用性：双主模式，双主当主从用（58的玩法）

2）读性能，三种方式扩充读性能

2.1）增加索引：主从上的索引可以不一样

2.2）增加从库

2.3）增加缓存：服务+缓存+数据一套（58的玩法）

3）一致性

3.1）主从不一致：引入中间层，读写都走主库（58的玩法）

3.2）缓存不一致：双淘汰来解决缓存不一致问题

总结-数据库架构设计思路

- 数据库架构设计思路

4）扩展性

4.1）数据扩容：提升从库，double主库，秒级扩容

4.2）字段扩展：追日志法，双写法

4.3）水平切分

（单key）用户库如何拆分：, user(uid XXOO)

（1对多）帖子库如何拆分： tiezi(tid, uid, XXOO)

（多对多）好友库如何拆分： friend(uid, friend_uid, XXOO)

（多key）订单库如何拆分：order(oid, buyer_id, seller_id, XXOO)

总结-数据库架构设计思路

5）SQL玩法

不这么玩：联合查询，子查询，触发器，自定义函数，事务

这么玩：

a）IN查询：分发MR，拼装成不同SQL语句

b）非partition key查询：定位一个库，分发MR

c）夸库分页

c.1）修改sql语句，服务内排序

c.2）引入特殊id，减少返回数量

c.3）业务优化，允许模糊查询

Q&A&讨论

谢谢！

58同城“架构师之路”


ChinaUnix
ITPUB
THANKS