

开源数据库中间件MYCAT 产品介绍与企业实战

王金剑

个人简介

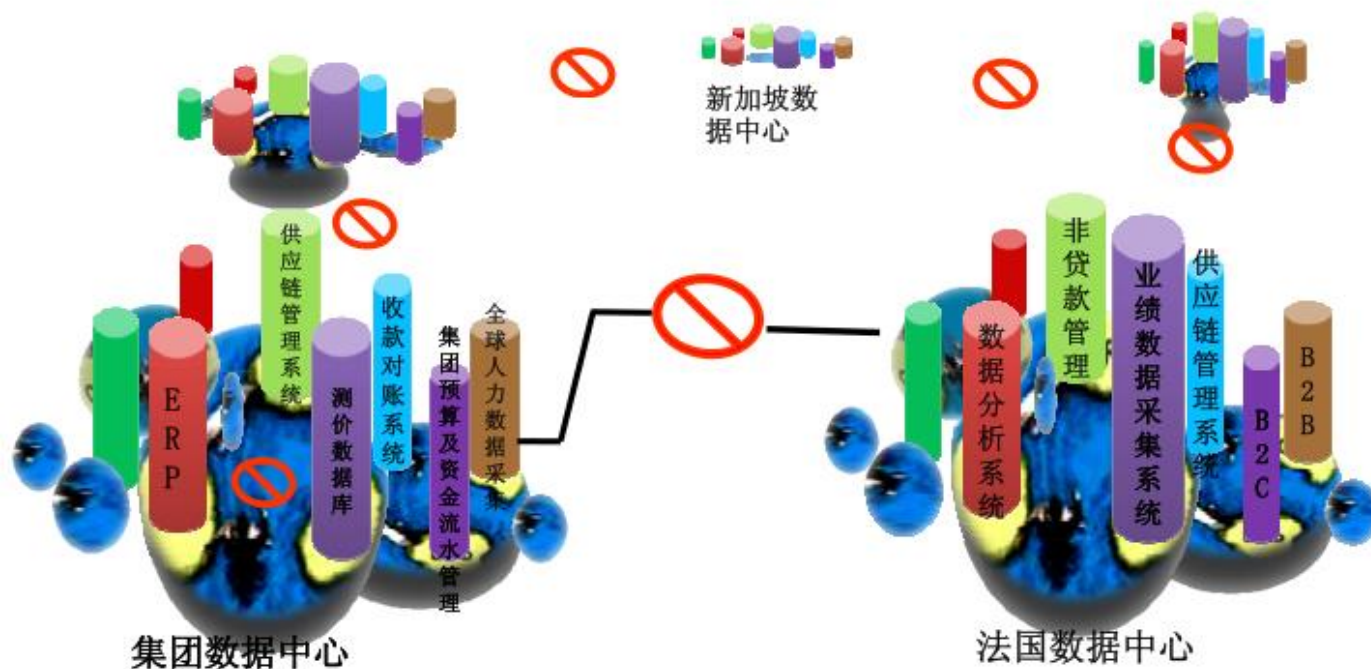
王金剑，CSDN认证专家，开源数据库中间件Mycat核心开发成员。从事软件工作10年，曾在金融和互联网行业企业担任高级软件工程师、项目经理、高级DBA工程师职位，有银行信用卡系统和电子商务系统基础架构设计与开发经验。现在天狮集团担任高级DBA工程师，负责公司电子商务网站数据库设计与优化、数据库架构规划及部署规范制定、核心应用的软件设计与开发等工作。最近代表MYCAT团队参加第四届全国创新创业大赛，并在天津赛区决赛中胜出。



问题与现状

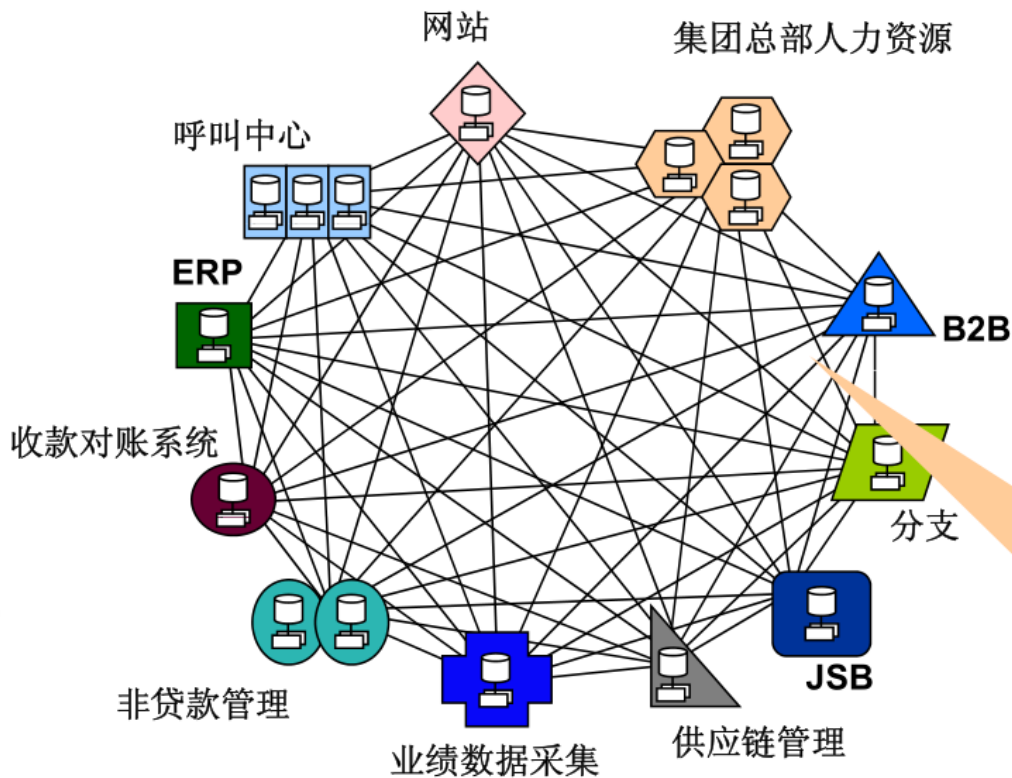
天狮业务快速扩张的同时，信息化建设不断快速发展,取得了很多的成就，但由于发展太快，发展的过程建设了很多的系统与数据库，形成了众多的信息孤岛。

往往为了连接这些信息孤岛，又产生了新的信息孤岛



问题与现状

天狮集团应用和数据众多，系统复杂，系统之间连接分散而众多，如图所示，这种点对点的连接方式造成系统维护复杂，扩展性差，新的业务需求和需求变化都难以及时响应。



- 系统之间直连，极端情况下 $n*(n-1)/2$ 条
- 服务的变动会影响该服务的所有使用者
- * 系统之间耦合紧密，牵一发而动全身，发展下去难以维护，不敢维护，无法维护

天狮集团信息化瓶颈主要表现在以下方面：



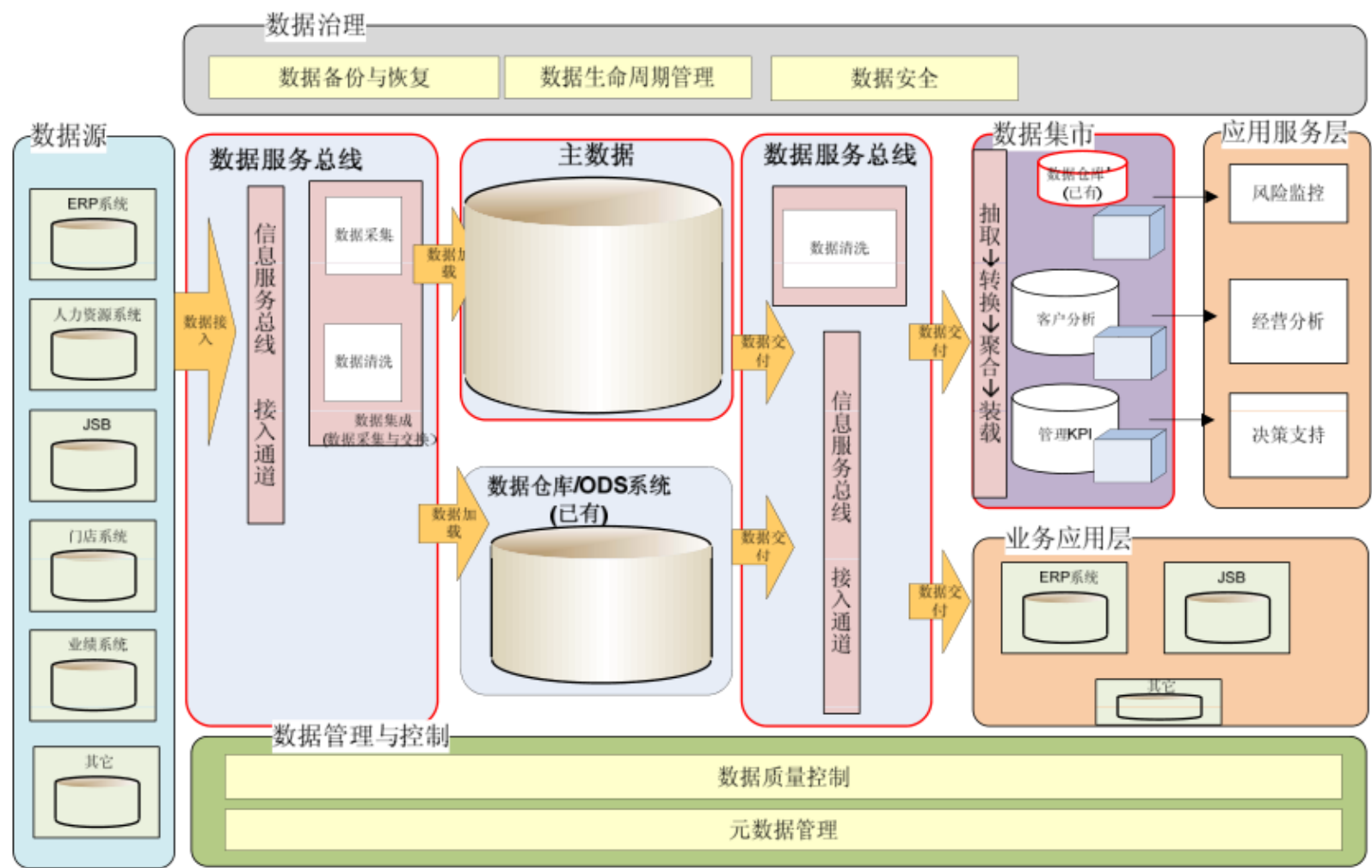
解决方案

针对上述问题，设计的解决方案包括建立总线、标准、架构



解决方案

通过建立的数据服务总线进行集成，做到统一标准，数据整合



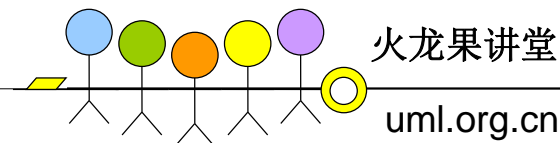
数据库中间件技术核心功能一

- 路由：基于数据路由功能，建立企业数据服务总线
 - ✓ 数据管理
 - ✓ 标准规范
 - ✓ 数据安全
 - ✓ 数据分析

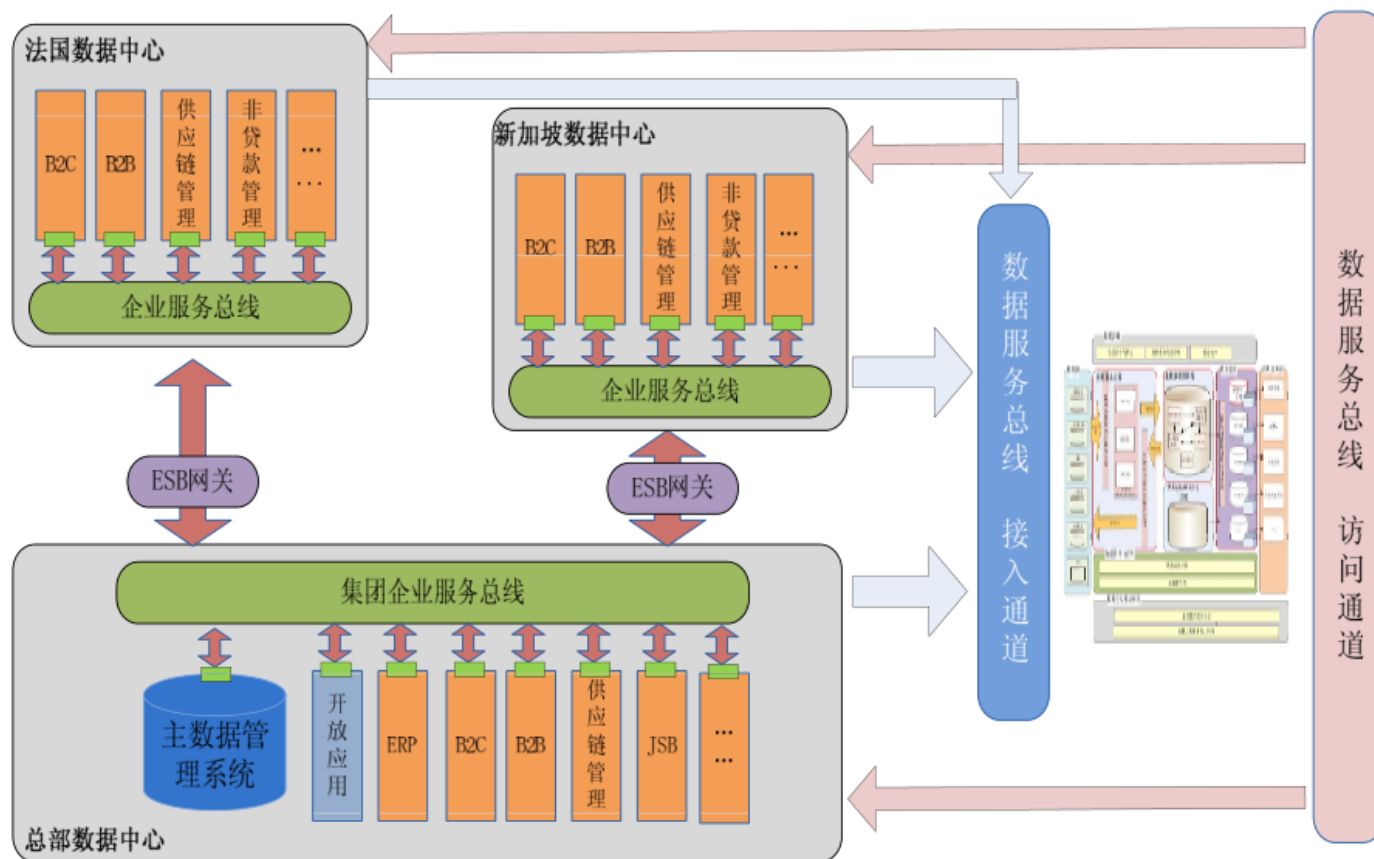
第一问题：为什么要用数据库中间件？

位于应用层与数据层之间的数据库中间件，为我们进行企业数据管理、确定标准规范，保障数据安全和进行数据分析创建了平台。这是我们引入该产品的根本原因。

数据库中间件核心功能介绍一

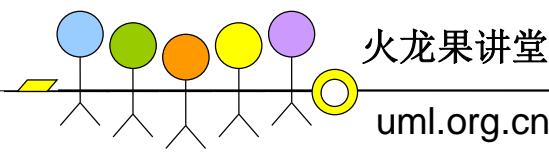


通过建立企业消息和数据服务总线进行应用集成和数据集成，为企业信息化打下了坚实的基础，下面是总体架构示意图



关于企业消息总线技术，请大家到火龙果官网收听我的另一个培训课程：
《基于ActiveMQ的消息总线逻辑与物理架构设计详解》

数据库中间件核心功能介绍一



什么是数据架构设计？

概述：

数据架构要对数据进行描述，识别企业数据源在哪，哪些数据是可信的，数据在不同类型数据结构（如操作数据、数据仓库，数据集市）中是如何流向的，详细说明关联的信息流转架构和分布。

目标：

- 建立涵盖主要业务流程和业务领域的企业级数据模型，提供完整而详细的数据定义，为企业业务运行及决策分析提供统一数据支撑。
- 明确数据在应用间的流转关系，为横向集成和纵向贯通提供支撑。
- 明确数据在业务及应用间的分布，为确定可信数据源提供支撑。

RACI

角色/RACI	负责/R	执行/A	询问/C	通知/I
业务架构师			√	√
应用架构师			√	√
数据架构师	√	√		√
数据责任人			√	√

输入：

- 业务能力
- 业务功能
- 业务流程
- 业务规则
- 业务数据标准
- 应用功能
- 应用集成
- 现有应用系统的数据模型
- 行业数据标准

输出：

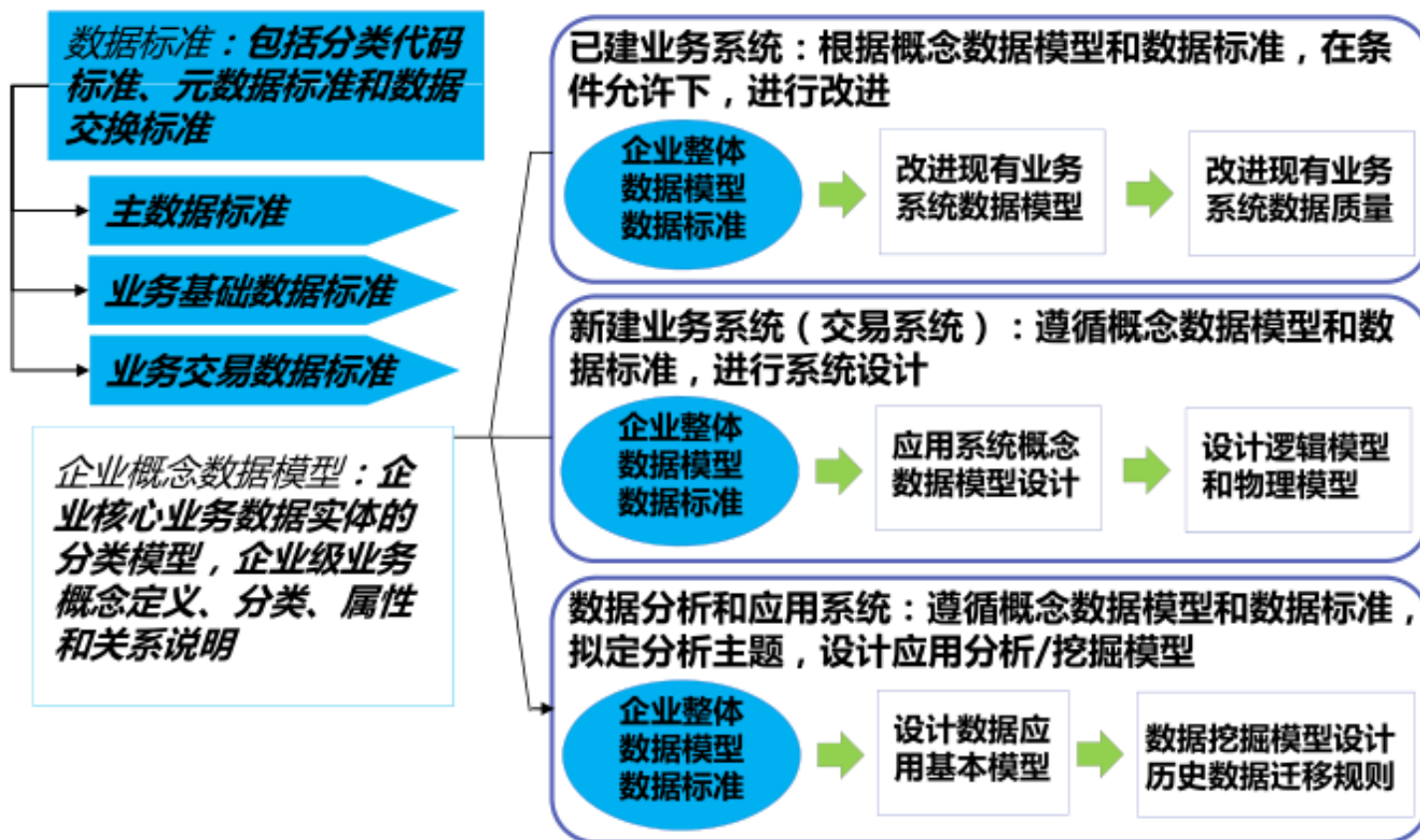
- 主题域模型
- 概念数据模型
- 逻辑数据模型
- 数据流转设计
- 数据分布设计

主要设计阶段



数据库中间件核心功能介绍一

通过数据库中间件产品不仅仅建立平台，更重要的的是规范了企业应用系统数据架构设计的流程、标准和规则，进而帮助我们建立了企业级的数据架构和数据模型。在规范标准指导下进行老系统改造以及新系统的建设，保障企业应用系统的常治久安。

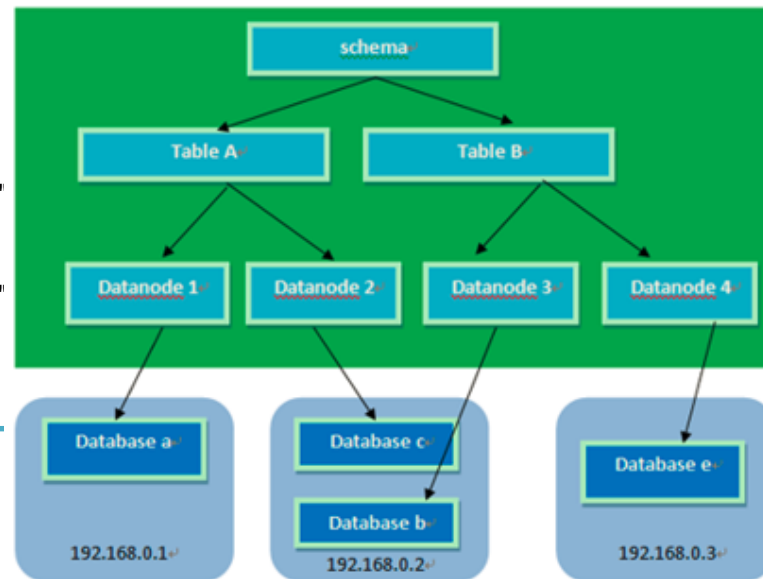


➤ Mycat 数据路由功能支持

通过配置文件，对底层数据库节点进行抽象，实现数据路由功能支持。

Schema.xml

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">
  <schema name="orderdb" checkSQLschema="true" sqlMaxLimit="100" >
    <table name="t_user" dataNode="dn1,dn2" rule="rule1"> </table>
  </schema>
  <dataNode name="dn1" dataHost="192.168.1.101_order_host1" database="order" />
  <dataNode name="dn2" dataHost="192.168.1.101_order_hos2" database="order" />
  <dataHost name="jdbchost" maxCon="2" minCon="1" balance="0"
    writeType="0" dbType="mysql" dbDriver="native">
    <heartbeat>select 1</heartbeat>
    <writeHost host="master" url="121.40.121.133:3306" user="root"
      password="zaq1xsw2cde3"></writeHost>
  </dataHost>
  <dataHost name="jdbchost2" maxCon="2" minCon="1" balance="0"
    writeType="0" dbType="mysql" dbDriver="native" >
    <heartbeat>select 1</heartbeat>
    <writeHost host="master" url="121.40.121.133:3306" user="root"
      password="zaq1xsw2cde3"></writeHost>
  </dataHost>
</mycat:schema>
```

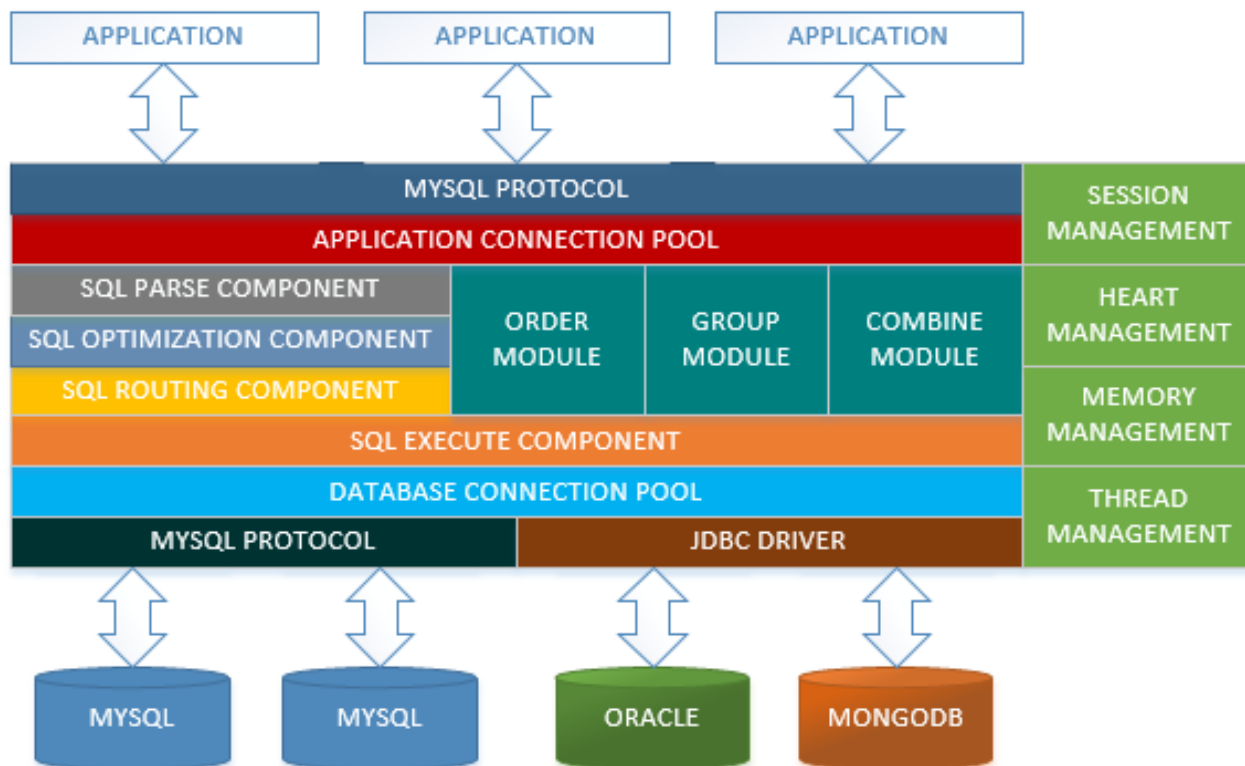


➤ Mycat 数据路由功能支持

我们可以把上层看作是对下层的抽象，例如操作系统是对各类计算机硬件的抽象。那么我们什么时候需要抽象？假如只有一种硬件的时候，我们需要开发一个操作系统吗？再比如一个项目需要很多人完成时，就应该有一个管理者，发挥管理、沟通、协调、约束等作用，类似总线的作用，而这个管理者对于他的上层来说就是对项目组的抽象。同样的，当应用系统中只有一台数据库服务器的时候并不需要中间件，而如果应用面对很多数据库的时候，这个时候就需要对数据库层做一个抽象，来管理这些分散的数据，最上面的应用只需要面对一个数据库层的抽象就可以了。这时下层无法解决的问题可以通过对下层的抽象在上层解决。

➤ Mycat 数据路由功能实现原理

Mycat 拦截到用户发送来的SQL语句，首先对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路由到后端数据库分片节点，并将返回的结果做适当的处理，最终再返回给用户。



➤ Mycat 数据路由功能实现

Mycat支持两种SQL解析器：

FoundationDB SQL Parser

该解析器来源于Apache Derby parser，FoundationDB SQL Parser解析器存在的问题：

- ✓ 解析器源码门槛高
- ✓ 支持的语句少
- ✓ 解析性能差

Druid SQL Parser

该解析器为阿里开发，Druid主要特点是连接池的监控功能，其SQL解析性能也非常出色，支持的语法更多，编码更容易。Druid 解析路由有两种方式：Visitor和Statement，其提供的接口可以比较方便的提取表名、条件表达式、字段列表、值列表等信息。而且可以很容易的通过AST（SQL Statement）语法树改写SQL，这对Mycat注解的实现提供了帮助。

➤ Mycat 数据路由功能实现

Visitor 方式说明

```
String sql = "select * from tableName";  
MySQLStatementParser parser = new MySQLStatementParser(sql);  
SQLStatement statement = parser.parseStatement();  
MycatSchemaStatVisitor visitor = new MycatSchemaStatVisitor();  
stmt.accept(visitor);
```

经过上面的步骤后，你可以很方便的从visitor中获取表名、条件、表别名map、字段列表、值类表等信息。用这些信息就可以做路由计算了。

Statement 方式说明

```
String sql = "select * from tableName" ;  
MySQLStatementParser parser = new MySQLStatementParser(sql);  
SQLStatement statement = parser.parseStatement();  
SQLSelectStatement selectStmt = (SQLSelectStatement) statement;
```

经过上面的步骤后，可以从selectStmt里面得到想要的信息。

如果sql = "delete from tableName" ;

转型为MySQLDeleteStatement

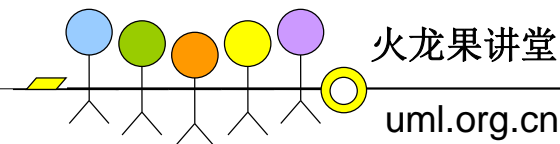
```
MySQLDeleteStatement deleteStmt = (MySQLDeleteStatement) statement;
```

数据库中间件技术核心功能二

- 分片：基于数据分片功能，建立分布式数据库架构
 - ✓ 性能优化
 - ✓ 水平扩展

数据分片是不得已而为之！

数据库中间件核心功能介绍二



系统优化中的一些错误观点：

“系统性能出现问题进行优化，一定要深入了解数据库内部参数、等待事件、Latch、缓冲池、trace文件等底层细节。”

往往出自数据库高手包括DBA，一些数据库爱好者，这部分人以了解数据库底层实现细节而感到非常骄傲。但是从优化角度讲数据库的等待事件、Latch等指标高，只是问题的表象，解决问题的关键往往是在应用层进行优化。

“只要系统参数调整了，性能就能提高。系统优化应该调整哪些参数？”

调整系统参数是非常重要的，但不一定能解决性能问题，否则就不会有去IOE了，问题可能性最大的还是应用设计和开发问题。

“系统性能问题从架构上解决，与应用开发关系不大。”

系统性能与各个层面都有关，其中应用开发是非常重要的一环。

看到很多企业使用数据库中间件做数据分片，说明我们的互联网应用发展很快令人高兴。但另一个方面，也令人担忧，我们的数据库压力真的已经到了必须切分不可的程度了吗？有没有更合适的优化方法。



王金剑
MySQL学习路径

MySQL高级开发



MySQL管理与维护



MySQL优化方法



MySQL深入研究



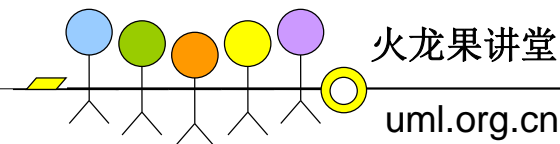
数据库中间件核心功能介绍二

《孙子兵法·谋攻篇》原文：
故上兵伐谋，
其次伐交，
其次伐兵，
其下攻城；
攻城之法为不得已。



用兵的最高境界是使用谋略胜敌，
“不战而屈人之兵”

数据库中间件核心功能介绍二



应用系统SQL优化举例：哈萨克B2B系统查询数据缓慢，直接影响代理商业绩上报，之前开发人员尝试通过减少历史数据的方法，提高查询执行，效果不明显。数据切分不是提高系统性能的银弹。

以下查询语句执行缓慢，执行一次大约需要1分钟，直接影响代理商业绩上报

/ Formatted on 2015/12/26 15:09:55 (QP5 v5.163.1008.3004) */*

```
SELECT sheetid,  
       bodgid,  
       bodunit,  
       bodname,  
       bodsl,  
       bodbatchno,  
       bodscrq,  
       bodbzbzrq  
FROM view_b2bsaleckex  
WHERE sheetid = '1512631KZ9556031282'  
ORDER BY bodgid
```

原SQL语句已使用绑定变量，并以在sheetid创建索引

查询缓慢的原因：使用了多层嵌套视图，其中视图VIEW_B2BSALECK 中的函数fgetb2blsckdh(sheetid)，致使sheetid索引失效；

修改建议：重写SQL语句，不再使用嵌套视图及该函数，或在该函数添加函数索引，因添加索引可能影响系统运行，因此建议晚上添加。创建函数索引，语句如下：

/ Formatted on 2015/12/26 16:01:55 (QP5 v5.163.1008.3004) */*

```
CREATE INDEX idx_boutdetail_fgetb2blsckdh  
ON boutdetail (fgetb2blsckdh(sheetid));
```

没有进行应用及SQL语句优化，直接创建索引，查询执行时间由1分钟降到不足1秒

➤ 为什么进行数据分片？

主要解决高并发下的底层数据库应用、网络以及各种物理设备瓶颈，而不是为了减少逻辑数据量。

➤ 什么是数据分片？

根据指定的分片规则，将同一个数据库中的数据，分布存放到多个数据库节点中，用以分散高并发下的底层数据库服务压力。

➤ 分片方式

- ✓ 垂直分片
- ✓ 水平分片

➤ 关于分区表与分片技术

正如培训PPT中数据库优化一节，分片技术位于“自顶向下优化法”的最后一层，是不得已而为之。在培训中特别提到了分片的目的不是针对数据量的优化，相反，表分区正好是对大数据表的一种优化技术。分区表是指按一定规则将大表进行拆分（按行水平拆分或按字段垂直拆分），拆分后的表依然在原数据库实例下进行管理，在物理上大表被拆分成小表，但在逻辑上还是一张表，应用无需任何改造。分区表优化属于数据库对象优化，位于自顶向下优化法的第一层。表分区后对于必须进行全表扫描的查询，性能必然提升。对于索引查询，使用分区索引后，由于索引树的高度低，查询速度也得到提高。同时分区技术提高了我们对大批量数据的管理效率，如数据清理，数据迁移，数据备份和恢复等。

使用一致性hash切分数据，减少了数据动荡，但是还是存在数据迁移问题？
这么说吧，如果你的表需要进行频繁的数据迁移、清理工作，即使你的数据量不是特别大，也会建议你做分区表，而如果仅仅从查询性能出发，对于亿级的表可能也没有分区的必要。关键还是看你的需求。

➤ 什么是垂直分片？

垂直分片是根据企业不同业务，或不同数据类型，将表进行分类，创建到不同的数据库节点上。

➤ 垂直分片的优点：

- ✓ 分片规则简单容易操作；
- ✓ 分片使业务更加清晰；
- ✓ 分片系统维护简单；

➤ 垂直分片的缺点：

- ✓ 依然存在单库性能瓶颈，扩展困难
- ✓ 事务处理复杂，提高了系统复杂度

➤ 什么是水平分片？

我们可以将数据的水平分片理解为是按照数据行的分片，就是将表中的某些行分片到一个数据库，而另外的某些行分片到其他的数据库中，其中选择合适的分片规则至关重要，因为它决定了数据聚合的难度。

有几种典型的分片规则包括：

- ✓ 按照用户主键ID求模，将数据分散到不同的数据库；
- ✓ 按照日期，将不同月、日的数据分散到不同的数据库；
- ✓ 按照某个特定的字段求摸，或者根据特定范围段分散到不同的数据库中。

水平分片的缺点：

- ✓ 技术比较复杂，维护难度大
- ✓ 事务处理复杂，系统复杂度高

➤ 分布式系统的优点

多节点，提高了系统整体的负载能力；可以使用多个处理器，同步查询，提高查询性能；可以有更多节点复制数据，增强了失败情况下的恢复能力，提高了系统整体的可用性。

但是，分布式处理增加了系统各个方面的复杂度，关键问题是解决如何**通信问题**。

➤ 分布式系统的先天不足—CAP理论

2000年，Eric Brewer教授在ACM分布式计算年会上指出了著名的CAP理论：分布式系统不可能同时满足一致性(Consistency)，可用性(Availability)和分区容错性(Tolerance of network Partition) 这三个需求。

- ✓ **一致性 Consistency**：系统在执行过某项操作后仍然处于一致的状态。在分布式系统中，更新操作执行成功后所有的用户都应该读到最新的值，这样的系统被认为是具有强一致性的。等同于所有节点访问同一份最新的数据副本。这个和数据库ACID的一致性类似，但这里关注的是所有数据节点上的数据一致性和正确性，而数据库的ACID关注的是在一个事务内，对数据的一些约束。
- ✓ **可用性 Availability**：每一个操作总是能够在一定的时间内返回结果，这里需要注意的是“一定时间内”和“返回结果”。一定时间指的是，在可以容忍的范围内返回结果，结果可以是成功或者失败。关注的是在某个结点的数据是否可用，可以认为某一个节点的系统是否可用，通信故障除外。
- ✓ **分区容错性 Partition Tolerance**：是否可以对数据进行分区。存在网络分区的情况下，仍然可以接受请求。

➤ 分布式事务与两阶段提交

为什么不能同时保证这个三点，主要是因为一旦分片，就必须在节点之间进行通信，涉及到通信，就无法确保在有限的时间内完成指定的功能，在通信完成的这一段时间内，数据就是不一致的。**要保证一致性，就必须在通信完成这一段时间内保护数据，使得任何访问这些数据的操作不可用。如果想保证一致性和可用性，那么数据就不能够分区。因此分布式造成的通信问题是CAP无法同时满足的根本原因，解决分布式事务的问题就是解决如何通信的问题。**

目前一般采用两段提交策略（Two-phase Commit）来实现，但是这是一个非常耗时的过程，会严重影响系统效率，在实践中我们应该尽量避免使用它。两段提交策略包含两种角色，事务管理器（DM, Transaction Manager）和资源管理器（RM, Resource Manager），执行过程如下：



明显问题是超时问题，比如当一个RM出问题了，那么整个事务只能处于等待状态。有可能产生连锁反应，导致整个系统都很慢，最终不可用。

➤ XA事务与最终一致性

XA事务就是基于两阶段提交实现的。但是XA事务的控制是在生产者、中间件、消费者三者之间的Session会话层实现的，因此这就要求业务操作的资源也要支持XA协议，这是一个比较大的限制。

如果有多个资源之间需要协调，而且都支持XA事务，相对比较方便。在JMS的API中，以XA开头的接口，都是支持XA协议的接口（比如XAConnection、XASession）。

XA事务的明显问题就是两段提交的超时timeout问题，当一个RM出问题了，那么整个事务只能处于等待状态。有可能产生连锁反应，导致整个系统都很慢，最终不可用，这也就是在CAP三者之间选择了C和P，放弃了A。互联网应用，牺牲了可用性，相当于间接的影响了用户体验。

避免使用XA事务的方法通常是最终一致性。可以使用先进先出的队列结构，如消息队列产品，从流程上保证最终一致性。

MyCAT 1.4 初步实现了不跨分片的XA事务。在手动事务提交模式下，执行“set xa=on”开启XA事务支持：

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> set xa=on;
Query OK, 0 rows affected (0.00 sec)
mysql> update travelrecord set days=3 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0
mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```


➤ 弱一致性、CAP、BASE与NoSQL

弱一致性就是不完全一致或者完全不一致。自然界没有两片完全一样的叶子，分布式系统本就是明知不可为而为之，CAP理论其实是证明了分布式系统先天不足。

如同人类发明了不合理的克隆技术，逻辑世界通过复制技术可以得到两份完全一致的数据，但是复制是需要时间的，类似《机器猫》里的时空穿梭门，一致性理论有一个“一致性窗口”的概念（即：时间窗口）。不过穿梭门是可以穿越到过去的，而一致性窗口只能穿越到未来，方法很简单，就是“等会儿”。

最终一致性就是等会儿就一致了，早晚会一致的。使用最终一致性的关键就是想方设法让用户“等会儿”。办法是让用户知道（对用户透明），这个方法有个学名叫“用户感知到的一致性”，意思就是让用户自己知道数据已经不一致了，你再忍会儿。

好像是缺什么就想补什么，一些原本就不可靠的分布式系统就总想证明自己的可靠性，发现CAP理论以后如获至宝，原来我们是符合“不可靠理论”的，分布式系统本来就不可靠。所以系统你就认了吧（之前是忍了吧）。

虽然我们NoSQL不符合ACID原则，但是我们符合“NO ACID原则”啊。我们换个词，就叫“BASE原则（基本可用性、软状态与最终一致性）”。BASE的含义就是指“NoSQL数据库设计可以通过牺牲一定的数据一致性和容错性来换取高性能的保持甚至提高”。其实所有的分布式系统都是牺牲C（一致性 Consistency）来换取P（分区容错性 Partition Tolerance），而不是牺牲A（可用性 Availability）。可用性是所有分布式系统共同追求的特性。

➤ Mycat 数据分片功能支持

Mycat 根据在配置文件中指定的路由规则，实现分片数据的路由功能。

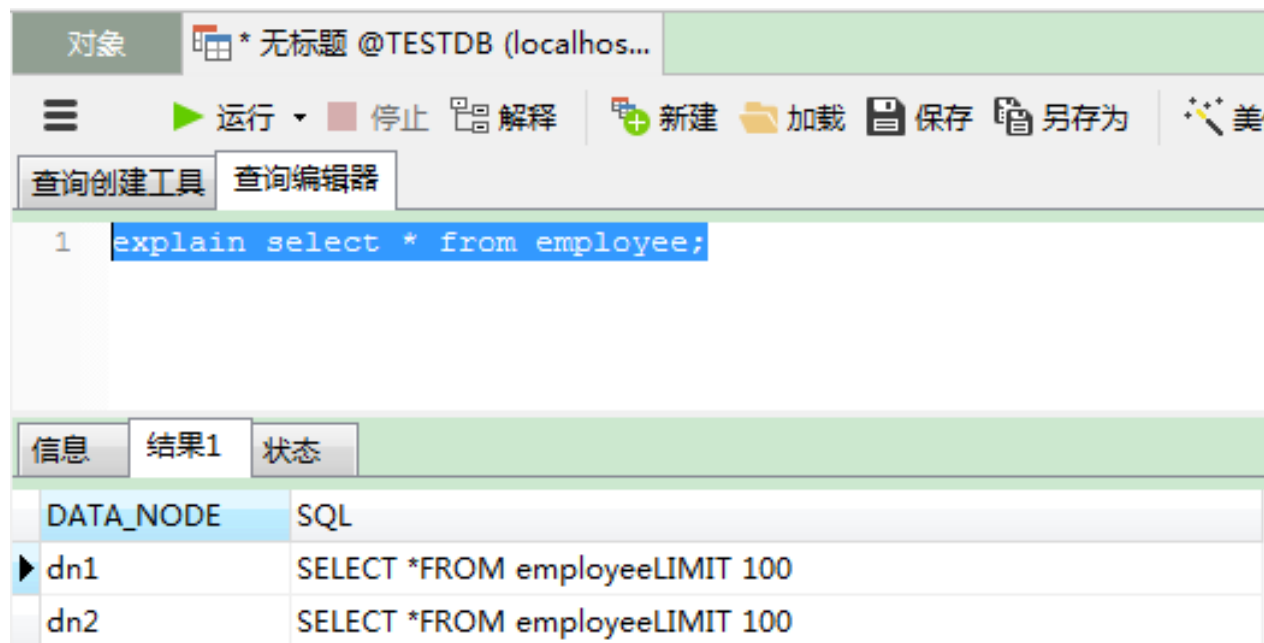
Rule.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mycat:rule SYSTEM "rule.dtd">
<mycat:rule xmlns:mycat="http://io.mycat/">
  <tableRule name="rule1">
    <rule>
      <columns>user_id</columns>
      <algorithm>func1</algorithm>
    </rule>
  </tableRule>
  <function name="func1"
    class="io.mycat.route.function.PartitionByLong">
    <property name="partitionCount">1</property>
    <property name="partitionLength">1024</property>
  </function>
</mycat:rule>
```

➤ Mycat 垂直分片

- ✓ 普通表：根据 “dataNode” 属性，将SQL语句路由到对应分片执行

```
<table name="employee" primaryKey="ID" dataNode="dn1,dn2"  
rule="sharding-by-intfile" />
```

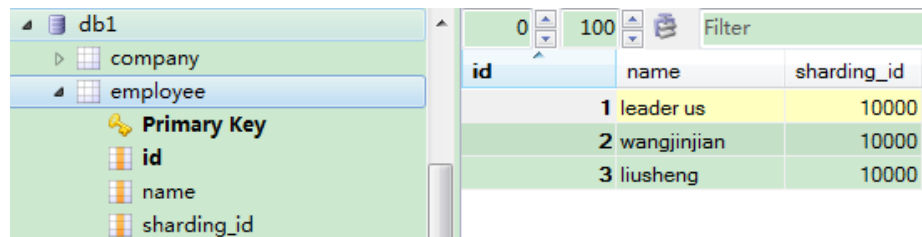


The screenshot shows a database management tool interface. At the top, there's a tab labeled "对象" and a dropdown menu showing "* 无标题 @TESTDB (localhos...". Below this is a toolbar with buttons for "运行", "停止", "解释", "新建", "加载", "保存", and "另存为". There are also tabs for "查询创建工具" and "查询编辑器". The main area displays a SQL query: "1 explain select * from employee;". Below the query, there's a table with two columns: "DATA_NODE" and "SQL". The table has two rows: "dn1" and "dn2".

DATA_NODE	SQL
dn1	SELECT *FROM employeeLIMIT 100
dn2	SELECT *FROM employeeLIMIT 100

➤ Mycat 垂直分片

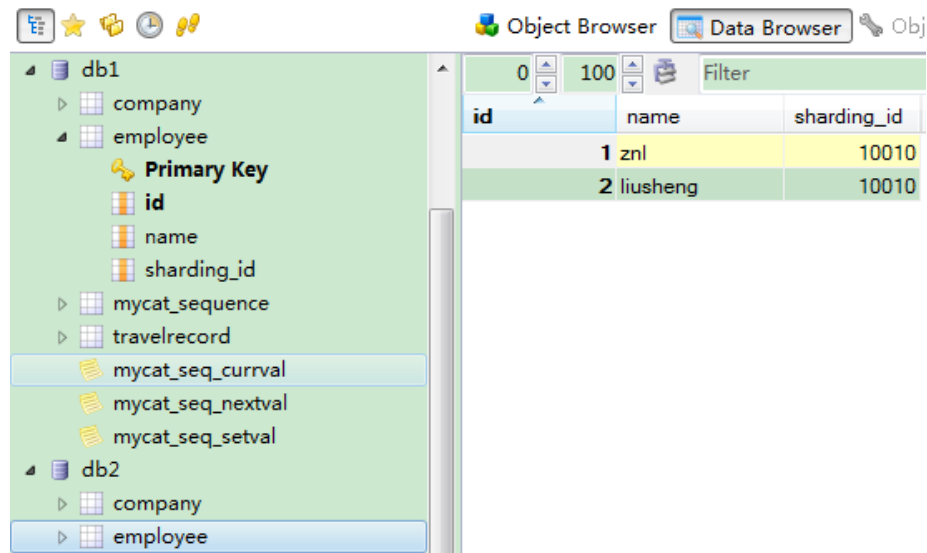
✓ 普通表：聚合查询结果：



db1

- company
- employee
 - Primary Key
 - id
 - name
 - sharding_id

id	name	sharding_id
1	leader us	10000
2	wangjinjian	10000
3	liusheng	10000



Object Browser Data Browser Obj

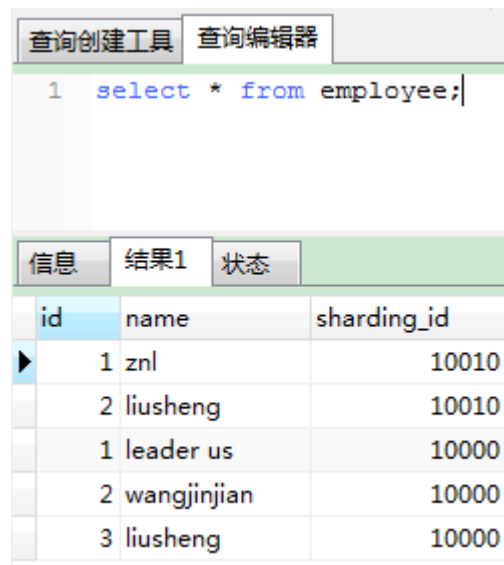
db1

- company
- employee
 - Primary Key
 - id
 - name
 - sharding_id
- mycat_sequence
- travelrecord
- mycat_seq_currval
- mycat_seq_nextval
- mycat_seq_setval

db2

- company
- employee

id	name	sharding_id
1	znl	10010
2	liusheng	10010



查询创建工具 查询编辑器

```
1 select * from employee;
```

信息 结果1 状态

id	name	sharding_id
1	znl	10010
2	liusheng	10010
1	leader us	10000
2	wangjinjian	10000
3	liusheng	10000

➤ Mycat 垂直分片

- ✓ 全局表：在每个分片节点，同步保存相同的一份数据；

```
<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />
```

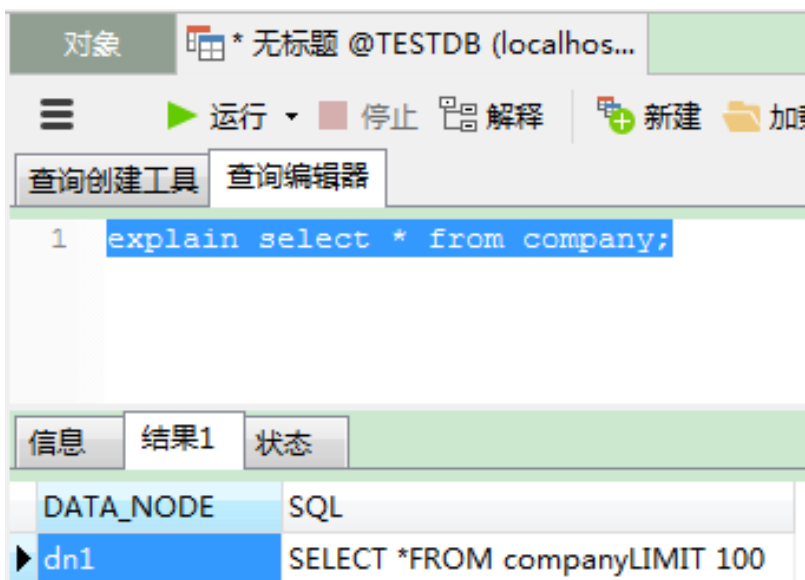


The screenshot shows a database management tool interface. The top bar includes a title bar with '对象' and a tab for '* 无标题 @TESTDB (localhos...'. Below the title bar is a menu bar with icons for '运行' (Run), '停止' (Stop), '解释' (Explain), '新建' (New), '加载' (Load), '保存' (Save), '另存为' (Save As), and '美化' (Beautify). The main area is divided into two tabs: '查询创建工具' (Query Creation Tool) and '查询编辑器' (Query Editor). The '查询编辑器' tab is active, showing a SQL query: `1 explain insert into company(id,name) values(1,'hp');`. Below the query editor is a table with three columns: '信息' (Information), '结果1' (Result 1), and '状态' (Status). The table contains three rows of data, each representing a data node (dn1, dn2, dn3) and the SQL statement executed on that node.

信息	结果1	状态
DATA_NODE	SQL	
dn1	insert into company(id,name) values(1,'hp')	
dn2	insert into company(id,name) values(1,'hp')	
dn3	insert into company(id,name) values(1,'hp')	

➤ Mycat 垂直分片

- ✓ 全局表：在任意节点中查询，获取结果，**缺少全局表的数据一致性检查**；



对象 * 无标题 @TESTDB (localhos...)

运行 停止 解释 新建 加时

查询创建工具 查询编辑器

```
1 explain select * from company;
```

信息	结果1	状态
DATA_NODE	SQL	
dn1	SELECT *FROM companyLIMIT 100	



对象 * 无标题 @TESTDB (localhos...)

运行 停止 解释 新建 加时

查询创建工具 查询编辑器

```
1 select * from company;
```

信息	结果1	状态
id	name	
1	hp	

➤ Mycat 水平分片规则

选择合适的水平分片规则非常重要，因为它决定了数据聚合的难度，原则就是尽量避免跨分片节点的数据聚合处理。

MYCAT 常用的分片规则如下，另外还有一些其他分片方式这里不全部列举：

- ✓ 分片枚举：sharding-by-intfile
- ✓ 主键范围：auto-sharding-long
- ✓ 一致性hash：sharding-by-murmur
- ✓ 字符串hash解析：sharding-by-stringhash
- ✓ 按日期（天）分片：sharding-by-date
- ✓ 按单月小时拆分：sharding-by-hour
- ✓ 自然月分片：sharding-by-month

➤ Mycat 水平分片配置举例—分片枚举

(1) 应用规则 : /src/main/resources/schema.xml

```
<table name="employee" primaryKey="ID" dataNode="dn1,dn2" rule="sharding-by-intfile" />
```

(2) 定义规则 : /src/main/resources/rule.xml

```
<tableRule name="sharding-by-intfile">
```

```
  <rule>
```

```
    <columns>sharding_id</columns>
```

```
    <algorithm>hash-int</algorithm>
```

```
  </rule>
```

```
</tableRule>
```

```
<function name="hash-int"
```

```
  class="org.opencloudb.route.function.PartitionByFileMap">
```

```
    <property name="mapFile">partition-hash-int.txt</property>
```

```
</function>
```

(3) 规则文件 : /src/main/resources/partition-hash-int.txt

```
10000=0
```

```
10010=1
```

```
public class Print {  
    public static void main(String[] args) {  
        System.out.println("Max:" + Integer.MAX_VALUE);  
        System.out.println("Min:" + Integer.MIN_VALUE);  
    }  
}
```

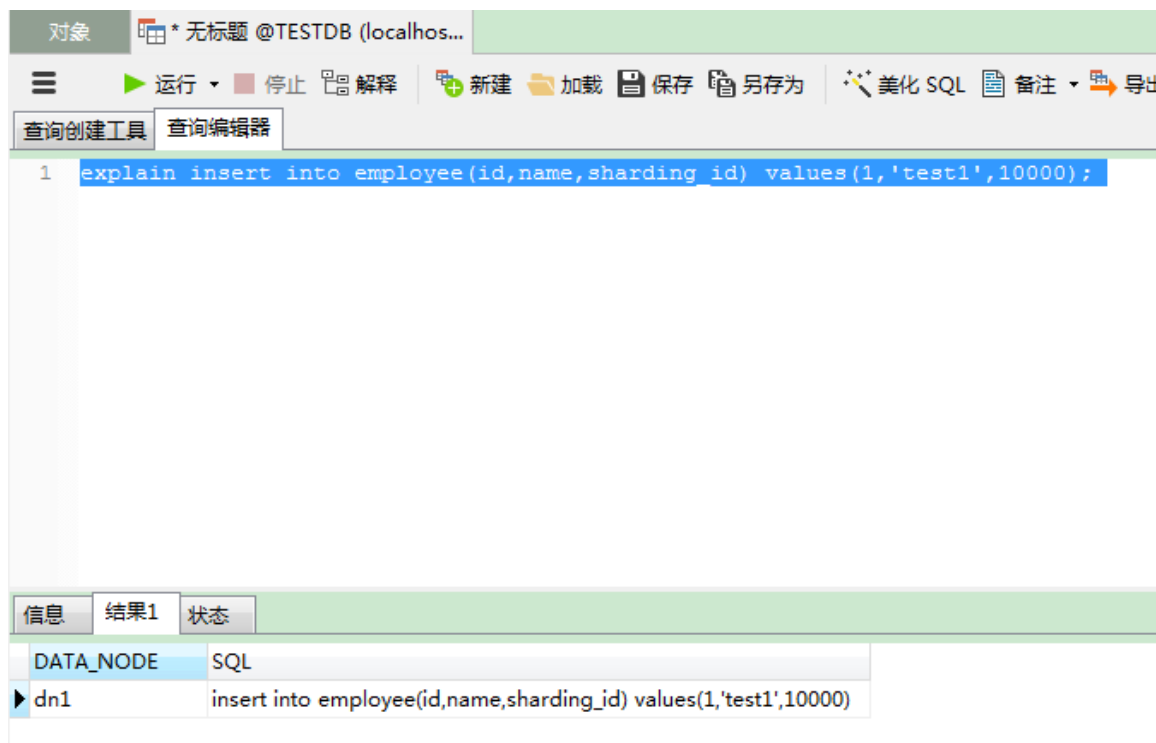
Max:2147483647 Min:-2147483648

整型最大值 : 2的31次方 - 1 整型最小值 : 2的负的31次方

➤ Mycat 水平分片配置举例—分片枚举

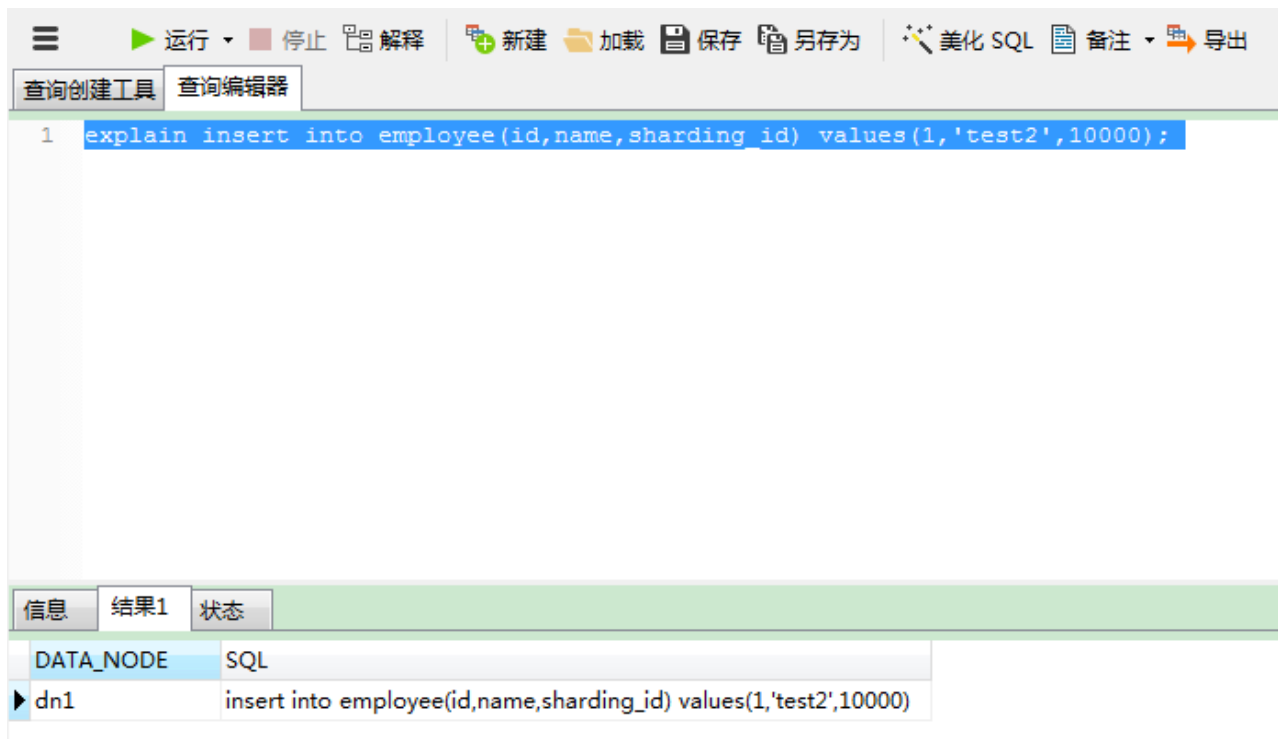
插入两条sharding_id=10000的数据，数据被插入到dn1节点

1) 语句1：explain insert into employee(id,name,sharding_id) values(1,'test1',10000);



➤ Mycat 水平分片配置举例—分片枚举

2) 语句2 : explain insert into employee(id,name,sharding_id)
values(1,'test2',10000);



The screenshot shows a SQL client interface with a menu bar at the top containing options like 运行 (Run), 停止 (Stop), 解释 (Explain), 新建 (New), 加载 (Load), 保存 (Save), 另存为 (Save As), 美化 SQL (Format SQL), 备注 (Remarks), and 导出 (Export). Below the menu bar are two tabs: 查询创建工具 (Query Creation Tool) and 查询编辑器 (Query Editor). The query editor contains the following SQL statement:

```
1 explain insert into employee(id,name,sharding_id) values(1,'test2',10000);
```

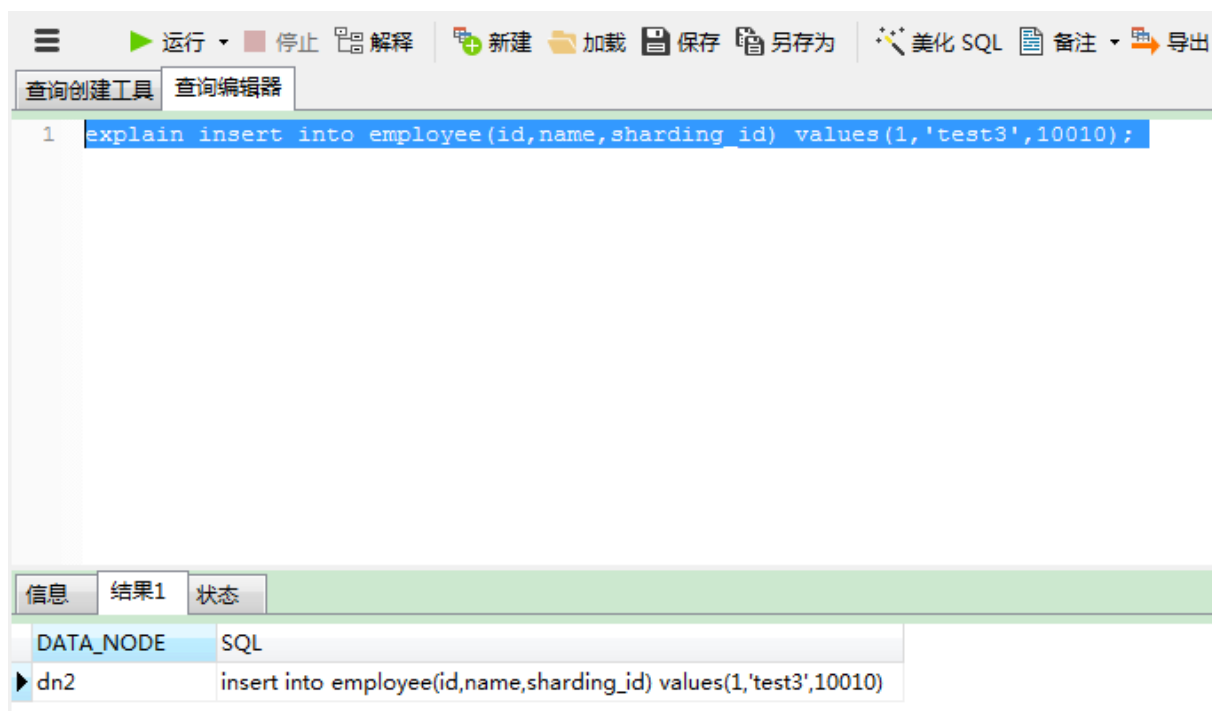
At the bottom of the interface, there is a tabbed view with three tabs: 信息 (Information), 结果1 (Result 1), and 状态 (Status). The 结果1 tab is selected, displaying a table with two columns: DATA_NODE and SQL. The table contains one row with the value 'dn1' in the DATA_NODE column and the SQL statement 'insert into employee(id,name,sharding_id) values(1,'test2',10000)' in the SQL column.

DATA_NODE	SQL
dn1	insert into employee(id,name,sharding_id) values(1,'test2',10000)

➤ Mycat 水平分片配置举例—分片枚举

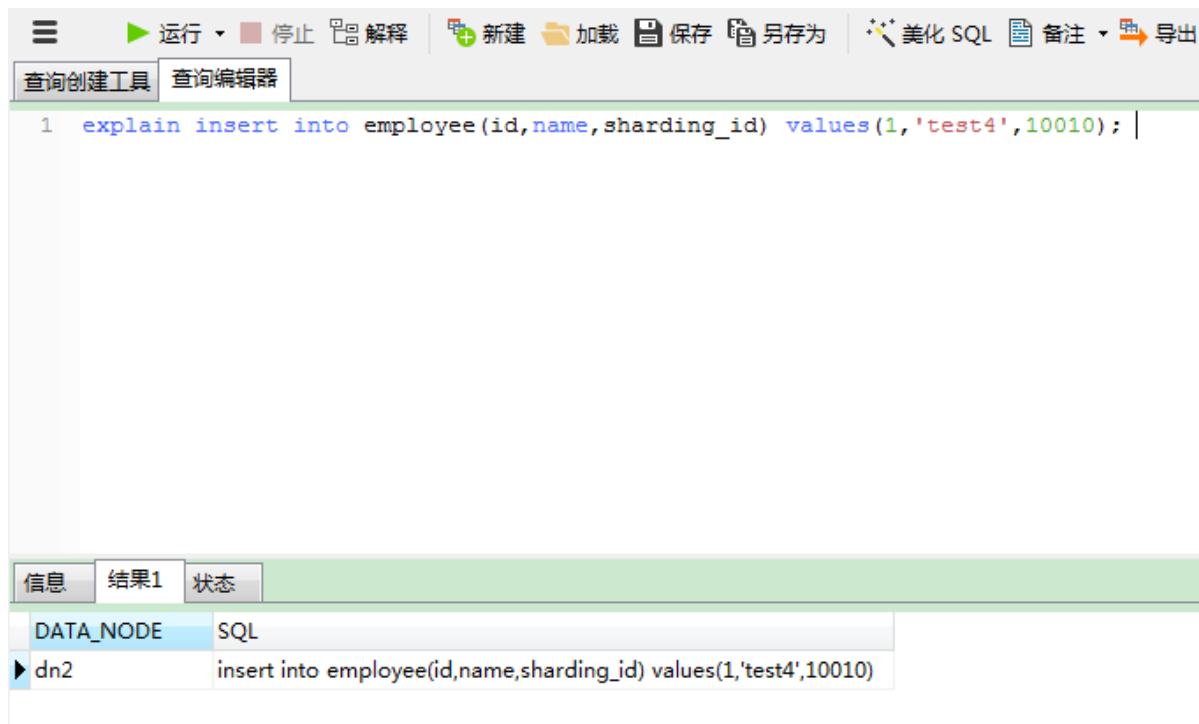
插入两条sharding_id=10010的数据，数据被插入到dn2节点

1) 语句1：explain insert into employee(id,name,sharding_id) values(1,'test3',10010);



➤ Mycat 水平分片配置举例—分片枚举

2) 语句2 : explain insert into employee(id,name,sharding_id) values(1,'test4',10010);



➤ Mycat 水平分片配置举例—主键范围

```
( 1 ) 使用规则 : /src/main/resources/schema.xml
      <table name="travelrecord" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />
( 2 ) 定义规则 : /src/main/resources/rule.xml
      <tableRule name="auto-sharding-long">
        <rule>
          <columns>id</columns>
          <algorithm>rang-long</algorithm>
        </rule>
      </tableRule>
      <function name="rang-long">
        class="org.opencloudb.route.function.AutoPartitionByLong">
          <property name="mapFile">autopartition-long.txt</property>
        </function>
( 3 ) 规则文件 : /src/main/resources/autopartition-long.txt
      # range start-end ,data node index
      # K=1000,M=10000.
      0-500M=0
      500M-1000M=1
      1000M-1500M=2
```

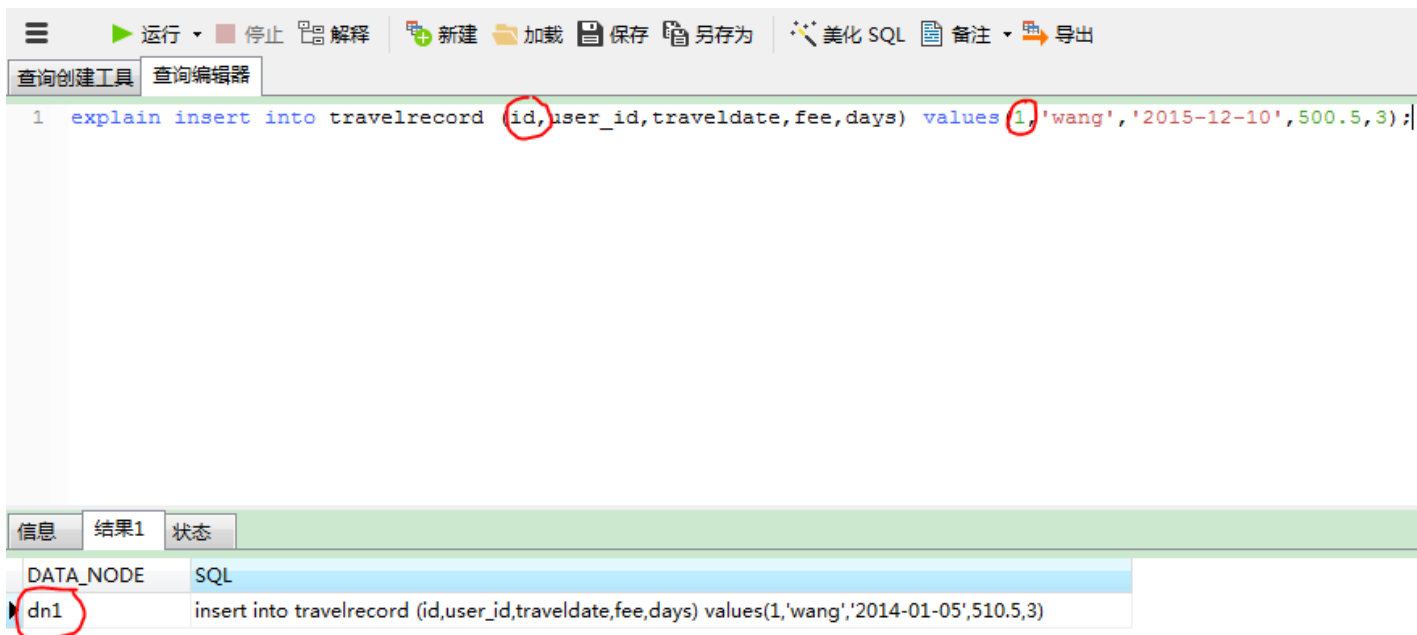
长整形的最大值：2的63次方-1

长整形的最小值：负的2的63次方-1

➤ Mycat 水平分片配置举例—主键范围

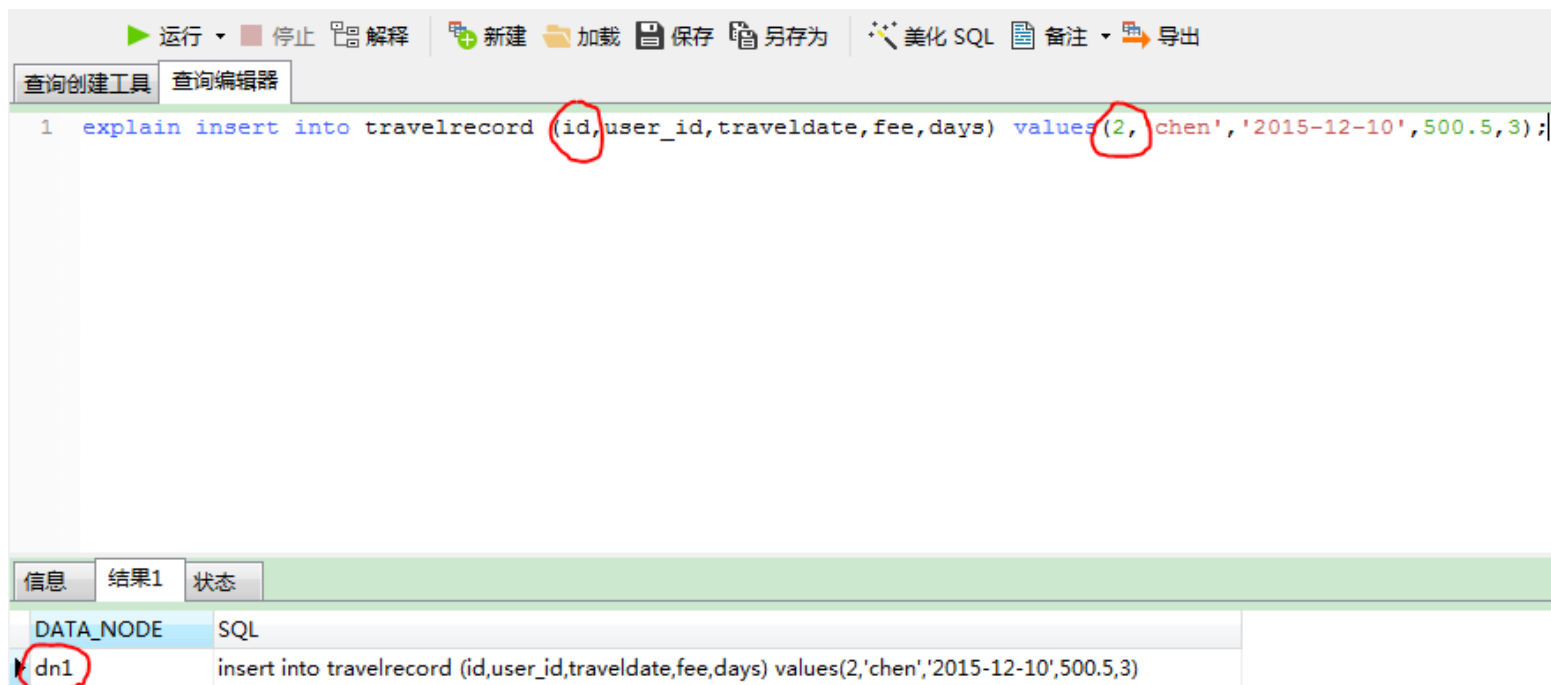
插入两条id范围在0-500M的数据，数据被插入到dn1节点

1) 语句1：explain insert into travelrecord (id,user_id,traveldate,fee,days) values(1,'wang','2015-12-10',500.5,3);



➤ Mycat 水平分片配置举例—主键范围

2) 语句2 : explain insert into travelrecord (id,user_id,traveldate,fee,days)
values(2,'chen','2015-12-10',500.5,3);



The screenshot shows a SQL client interface with a toolbar at the top containing buttons for '运行' (Run), '停止' (Stop), '解释' (Explain), '新建' (New), '加载' (Load), '保存' (Save), '另存为' (Save As), '美化 SQL' (Format SQL), '备注' (Comment), and '导出' (Export). Below the toolbar are two tabs: '查询创建工具' (Query Creation Tool) and '查询编辑器' (Query Editor). The '查询编辑器' tab is active, displaying the following SQL query:

```
1 explain insert into travelrecord (id,user_id,traveldate,fee,days) values(2,'chen','2015-12-10',500.5,3);
```

The values '2' and 'chen' in the query are circled in red. Below the query editor is a table with three columns: '信息' (Information), '结果1' (Result 1), and '状态' (Status). The table has one row with the following data:

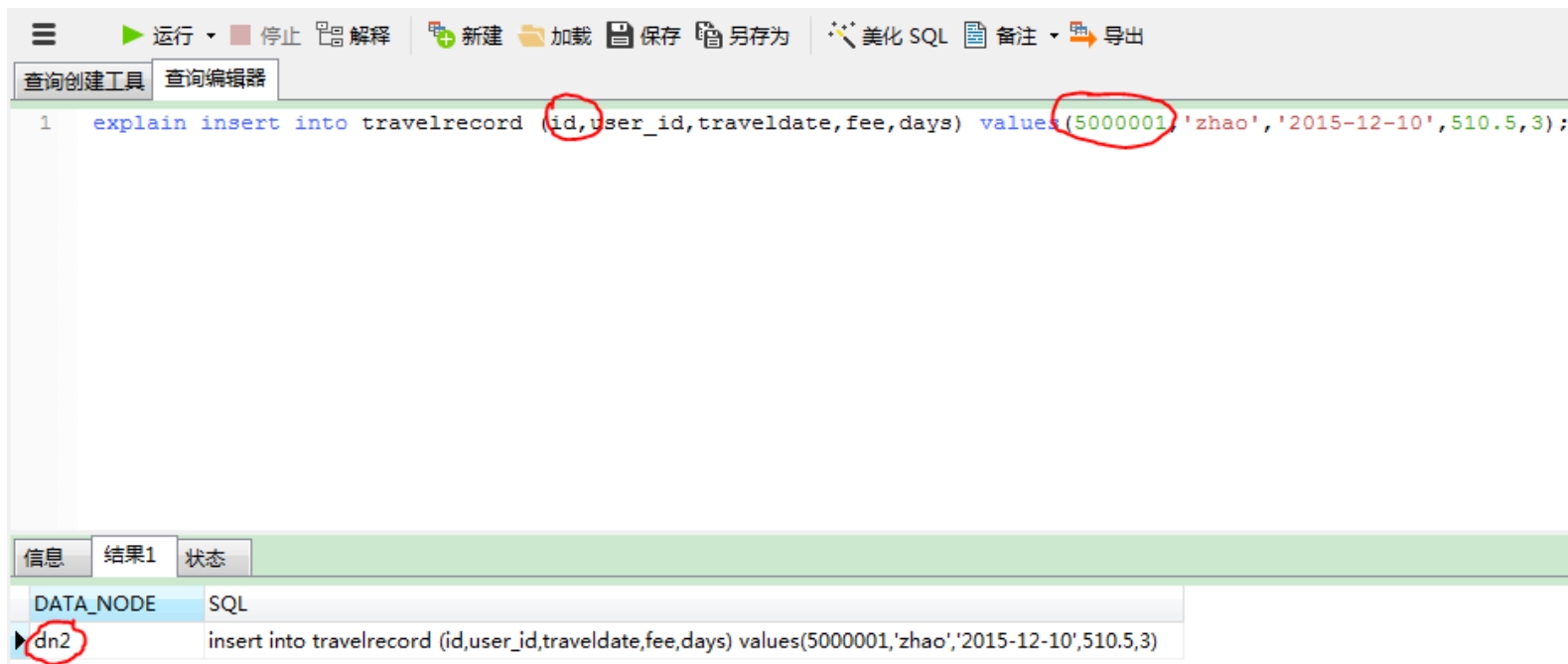
信息	结果1	状态
dn1	insert into travelrecord (id,user_id,traveldate,fee,days) values(2,'chen','2015-12-10',500.5,3)	

The 'dn1' value in the first column is circled in red.

➤ Mycat 水平分片配置举例—主键范围

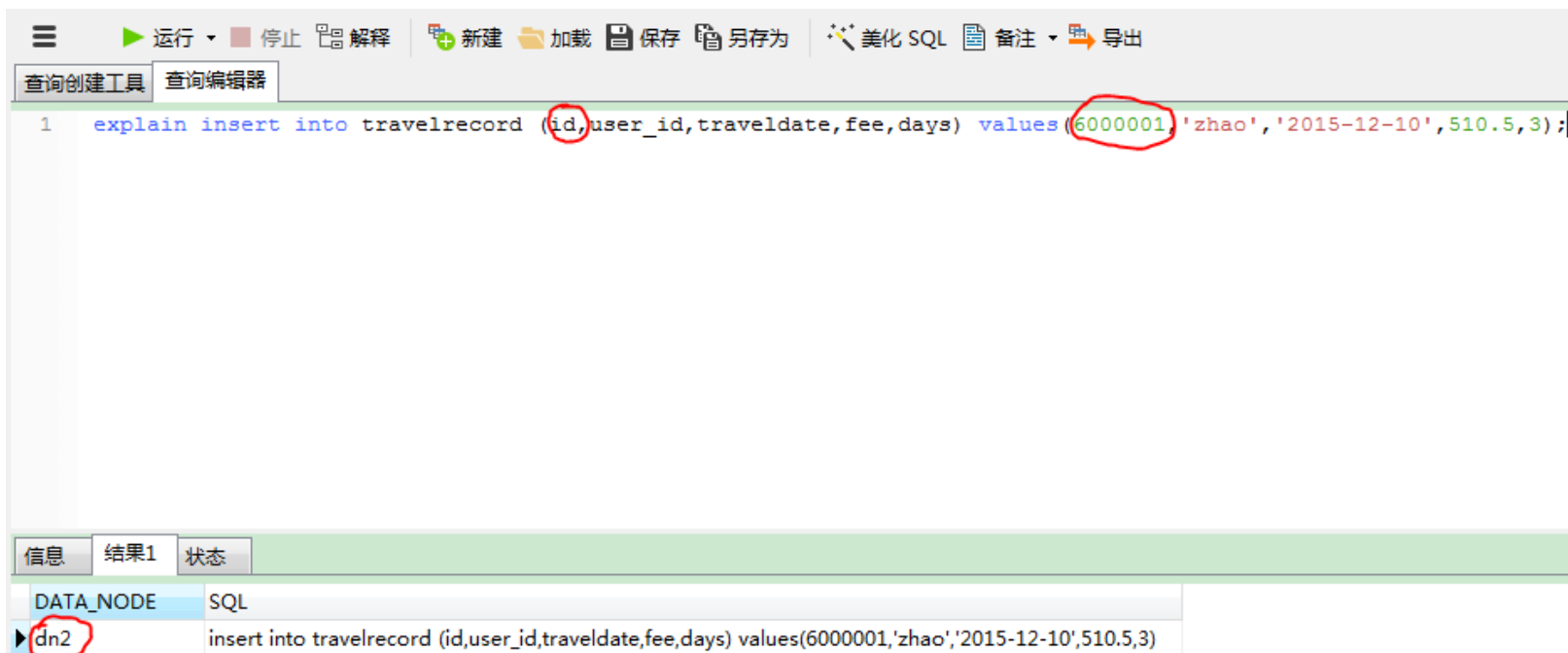
插入两条id范围在500M-1000M的数据，数据被插入到dn2节点

1) 语句1：explain insert into travelrecord (id,user_id,traveldate,fee,days) values(5000001,'zhao','2015-12-10',510.5,3);



➤ Mycat 水平分片配置举例—主键范围

2) 语句2 : explain insert into travelrecord (id,user_id,traveldate,fee,days)
values(6000001,'zhao','2015-12-10',510.5,3);



运行 停止 解释 新建 加载 保存 另存为 美化 SQL 备注 导出

查询创建工具 查询编辑器

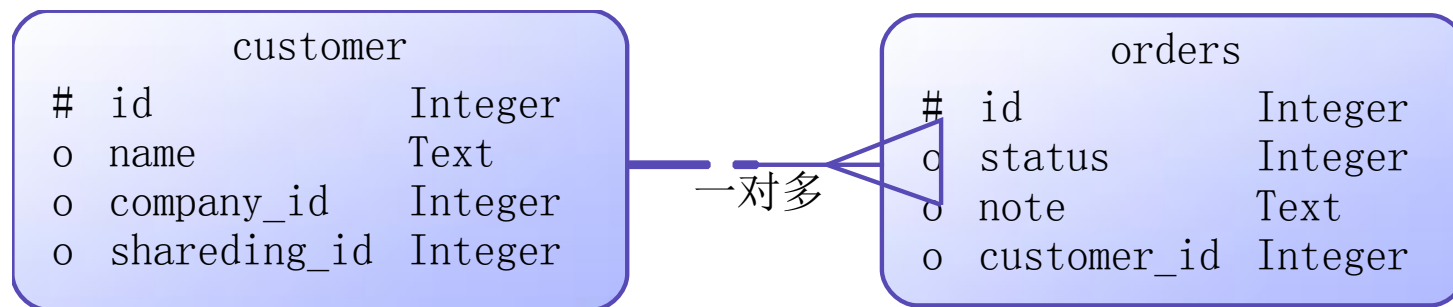
```
1 explain insert into travelrecord (id,user_id,traveldate,fee,days) values(6000001,'zhao','2015-12-10',510.5,3);
```

信息	结果1	状态
DATA_NODE	SQL	
dn2	insert into travelrecord (id,user_id,traveldate,fee,days) values(6000001,'zhao','2015-12-10',510.5,3)	

➤ Mycat 水平分片—ER分片

根据数据库逻辑数据ER关系模型（一对一、一对多、多对多），将拆分规则下的关联数据，放到同一个分片节点上，避免出现跨分片连接操作。

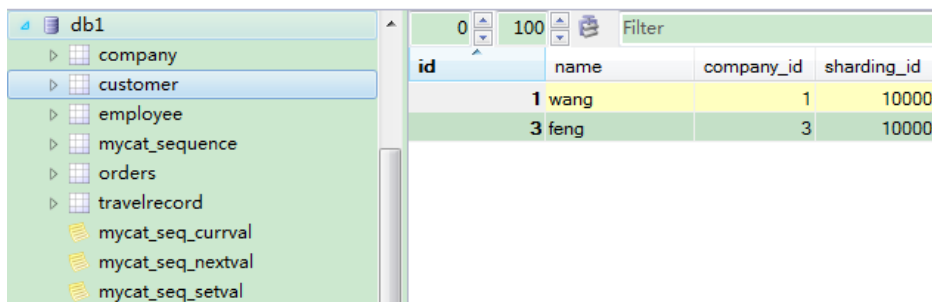
```
<table name="customer" primaryKey="ID" dataNode="dn1,dn2" rule="sharding-by-intfile">  
  <childTable name="orders" primaryKey="ID" joinKey="customer_id" parentKey="id">  
    </childTable>  
</table>
```



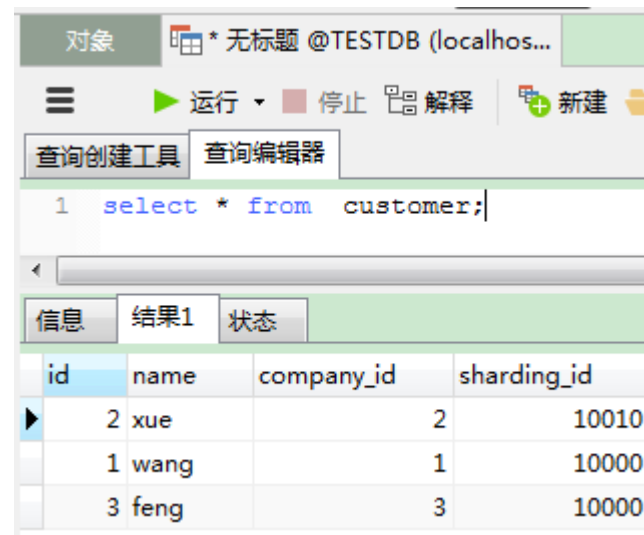
➤ Mycat 水平分片—ER分片

```
CREATE customer : CREATE TABLE customer (  
    id INT NOT NULL PRIMARY KEY,  
    NAME VARCHAR (100),  
    company_id INT NOT NULL,  
    sharding_id INT NOT NULL  
);
```

```
insert into customer (id,name,company_id,sharding_id )values(1,'wang',1,10000); /*stored in db1*/  
insert into customer (id,name,company_id,sharding_id )values(2,'xue',2,10010); /*stored in db2*/  
insert into customer (id,name,company_id,sharding_id )values(3,'feng',3,10000); /*stored in db3*/
```



id	name	company_id	sharding_id
1	wang	1	10000
3	feng	3	10000

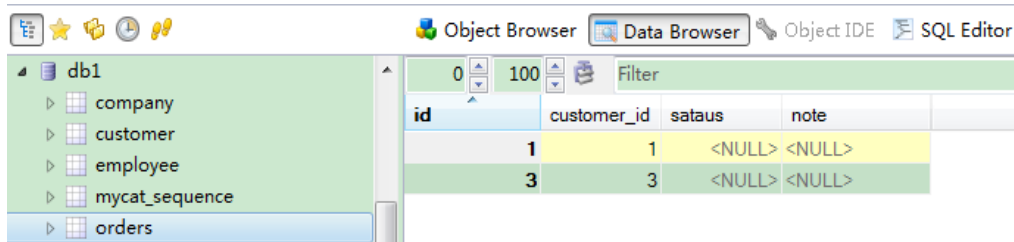


id	name	company_id	sharding_id
2	xue	2	10010
1	wang	1	10000
3	feng	3	10000

➤ Mycat 水平分片—ER分片

```
CREATE TABLE orders (  
    id INT NOT NULL PRIMARY KEY,  
    customer_id INT NOT NULL,  
    sataus INT,  
    note VARCHAR (100)  
);
```

```
insert into orders(id,customer_id) values(1,1); /*stored in db1 because customer table with id=1 stored in db1*/  
insert into orders(id,customer_id) values(2,2); /*stored in db2 because customer table with id=2 stored in db2*/  
insert into orders(id,customer_id) values(3,3); /*stored in db1 because customer table with id=3 stored in db1*/
```



Object Browser | Data Browser | Object IDE | SQL Editor

db1

- company
- customer
- employee
- mycat_sequence
- orders

id	customer_id	sataus	note
1	1	<NULL>	<NULL>
3	3	<NULL>	<NULL>

查询创建工具

查询编辑器

```
1 select * from orders;
```

信息

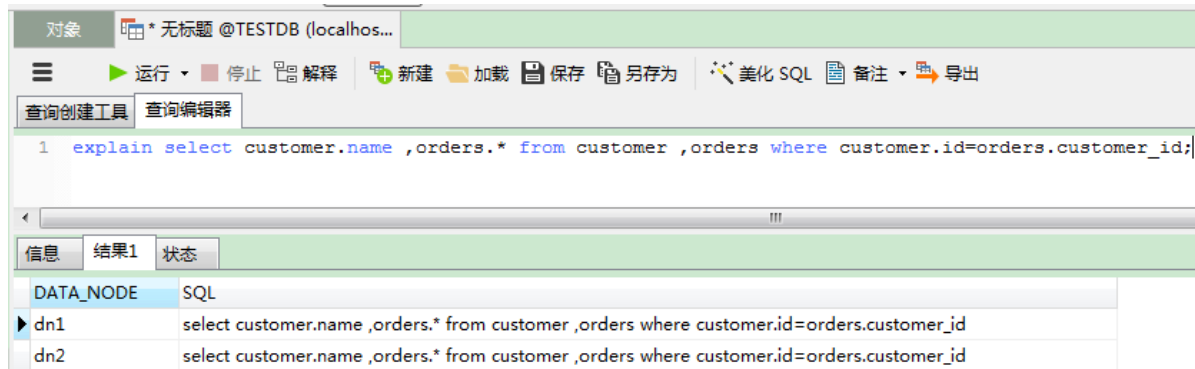
结果1

状态

id	customer_id	sataus	note
2	2	(Null)	(Null)
1	1	(Null)	(Null)
3	3	(Null)	(Null)

➤ Mycat 水平分片—ER分片

ER关系表联合查询测试：select customer.name ,orders.* from customer ,orders where customer.id=orders.customer_id;



对象 * 无标题 @TESTDB (localhos...)

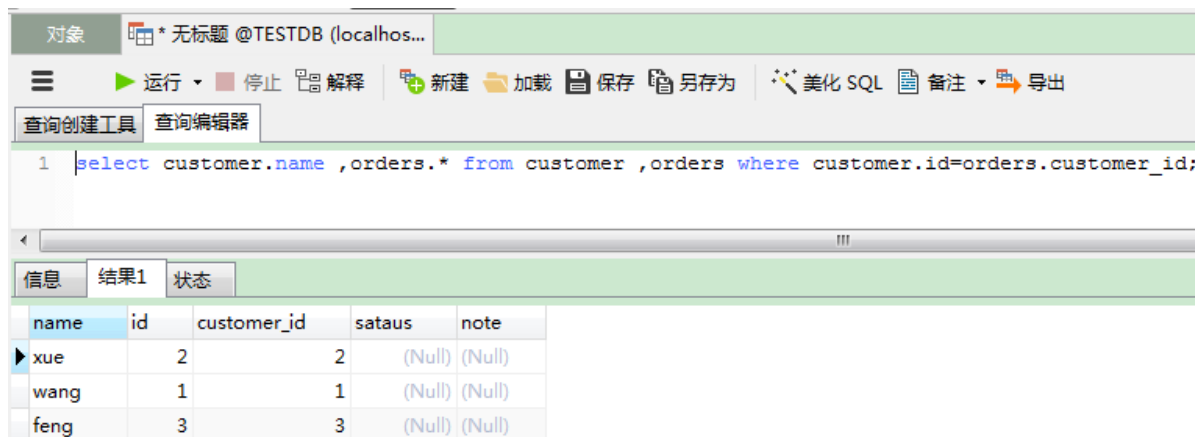
运行 停止 解释 新建 加载 保存 另存为 美化 SQL 备注 导出

查询创建工具 查询编辑器

```
1 explain select customer.name ,orders.* from customer ,orders where customer.id=orders.customer_id;
```

信息 结果1 状态

DATA_NODE	SQL
dn1	select customer.name ,orders.* from customer ,orders where customer.id=orders.customer_id
dn2	select customer.name ,orders.* from customer ,orders where customer.id=orders.customer_id



对象 * 无标题 @TESTDB (localhos...)

运行 停止 解释 新建 加载 保存 另存为 美化 SQL 备注 导出

查询创建工具 查询编辑器

```
1 select customer.name ,orders.* from customer ,orders where customer.id=orders.customer_id;
```

信息 结果1 状态

name	id	customer_id	sataus	note
xue	2	2	(Null)	(Null)
wang	1	1	(Null)	(Null)
feng	3	3	(Null)	(Null)

➤ 分片全局序列号

分片以后MySQL 的自增主键就不能使用了，为了实现分片情况下表的主键是全局唯一的，可以使用Mycat 提供的全局序列号功能：

格式为：MYCATSEQ_*seqname*

MYCATSEQ 必须大写

seqname 为具体定义的sequence的名称

Mycat 自身提供了一个全局序列号，名称为：MYCATSEQ_GLOBAL

1. 使用默认的全局sequence：

```
insert into table1(id,name) values(next value for  
MYCATSEQ_GLOBAL,'test');
```

2. 使用自定义的sequence：

```
insert into table2(id,name) values(next value for  
MYCATSEQ_MYSEQ,'test');
```

3. 获取最新的值

```
Select next value for MYCATSEQ_MYSEQ
```

➤ Mycat 全局序列号获取方式

Mycat 全局序列号提供了三种获取方式：

- ✓ 文件获取方式
- ✓ 数据库获取方式
- ✓ 时间戳获取方式

(1) 文件获取方式

1. 配置 server.xml
`<property name="sequenceHandlerType">0</property>`
2. 配置 `sequence_conf.properties`
default global sequence
GLOBAL.HISIDS=
GLOBAL.MINID=10001
GLOBAL.MAXID=20000
GLOBAL.CURID=10000

self define sequence
MYSEQ.HISIDS=
MYSEQ.MINID=1001
MYSEQ.MAXID=2000
MYSEQ.CURID=1000

➤ Mycat 全局序列号获取方式

(2) 数据库获取方式:

1. 配置 server.xml
`<property name="sequenceHandlerType">0</property>`
2. 配置 sequence_db_conf.properties
#sequence stored in datanode
GLOBAL=dn1
MYSEQ=dn1
3. 在对应数据库中执行创建获取序列号方法脚本
 - (1) 创建MYCAT_SEQUENCE 表
 - (2) 创建mycat_seq_currval方法，获取当前 sequence的值
 - (3) 创建mycat_seq_nextval方法，获取下一个sequence值
 - (4) 创建mycat_seq_setval方法，设置sequence值

建议采用数据库获取方式，脚本全文如下：



全局序列号数据

➤ Mycat 全局序列号获取方式

(3) 时间戳获取方式:

时间戳ID为64位二进制数，换算成十进制为18位数的long类型

1.配置server.xml

```
<property name="sequenceHandlerType">2</property>
```

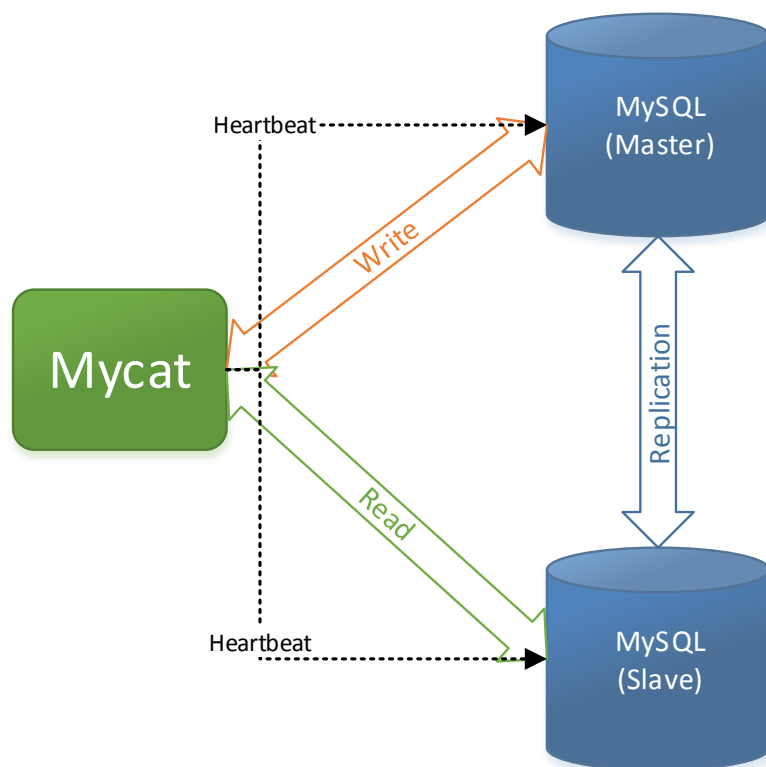
2.配置：sequence_time_conf.properties

WORKID=0-31 任意整数

DATAACENTERID=0-31 任意整数

➤ Mycat 读写分离与自动切换

Mycat 支持基于MySQL主从复制状态监控，而一旦检测到主从同步出错或者延时，则自动排除readHost，防止程序读到不一致的旧数据。



➤ Mycat 读写分离与自动切换

在schema.xml 配置文件的数据Host 元素上定义两个属性，则开启MySQL 主从复制状态绑定的读写分离与切换机制：

switchType=2

slaveThreshold=100

switchType 属性有三个值：

- ✓ switchType=-1 表示不自动切换
- ✓ switchType=1 默认值，自动切换
- ✓ switchType=2 基于MySQL 主从同步的状态决定是否切换

```
<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
  writeType="0" dbType="mysql" dbDriver="native" switchType="2" slaveThreshold="100">
  <heartbeat>show slave status </heartbeat>
</dataHost>
```

writeType=0 是默认配置，writeType=1 代表配置多主，此模式下无读节点。Mycat 往所有写节点随机写数据，但是每次只会写入一个节点，节点之间开启数据库同步，多主会带来同步一致性问题。

➤ Mycat 读写分离与自动切换

Mycat 主从复制状态监控，通过执行show slave status命令的

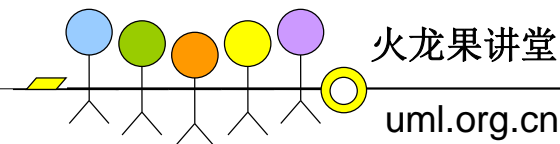
- ✓ Seconds_Behind_Master
- ✓ Slave_IO_Running
- ✓ Slave_SQL_Running

三个字段来确定当前主从同步的状态：

```
mysql> show slave status\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      ...
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Seconds_Behind_Master: 0
      ...
1 row in set (0.00 sec)
```

Seconds_Behind_Master>slaveThreshold，读写分离筛选器会过滤掉此Slave，防止读到旧数据；
切换逻辑会检查从节点上的Seconds_Behind_Master是否为0，为0时则表示主从同步，可以安全切换，否则不会切换。

数据库中间件核心功能介绍二



➤ balance 属性设置

balance=0：不开启读写分离

balance=1：全部的readHost与stand by writeHost参与select语句的负载均衡，简单的说，当双主双从模式(M1->S1，M2->S2，并且M1与M2互为主备)，正常情况下，M2,S1,S2都参与select语句的负载均衡。

balance=2：所有的readHost与writeHost都参与select语句的负载均衡，也就是说，当系统的写操作压力不大的情况下，所有主机都可以承担负载均衡。

balance=3：全部的读节点参数读，写节点不参与，如果配置了多个writerhost，则多个writerhost下面的readhost参数读负载。

➤ 读写分离机制

Mycat 读写分离机制如下：

- 1、事务内的SQL，默认走写节点，以注释/*balance*/开头，则会根据balance去获取；
- 2、自动提交的select语句会走读节点，并在所有可用读节点中间随机负载均衡,默认根据balance去获取，以注释/*balance*/开头则会走写，解决部分已经开启读写分离，但是需要强一致性数据实时获取数据的场景走写；
- 3、当某个主节点宕机，则其全部读节点都不再被使用，因为此时，同步失败，数据已经不是最新的，MYCAT会采用另外一个主节点所对应的全部读节点来实现select负载均衡；
- 4、当所有主节点都失败，则为了系统高可用性，自动提交的所有select语句仍将提交到全部存活的读节点上执行，此时系统的很多页面还是能出来数据，只是用户修改或提交会失败。

➤ Mycat 注解简介

注解格式如下：

`/*!mycat: sql=SQL*/`，使用时将=号后的SQL语句替换为需要的语句即可。

注解是告诉Mycat按照注解内的SQL（称之为注解SQL）去进行解析处理，解析出分片信息后，将注解后真正要执行的SQL语句（称之为原始SQL）发送到该分片对应的物理库去执行。注解只是告诉Mycat到何处去执行原始SQL，因而使用注解前，要提前知道原始SQL去哪个分片执行，然后在注解SQL中指向该分片，这样才能使用。例如例子中的`sharding_id=10010`

例如，需要执行如下语句：

```
insert into persons(id,name,sharding_id) values(1,' abc' ,10010),(2,' xyz' ,10010);
```

Mycat不支持这种语句的执行，如果需要执行这种语句，可在这个语句前添加

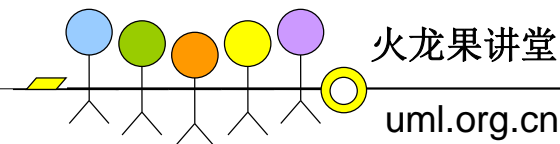
```
/*!mycat: sql=select id from persons where sharding_id=10010 */
```

语句变为如下形式即可执行

```
/*!mycat: sql=select id from persons where sharding_id=10010 */insert into  
persons(id,name,sharding_id) values(1,' abc' ,10010),(2,' xyz' ,10010);
```

注意：如果例子的注解没有添加`sharding_id=10010`这个条件，则Mycat会将SQL发送到persons表所在的所有分片上去执行去，这样造成语句在多个分片上执行。

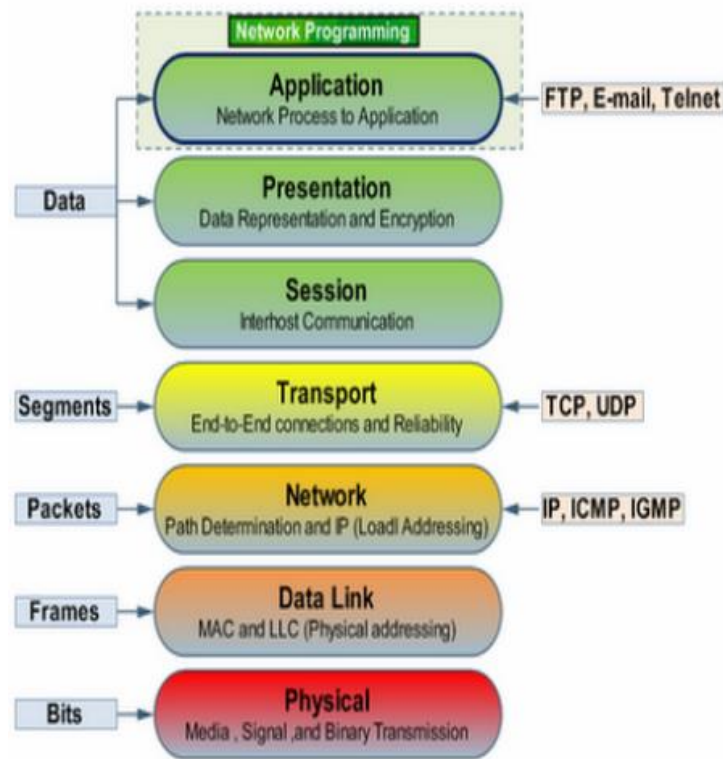
Mycat 高可用与一致性分析



互联网系统与七层协议模型

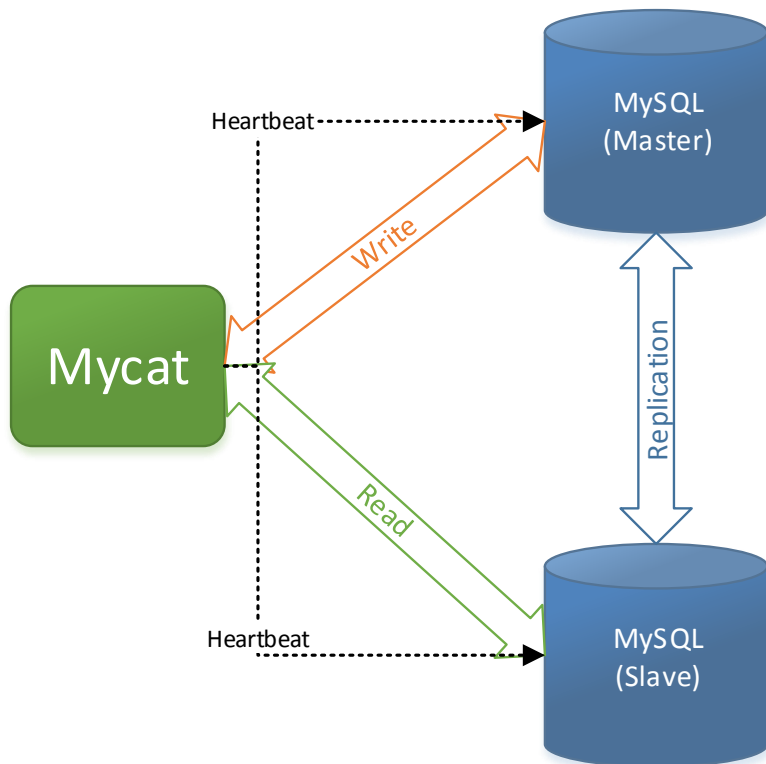
- ✓ 可用性
- ✓ 可靠性
- ✓ 安全性
- ✓ 一致性
- ✓

互联网系统中的任何产品都不是孤立的，因此当我们讨论产品特性时，其实是在讨论与该产品相关的系统整体的架构设计，而系统各层间的通信是通过各种协议实现的，因此好的产品应尽可能多的支持各种通信协议，同时在协议模型的每一层都应该考虑这些特性的实现。另外在分析CAP理论的时候我们可以了解到，这些产品特性不可兼得，彼此之间是此消彼长的关系，我们只能根据业务特点，找到最佳的平衡点。



Mycat 集群的高可用与一致性分析一

```
<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"  
  writeType="0" dbType="mysql" dbDriver="native" switchType="2" slaveThreshold="100">  
  <heartbeat>show slave status </heartbeat>  
</dataHost>
```

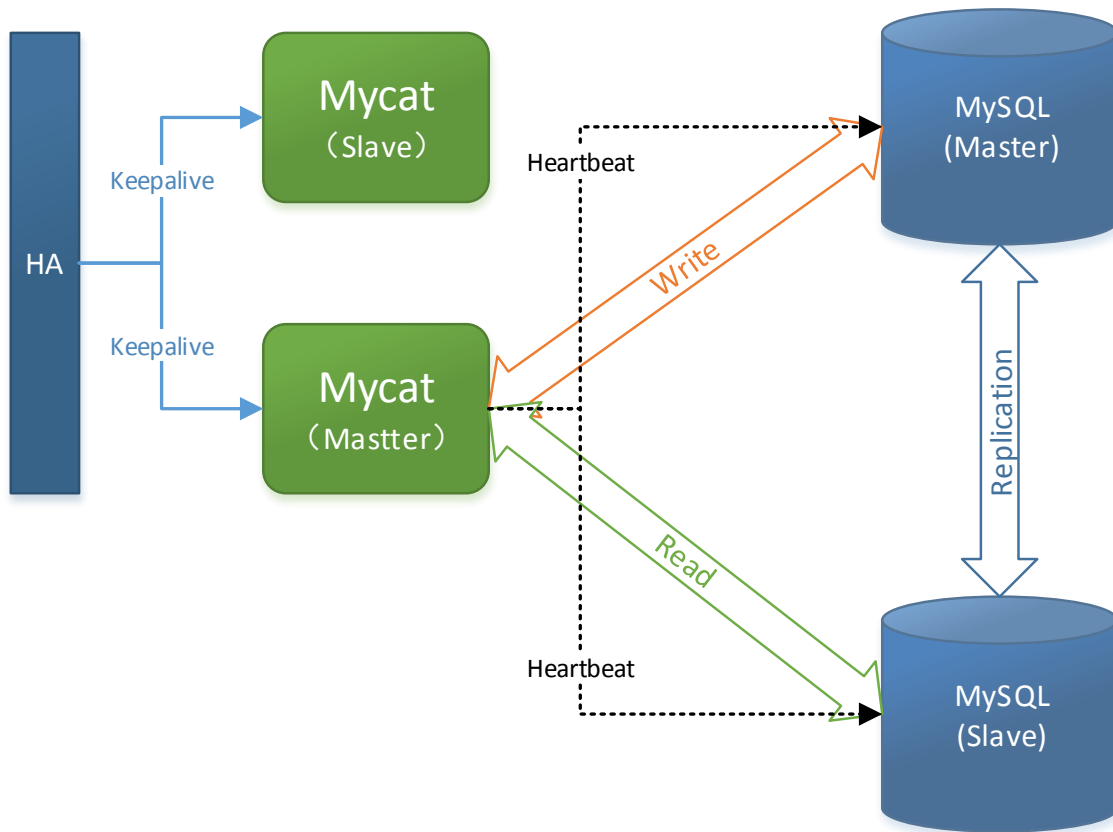


架构特点：

- ✓ MySQL节点做冗余，并通过Mycat 进行数据一致性检测和自动切换，数据库层可用性增强
- ✓ 冗余和读写分离使系统数据一致性降低
- ✓ Mycat是单点故障，可用性低

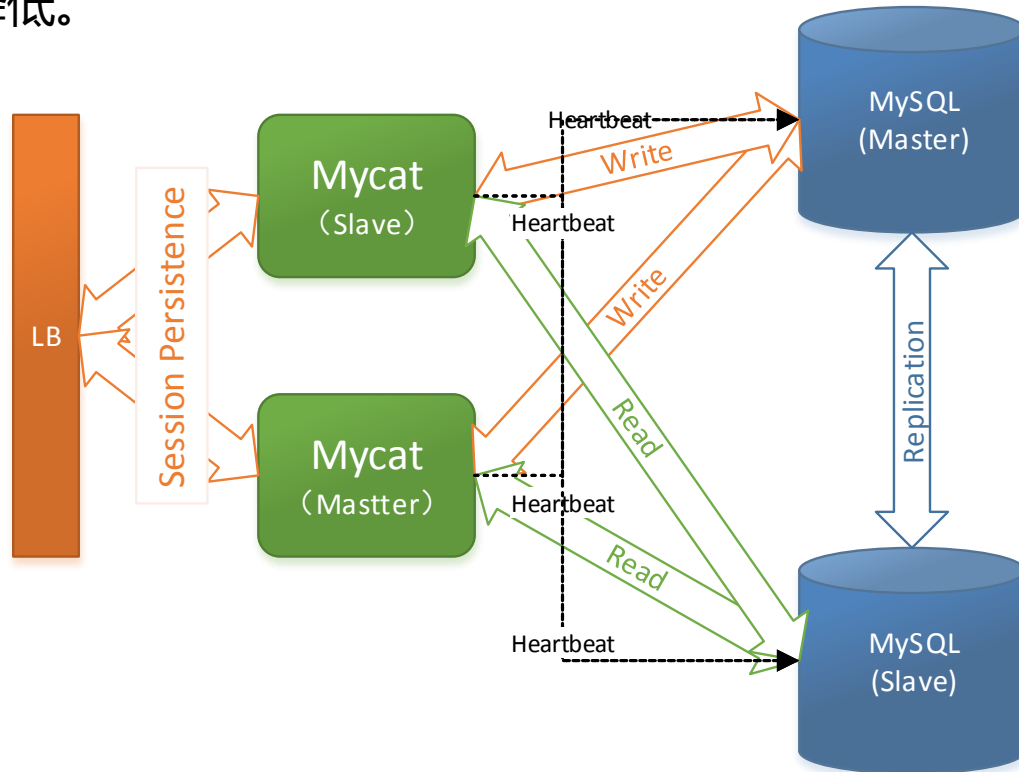
Mycat 集群的高可用与一致性分析二

MySQL节点做冗余，并通过Mycat进行数据一致性检测和自动切换，数据库层可用性增强，冗余和读写分离使系统数据一致性降低。建立Mycat主备架构集群，提高了Mycat可用性，但架构中依然只有一个Mycat节点工作，高并发下的抗压能力不强。



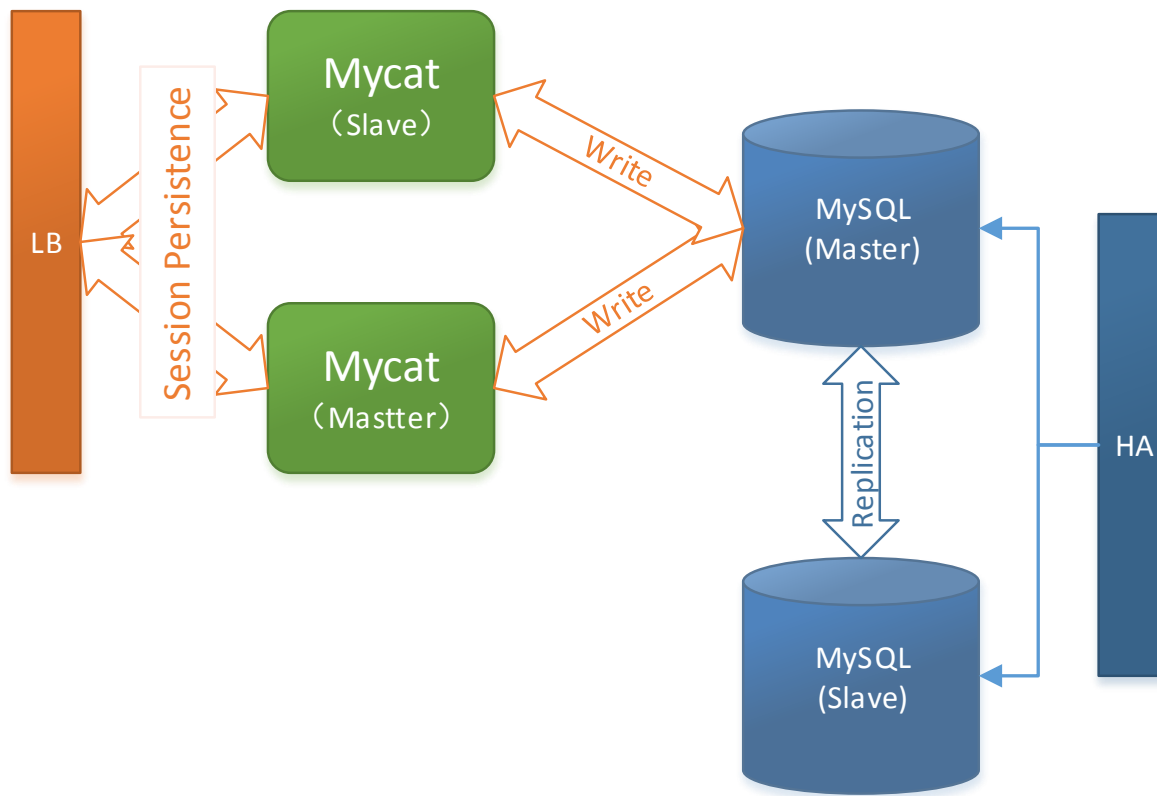
Mycat 集群的高可用与一致性分析三

MySQL节点做冗余，并通过Mycat进行数据一致性检测和自动切换，数据库层可用性增强，冗余和读写分离使系统数据一致性降低。建立Mycat负载均衡架构集群，提高Mycat可用性和高并发下的抗压能力，LB设备必须支持会话保持功能，以保证会话结果正确返回，但是由于可能产生多个Mycat节点心跳不一致问题，系统数据一致性进一步降低。



Mycat 集群的高可用与一致性分析四

MySQL做冗余，可用性增强，建立Mycat负载均衡架构集群，提高Mycat可用性，高并发下的抗压能力。不使用Mycat心跳检测，因此也无法使用Mycat读写分离和自动切换功能。使用其他HA设备保证MySQL的高可用性，由于没有启用读写分离策略，所以数据层抗压能力被降低。



Mycat

- ✓ 支持SQL92标准；
- ✓ 支持JDBC连接Oracle、MongoDB；
- ✓ 支持MySQL及其变种的数据分片集群；
- ✓ 自动故障切换支持读写分离；
- ✓ 支持全局表，数据自动分片到多个节点，用于高效表关联查询；
- ✓ 独有的基于ER关系的分片策略，实现了高效的表关联查询
- ✓ 多平台支持，部署简单



360 Atlas

由奇虎360公司的Web平台部基础架构团队开发维护，是一个基于MySQL协议的数据中间层项目。它是在MySQL Proxy 0.8.2 版本的基础上，对其进行了优化，增加了一些新的功能特性。360内部使用Atlas运行的MySQL业务，每天承载的读写请求数达几十亿条。

主要功能：

- ✓ 读写分离；
- ✓ 从库负载均衡；
- ✓ IP过滤；
- ✓ SQL语句黑白名单；
- ✓ 自动分表

百度 Heisenberg

由百度员工编写的MySQL分片中间件

主要功能如下：

- ✓ 分片与应用脱离，分库表如同使用单库表一样；
- ✓ 整合数据源；
- ✓ 支持热重启配置；
- ✓ 支持水平扩容；
- ✓ 支持MySQL原生协议；
- ✓ 支持多语言，C、Java等都可以使用；
- ✓ 可以通过管理命令调整参数，如连接数、线程池、节点等

腾讯 TDSQL

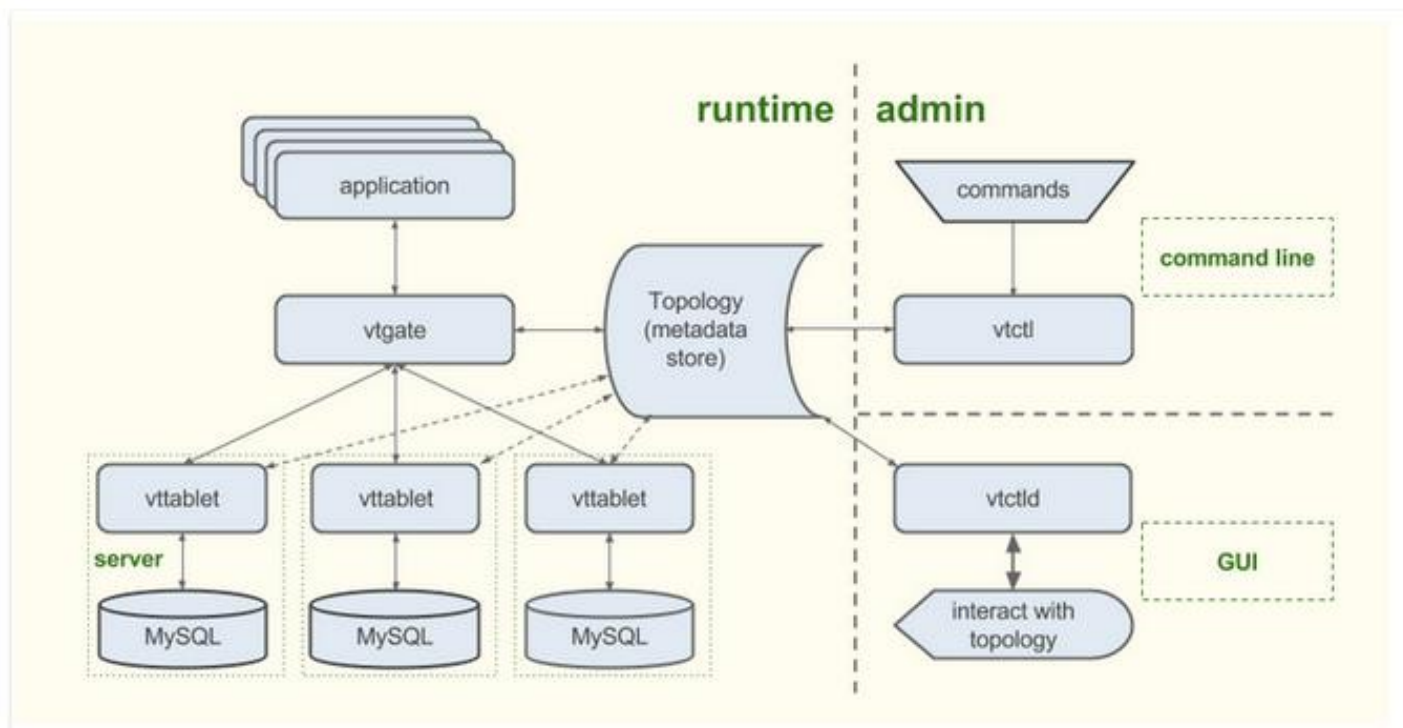
由腾讯编写的MySQL中间件，主要用户数据库集群管理，包括自动切换、自动恢复、主备一致性检测、跨IDC容灾

主要功能如下：

- ✓ 通过MySQL API和SQL接口访问集群；
- ✓ 节点异常自动切换，切换过程保证数据零丢失，管好钱袋子；
- ✓ 按需自动扩容/缩容，以支撑业务爆发式增长，扩容过程对业务基本上无感知；
- ✓ 业务之间支持隔离，集群自身具备流控机制；
- ✓ 配套工具完善：时耗分析、慢查询分析、冷备及恢复中心、异常SQL拦截、流控等

谷歌 Vitess

谷歌开发的数据库中间件，集群基于ZooKeeper管理，通过RPC方式进行数据处理，总体分为服务器，命令行，GUI监控三个部分。



感谢您的观赏！

开源产品就像一块布，不能拿来就用，我们要量体裁衣，把它变成一件衣服。

互联网开发无小事，因为互联网应用要面对海量信息，一个小小的问题也能被无限放大，就像汪洋中的蝴蝶效应。

一个互联网应用遇到的问题可能与系统、网络、应用、架构、数据等各个层面的问题有关，只有耐心发现并处理好每一个相关层面的问题，才能最终形成一款好的互联网应用。时刻提醒自己当下一波巨浪到来前我们会准备的更好！

