**Mycat与Mysql跨库JOIN与性能测试**

文档撰写：彭军林

完成日期：2017-6-14

|  |  |
| --- | --- |
| **工作事项** | **执行人** |
| 服务器数据准备 | 宋旭朋 |
| 服务器分片搭建及测试 | 崔锦璐 |
| JDBC程序测试及跨库测试 | 彭军林 |

上海势航网络科技有限公司

[1. 跨库JOIN分库不分片 2](#_Toc18541)

[1.1. Mysql分库操作 3](#_Toc3306)

[1.2. ※Mycat 跨库JOIN※ 3](#_Toc9413)

[1.3. 人工智能JOIN测试数据 5](#_Toc23341)

[1.3.1. 非limit查询 5](#_Toc29758)

[1.3.2. limit查询 7](#_Toc4513)

[2. 分库-分片JOIN方案 8](#_Toc26469)

[2.1. JOIN概述 8](#_Toc25921)

[2.1.1. INNER JOIN 9](#_Toc4965)

[2.1.2. LEFT JOIN 9](#_Toc25150)

[2.1.3. RIGHT JOIN 10](#_Toc14488)

[2.1.4. FULL JOIN 10](#_Toc16588)

[2.2. 全局表 11](#_Toc19095)

[2.3. ER Join 11](#_Toc14708)

[2.4. Share join 12](#_Toc11731)

[2.5. ※catlet（人工智能）※ 14](#_Toc7362)

[3. Mysql与Mycat插入性能测试 16](#_Toc32515)

[3.1. 基于Mysql的JDBC测试 16](#_Toc20868)

[3.1.1. 多线程插入性能测试 16](#_Toc6886)

[3.1.2. 多线程读取性能测试 17](#_Toc13822)

[3.1.3. 单线程读写性能测试 19](#_Toc4394)

[3.2. 基于Mycat的JDBC读写测试 21](#_Toc956)

[3.2.1. Windows Cmd单线读写测试 21](#_Toc17271)

[3.2.2. Linux批处理读写测试 21](#_Toc4836)

[3.3. Sysbench工具的Mycat测试 22](#_Toc14833)

[3.3.1. Mycat查询统计 22](#_Toc317)

[3.3.2. Mycat插入统计 23](#_Toc16559)

[3.3.3. Mycat读写统计 23](#_Toc8773)

[3.4. 测试对比和结论 24](#_Toc15860)

[3.4.1. 测试数据对比 24](#_Toc2306)

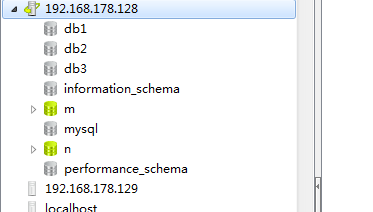
[3.4.2. 测试结论 24](#_Toc28467)

# 跨库JOIN分库不分片

本问题理解存在的跨库JOIN问题可以理解为不同的分片进行JOIN，但JOIN不是基于全局表设置。Mycat提供catlet（人工智能）可对不同分片（数据库实例）的表进行2表JOIN查询。

## Mysql分库操作

这里我们将基础和业务分别放在不同的数据库分片上，创建m和n数据库实例名。



m基础配置数据库：包含t\_user

n业务数据数据库：包含t\_service

## ※Mycat 跨库JOIN※

Mycat基于catlet的分库JOIN为数据库表按类型分库提供了很好的支持，而全局表在每个节点都有DDL实现也可以实现直接JOIN操作。当表设置为全局表后可以与任意一个表进行JOIN操作。

只需要修改${MYCAT\_HOME}/conf/schema.xml

<?xml version="1.0"?>

<!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://io.mycat/">

<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">

<!-- auto sharding by id (long) -->

<table name="t\_user" primaryKey="u\_id" autoIncrement="true" dataNode="dn1" rule="mod-long" >

<!-- <childTable name="t\_service" primaryKey="s\_id" joinKey="s\_uid" parentKey="u\_id"/> -->

</table>

<table name="t\_service" primaryKey="s\_id" autoIncrement="true" dataNode="dn2" rule="mod-long" />

<!-- random sharding using mod sharind rule -->

<!-- <table name="dual" primaryKey="ID" dataNode="dnx,dnoracle2" type="global"

needAddLimit="false"/> <table name="worker" primaryKey="ID" dataNode="jdbc\_dn1,jdbc\_dn2,jdbc\_dn3"

rule="mod-long" /> -->

</schema>

<!-- <dataNode name="dn1$0-743" dataHost="localhost1" database="db$0-743"

/> -->

<dataNode name="dn1" dataHost="localhost1" database="m" />

<dataNode name="dn2" dataHost="localhost1" database="n" />

<!--<dataNode name="dn4" dataHost="sequoiadb1" database="SAMPLE" />

<dataNode name="jdbc\_dn1" dataHost="jdbchost" database="db1" />

<dataNode name="jdbc\_dn2" dataHost="jdbchost" database="db2" />

<dataNode name="jdbc\_dn3" dataHost="jdbchost" database="db3" /> -->

<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

<!-- can have multi write hosts -->

<writeHost host="hostM1" url="192.168.178.128:3306" user="root" password="123456">

<!-- can have multi read hosts -->

<readHost host="hostS2" url="192.168.178.128:3306" user="root" password="123456" />

</writeHost>

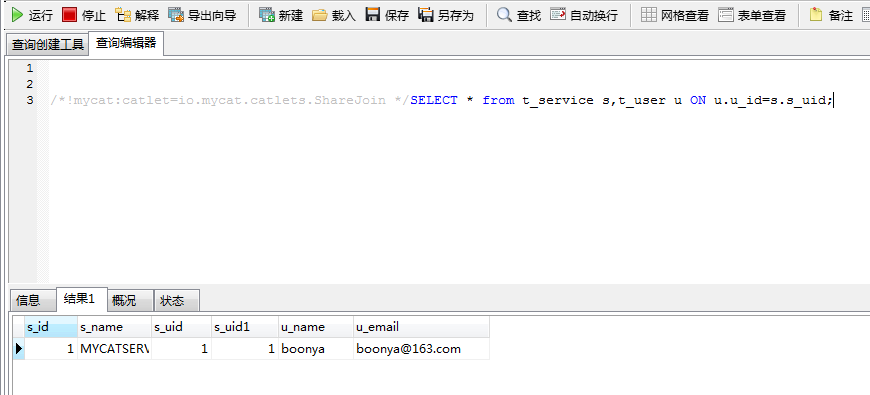
<!-- <writeHost host="hostM2" url="localhost:3316" user="root" password="123456"/> -->

</dataHost>

</mycat:schema>

利用catlets人工智能解析工具JOIN：

/\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid;



## 人工智能JOIN测试数据

### 非limit查询

boonya@ubuntu:~$ mysql -h192.168.178.128 -uroot -p123456 -P8666

Warning: Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 5

Server version: 5.6.29-mycat-1.6-RELEASE-20161028204710 MyCat Server (OpenCloundDB)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> /\*!mycat:catlet=demo.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid;

ERROR 1064 (HY000): java.lang.ClassNotFoundException: demo.catlets.ShareJoin

mysql> /\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid;

Empty set (0.15 sec)

mysql> show databases;

+----------+

| DATABASE |

+----------+

| TESTDB |

+----------+

1 row in set (0.00 sec)

mysql> /\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid;

+------+--------------+-------+-------+--------+----------------+

| s\_id | s\_name | s\_uid | s\_uid | u\_name | u\_email |

+------+--------------+-------+-------+--------+----------------+

| 1 | MYCATSERVICE | 1 | 1 | boonya | boonya@163.com |

+------+--------------+-------+-------+--------+----------------+

1 row in set (0.02 sec)

mysql> /\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid;

ERROR 2006 (HY000): MySQL server has gone away

No connection. Trying to reconnect...

Connection id: 15

Current database: \*\*\* NONE \*\*\*

+------+--------------+-------+-------+--------+-----------------+

| s\_id | s\_name | s\_uid | s\_uid | u\_name | u\_email |

+------+--------------+-------+-------+--------+-----------------+

| 1 | MYCATSERVICE | 1 | 1 | boonya | boonya@163.com |

| 2 | SHOPPING | 2 | 2 | niuniu | niuniu@sina.com |

+------+--------------+-------+-------+--------+-----------------+

2 rows in set (0.02 sec)

mysql>

JOIN查询有效。

### limit查询

+------+--------------+-------+-------+--------+-----------------+

| s\_id | s\_name | s\_uid | s\_uid | u\_name | u\_email |

+------+--------------+-------+-------+--------+-----------------+

| 1 | MYCATSERVICE | 1 | 1 | boonya | boonya@163.com |

| 2 | SHOPPING | 2 | 2 | niuniu | niuniu@sina.com |

+------+--------------+-------+-------+--------+-----------------+

2 rows in set (0.37 sec)

mysql> /\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid limit 2;

+------+--------------+-------+-------+--------+-----------------+

| s\_id | s\_name | s\_uid | s\_uid | u\_name | u\_email |

+------+--------------+-------+-------+--------+-----------------+

| 1 | MYCATSERVICE | 1 | 1 | boonya | boonya@163.com |

| 2 | SHOPPING | 2 | 2 | niuniu | niuniu@sina.com |

+------+--------------+-------+-------+--------+-----------------+

2 rows in set (0.06 sec)

mysql> /\*!mycat:catlet=io.mycat.catlets.ShareJoin \*/SELECT \* from t\_service s,t\_user u ON u.u\_id=s.s\_uid limit 1;

+------+--------------+-------+-------+--------+----------------+

| s\_id | s\_name | s\_uid | s\_uid | u\_name | u\_email |

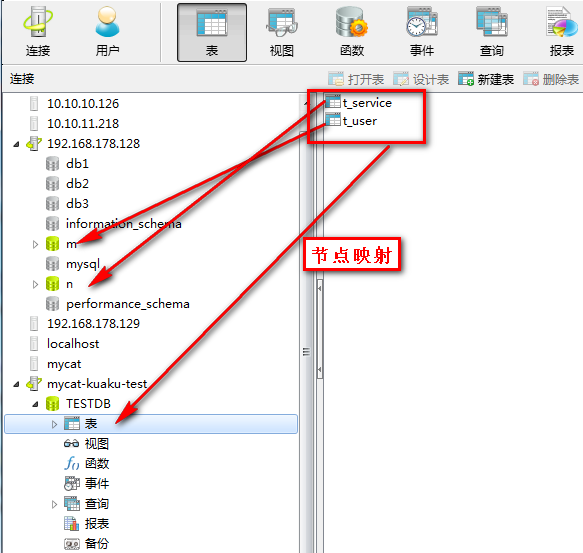
+------+--------------+-------+-------+--------+----------------+

| 1 | MYCATSERVICE | 1 | 1 | boonya | boonya@163.com |

+------+--------------+-------+-------+--------+----------------+

1 row in set (0.02 sec)

最终看得到数据表对应关系如下：



# 分库-分片JOIN方案

根据业务表或日志表进行分片处理，主表-字表之间存在ER关系，分在同一个数据库分片上避免跨库JOIN操作。

表的设置可以在任意分片上。

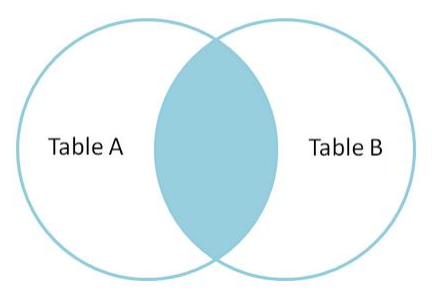
## JOIN概述

Join 绝对是关系型数据库中最常用一个特性，然而在分布式环境中,跨分片的 join 确是最复杂的，最难解决一个问题。下面我们简单介绍下各种 Join 操作。

### INNER JOIN

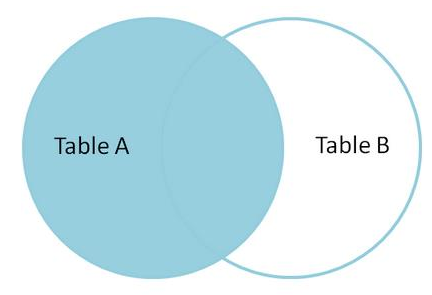
内连接，也叫等值连接，inner join 产生同时符合 A 表和 B 表的一组数据。

如图：



### LEFT JOIN

左连接从 A 表(左)产生一套完整的记录,与匹配的 B 表记录(右表) .如果没有匹配,右侧将包含 null,在 Mysql 中等同于 left outer join。



### RIGHT JOIN

同 Left join,AB 表互换即可。

Cross join

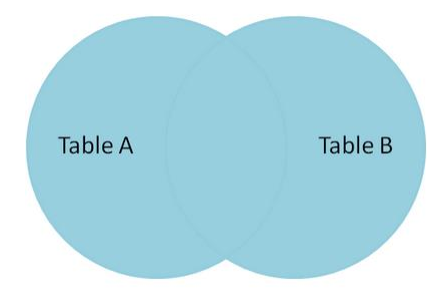
交叉连接，得到的结果是两个表的乘积，即笛卡尔积。笛卡尔（Descartes）乘积又叫直积。假设集合

A={a,b}，集合 B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1), (b,2)}。可以扩展到多个集合的情

况。类似的例子有，如果 A 表示某学校学生的集合，B 表示该学校所有课程的集合，则 A 与 B 的笛卡尔积表示所

有可能的选课情况。

### FULL JOIN



全连接产生的所有记录（双方匹配记录）在表 A 和表 B。如果没有匹配,则对面将包含 null。

**性能建议**

尽量避免使用 Left join 或 Right join,而用 Inner join在使用 Left join 或 Right join 时，ON 会优先执行，where 条件在最后执行，所以在使用过程中，条件尽可能的在 ON 语句中判断，减少 where 的执行少用子查询，而用 join。

Mycat 目前版本支持跨分片的 join,主要实现的方式有四种。

全局表，ER 分片，catletT(人工智能)和 ShareJoin，ShareJoin 在开发版中支持，前面三种方式 1.3.0.1 支持。

## 全局表

一个真实的业务系统中，往往存在大量的类似字典表的表格，它们与业务表之间可能有关系，这种关系，可

以理解为“标签”，而不应理解为通常的“主从关系”，这些表基本上很少变动，可以根据主键 ID 进行缓存，下

面这张图说明了一个典型的“标签关系”图：

在分片的情况下，当业务表因为规模而进行分片以后，业务表与这些附属的字典表之间的关联，就成了比较

棘手的问题，考虑到字典表具有以下几个特性：

• 变动不频繁

• 数据量总体变化不大

• 数据规模不大，很少有超过数十万条记录。

鉴于此，MyCAT 定义了一种特殊的表，称之为“全局表”，全局表具有以下特性：

• 全局表的插入、更新操作会实时在所有节点上执行，保持各个分片的数据一致性

• 全局表的查询操作，只从一个节点获取

• 全局表可以跟任何一个表进行 JOIN 操作

将字典表或者符合字典表特性的一些表定义为全局表，则从另外一个方面，很好的解决了数据 JOIN 的难题。

通过全局表+基于 E-R 关系的分片策略，MyCAT 可以满足 80%以上的企业应用开发。

配置100

全局表配置比较简单，不用写 Rule 规则，如下配置即可：

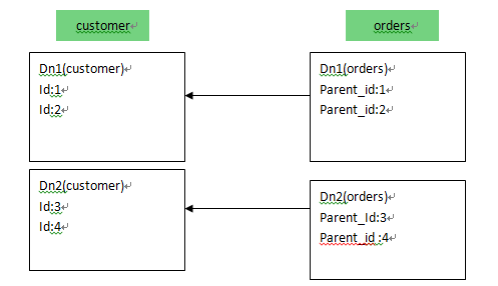
<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />

需要注意的是，全局表每个分片节点上都要有运行创建表的 DDL 语句。

## ER Join

MyCAT 借鉴了 NewSQL 领域的新秀 Foundation DB 的设计思路，Foundation DB 创新性的提出了 TableGroup 的概念，其将子表的存储位置依赖于主表，并且物理上紧邻存放，因此彻底解决了 JION 的效率和性能问题，根据这一思路，提出了基于 E-R 关系的数据分片策略，子表的记录与所关联的父表记录存放在同一个数据分片上。

customer 采用 sharding-by-intfile 这个分片策略，分片在 dn1,dn2 上，orders 依赖父表进行分片，两个表的关联关系为 orders.customer\_id=customer.id。于是数据分片和存储的示意图如下：



这样一来，分片 Dn1 上的的 customer 与 Dn1 上的 orders 就可以进行局部的 JOIN 联合，Dn2 上也如此，再合并两个节点的数据即可完成整体的 JOIN，试想一下，每个分片上 orders 表有 100 万条，则 10 个分片就有 1 个亿，基于 E-R 映射的数据分片模式，基本上解决了 80%以上的企业应用所面临的问题。

**配置**

以上述例子为例，schema.xml 中定义如下的分片配置：

<table name="customer" dataNode="dn1,dn2" rule="sharding-by-intfile">

<childTable name="orders" joinKey="customer\_id" parentKey="id"/>

</table>101

## Share join

ShareJoin 是一个简单的跨分片 Join,基于 HBT 的方式实现。目前支持 2 个表的 join,原理就是解析 SQL 语句，拆分成单表的 SQL 语句执行，然后把各个节点的数据汇集。

**配置**

支持任意配置的 A,B 表如：

A,B 的 dataNode 相同

<table name="A" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />

<table name="B" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />

A,B 的 dataNode 不同

<table name="A" dataNode="dn1,dn2 " rule="auto-sharding-long" />

<table name="B" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" />

或

<table name="A" dataNode="dn1 " rule="auto-sharding-long" />

<table name="B" dataNode=" dn2,dn3" rule="auto-sharding-long" />

**代码测试**

先把表 company 从全局表修改下配置

<table name="company" primaryKey="ID" dataNode="dn1,dn2,dn3" rule="mod-long" />

**重新插入数据**

mysql> delete from company;

Query OK, 9 rows affected (0.19 sec)

mysql> insert company (id,name) values(1,'mycat');

Query OK, 1 row affected (0.08 sec)

mysql> insert company (id,name) values(2,'ibm');

Query OK, 1 row affected (0.03 sec)102

mysql> insert company (id,name) values(3,'hp');

Query OK, 1 row affected (0.03 sec)

下面可以看下普通的 join 和 sharejoin 的区别

mysql> select a.\*,b.id, b.name as tit from customer a,company b where a.company\_id=b.id;

+----+------+------------+-------------+----+------+

| id | name | company\_id | sharding\_id | id | tit |

+----+------+------------+-------------+----+------+

| 3 | feng | 3 | 10000 | 3 | hp |

+----+------+------------+-------------+----+------+

1 row in set (0.03 sec)

mysql> /\*!mycat:catlet=demo.catlets.ShareJoin \*/ select a.\*,b.id, b.name as tit from customer a,company b

on a.company\_id=b.id;

+----+------+------------+-------------+----+-------+

| id | name | company\_id | sharding\_id | id | tit |

+----+------+------------+-------------+----+-------+

| 3 | feng | 3 | 10000 | 3 | hp |

| 1 | wang | 1 | 10000 | 1 | mycat |

| 2 | xue | 2 | 10010 | 2 | ibm |

+----+------+------------+-------------+----+-------+

3 rows in set (0.05 sec)

其他两种写法

/\*!mycat:catlet=demo.catlets.ShareJoin \*/ select a.\*,b.id, b.name as tit from customer a join company b on

a.company\_id=b.id;

+----+------+------------+-------------+----+-------+

| id | name | company\_id | sharding\_id | id | tit |

+----+------+------------+-------------+----+-------+

| 3 | feng | 3 | 10000 | 3 | hp |

| 1 | wang | 1 | 10000 | 1 | mycat |103

| 2 | xue | 2 | 10010 | 2 | ibm |

+----+------+------------+-------------+----+-------+

3 rows in set (0.01 sec)

/\*!mycat:catlet=demo.catlets.ShareJoin \*/ select a.\*,b.id, b.name as tit from customer a join company b where

a.company\_id=b.id;

+----+------+------------+-------------+----+-------+

| id | name | company\_id | sharding\_id | id | tit |

+----+------+------------+-------------+----+-------+

| 3 | feng | 3 | 10000 | 3 | hp |

| 1 | wang | 1 | 10000 | 1 | mycat |

| 2 | xue | 2 | 10010 | 2 | ibm |

+----+------+------------+-------------+----+-------+

3 rows in set (0.01 sec)

对\*的支持，还可以这样写 SQL

mysql> /\*!mycat:catlet=demo.catlets.ShareJoin \*/ select a.\*,b.\* from customer a join company b on

a.company\_id=b.id;

+----+------+------------+-------------+-------+

| id | name | company\_id | sharding\_id | name |

+----+------+------------+-------------+-------+

| 1 | wang | 1 | 10000 | mycat |

| 2 | xue | 2 | 10010 | ibm |

| 3 | feng | 3 | 10000 | hp |

+----+------+------------+-------------+-------+

3 rows in set (0.02 sec)

mysql> /\*!mycat:catlet=demo.catlets.ShareJoin \*/ select \* from customer a join company b on

a.company\_id=b.id;

+----+------+------------+-------------+-------+104

| id | name | company\_id | sharding\_id | name |

+----+------+------------+-------------+-------+

| 1 | wang | 1 | 10000 | mycat |

| 2 | xue | 2 | 10010 | ibm |

| 3 | feng | 3 | 10000 | hp |

+----+------+------------+-------------+-------+

3 rows in set (0.02 sec)

/\*!mycat:catlet=demo.catlets.ShareJoin \*/ select a.id,a.user\_id,a.traveldate,a.fee,a.days,b.id as nnid, b.title as

tit from travelrecord a join hotnews b on b.id=a.days order by a.id ;

## ※catlet（人工智能）※

解决跨分片的 SQL JOIN 的问题，远比想象的复杂，而且往往无法实现高效的处理，既然如此，就依靠人工的智力，去编程解决业务系统中特定几个必须跨分片的 SQL 的 JOIN 逻辑，MyCAT 提供特定的 API 供程序员调用，这就是 MyCAT 创新性的思路——人工智能。以一个跨节点的 SQL 为例。

Select a.id,a.name,b.title from a,b where a.id=b.id

其中 a 在分片 1，2，3 上，b 在 4，5，6 上，需要把数据全部拉到本地（MyCAT 服务器）,执行 JOIN 逻辑，具体过程如下（只是一种可能的执行逻辑）：

EngineCtx ctx=new EngineCtx();//包含 MyCat.SQLEngine

String sql=,“select a.id ,a.name from a ” ;

//在 a 表所在的所有分片上顺序执行下面的本地 SQL

ctx.executeNativeSQLSequnceJob(allAnodes,new DirectDBJoinHandler());

DirectDBJoinHandler 类是一个回调类，负责处理 SQL 执行过程中返回的数据包，这里的这个类，主要目的

是用 a 表返回的 ID 信息，去 b 表上查询对于的记录，做实时的关联：

DirectDBJoinHandler{

Private HashMap<byte[],byte[]> rows;//Key 为 id,value 为一行记录的 Column 原始 Byte 数组，这里是

a.id,a.name,b.title 这三个要输出的字段105

Public Boolean onHeader(byte[] header)

{

//保存 Header 信息，用于从 Row 中获取 Field 字段值

}

Public Boolean onRowData(byte[] rowData)

{

String id=getColumnAsString(“id” );

//放入结果集,b.title 字段未知，所以先空着

rows.put(getColumnRawBytes(“id” ),rowData);

//满 1000 条，发送一个查询请求

String sql=” select b.id, b.name from b where id in (………….)” ;

//此 SQL 在 B 的所有节点上并发执行，返回的结果直接输出到客户端

ctx.executeNativeSQLParallJob(allBNodes,sql ,new MyRowOutPutDataHandler(rows));

}

Public Boolean onRowFinished()

{

}

Public void onJobFinished()

{

If(ctx.allJobFinished())

{///used total time ….

}

} }

最后，增加一个 Job 事件监听器，这里是所有 Job 完成后，往客户端发送 RowEnd 包，结束整个流程。

ctx.setJobEventListener(new JobEventHandler(){public void onJobFinished(){ client.writeRowEndPackage()}});

以上提供一个 SQL 执行框架，完全是异步的模式执行，并且以后会提供更多高质量的 API，简化分布式数据

处理，比如内存结合文件的数据 JOIN 算

# Mysql与Mycat插入性能测试

## 基于Mysql的JDBC测试

### 多线程插入性能测试

==============================================JDBC测试10.10.11.218=================

java -jar SQLTEST.jar 参数说明

============================================================插入性能===========================================

[0] -INTEGER 线程个数

[1] -STRING 操作类型 参数请输query|insert|queryandinsert|insertandquery

[2] -INTEGER 每次执行的记录数

[3] -STRING 是否限制查询记录数 unlimit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 20 insert 5000

线程20执行完成，耗时：1227ms,约1s

线程19执行完成，耗时：1946ms,约1s

线程18执行完成，耗时：2745ms,约2s

线程17执行完成，耗时：3628ms,约3s

线程16执行完成，耗时：4353ms,约4s

线程15执行完成，耗时：5827ms,约5s

线程14执行完成，耗时：6588ms,约6s

线程13执行完成，耗时：7337ms,约7s

线程11执行完成，耗时：8161ms,约8s

线程12执行完成，耗时：8908ms,约8s

线程9执行完成，耗时：9829ms,约9s

线程7执行完成，耗时：10640ms,约10

线程6执行完成，耗时：13758ms,约13

线程5执行完成，耗时：15526ms,约15

线程4执行完成，耗时：16393ms,约16

线程3执行完成，耗时：16991ms,约16

线程2执行完成，耗时：17891ms,约17

线程8执行完成，耗时：18898ms,约18

线程10执行完成，耗时：19728ms,约1

线程1执行完成，耗时：20692ms,约20

线程运行完成，共计耗时：20699ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 10 insert 10000

线程10执行完成，耗时：1767ms,约1s

线程9执行完成，耗时：3326ms,约3s

线程7执行完成，耗时：4773ms,约4s

线程8执行完成，耗时：6540ms,约6s

线程6执行完成，耗时：7962ms,约7s

线程5执行完成，耗时：9634ms,约9s

线程4执行完成，耗时：11377ms,约11s

线程3执行完成，耗时：12514ms,约12s

线程2执行完成，耗时：14116ms,约14s

线程1执行完成，耗时：15692ms,约15s

线程运行完成，共计耗时：15694ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 5 insert 20000

线程5执行完成，耗时：2930ms,约2s

线程1执行完成，耗时：4906ms,约4s

线程4执行完成，耗时：6916ms,约6s

线程3执行完成，耗时：9016ms,约9s

线程2执行完成，耗时：10661ms,约10s

线程运行完成，共计耗时：10666ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 2 insert 50000

线程2执行完成，耗时：6022ms,约6s

线程1执行完成，耗时：10487ms,约10

线程运行完成，共计耗时：10490ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 insert 50000

线程1执行完成，耗时：15230ms,约15s

线程运行完成，共计耗时：15235ms

### 多线程读取性能测试

============================================================读取性能===========================================

[0] -INTEGER 线程个数

[1] -STRING 操作类型 参数请输query|insert|queryandinsert|insertandquery

[2] -INTEGER 每次执行的记录数

[3] -STRING 是否限制查询记录数 unlimit

SELECT \* FROM mycat\_hsae\_db.t\_location t LEFT JOIN data\_number.t\_vehicle v ON t.F\_VEHICLE\_ID=v.F\_ID

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 20 query 5000

数据库已连接:true

线程1执行完成，耗时：193ms,约0s

线程18执行完成，耗时：338ms,约0s

线程14执行完成，耗时：502ms,约0s

线程10执行完成，耗时：672ms,约0s

线程6执行完成，耗时：847ms,约0s

线程19执行完成，耗时：1004ms,约1s

线程17执行完成，耗时：1163ms,约1s

线程13执行完成，耗时：1323ms,约1s

线程11执行完成，耗时：1487ms,约1s

线程9执行完成，耗时：1651ms,约1s

线程20执行完成，耗时：1818ms,约1s

线程16执行完成，耗时：1977ms,约1s

线程12执行完成，耗时：2137ms,约2s

线程8执行完成，耗时：2318ms,约2s

线程4执行完成，耗时：2482ms,约2s

线程7执行完成，耗时：2650ms,约2s

线程5执行完成，耗时：2810ms,约2s

线程15执行完成，耗时：2975ms,约2s

线程3执行完成，耗时：3133ms,约3s

线程2执行完成，耗时：3297ms,约3s

线程运行完成，共计耗时：3299ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 10 query 10000

线程1执行完成，耗时：345ms,约0s

线程10执行完成，耗时：675ms,约0s

线程6执行完成，耗时：1001ms,约1s

线程9执行完成，耗时：1323ms,约1s

线程5执行完成，耗时：1643ms,约1s

线程8执行完成，耗时：1966ms,约1s

线程7执行完成，耗时：2307ms,约2s

线程4执行完成，耗时：2670ms,约2s

线程3执行完成，耗时：2992ms,约2s

线程2执行完成，耗时：3326ms,约3s

线程运行完成，共计耗时：3328ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 5 query 20000

线程1执行完成，耗时：671ms,约0s

线程5执行完成，耗时：1333ms,约1s

线程4执行完成，耗时：1982ms,约1s

线程3执行完成，耗时：2690ms,约2s

线程2执行完成，耗时：3339ms,约3s

线程运行完成，共计耗时：3342ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 2 query 50000

线程1执行完成，耗时：1638ms,约1s

线程2执行完成，耗时：3258ms,约3s

线程运行完成，共计耗时：3260ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 query 100000

线程2执行完成，耗时：3211ms,约3s

线程运行完成，共计耗时：3218ms

### 单线程读写性能测试

============================================================单线程读写性能（读取不限条数-unlimit）===========================================

============================================================读取性能===========================================

[0] -INTEGER 线程个数

[1] -STRING 操作类型 参数请输query|insert|queryandinsert|insertandquery

[2] -INTEGER 每次执行的记录数

[3] -STRING 是否限制查询记录数 unlimit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 10000 unlimit

线程2执行完成，耗时：1721ms,约2s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 20000 unlimit

线程2执行完成，耗时：3627ms,约4s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 30000 unlimit

线程2执行完成，耗时：5176ms,约5s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 40000 unlimit

线程2执行完成，耗时：6736ms,约7s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 50000 unlimit

线程2执行完成，耗时：9596ms,约10s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 60000 unlimit

线程2执行完成，耗时：11642ms,约12s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 70000 unlimit

线程2执行完成，耗时：15830ms,约16s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 80000 unlimit

线程2执行完成，耗时：21860ms,约22s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 90000 unlimit

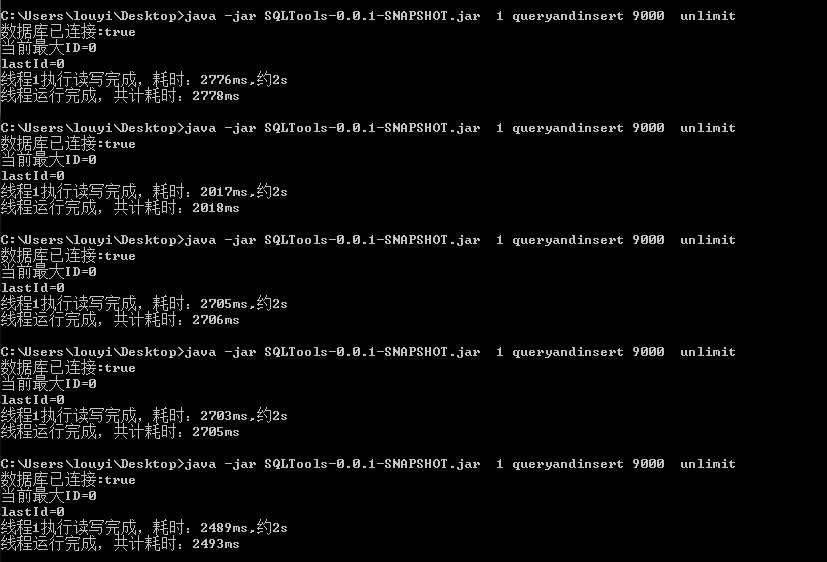
线程2执行完成，耗时：30311ms,约30s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======java -jar SQLTEST.jar 1 queryandinsert 100000 unlimit

线程2执行完成，耗时：37585ms,约38s

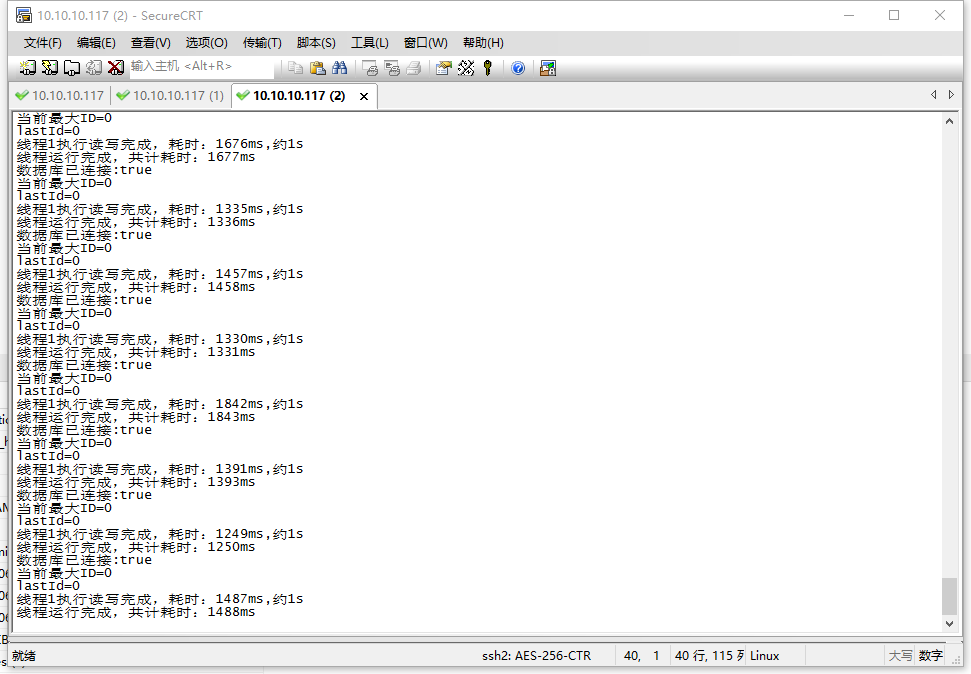
## 基于Mycat的JDBC读写测试

### Windows Cmd单线读写测试



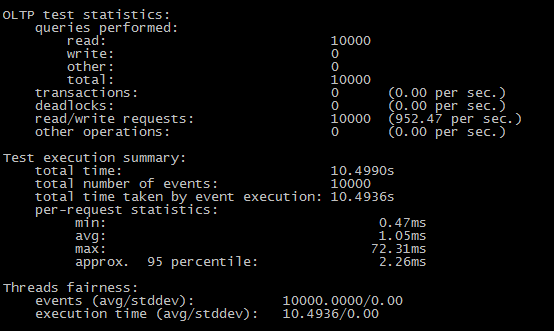
### Linux批处理读写测试

测试数据量每次插入9000

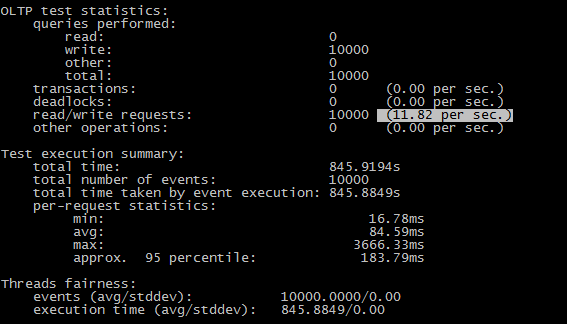


## Sysbench工具的Mycat测试

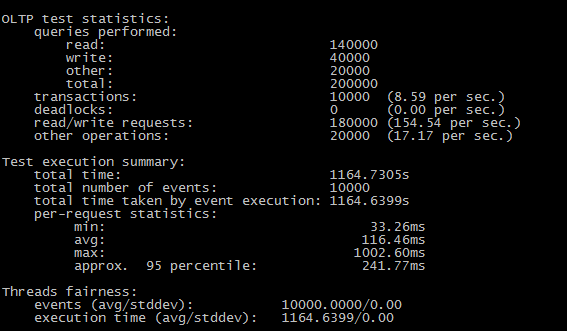
### Mycat查询统计



### Mycat插入统计



### Mycat读写统计



## 测试对比和结论

### 测试数据对比

|  |  |  |
| --- | --- | --- |
| **测试属性** | **MySQL-InnoDB(v)** | **MYCAT(v)** |
| 多线程插入10w | 5000/s左右 | 1000/s左右 |
| 多线程读取10w | 3000/s左右 | 10000/s左右 |
| 单线程读写10w | 5000/s左右 | 4500/s左右(数据量大于400W速度为3000/s左右) |

### 测试结论

1. Mysqld单线程读写性能比较稳定一般5000/s,优于Mycat性能。
2. 多线程读取性能Mycat优于MySQL。
3. 多线程插入Mysql性能优于Mycat。
4. Mycat测试过程中性能不稳定，并且因环境差异性能也不一样。
5. Mycat批量处理大于10000可能产生插入异常（因环境而异，虚拟机测试过程中达到36000）。