开源数据库中间件DBLE对MyCAT的增强与改进

爱可生—阎虎青

TECHNOLOGY
ACTION
爱可生

分布式系统需求

大型应用
（高并发实时交易场景）

电商、金融、O2O、社交应用、零售等，普遍存在用户基数大（百万级或以上）、营销活动频繁、核心交易系统数据库响应不及时问题，制约业务发展。

物联网数据
（大数据量存储访问场景）

在工业监控和远程控制、智慧城市的延展、智能家居、车联网等物联网场景下；传感监控设备多，采样率高，数据规模大。存储一年数据就可以达到PB级甚至EB。

云端服务
（云计算中应用服务场景）

PaaS、SaaS等服务提供商，需要对客户提供一套动态按需索取的并支持大容量的数据库服务，同时对服务质量和性能要求极高。

分布式系统架构示意图

开源分布式中间件d[ou]ble

项目地址:https://github.com/actiontech/dble

TECHNOLOGY ACTION 爱可生

开源分布式中间件d[ou]ble

dble是基于MyCAT的增强改进版的分布式数据库中间件

MyCAT优点：

开源，社区知名度高，出身名门(Ali Cobar)

dble 解决了MyCAT的哪些问题/限制？

- 一、系统性解决复杂查询
- 二、明晰SQL语法支持边界
- 三、增强了语法兼容性
- 四、保障代码质量

系统性解决复杂查询

解决复杂查询需要的系统方法

什么是复杂查询？

单表的
聚合函数
矢量函数
运算表达式

多表之间
JOIN
UNION
SUBQUERY

以上组合

什么是系统性方法？

1.《周髀算经》中记载 “句广三，股修四，径隅五” 。

2.在平面上的一个直角三角形中，两个直角边边长的平方加起来等于斜边长的平方。

解决复杂查询需要的系统方法

MyCAT的“勾股数”1：GLOBAL表

GLOBAL表：
对于数据量不大的字典表（例:超市商品）在多个分片上都有一份同样的副本

相关JOIN语句可以直接下发给各个结点，直接合并结果集就行。

解决复杂查询需要的系统方法(特例GLOBAL表)

JOIN 例子(伪SQL):

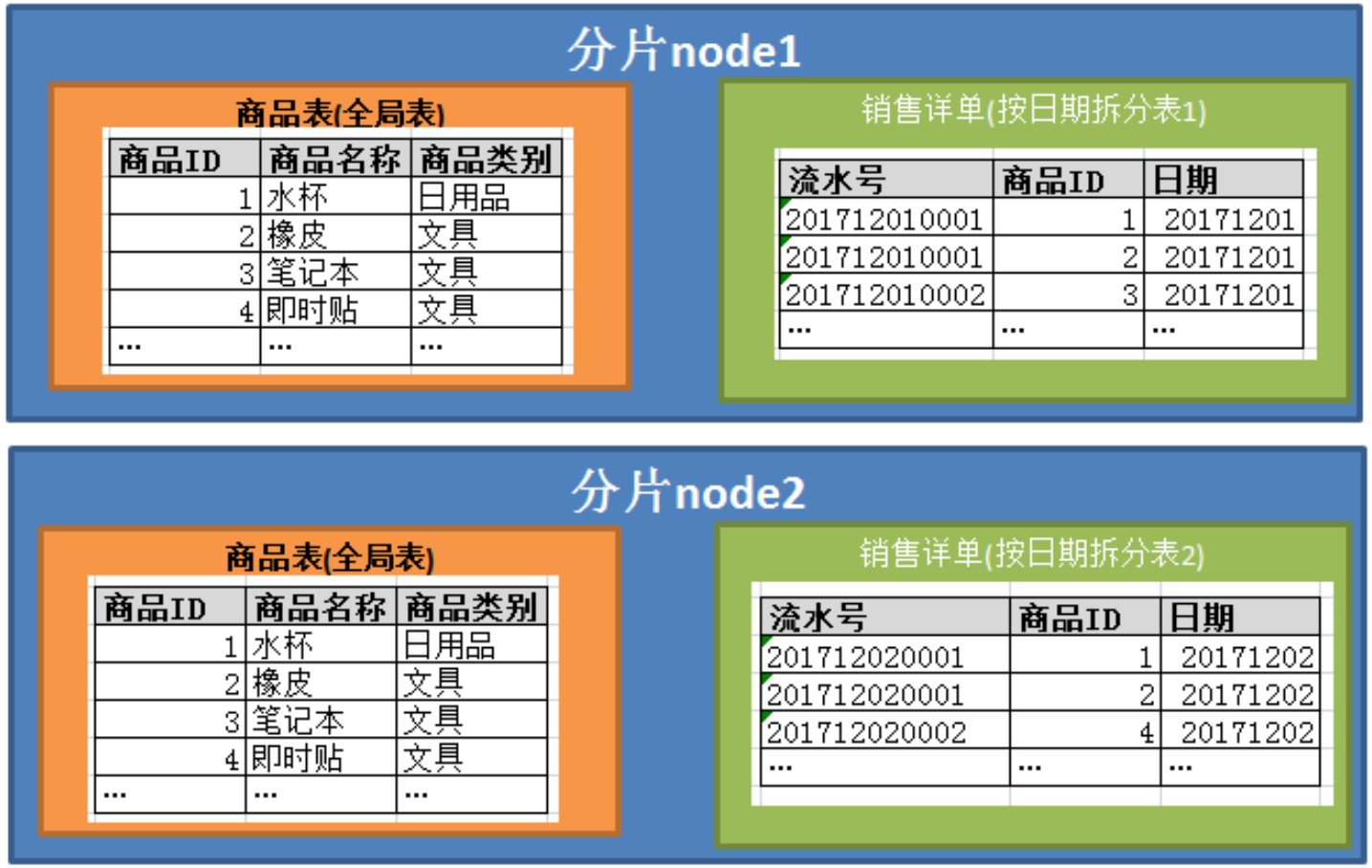
SELECT 日期,商品名,COUNT(*) AS 订单量
FROM 商品表
JOIN 销售详单
USING(商品ID)
WHERE 日期范围(跨结点)
GROUP BY 日期,商品名。

解决复杂查询需要的系统方法(特例GLOBAL表)

如果QUERY是这样呢：

```
SELECT 商品类别
,COUNT(DISTINCT 商品ID) AS 卖出种类
FROM 商品表
JOIN 销售详单
USING(商品ID)
WHERE 日期范围(跨结点)
GROUP BY 商品类别
```

解决复杂查询需要的系统方法(特例GLOBAL表)



商品类别	卖出种类
日用品	1
文具	2

+

商品类别	卖出种类
日用品	1
文具	2

≠

商品类别	卖出种类
日用品	1
文具	3

```
SELECT 商品类别,COUNT(DISTINCT 商品ID) AS 卖出种类
FROM 商品表 JOIN 销售详单 USING(商品ID)
WHERE 日期范围(跨结点) GROUP BY 商品类别
```

TECHNOLOGY ACTION 爱可生

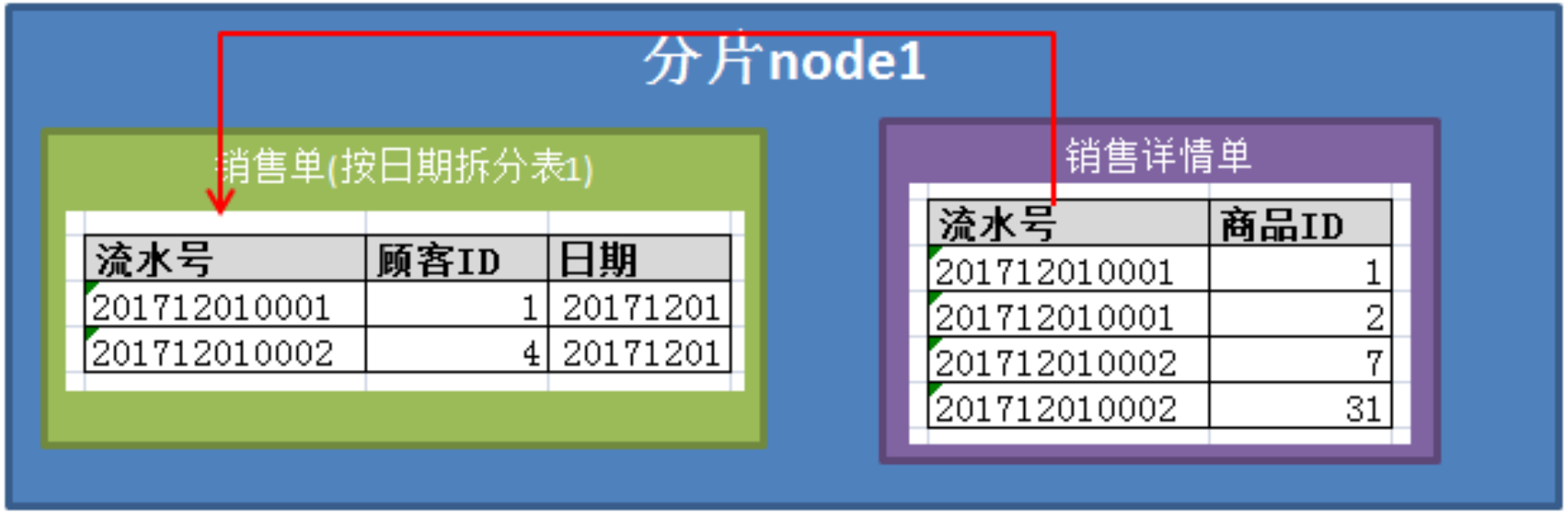
解决复杂查询需要的系统方法

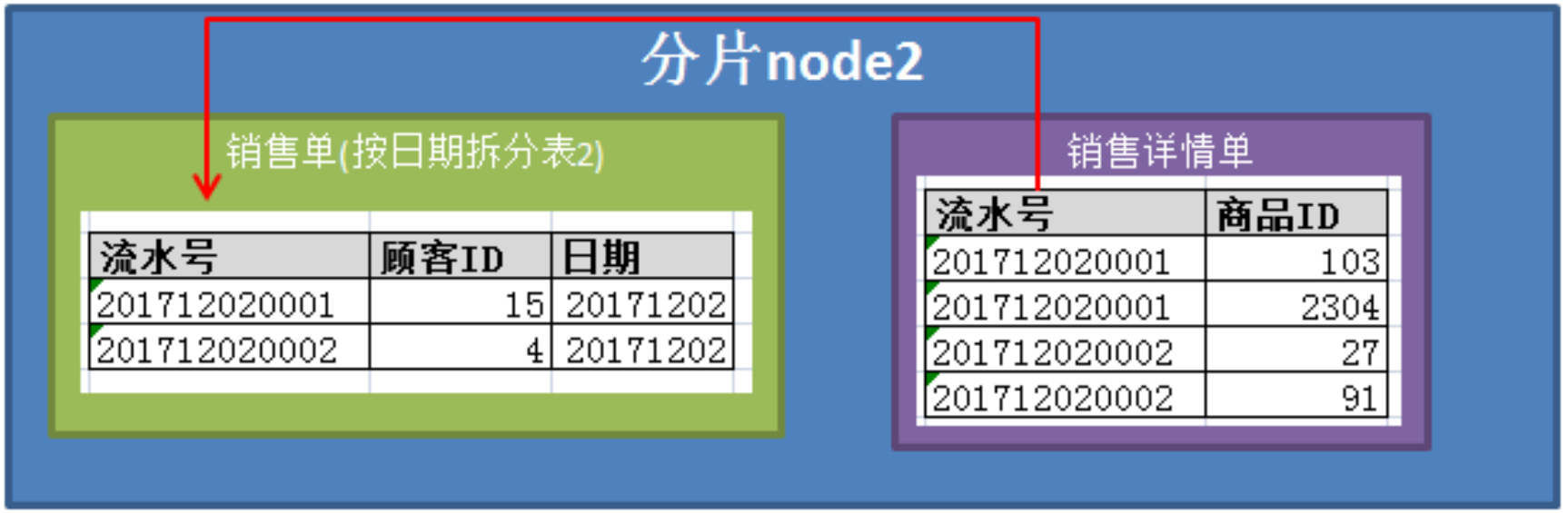
MyCAT的“勾股数”2：ER表

ER表: 对于有外键关系的表，可以根据外键关系拆分

注意，外键列需要依赖于拆分列，不能有1:N的关系

解决复杂查询需要的系统方法(特例ER表)





ER表拆分方法

假设按照销售单表的日期拆分，流水号作为外键

解决复杂查询需要的系统方法(特例ER表)

当ER关系是多个维度时，无法用ER表拆分方法解决所有问题

解决复杂查询需要的系统方法（特例Hint）

MyCAT的“勾股数”3：Hint

```
[redacted] [testdb]>/*!mycat:catlet=io.mycat.catlets.ShareJoin*/
    -> select a.*,b.id from sharding_four_node a
    -> inner join sharding_two_node b
    -> on a.id =b.id;
+----+--------+-----------+----+----+
| id | c_flag | c_decimal | id | id |
+----+--------+-----------+----+----+
|  1 | 1      |    6.0000 |  1 |  1 |
|  2 | 2      |    6.0000 |  2 |  2 |
|  3 | 3      |    6.0000 |  3 |  3 |
+----+--------+-----------+----+----+
3 rows in set (0.02 sec)
```

Hint:对于两个表的跨结点查询使用hint的方式做

其实就是这样的nest loop join

```
select *, id from sharding_four_node
select id, id from sharding_two_node where id in (1,2,3,4)
```

未解决多于两个表的问题
未解决JOIN 条件复杂的问题
增加Hint的兼容性代价

系统解决复杂查询-关系代数

复杂查询实现思路:**关系代数 查询树**

1.解析SQL时候，将SQL转为基本元组，以及对元组进行关系运算，构建查询树 。

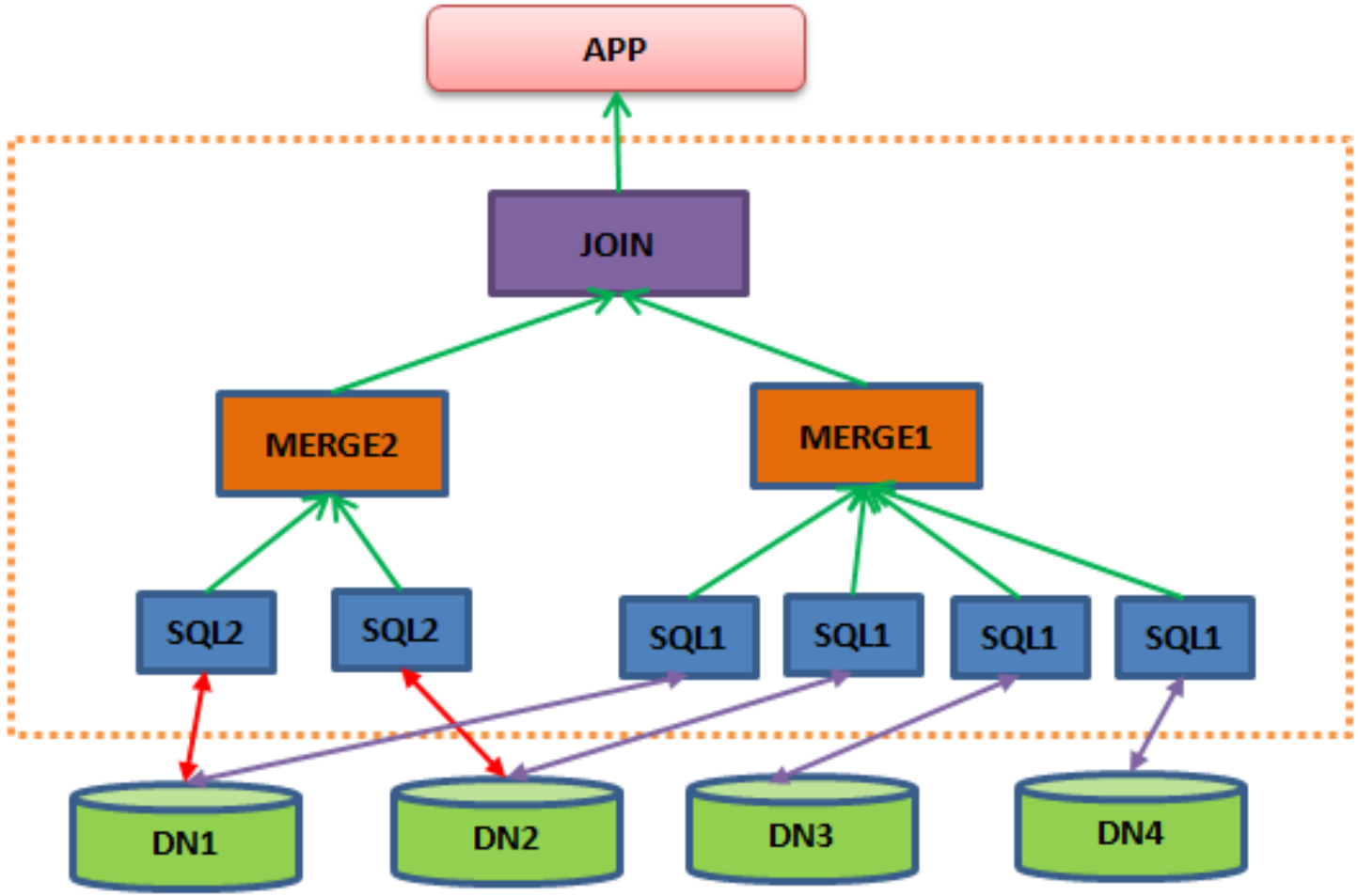
2.将基本元组及可下发的运算作为查询树的叶子节点，下发到物理数据结点查询。

3.结果集返回后，不可下发的运算作为查询树的非叶子节点对子树返回结果处理。

关系代数（部分）

Name	Symbol	对应sql部分
Selection	$\sigma_{a\theta b}(R)$ or $\sigma_{a\theta v}(R)$ θ包含$\leqslant, \geqslant, \neq, <, >, =$	WHERE
Projection	$\Pi_{a_1,\ldots,a_n}(R)$	SELECT
Cartesian product	$A \times B = \{(a,b) \mid a \in A \text{ and } b \in B\}.$	笛卡儿积
set Union	$A \cup B = \{x : x \in A \text{ or } x \in B\}$	UNION
Rename	$\rho_{a/b}(R) = \{x \in B \mid x \notin A\}.$	别名
Natural join	$R \bowtie S = \{r \cup s \mid r \in R \wedge s \in S \wedge Fun(r \cup s)\}$	Natural join
θ-join	$R \bowtie_\theta S = \sigma_\theta (R \times S)$	JOIN
equijoin	θ为"=" 的θ-join	
Semijoin	$R \ltimes S = \pi_{a1,\ldots,an}(R \bowtie S)$	
Left outer join	$R \bowtie S = (R \bowtie S) \cup ((R - \pi_{r_1,r_2,\ldots,r_n}(R \bowtie S)) \times \{(\omega,\ldots\omega)\})$	
Right outer join	$R \bowtie S = (R \bowtie S) \cup (\{(\omega,\ldots,\omega)\} \times (S - \pi_{s_1,s_2,\ldots,s_n}(R \bowtie S)))$	
Full outer join	$R \bowtie S = R \bowtie S = (R \bowtie S) \cup (R \bowtie S)$	
Aggregation	Exp1, Exp2, Exp3...Gfunc1, func2, func3...	聚合

系统解决复杂查询-JOIN



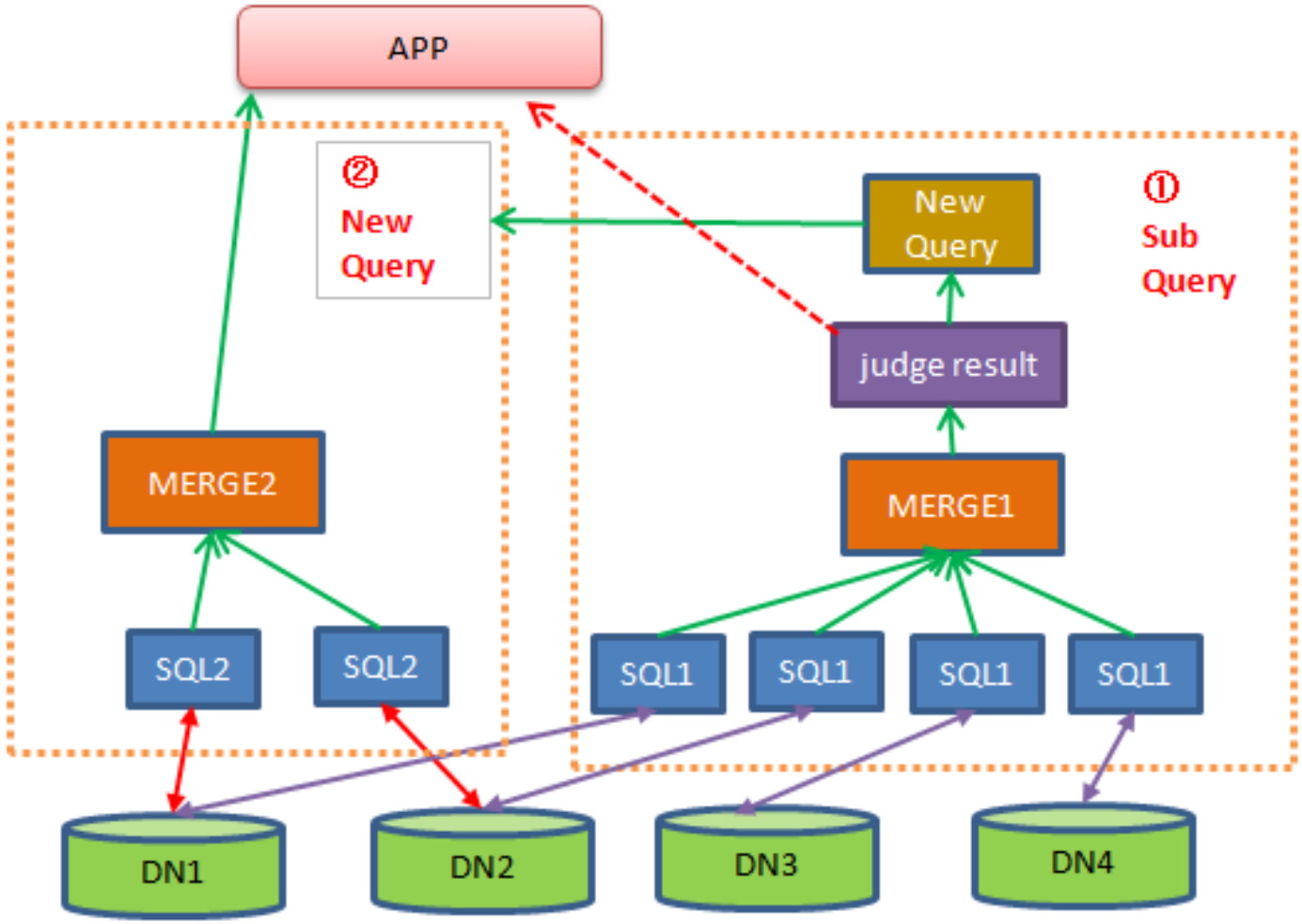
JOIN 举例：
select * from table1 a inner join table2 b on a.id =b.id ;
table1 有四个分片dn1~4
table2 有2个分片 dn1,dn2

查询树：

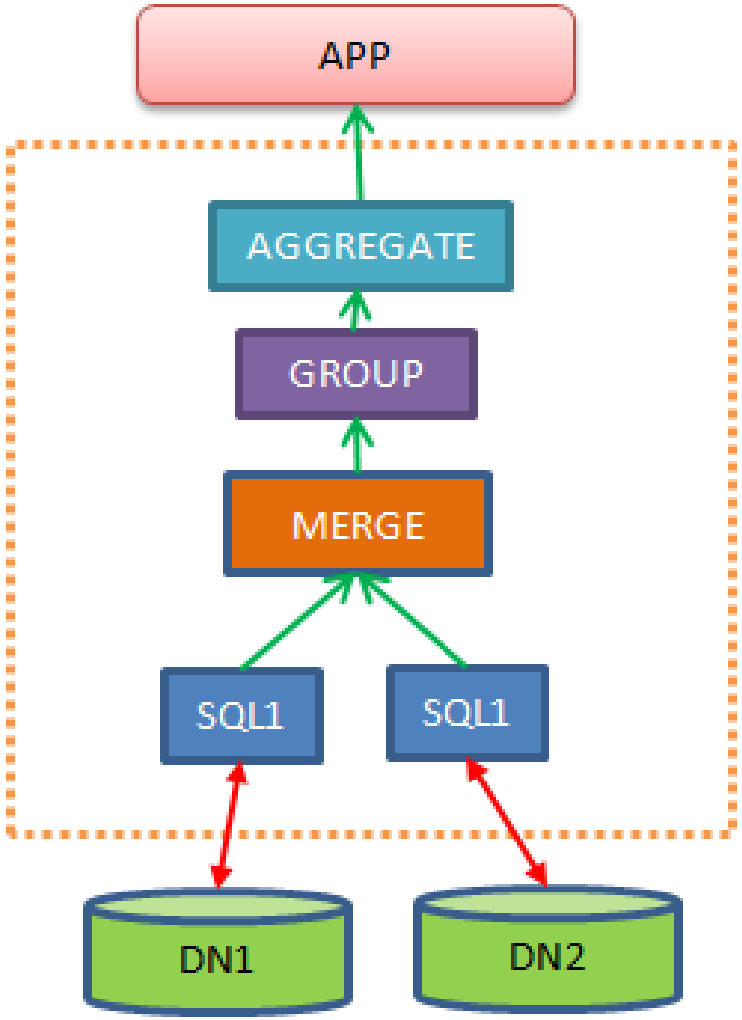
下发语句：
SQL1:select * from table1;
SQL2:select * from table2;

中间件后续操作：
1.各个节点结果合并
2.将结果集按照 a.id =b.id过滤及JOIN

系统解决复杂查询

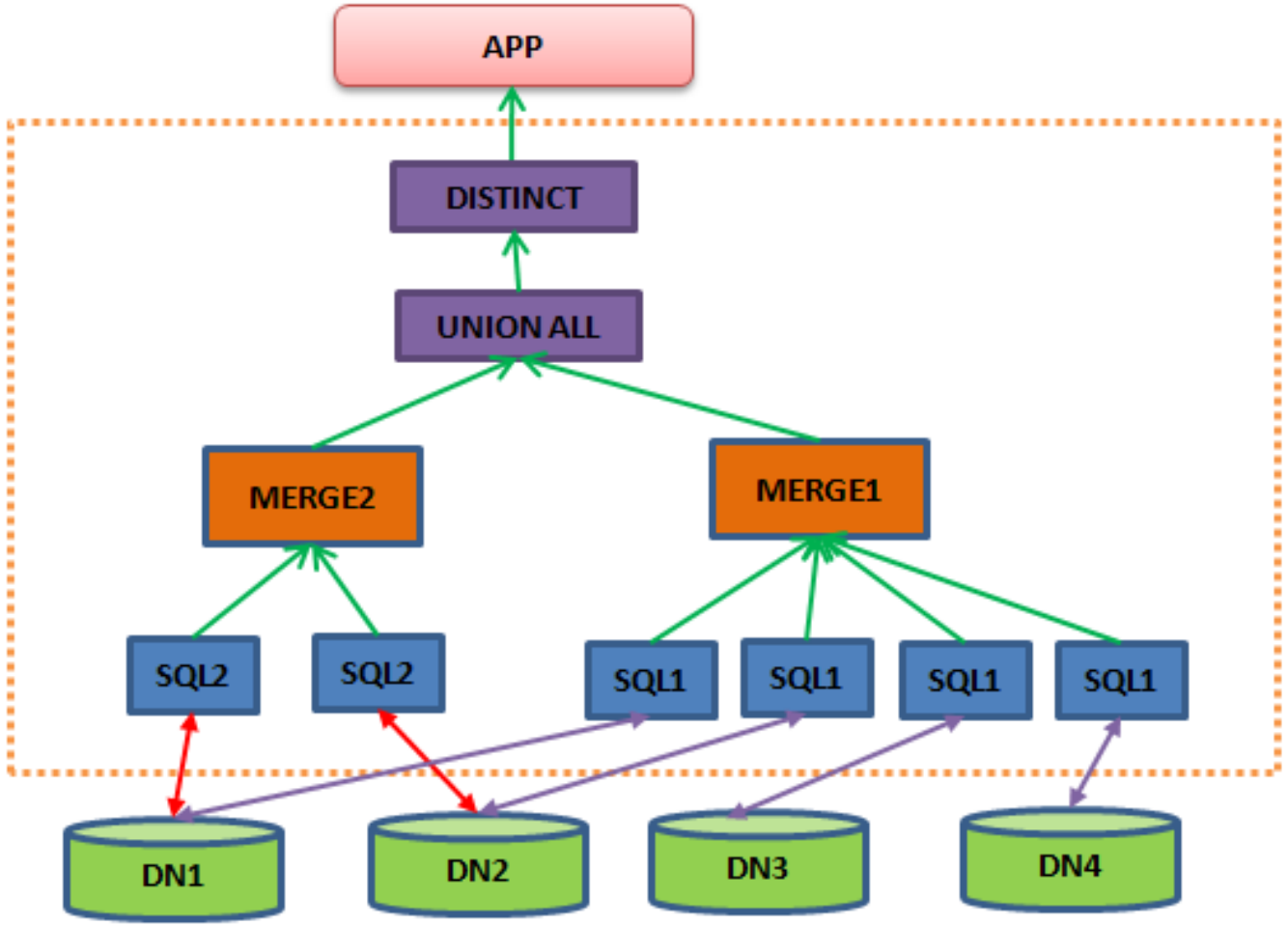


子查询



复杂聚合

系统解决复杂查询



UNION

具体实现细节可以下载本人的另一次分享
《dble-开源MySQL分布式中间件实现剖析》

系统解决复杂查询-案例

```
[redacted] [testdb]>select * from sharding_four_node;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
|  1 | 1      |    6.0000 |
+----+--------+-----------+
4 rows in set (0.01 sec)

[redacted] [testdb]>select * from sharding_two_node;
+-----+--------+---------------------+-----+------+
| id  | c_char | ts                  | si  | id2  |
+-----+--------+---------------------+-----+------+
|   1 | 1      | 2017-12-01 14:57:13 |   1 |    1 |
|   2 | 2      | 2017-11-30 10:50:59 |   2 |    1 |
|   3 | 3      | 2017-12-01 14:57:13 |   3 |    1 |
| 514 | 514    | 2017-12-01 14:57:13 | 514 |    1 |
| 515 | 515    | 2017-12-01 14:57:13 | 515 |    1 |
+-----+--------+---------------------+-----+------+
5 rows in set (0.00 sec)
```

sharding_four_node 按照ID拆分为4个节点。
拆分规则：对4求模。

sharding_two_node 按照ID拆分为2个结点。
拆分规则：对1024求模，0~511分到第一个结点，512~1023拆分到第二个结点。

系统解决复杂查询-案例 union

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_two_node

id	c_char	ts
1	1	2017-12-01
2	2	2017-11-30
3	3	2017-12-01
514	514	2017-12-01
515	515	2017-12-01

select id from sharding_two_node a union
select id from sharding_four_node ;

MyCAT结果：

```
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
|   4 |
| 514 |
| 515 |
|   1 |
+-----+
7 rows in set (0.01 sec)
```

dble结果

```
+-----+
| id  |
+-----+
|   1 |
|   4 |
|   2 |
|   3 |
| 514 |
| 515 |
+-----+
6 rows in set (0.03 sec)
```

系统解决复杂查询-案例 union all

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_two_node

id	c_char	ts
1	1	2017-12-01
2	2	2017-11-30
3	3	2017-12-01
514	514	2017-12-01
515	515	2017-12-01

```
select id from sharding_two_node a union all
select id from sharding_four_node ;
```

MyCAT结果

id
1
2
3
4
514
515
1

7 rows in set (0.00 sec)

dble结果

id
514
515
1
2
3
4
2
1
3

9 rows in set (0.01 sec)

系统解决复杂查询-案例 复杂表达式

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_two_node

id	c_char	ts
1	1	2017-12-01
2	2	2017-11-30
3	3	2017-12-01
514	514	2017-12-01
515	515	2017-12-01

select abs(sum(c_decimal))
from sharding_four_node

abs(SUM(c_decimal))
6.0000
6.0000
6.0000
6.0000

4 rows in set (0.02 sec)

MyCAT结果 

abs(SUM(c_decimal))
24.0000

1 row in set (0.12 sec)

dble结果 

系统解决复杂查询-案例 子查询

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_two_node

id	c_char	ts
1	1	2017-12-01
2	2	2017-11-30
3	3	2017-12-01
514	514	2017-12-01
515	515	2017-12-01

select id from sharding_two_node where id =(select min(id) from sharding_four_node)

```
Empty set (0.05 sec)
```

MyCAT1.6.1结果 

MyCAT1.6.5结果：
连接断开，有时会hang住

```
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.10 sec)
```

dble结果 

系统解决复杂查询-案例 JOIN

表格数据

sharding_four_node

```
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
|  1 | 1      |    6.0000 |
+----+--------+-----------+
```

sharding_two_node

```
+-----+--------+-------------
| id  | c_char | ts
+-----+--------+-------------
|   1 | 1      | 2017-12-01 1
|   2 | 2      | 2017-11-30 1
|   3 | 3      | 2017-12-01 1
| 514 | 514    | 2017-12-01 1
| 515 | 515    | 2017-12-01 1
+-----+--------+-------------
```

```
select a.id,b.* from
sharding_two_node a inner join
sharding_four_node b on a.id =b.id;
```

```
Empty set (0.05 sec)
```

MyCAT1.6.1结果 

MyCAT1.6.5结果： 略 

```
+----+----+--------+-----------+
| id | id | c_flag | c_decimal |
+----+----+--------+-----------+
|  1 |  1 | 1      |    6.0000 |
|  2 |  2 | 2      |    6.0000 |
|  3 |  3 | 3      |    6.0000 |
+----+----+--------+-----------+
3 rows in set (0.02 sec)
```

dble结果 

系统解决复杂查询-案例 JOIN改动

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_two_node

id	c_char	ts
1	1	2017-12-01
2	2	2017-11-30
3	3	2017-12-01
514	514	2017-12-01
515	515	2017-12-01

select a.id,b.* from sharding_two_node a inner join sharding_four_node b on a.id =b.id+1;

```
Empty set (0.05 sec)
```

MyCAT1.6.5结果：

原来1.6.5只是把Hint固化到代码里，判断两表拆分规则不同就直接用hint逻辑

id	id	c_flag	c_decimal
2	1	1	6.0000
3	2	2	6.0000

2 rows in set (0.01 sec)

dble结果

系统解决复杂查询-案例 JOIN for1.6.5

```
<table name="sharding_four_node" primaryKey="id" dataNode="dn1,dn2,dn3,dn4" rule="rule1plus" />
<table name="sharding_four_node2" primaryKey="id" dataNode="dn4,dn3,dn2,dn1" rule="rule1plus" />
```

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_four_node2

id	c_flag	c_decimal
2	2	2.0000
3	3	3.0000
1	1	1.0000
4	4	4.0000

sharding_four_node
sharding_four_node2
均按照ID拆分为4个节点。
拆分规则：对4求模。

唯一不同在于分布结点的顺序不同

系统解决复杂查询-JOIN for1.6.5

表格数据

sharding_four_node

id	c_flag	c_decimal
2	2	6.0000
3	3	6.0000
4	4	6.0000
1	1	6.0000

sharding_four_node2

id	c_flag	c_decimal
2	2	2.0000
3	3	3.0000
1	1	1.0000
4	4	4.0000

select * from sharding_four_node2 a inner join sharding_four_node b on a.id =b.id;

```
Empty set (0.01 sec)
```

1.6.5版判断拆分规则不同就使用hint逻辑.
这里规则相同，就回到了不支持跨结点查询的老路子上

id	c_flag	c_decimal	id	c_flag	c_decimal
1	1	1.0000	1	1	6.0000
2	2	2.0000	2	2	6.0000
3	3	3.0000	3	3	6.0000
4	4	4.0000	4	4	6.0000

4 rows in set (0.81 sec)

dble结果

系统解决复杂查询-复杂查询的查询计划

explain select * from sharding_two_node a
inner join sharding_four_node b on a.id =b.id;

```
+------------+----------+------------------------------------------------------------------------------------------+
| DATA_NODE  | TYPE     | SQL/REF                                                                                  |
+------------+----------+------------------------------------------------------------------------------------------+
| dn1.0      | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn2.0      | BASE SQL | select `a`.`id`,`a`.`c_char`,`a`.`ts`,`a`.`si` from  `sharding_two_node` `a` ORDER BY `a`.`id` ASC |
| dn1.1      | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn2.1      | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn3.0      | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| dn4.0      | BASE SQL | select `b`.`id`,`b`.`c_flag`,`b`.`c_decimal` from  `sharding_four_node` `b` ORDER BY `b`.`id` ASC  |
| merge.1    | MERGE    | dn1.0, dn2.0                                                                             |
| merge.2    | MERGE    | dn1.1, dn2.1, dn3.0, dn4.0                                                               |
| join.1     | JOIN     | merge.1, merge.2                                                                         |
+------------+----------+------------------------------------------------------------------------------------------+
9 rows in set (0.00 sec)
```

明晰语法支持边界

明晰SQL支持边界（MYCAT SET 语句)

```
[testdb]>select * from sharding_four_node order by id;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1      |    6.0000 |
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
+----+--------+-----------+
4 rows in set (0.03 sec)

[testdb]>insert into sharding_four_node(id,c_flag,c_decimal) values(5,'5',5.0);
Query OK, 1 row affected (0.05 sec)

[testdb]>set @@session.tx_read_only =1;
Query OK, 0 rows affected (0.00 sec)

[testdb]>insert into sharding_four_node(id,c_flag,c_decimal) values(6,'6',6.0);
Query OK, 1 row affected (0.12 sec)

[testdb]>select * from sharding_four_node order by id;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1      |    6.0000 |
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
|  5 | 5      |    5.0000 |
|  6 | 6      |    6.0000 |
+----+--------+-----------+
6 rows in set (0.01 sec)
```

MyCAT的 set@@session.tx_read_only

明晰SQL支持边界（ MYCAT SET 语句)

```
[testdb]>select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)

[testdb]>set @@session.tx_isolation = 'SERIALIZABLE';
Query OK, 0 rows affected (0.00 sec)

[testdb]>select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)
```

甚至

```
[testdb]>set you =me;
Query OK, 0 rows affected (0.00 sec)
```

明晰SQL支持边界（ MYCAT SET 语句)

Set语句为什么会出现这种情况呢，我们看一下源码。

```
157         default:
158             boolean ignore = SetIgnoreUtil.isIgnoreStmt(stmt);
159             if ( !ignore ) {
160                 StringBuilder s = new StringBuilder();
161                 logger.warn(s.append(c).append(stmt).append(" is not recoginized and ignored").toString());
162             }
163             c.write(c.writeToBuffer(OkPacket.OK, c.allocate()));
```

MYCAT 在枚举了一些set选项之后，其余set一律返回OK

明晰SQL支持边界（DBLE SET 语句)

```
[testdb]>select * from sharding_four_node order by id;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1      |    6.0000 |
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
+----+--------+-----------+
4 rows in set (0.01 sec)

[testdb]>insert into sharding_four_node(id,c_flag,c_decimal) values(5,'5',5.0);
Query OK, 1 row affected (0.02 sec)

[testdb]>set @@session.tx_read_only =1;
Query OK, 0 rows affected (0.00 sec)

[testdb]>insert into sharding_four_node(id,c_flag,c_decimal) values(6,'6',6.0);
ERROR 1792 (HY000): Cannot execute statement in a READ ONLY transaction.
[testdb]>select * from sharding_four_node order by id;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  1 | 1      |    6.0000 |
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
|  4 | 4      |    6.0000 |
|  5 | 5      |    5.0000 |
+----+--------+-----------+
5 rows in set (0.01 sec)
```

dble的 set@@session.tx_read_only

明晰SQL支持边界（DBLE SET 语句)

```
[testdb]>select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.01 sec)

[testdb]>set @@session.tx_isolation ='SERIALIZABLE';
Query OK, 0 rows affected (0.00 sec)

[testdb]>select @@tx_isolation;
+----------------+
| @@tx_isolation |
+----------------+
| SERIALIZABLE   |
+----------------+
1 row in set (0.00 sec)


[testdb]>set you = me;
ERROR 3010 (HY000): system variable you is not supported
```

明晰SQL支持边界（不支持语句明确提示)

例1.dble不支持设置全局只读的功能，会有明确报错

```
[testdb]>set @@global.tx_read_only =1;
ERROR 3010 (HY000): setting GLOBAL value is not supported
```

例2.根据MySQL官方文档，没有SESSION 或者GLOBAL关键字，事务变量只对下一条事务生效

- Without any `SESSION` or `GLOBAL` keyword, the statement applies to the next (not started) transaction performed within the current session. Subsequent transactions revert to using the `SESSION` isolation level.

dble还没实现此功能，就会明确报不支持

```
[testdb]>set @@tx_read_only =1;
ERROR 3010 (HY000): setting transaction without any SESSION or GLOBAL keyword is not supported now
```

增强语法兼容性

语法兼容性

例1: MyCAT的insert强制必须写完整列名，否则报错：

```
[testdb]>insert into sharding_four_node values(5,'5',5.0);
ERROR 1064 (HY000): partition table, insert must provide ColumnList
```

dble语法兼容性更强：

```
[testdb]>insert into sharding_four_node values(5,'5',5.0);
Query OK, 1 row affected (0.20 sec)
```

例2. MyCAT的全局序列自定义了语法

```
[testdb]>insert into sharding_four_node_autoinc(id,c_char) values(next value for MYCATSEQ_GLOBAL,'1');
fected (0.33 sec)
```

dble和普通的mysql自增长列语法相同：

```
[testdb]>insert into sharding_four_node_autoinc(c_char) values('1');
Query OK, 1 row affected (0.22 sec)
```

语法兼容性

例3: MyCAT的insert拼写错误

```
[testdb]>inset into  sharding_four_node(id,c_flag,c_decimal) values(5,'5',5.0);
Query OK, 0 rows affected (0.00 sec)

[testdb]>select * from sharding_four_node ;
+----+--------+-----------+
| id | c_flag | c_decimal |
+----+--------+-----------+
|  4 | 4      |    6.0000 |
|  1 | 1      |    6.0000 |
|  2 | 2      |    6.0000 |
|  3 | 3      |    6.0000 |
+----+--------+-----------+
4 rows in set (0.01 sec)
```

dble会提示语法错误

```
[testdb]>inset into sharding_four_node (id,c_flag,c_decimal) values(5,'5',5.0);
: syntax error, error in :'inset into sharding_four_node ',expect IDENTIFIER, actual IDENTIFIER inset
```

保障软件质量

保证软件质量

MyCAT开源以来, 以下几个方面影响了代码的质量

- 没有开发团队，无人重构代码，旧代码残留
- 没有测试团队，无法及时发现BUG
- 没有良好的review机制，贡献者代码质量良莠不齐
- 没有运维使用团队，仅从开发者角度，很难发现设计缺陷

MyCAT启动类的残留代码

```
350         int bufferPoolType = system.getProcessorBufferPoolType();
351
352         switch (bufferPoolType){
353             case 0:
354                 bufferPool = new DirectByteBufferPool(bufferPoolPageSize,bufferPoolChunkSize,
355                     bufferPoolPageNumber,system.getFrontSocketSoRcvbuf());
356
357
358                 totalNetWorkBufferSize = bufferPoolPageSize*bufferPoolPageNumber;
359                 break;
360             case 1:
361                 /**
362                  * todo 对应权威指南修改：
363                  *
364                  * bytebufferarena由6个bytebufferlist组成，这六个list有减少内存碎片的机制
365                  * 每个bytebufferlist由多个bytebufferchunk组成，每个list也有减少内存碎片的机制
366                  * 每个bytebufferchunk由多个page组成，平衡二叉树管理内存使用状态，计算灵活
367                  * 设置的pagesize对应bytebufferarena里面的每个bytebufferlist的每个bytebufferchunk的buffer长度
368                  * bufferPoolChunkSize对应每个bytebufferchunk的每个page的长度
369                  * bufferPoolPageNumber对应每个bytebufferlist有多少个bytebufferchunk
370                  */
371
372                 totalNetWorkBufferSize = 6*bufferPoolPageSize * bufferPoolPageNumber;
373                 break;
```

当把内存类型参数设为1时，

对应代码分支没有初始化内存缓冲池

MyCAT启动类的残留代码

现象也很简单，启动后客户端连不上。
日志里全是NP异常

```
2017-12-02 15:49:51.385  WARN [$_MyCatServer] (io.mycat.net.NIOAcceptor.accept(NIOAcceptor.java:143)) - $_MyCatServer
java.lang.NullPointerException
        at io.mycat.net.AbstractConnection.setProcessor(AbstractConnection.java:213) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.FrontendConnection.setProcessor(FrontendConnection.java:130) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.NIOAcceptor.accept(NIOAcceptor.java:137) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.NIOAcceptor.run(NIOAcceptor.java:102) ~[Mycat-server-1.6.5-release.jar:?]
2017-12-02 15:49:55.602  WARN [$_MyCatServer] (io.mycat.net.NIOAcceptor.accept(NIOAcceptor.java:143)) - $_MyCatServer
java.lang.NullPointerException
        at io.mycat.net.AbstractConnection.setProcessor(AbstractConnection.java:213) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.FrontendConnection.setProcessor(FrontendConnection.java:130) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.NIOAcceptor.accept(NIOAcceptor.java:137) ~[Mycat-server-1.6.5-release.jar:?]
        at io.mycat.net.NIOAcceptor.run(NIOAcceptor.java:102) ~[Mycat-server-1.6.5-release.jar:?]
```

这个bug一定程度反映了代码质量。
如果认真阅读过启动类代码，能够发现这里没有初始化的。

无人发现的内存double free Bug

重现步骤：设置参数为使用堆外内存处理合并结果集，用sysbench单线程压，SQL语句为一个跨结点的带聚合函数的查询，几十秒内必然出现中间件进程崩溃，并报出double free 的错误。而中间件守护进程会重新起一个进程，如果持续有压力，仍然会复现。

从dble的issue list可以看出，这个编号为4 的bug在我司测试团队那里是很早就发现的bug。

在修复过程中我们发现，这是一个中间件内部的线程安全的问题，不做压力测试很难发现。

贡献者代码没有严格review

mycat版本1.5和1.6 旧内存管理模式下查询多个AVG列抛`java.lang.IndexOutOfBoundsException`

BUG重现:

```
mysql> select avg(id),avg(id) from product;
ERROR 1105 (HY000): java.lang.IndexOutOfBoundsException: Index: 3, Size: 3
```

mycat后台异常堆栈:

```
caught exception  java.lang.IndexOutOfBoundsException: Index: 3, Size: 3
        at java.util.ArrayList.rangeCheck(ArrayList.java:653)
        at java.util.ArrayList.get(ArrayList.java:429)
        at io.mycat.sqlengine.mpp.RowDataPacketGrouper.mergAvg(RowDataPacketGrouper.java:240)
        at io.mycat.sqlengine.mpp.RowDataPacketGrouper.getResult(RowDataPacketGrouper.java:68
        at io.mycat.sqlengine.mpp.DataMergeService.getResults(DataMergeService.java:272)
        at io.mycat.sqlengine.mpp.DataMergeService.run(DataMergeService.java:228)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
```

定位到`RowDataPacketGrouper`类的`mergAvg`方法，发现处理逻辑有问题:

在处理avg结果的for循环里面删除了数组元素，导致下一个avg的处理抛数组下表越界的Exception。

Bug #1194截图

贡献者代码没有严格review

```
Set<Integer> rmIndexSet = new HashSet<Integer>();
for (MergeCol merg : mergCols) {
        if(merg.mergeType==MergeCol.MERGE_AVG)
        {
,16 @@ private void mergAvg(RowDataPacket toRow) {
                if (result != null)
                {
                        toRow.fieldValues.set(merg.colMeta.avgSumIndex, result);
                        toRow.fieldValues.remove(merg.colMeta.avgCountIndex) ;
                        toRow.fieldCount=toRow.fieldCount-1;
                        toRow.fieldValues.remove(merg.colMeta.avgCountIndex) ;
                        toRow.fieldCount=toRow.fieldCount-1;
                        rmIndexSet.add(merg.colMeta.avgCountIndex);
                }
        }
}

for(Integer index : rmIndexSet) {
        toRow.fieldValues.remove(index);
        toRow.fieldCount = toRow.fieldCount - 1;
}
```

Bug #1194 的修复：

结果正确，逻辑奇怪

1.仍然是在循环中尝试按照索引移除ArrayList中某些值。

2.使用了自动装箱导致remove 方法不起任何作用

3.改变大小是有用的。

开发者容易忽视的设计缺陷（需要真正的使用者）

例1.
任何一个数据库用户都可以登陆管理端，登陆上去后能够执行很多高权限的命令，比如服务下线，修改配置之类的操作。

例2.
时间戳全局序列的实现方式是这样的
64 位二进制 (42(毫秒)+5(机器ID)+5(业务编码)+12(毫秒内重复累加)
在低并发（1000qps以下）的情况下，1毫秒内只有一条数据，则这条数据后12位是固定值0，也就是说生成的值一定是4096的倍数 。

如果拆分算法是hash拆分，将自增列的值做了模1024的运算。
则得到的结果一定是0，则拆分的数据将发生严重倾斜，
数据会全部集中到第一个结点上。

dble的自动化工具的引入

静态代码分析工具

可持续集成

自动化测试

其他重要bug举例

序号	Bug描述	状态
1	全局序列批量写入值会重复	已修复
2	不支持alter修改全局表结构	已修复
3	ER表事务中不支持父子表同时插入数据	已修复
4	useOffHeapForMerge 和useStreamOutput开启导致NP异常	已修复
5	固定大小的线程池在含有子任务的压力下可能死锁	持续修复中
6	针对between A and B语法，hash拆分算法计算出来的范围有误	已修复
7	global表对update where in语句改写错误	已修复
8	查询返回报文中的meta信息不正确	已修复，测试中
...		

dble 功能特性

项目地址:https://github.com/actiontech/dble

TECHNOLOGY ACTION 爱可生

QQ Group: 669663113

微信交流群

GitHub

https://github.com/actiontech/dble

TECHNOLOGY
ACTION
爱可生