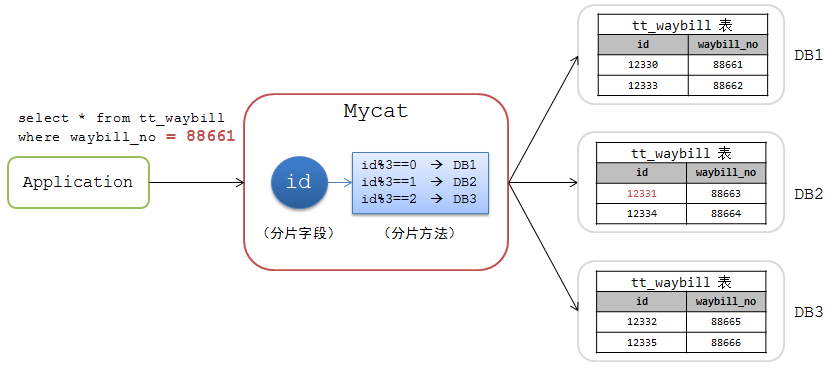
**Mycat从入门到放弃**

参考：<http://blog.csdn.net/u013235478/article/details/53178657>

## 1.非分片字段查询

Mycat中的路由结果是通过**分片字段**和**分片方法**来确定的。例如下图中的一个Mycat分库方案：

* 根据 **tt\_waybill** 表的 **id** 字段来进行分片
* 分片方法为 **id** 值取 **3** 的模，根据模值确定在DB1，DB2，DB3中的某个分片



如果查询条件中有 **id** 字段的情况还好，查询将会落到某个具体的分片。例如：

mysql>select \* from tt\_waybill where **id** = 12330;

此时Mycat会计算路由结果

12330 % 3 = 0 –> DB1

并将该请求路由到DB1上去执行。

如果查询条件中没有 **分片字段** 条件，例如：

mysql>select \* from tt\_waybill where waybill\_no =88661;

此时Mycat无法计算路由，便发送到所有节点上执行：

DB1 –> select \* from tt\_waybill where waybill\_no =88661;   
DB2 –> select \* from tt\_waybill where waybill\_no =88661;   
DB3 –> select \* from tt\_waybill where waybill\_no =88661;

如果该分片字段选择度高，也是业务常用的查询维度，一般只有一个或极少数个DB节点命中（返回结果集）。示例中只有3个DB节点，而实际应用中的DB节点数远超过这个，假如有50个，那么前端的一个查询，落到MySQL数据库上则变成50个查询，会极大消耗Mycat和MySQL数据库资源。

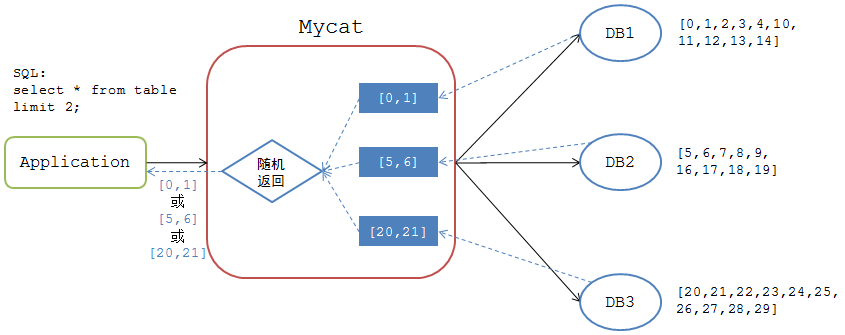
如果设计使用Mycat时有非分片字段查询（即非分片字段作为查询条件会导致将一条sql映射多台机器并行执行查询），请考虑放弃！

## 2.分页排序

先看一下Mycat是如何处理分页操作的，假如有如下Mycat分库方案：   
一张表有30份数据分布在3个分片DB上，具体数据分布如下

DB1：[0,1,2,3,4,10,11,12,13,14]   
DB2：[5,6,7,8,9,16,17,18,19]   
DB3：[20,21,22,23,24,25,26,27,28,29]

（这个示例的场景中没有查询条件，所以都是全分片查询，也就没有假定该表的分片字段和分片方法）



当应用执行如下分页查询时

mysql>select \* from table limit 2;

Mycat将该SQL请求分发到各个DB节点去执行，并接收各个DB节点的返回结果

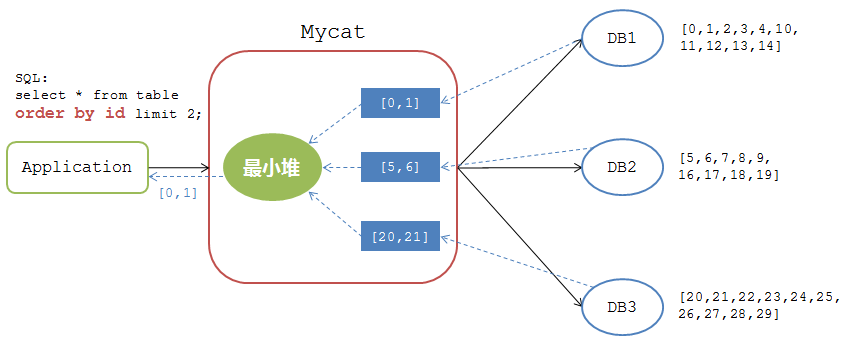
DB1: [0,1]   
DB2: [5,6]   
DB3: [20,21]

但Mycat向应用返回的结果集取决于哪个DB节点最先返回结果给Mycat。如果Mycat最先收到DB1节点的结果集，那么Mycat返回给应用端的结果集为 **[0,1]**，如果Mycat最先收到DB2节点的结果集，那么返回给应用端的结果集为 **[5,6]**。也就是说，相同情况下，同一个SQL，在Mycat上执行时会有不同的返回结果。

在Mycat中执行分页操作时必须显示加上排序条件才能保证结果的正确性，下面看一下Mycat对排序分页的处理逻辑。   
假如在前面的分页查询中加上了排序条件（假如表数据的列名为**id**）

mysql>select \* from table **order by id** limit 2;

Mycat的处理逻辑如下图：

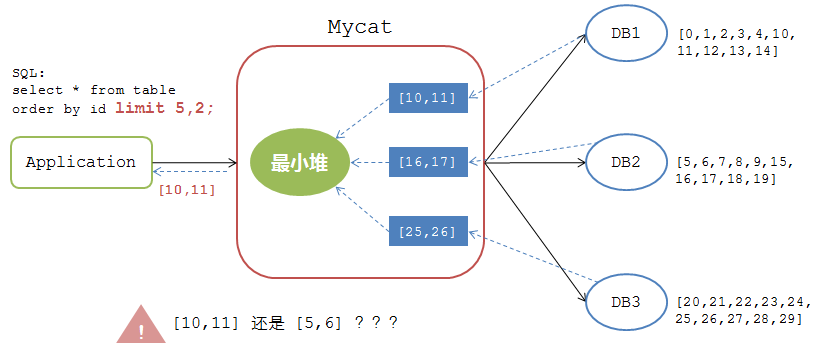


在有排序条件的情况下，Mycat接收到各个DB节点的返回结果后，对其进行最小堆运算，计算出所有结果集中最小的两条记录 **[0,1]** 返回给应用。

但是，当排序分页中有 **偏移量** （offset）时，处理逻辑又有不同。假如应用的查询SQL如下：

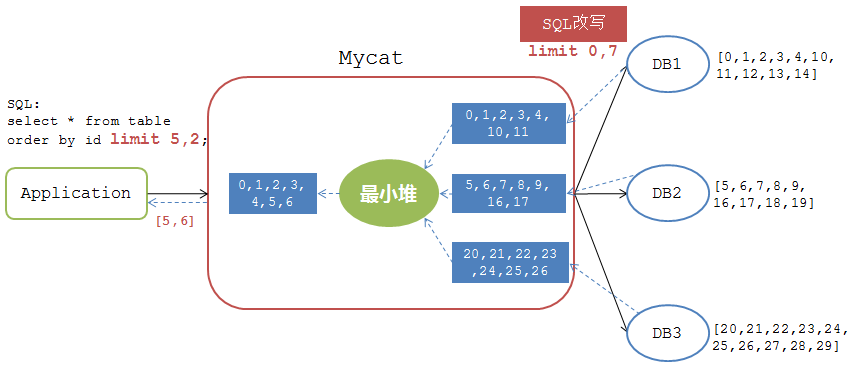
mysql>select \* from table order by id **limit 5,2**;

如果按照上述排序分页逻辑来处理，那么处理结果如下图：



Mycat将各个DB节点返回的数据 **[10,11], [16,17], [20,21]** 经过最小堆计算后返回给应用的结果集是 **[10,11]**。可是，对于应用而言，该表的所有数据明明是 **0-29** 这30个数据的集合，**limit 5,2** 操作返回的结果集应该是 **[5,6]**，如果返回 **[10,11]** 则是错误的处理逻辑。

所以Mycat在处理 **有偏移量的排序分页** 时是另外一套逻辑——**改写SQL** 。如下图：



Mycat在下发有 **limit m,n** 的SQL语句时会对其进行改写，改写成 **limit 0, m+n** 来保证查询结果的逻辑正确性。所以，Mycat发送到后端DB上的SQL语句是

mysql>select \* from table order by id limit 0,7;

各个DB返回给Mycat的结果集是

DB1: [0,1,2,3,4,10,11]   
DB2: [5,6,7,8,9,16,17]   
DB3: [20,21,22,23,24,25,26]

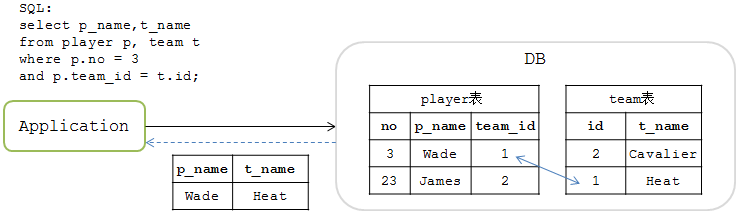
经过最小堆计算后得到最小序列 **[0,1,2,3,4,5,6]** ，然后返回偏移量为5的两个结果为 **[5,6]** 。

虽然Mycat返回了正确的结果，但是仔细推敲发现这类操作的处理逻辑是及其消耗（浪费）资源的。应用需要的结果集为2条，Mycat中需要处理的结果数为21条。也就是说，对于有 **t** 个DB节点的全分片 **limit m, n** 操作，Mycat需要处理的数据量为 **(m+n)\*t** 个。比如实际应用中有50个DB节点，要执行limit 1000,10操作，则Mycat处理的数据量为 **50500** 条，返回结果集为**10**，当偏移量更大时，内存和CPU资源的消耗则是数十倍增加。

如果设计使用Mycat时有分页排序，请考虑放弃！

## 3.任意表JOIN

先看一下在单库中JOIN中的场景。假设在某单库中有 **player** 和 **team** 两张表，**player** 表中的 **team\_id** 字段与 **team** 表中的 **id** 字段相关联。操作场景如下图：



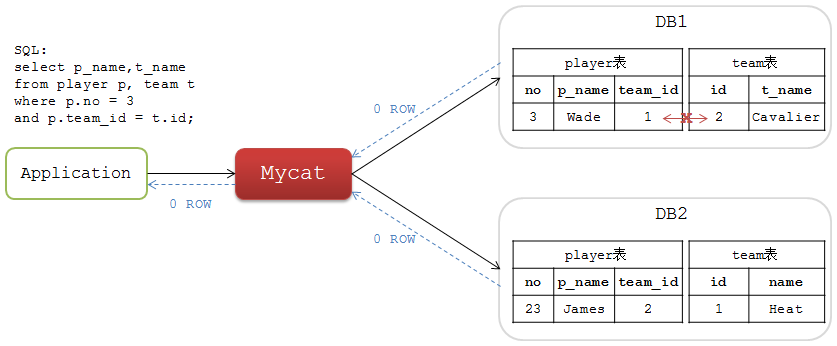
JOIN操作的SQL如下

mysql>select p\_name,t\_name from player p, team t where p.no = 3 and p.team\_id = t.id;

此时能查询出结果

| **p\_name** | **t\_name** |
| --- | --- |
| Wade | Heat |

如果将这两个表的数据分库后，相关联的数据可能分布在不同的DB节点上，如下图：



这个SQL在各个单独的分片DB中都查不出结果，也就是说Mycat不能查询出正确的结果集。

设计使用Mycat时如果要进行表JOIN操作，要确保两个表的关联字段具有相同的数据分布，否则请考虑放弃！

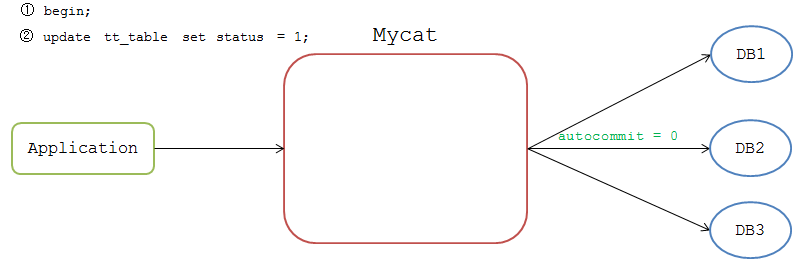
## 4.分布式事务

Mycat并没有根据二阶段提交协议实现 **XA事务**，而是只保证 **prepare** 阶段数据一致性的 **弱XA事务** ，实现过程如下：

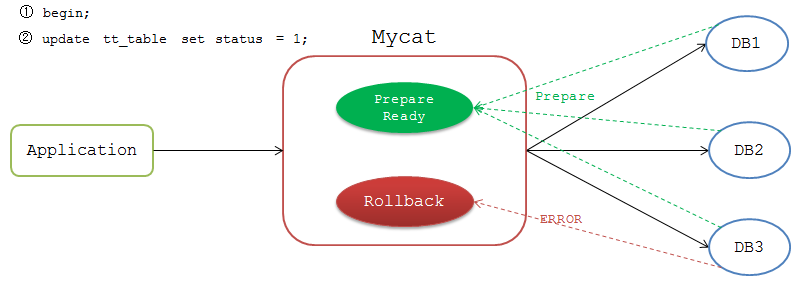
应用开启事务后Mycat标识该连接为非自动提交，比如前端执行

mysql>begin;

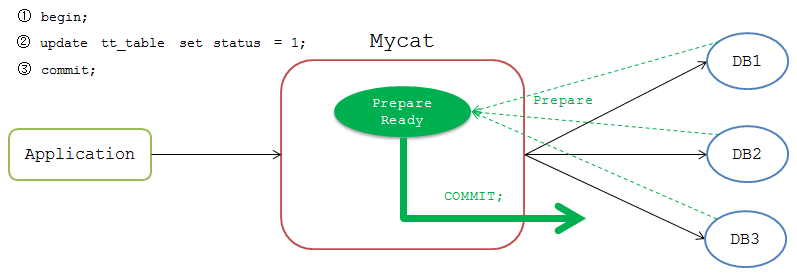
Mycat不会立即把命令发送到DB节点上，等后续下发SQL时，Mycat从连接池获取非自动提交的连接去执行。



Mycat会等待各个节点的返回结果，如果都执行成功，Mycat给该连接标识为 **Prepare Ready** 状态，如果有一个节点执行失败，则标识为 **Rollback** 状态。



执行完成后Mycat等待前端发送 commit 或 rollback 命令。发送 commit 命令时，Mycat检测当前连接是否为 **Prepare Ready** 状态，若是，则将 commit 命令发送到各个DB节点。



但是，这一阶段是无法保证一致性的，如果一个DB节点在 commit 时故障，而其他DB节点 commit 成功，Mycat会一直等待故障DB节点返回结果。Mycat只有收到所有DB节点的成功执行结果才会向前端返回 执行成功 的包，此时Mycat只能一直 waiting 直至TIMEOUT，导致事务一致性被破坏。

设计使用Mycat时如果有分布式事务，得先看是否得保证事务得强一致性，否则请考虑放弃！