### 1.名称空间

#### 1.1说明

* 在一个项目中，需要使用HBase保存多张表，这些表会按照业务域来划分
* 为了方便管理，不同的业务域以名称空间（namespace)来划分，这样管理起来会更加容易
* 类似于Hive中的数据库，不同的数据库下可以放不同类型的表
* HBase默认的名称空间是「default」，默认情况下，创建表时表都将创建在 default 名称空间下
* HBase中还有一个命名空间「hbase」，用于存放系统的内建表（namespace、meta）

#### 语法

##### 创建命名空间

create\_namespace 'CHAT'

##### 查看命名空间列表

list\_namespace

##### 查看命名空间

describe\_namespace 'CHAT'

##### 命名空间创建表

在命令CHAT命名空间下创建名为：MSG的表，该表包含一个名为C1的列蔟。

注意：带有命名空间的表，使用**冒号**将命名空间和表名连接到一起。

create 'CHAT:MSG','C1'

##### 删除命名空间

* 删除命名空间，命名空间中必须没有表，如果命名空间中有表，是无法删除的

drop\_namespace 'CHAT'

### 列蔟设计

* HBase列蔟的数量应该越少越好
  + 两个及以上的列蔟HBase性能并不是很好
  + 一个列蔟所存储的数据达到flush的阈值时，表中所有列蔟将同时进行flush操作
  + 这将带来不必要的I/O开销，列蔟越多，对性能影响越大
* 本次项目中我们只设计一个列蔟：C1

### 版本设计

#### 说明

* 此处，我们需要保存的历史聊天记录是不会更新的，一旦数据保存到HBase中，就不会再更新
* 无需考虑版本问题
* 本次项目中只保留一个版本即可，这样可以节省大量空间
* HBase默认创建表的版本为1，故此处保持默认即可

#### 查看表

通过以下输出可以看到：

* 版本是相对于列蔟而言
* 默认列蔟的版本数为1

|  |
| --- |
| hbase(main):015:0> **describe "CHAT:MSG"**  Table CHAT:MSG is ENABLED  CHAT:MSG  **COLUMN FAMILIES DESCRIPTION**  {NAME => 'C1', **VERSIONS => '1'**, EVICT\_BLOCKS\_ON\_CLOSE => 'false', NEW\_VERSION\_BEHAVIOR => 'false', KEEP\_DELETED\_CELLS => 'FALSE', CACHE\_DATA\_ON\_WRITE => 'false', DATA\_BLOCK\_ENCODING => 'NONE', TTL => 'FOREVER', MIN\_VERSIONS => '0', REPLI  CATION\_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE\_INDEX\_ON\_WRITE => 'false', IN\_MEMORY => 'false', CACHE\_BLOOMS\_ON\_WRITE => 'false', PREFETCH\_BLOCKS\_ON\_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}  1 row(s) |

### 数据压缩

#### 压缩算法

在HBase可以使用多种压缩编码，包括LZO、SNAPPY、GZIP。只在硬盘压缩，内存中或者网络传输中没有压缩。

|  |  |  |  |
| --- | --- | --- | --- |
| **压缩算法** | **压缩后占比** | **压缩** | **解压缩** |
| GZIP | 13.4% | 21 MB/s | 118 MB/s |
| LZO | 20.5% | 135 MB/s | 410 MB/s |
| Zippy/Snappy | 22.2% | 172 MB/s | 409 MB/s |

* GZIP的压缩率最高，但是其实CPU密集型的，对CPU的消耗比其他算法要多，压缩和解压速度也慢；
* LZO的压缩率居中，比GZIP要低一些，但是压缩和解压速度明显要比GZIP快很多，其中解压速度快的更多；
* Zippy/Snappy的压缩率最低，而压缩和解压速度要稍微比LZO要快一些
* **本案例采用GZ算法，这样可以确保的压缩比最大化，更加节省空间**

#### 查看表数据压缩方式

通过以下输出可以看出，HBase创建表默认是没有指定压缩算法的

|  |
| --- |
| hbase(main):015:0> describe " CHAT:MSG"  Table CHAT:MSG is ENABLED  CHAT:MSG  COLUMN FAMILIES DESCRIPTION  {NAME => 'C1', VERSIONS => '1', EVICT\_BLOCKS\_ON\_CLOSE => 'false', NEW\_VERSION\_BEHAVIOR => 'false', KEEP\_DELETED\_CELLS => 'FALSE', CACHE\_DATA\_ON\_WRITE => 'false', DATA\_BLOCK\_ENCODING => 'NONE', TTL => 'FOREVER', MIN\_VERSIONS => '0', REPLI  CATION\_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE\_INDEX\_ON\_WRITE => 'false', IN\_MEMORY => 'false', CACHE\_BLOOMS\_ON\_WRITE => 'false', PREFETCH\_BLOCKS\_ON\_OPEN => 'false', **COMPRESSION => 'NONE'**, BLOCKCACHE => 'true', BLOCKSIZE => '65536'}  1 row(s) |

#### 设置数据压缩

本案例中，我们使用GZ压缩算法，语法如下：

* 创建新的表，并指定数据压缩算法

create "CHAT:MSG", {NAME => "C1", COMPRESSION => "GZ"}

* 修改已有的表，并指定数据压缩算法

alter " CHAT:MSG", {NAME => "C1", COMPRESSION => "GZ"}

### ROWKEY设计原则

#### HBase官方的设计原则

##### 避免使用递增行键/时序数据

如果ROWKEY设计的都是按照顺序递增（例如：时间戳），这样会有很多的数据写入时，负载都在一台机器上。我们尽量应当将写入大压力均衡到各个RegionServer

##### 避免ROWKEY和列的长度过大

* 在HBase中，要访问一个Cell（单元格），需要有ROWKEY、列蔟、列名，如果ROWKEY、列名太大，就会占用较大内存空间。所以ROWKEY和列的长度应该尽量短小
* ROWKEY的最大长度是64KB，建议越短越好

##### 使用long等类型比String类型更省空间

long类型为8个字节，8个字节可以保存非常大的无符号整数，例如：18446744073709551615。如果是字符串，是按照一个字节一个字符方式保存，需要快3倍的字节数存储。

##### ROWKEY唯一性

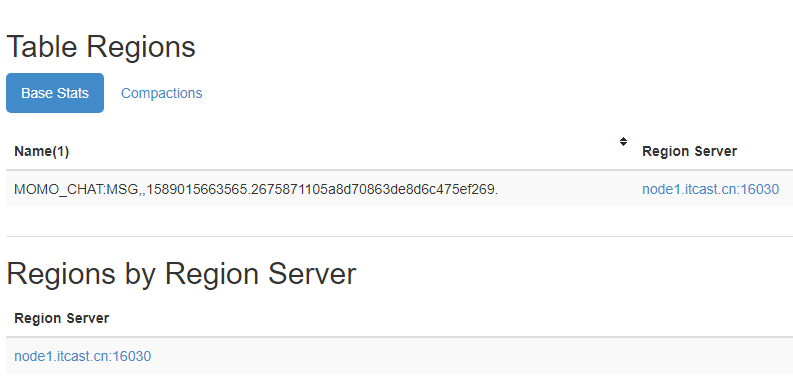
* 设计ROWKEY时，必须保证RowKey的唯一性
* 由于在HBase中数据存储是Key-Value形式，若向HBase中同一张表插入相同RowKey的数据，则原先存在的数据会被新的数据覆盖。

#### 避免数据热点

* 热点是指大量的客户端（client）直接访问集群的一个或者几个节点（可能是读、也可能是写）
* 大量地访问量可能会使得某个服务器节点超出承受能力，导致整个RegionServer的性能下降，其他的Region也会受影响

##### 预分区

* 默认情况，一个HBase的表只有一个Region，被托管在一个RegionServer中



* 每个Region有两个重要的属性：Start Key、End Key，表示这个Region维护的ROWKEY范围
* 如果只有一个Region，那么Start Key、End Key都是空的，没有边界。所有的数据都会放在这个Region中，但当数据越来越大时，会将Region分裂，取一个Mid Key来分裂成两个Region
* 预分区个数 = 节点的倍数。默认Region的大小为10G，假设我们预估1年下来的大小为10T，则10000G / 10G = 1000个Region，所以，我们可以预设为1000个Region，这样，1000个Region将均衡地分布在各个节点上

##### ROWKEY避免热点设计

1. 反转策略

* 如果设计出的ROWKEY在数据分布上不均匀，但ROWKEY尾部的数据却呈现出了良好的随机性，可以考虑**将ROWKEY的翻转**，或者直接将尾部的bytes提前到ROWKEY的开头。
* 反转策略可以使ROWKEY随机分布，但是牺牲了ROWKEY的有序性
* 缺点：利于Get操作，但不利于Scan操作，因为数据在原ROWKEY上的自然顺序已经被打乱

1. 加盐策略

* Salting（加盐）的原理是在原ROWKEY的**前面**添加固定长度的随机数，也就是给ROWKEY分配一个随机前缀使它和之间的ROWKEY的开头不同
* 随机数能保障数据在所有Regions间的负载均衡
* 缺点：因为添加的是随机数，基于原ROWKEY查询时无法知道随机数是什么，那样在查询的时候就需要去各个可能的Regions中查找，加盐对比读取是无力的

1. 哈希策略

* 基于 ROWKEY的完整或部分数据进行 Hash，而后将Hashing后的值完整替换或部分替换原ROWKEY的前缀部分
* 这里说的 hash 包含 MD5、sha1、sha256 或 sha512 等算法
* 缺点：Hashing 也不利于 Scan，因为打乱了原RowKey的自然顺序

#### 数据预分区

##### 预分区

在HBase中，可以通过指定start key、end key来进行分区，还可以直接指定Region的数量，指定分区的策略。

1. 指定 start key、end key来分区

|  |
| --- |
| **hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '20', '30', '40']**  **hbase> create 't1', 'f1', SPLITS => ['10', '20', '30', '40']**  **hbase> create 't1', 'f1', SPLITS\_FILE => 'splits.txt', OWNER => 'johndoe'** |

1. 指定分区数量、分区策略

|  |
| --- |
| **hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}** |

分区策略

* HexStringSplit: ROWKEY是十六进制的字符串作为前缀的
* DecimalStringSplit: ROWKEY是10进制数字字符串作为前缀的
* UniformSplit: ROWKEY前缀完全随机

Region的数量可以按照数据量来预估。本次案例，因为受限于虚拟机，所以我们设计为6个Region。因为ROWKEY我们是使用多个字段拼接，而且前缀不是完全随机的，所以需要使用HexStringSplit。

## 性能问题

* Hbase默认只支持对行键的索引，那么如果要针对其它的列来进行查询，就只能全表扫描
* 上述的查询是使用scan + filter组合来进行查询的，但查询地效率不高，因为要进行顺序全表扫描而没有其他索引。如果数据量较大，只能在客户端（client）来进行处理，如果要传输到Client大量的数据，然后交由客户端处理
  + 网络传输压力很大
  + 客户端的压力很大
* 如果表存储的数据量很大时，效率会非常低下，此时需要使用二级索引
* 也就是除了ROWKEY的索引外，还需要人为添加其他的方便查询的索引

如果每次需要我们开发二级索引来查询数据，这样使用起来很麻烦。再者，查询数据都是HBase Java API，使用起来不是很方便。为了让其他开发人员更容易使用该接口。如果有一种SQL引擎，通过SQL语句来查询数据会更加方便。

此时，使用Apache Phoenix就可以解决我们上述问题。



## Apache Phoenix

### Phoenix介绍

Phoenix官方网址：<http://phoenix.apache.org/>

#### 简介



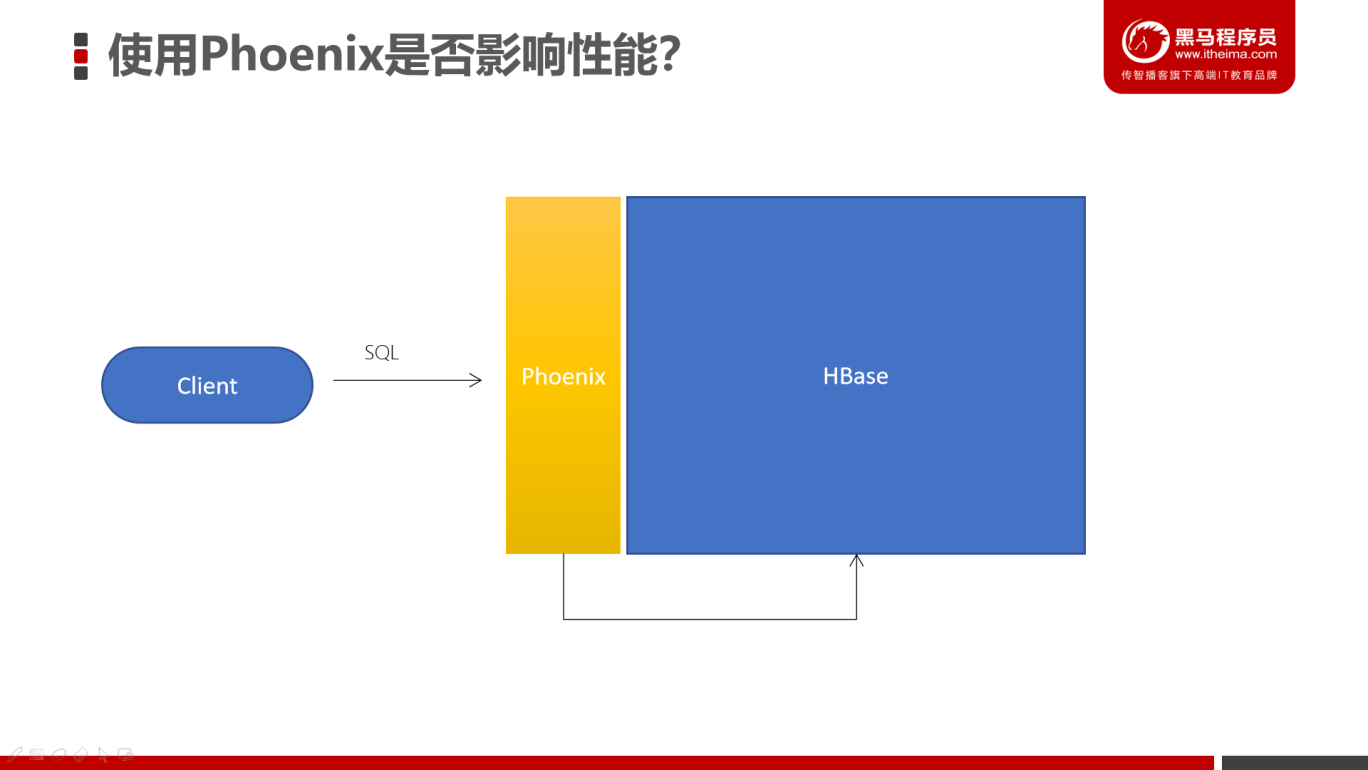
* Phoenix官网：「We put the SQL back in NoSQL」

Apache Phoenix让Hadoop中支持低延迟OLTP和业务操作分析。

* 提供标准的SQL以及完备的ACID事务支持
* 通过利用HBase作为存储，让NoSQL数据库具备通过有模式的方式读取数据，我们可以使用SQL语句来操作HBase，例如：创建表、以及插入数据、修改数据、删除数据等。
* Phoenix通过**协处理器在服务器端执行操作**，最小化客户机/服务器数据传输

Apache Phoenix可以很好地与其他的Hadoop组件整合在一起，例如：Spark、Hive、Flume以及MapReduce。

#### 使用Phoenix是否会影响HBase性能



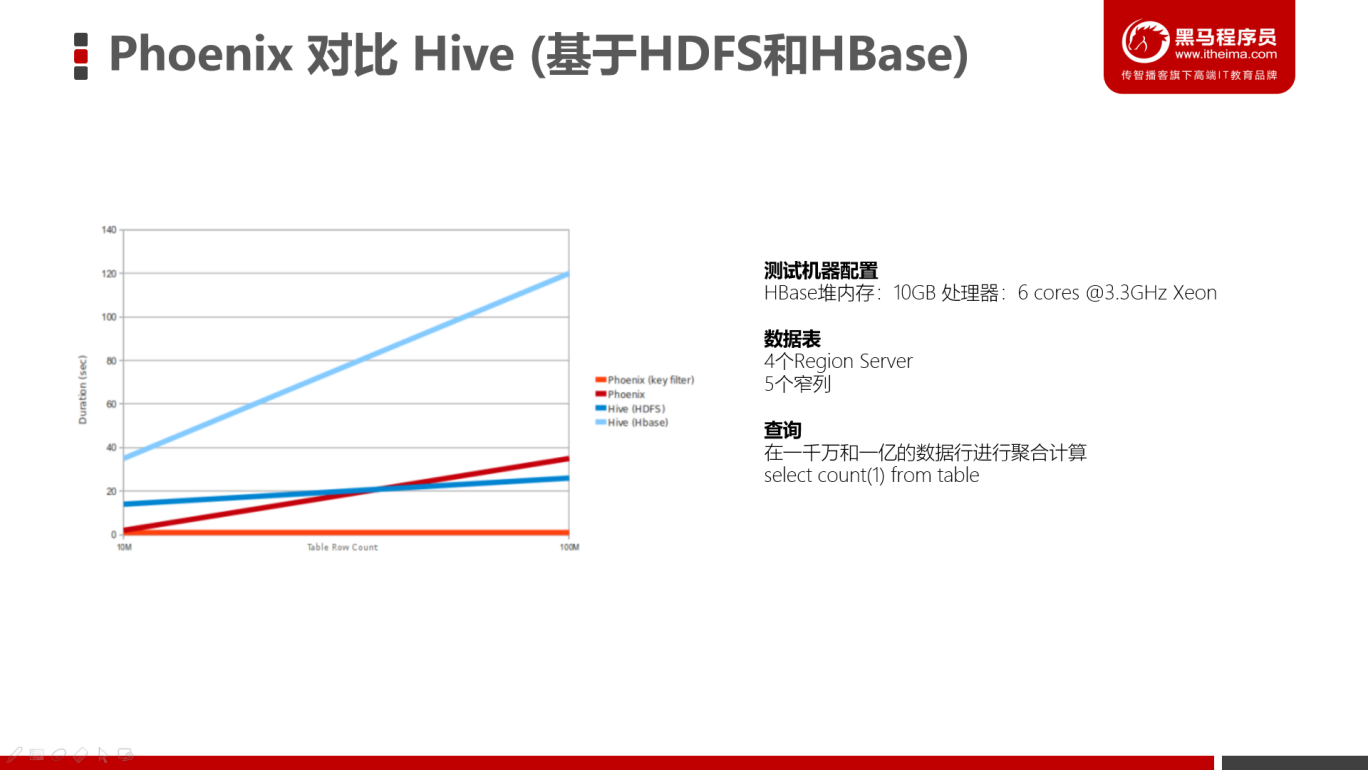
* Phoenix不会影响HBase性能，反而会提升HBase性能
* Phoenix将SQL查询编译为本机HBase扫描
* 确定scan的key的最佳startKey和endKey
* 编排scan的并行执行
* 将WHERE子句中的谓词推送到服务器端
* 通过协处理器执行聚合查询
* 用于提高非行键列查询性能的二级索引
* 统计数据收集，以改进并行化，并指导优化之间的选择
* 跳过扫描筛选器以优化IN、LIKE和OR查询
* 行键加盐保证分配均匀，负载均衡

#### 哪些公司在使用Phoenix

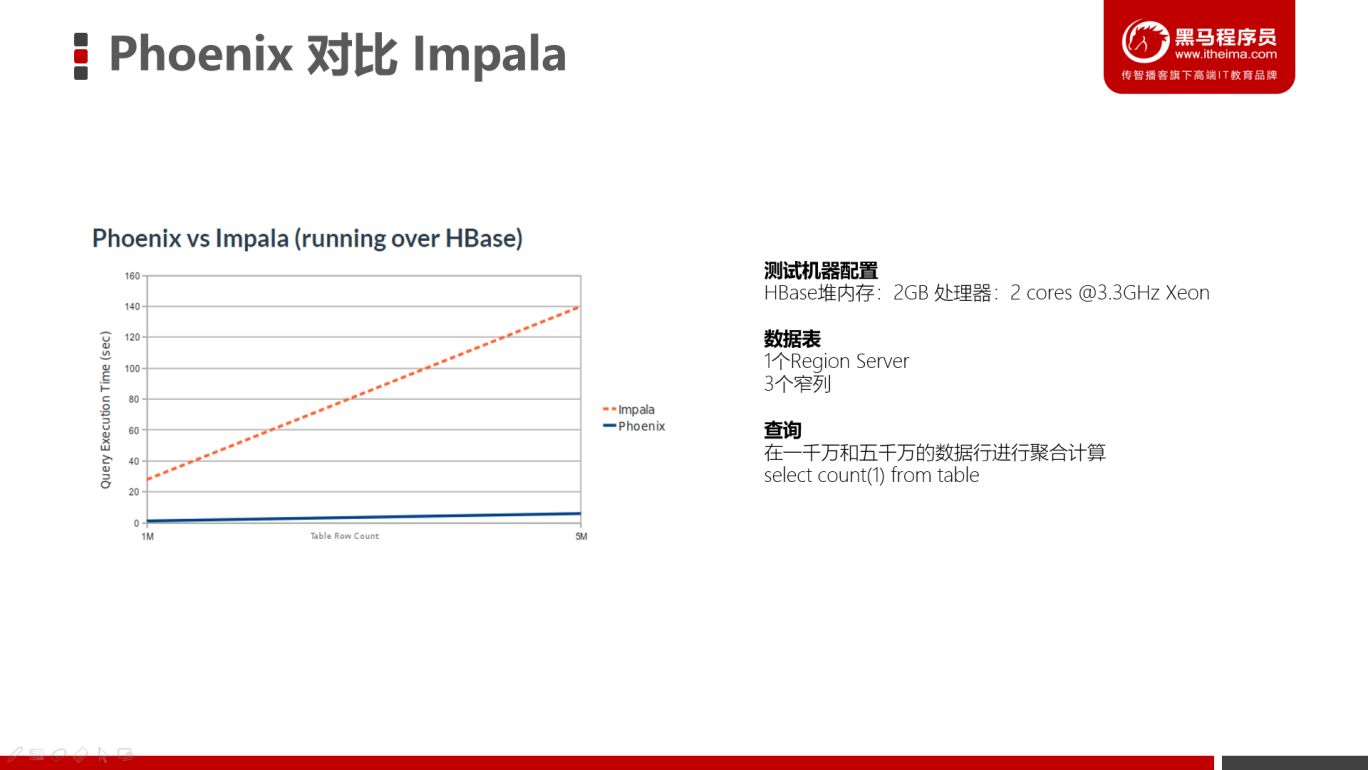


#### 官方性能测试

##### Phoenix对标Hive（基于HDFS和HBase）



##### Phoenix对标Impala



##### 关于上述官网两张性能测试的说明

上述两张图是从Phoenix官网拿下来的，这容易引起一个歧义。就是：有了HBase + Phoenix，那是不是意味着，我们将来做数仓（OLAP）就可以不用Hadoop + Hive了？

千万不要这么以为，HBase + Phoenix是否适合做OLAP取决于HBase的定位。Phoenix只是在HBase之上构建了SQL查询引擎（注意：我称为SQL查询引擎，并不是像MapReduce、Spark这种大规模数据计算引擎）。HBase的定位是在高性能随机读写，Phoenix可以使用SQL快插查询HBase中的数据，但数据操作底层是必须符合HBase的存储结构，例如：必须要有ROWKEY、必须要有列蔟。因为有这样的一些限制，绝大多数公司不会选择HBase + Phoenix来作为数据仓库的开发。而是用来快速进行海量数据的随机读写。这方面，HBase + Phoenix有很大的优势。

### 安装Phoenix

#### 下载

大家可以从官网上下载与HBase版本对应的Phoenix版本。对应到HBase 2.1，应该使用版本「5.0.0-HBase-2.0」。

<http://phoenix.apache.org/download.html>

也可以使用资料包中的安装包。

#### 安装

1. 上传安装包到Linux系统，并解压

|  |
| --- |
| cd /export/software  tar -xvzf apache-phoenix-5.0.0-HBase-2.0-bin.tar.gz -C ../server/ |

1. 将phoenix的所有jar包添加到所有HBase RegionServer和Master的复制到HBase的lib目录

|  |
| --- |
| *#  拷贝jar包到hbase lib目录*  cp /export/server/apache-phoenix-5.0.0-HBase-2.0-bin/phoenix-**\***.jar /export/server/hbase-2.1.0/lib/  *#  进入到hbase lib  目录*  cd /export/server/hbase-2.1.0/lib/  *# 分发jar包到每个HBase 节点*  scp phoenix-**\***.jar node2:**$PWD**  scp phoenix-**\***.jar node3:**$PWD** |

1. 修改配置文件

|  |
| --- |
| cd /export/server/hbase-2.1.0/conf/  vim hbase-site.xml  ------  # 1. 将以下配置添加到 hbase-site.xml 后边  <!-- 支持HBase命名空间映射 -->  <property>  <name>phoenix.schema.isNamespaceMappingEnabled</name>  <value>true</value>  </property>  <!-- 支持索引预写日志编码 -->  <property>  <name>hbase.regionserver.wal.codec</name>  <value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>  </property>  # 2. 将hbase-site.xml分发到每个节点  scp hbase-site.xml node2:$PWD  scp hbase-site.xml node3:$PWD |

1. 将配置后的hbase-site.xml拷贝到phoenix的bin目录

|  |
| --- |
| cp /export/server/hbase-2.1.0/conf/hbase-site.xml /export/server/apache-phoenix-5.0.0-HBase-2.0-bin/bin/ |

1. 重新启动HBase

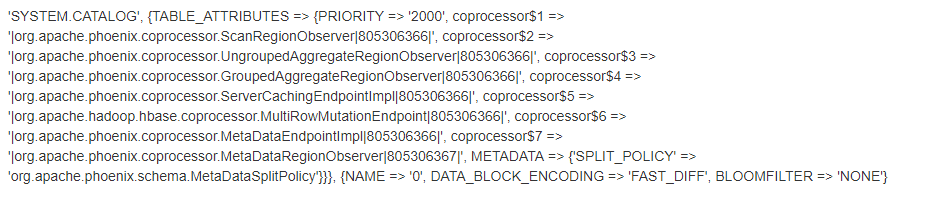
|  |
| --- |
| stop-hbase.sh  start-hbase.sh |

1. 启动Phoenix客户端，连接Phoenix Server

注意：第一次启动Phoenix连接HBase会稍微慢一点。

|  |
| --- |
| cd /export/server/apache-phoenix-5.0.0-HBase-2.0-bin/  bin/sqlline.py node1:2181  # 输入!table查看Phoenix中的表  !table |

1. 查看HBase的Web UI，可以看到Phoenix在default命名空间下创建了一些表，而且该系统表加载了大量的协处理器。





### 快速入门

#### 需求

本次的小DEMO，我们沿用之前的订单数据集。我们将使用Phoenix来创建表，并进行数据增删改查操作。



|  |  |
| --- | --- |
| 列名 | 说明 |
| id | 订单ID |
| status | 订单状态 |
| money | 支付金额 |
| pay\_way | 支付方式ID |
| user\_id | 用户ID |
| operation\_time | 操作时间 |
| category | 商品分类 |

#### 创建表语法

在Phoenix中，我们可以使用类似于MySQL DDL的方式快速创建表。例如：

|  |
| --- |
| CREATE TABLE IF NOT EXISTS 表名 (  ROWKEY名称 数据类型 PRIMARY KEY  列蔟名.列名1 数据类型 NOT NULL,  列蔟名.列名2 数据类型 NOT NULL,  列蔟名.列名3 数据类型); |

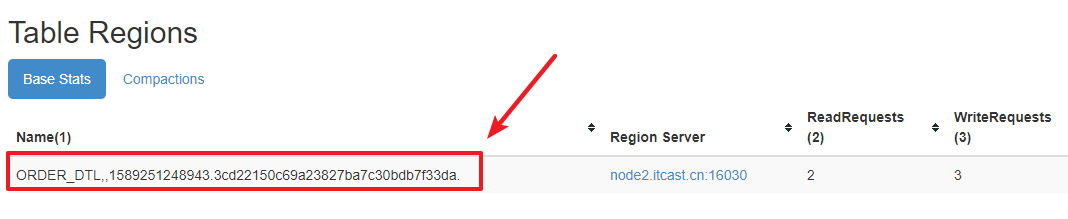
订单明细建表语句：

|  |
| --- |
| **create** **table** **if** **not** **exists** ORDER\_DTL(      ID **varchar** **primary** **key**,      C1.STATUS **varchar**,      C1.MONEY **float**,      C1.PAY\_WAY **integer**,      C1.USER\_ID **varchar**,      C1.OPERATION\_TIME **varchar**,      C1.CATEGORY **varchar**  ); |

通过HBase的Web UI，我们可以看到Phoenix帮助我们自动在HBase中创建了一张名为 ORDER\_DTL 的表格，可以看到里面添加了很多的协处理器。

|  |
| --- |
| 'ORDER\_DTL', {TABLE\_ATTRIBUTES => {coprocessor$1 => '|org.apache.phoenix.coprocessor.ScanRegionObserver|805306366|', coprocessor$2 => '|org.apache.phoenix.coprocessor.UngroupedAggregateRegionObserver|805306366|', coprocessor$3 => '|org.apache.phoenix.coprocessor.GroupedAggregateRegionObserver|805306366|', coprocessor$4 => '|org.apache.phoenix.coprocessor.ServerCachingEndpointImpl|805306366|', coprocessor$5 => '|org.apache.phoenix.hbase.index.Indexer|805306366|index.builder=org.apache.phoenix.index.PhoenixIndexBuilder,org.apache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexCodec'}}, {NAME => '0', VERSIONS => '1', EVICT\_BLOCKS\_ON\_CLOSE => 'false', NEW\_VERSION\_BEHAVIOR => 'false', KEEP\_DELETED\_CELLS => 'FALSE', CACHE\_DATA\_ON\_WRITE => 'false', DATA\_BLOCK\_ENCODING => 'FAST\_DIFF', TTL => 'FOREVER', MIN\_VERSIONS => '0', REPLICATION\_SCOPE => '0', BLOOMFILTER => 'NONE', CACHE\_INDEX\_ON\_WRITE => 'false', IN\_MEMORY => 'false', CACHE\_BLOOMS\_ON\_WRITE => 'false', PREFETCH\_BLOCKS\_ON\_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'} |

同时，我们也看到这个表格默认只有一个Region，也就是没有分区的。



**常见问题：**

1. **The table does not have a primary key**

|  |
| --- |
| 0: jdbc:phoenix:node1:2181> create table if not exists ORDER\_DTL(  . . . . . . . . . . . . . . . . . . > C1.ID varchar,  . . . . . . . . . . . . . . . . . . > C1.STATUS varchar,  . . . . . . . . . . . . . . . . . . > C1.MONEY double,  . . . . . . . . . . . . . . . . . . > C1.PAY\_WAY integer,  . . . . . . . . . . . . . . . . . . > C1.USER\_ID varchar,  . . . . . . . . . . . . . . . . . . > C1.OPERATION\_TIME varchar,  . . . . . . . . . . . . . . . . . . > C1.CATEGORY varchar  . . . . . . . . . . . . . . . . . . > );  **Error: ERROR 509 (42888): The table does not have a primary key. tableName=ORDER\_DTL (state=42888,code=509)**  java.sql.SQLException: ERROR 509 (42888): The table does not have a primary key. tableName=ORDER\_DTL  at org.apache.phoenix.exception.SQLExceptionCode$Factory$1.newException(SQLExceptionCode.java:494)  at org.apache.phoenix.exception.SQLExceptionInfo.buildException(SQLExceptionInfo.java:150)  at org.apache.phoenix.schema.MetaDataClient.createTableInternal(MetaDataClient.java:2440)  at org.apache.phoenix.schema.MetaDataClient.createTable(MetaDataClient.java:1114)  at org.apache.phoenix.compile.CreateTableCompiler$1.execute(CreateTableCompiler.java:192)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:408)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:391)  at org.apache.phoenix.call.CallRunner.run(CallRunner.java:53)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:390)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:378)  at org.apache.phoenix.jdbc.PhoenixStatement.execute(PhoenixStatement.java:1825)  at sqlline.Commands.execute(Commands.java:822)  at sqlline.Commands.sql(Commands.java:732)  at sqlline.SqlLine.dispatch(SqlLine.java:813)  at sqlline.SqlLine.begin(SqlLine.java:686)  at sqlline.SqlLine.start(SqlLine.java:398)  at sqlline.SqlLine.main(SqlLine.java:291) |

原因：

* 表没有主键，创建表时必须要指定主键，因为HBase数据存储必须要有rowkey

解决办法：

* 在id后面加一个primary key

1. **Error: ERROR 1003 (42J01): Primary key columns must not have a family name**

|  |
| --- |
| Error: ERROR 1003 (42J01): Primary key columns must not have a family name. columnName=C1.ID (state=42J01,code=1003)  **java.sql.SQLException: ERROR 1003 (42J01): Primary key columns must not have a family name. columnName=C1.ID**  at org.apache.phoenix.exception.SQLExceptionCode$Factory$1.newException(SQLExceptionCode.java:494)  at org.apache.phoenix.exception.SQLExceptionInfo.buildException(SQLExceptionInfo.java:150)  at org.apache.phoenix.schema.MetaDataClient.newColumn(MetaDataClient.java:1028)  at org.apache.phoenix.schema.MetaDataClient.createTableInternal(MetaDataClient.java:2396)  at org.apache.phoenix.schema.MetaDataClient.createTable(MetaDataClient.java:1114)  at org.apache.phoenix.compile.CreateTableCompiler$1.execute(CreateTableCompiler.java:192)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:408)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:391)  at org.apache.phoenix.call.CallRunner.run(CallRunner.java:53)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:390)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:378)  at org.apache.phoenix.jdbc.PhoenixStatement.execute(PhoenixStatement.java:1825)  at sqlline.Commands.execute(Commands.java:822)  at sqlline.Commands.sql(Commands.java:732)  at sqlline.SqlLine.dispatch(SqlLine.java:813)  at sqlline.SqlLine.begin(SqlLine.java:686)  at sqlline.SqlLine.start(SqlLine.java:398)  at sqlline.SqlLine.main(SqlLine.java:291)  0: jdbc:phoenix:node1:2181>  Error: (state=,code=0)  java.sql.SQLFeatureNotSupportedException  at org.apache.phoenix.jdbc.PhoenixStatement.cancel(PhoenixStatement.java:1691)  at sqlline.DispatchCallback.forceKillSqlQuery(DispatchCallback.java:83)  at sqlline.SqlLine.begin(SqlLine.java:700)  at sqlline.SqlLine.start(SqlLine.java:398)  at sqlline.SqlLine.main(SqlLine.java:291) |

问题原因：

* 给Primary Key指定了主键

解决办法：

* 移除primary key上的列蔟

### 查看表的信息

|  |
| --- |
| !desc ORDER\_DTL |

注意：

* 一定要加上 !

#### 删除表语法

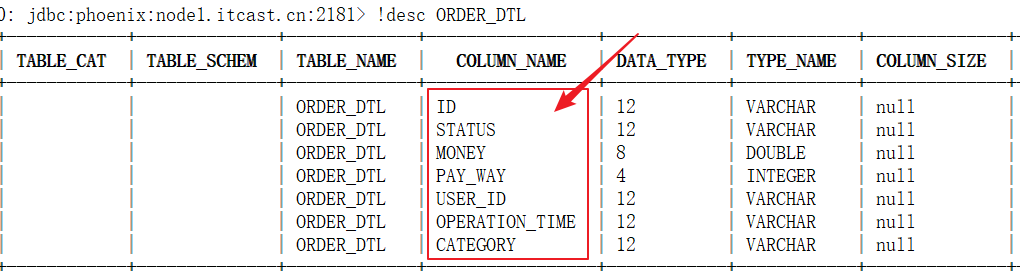
|  |
| --- |
| drop table if exists ORDER\_DTL; |

#### 大小写问题

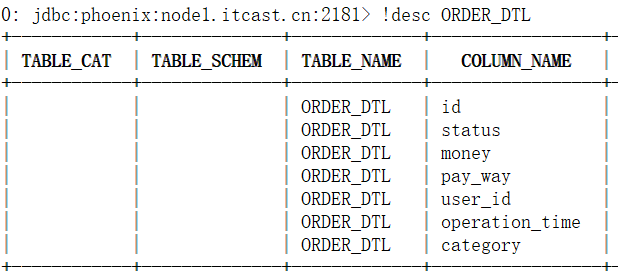
在HBase中，如果在列蔟、列名没有添加双引号。Phoenix会自动转换为大写。

例如：

|  |
| --- |
| create table if not exists ORDER\_DTL(      id varchar primary key,      C1.status varchar,      C1.money double,      C1.pay\_way integer,      C1.user\_id varchar,      C1.operation\_time varchar,      C1.category varchar  ); |



如果要将列的名字改为小写，需要使用双引号，如下：



注意：

* 一旦加了小写，后面都得任何应用该列的地方都得使用双引号，否则将报以下错误：

|  |
| --- |
| Error: ERROR 504 (42703): Undefined column. columnName=ORDER\_DTL.ID |

#### 插入数据

在Phoenix中，插入并**不是**使用insert来实现的。而是 「upsert 」命令。它的功能为insert + update，与HBase中的put相对应。如果不存在则插入，否则更新。列表是可选的，如果不存在，值将按模式中声明的顺序映射到列。这些值必须计算为常量。

|  |
| --- |
| upsert into 表名(列蔟列名, xxxx, ) VALUES(XXX, XXX, XXX) |

插入一条数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 订单ID | 订单状态 | 支付金额 | 支付方式ID | 用户ID | 操作时间 | 商品分类 |
| ID | STATUS | PAY\_MONEY | PAYWAY | USER\_ID | OPERATION\_DATE | CATEGORY |
| 000001 | 已提交 | 4070 | 1 | 4944191 | 2020-04-25 12:09:16 | 手机; |

参考代码：

|  |
| --- |
| UPSERT INTO ORDER\_DTL VALUES('000001', '已提交', 4070, 1, '4944191', '2020-04-25 12:09:16', '手机;'); |

#### 查询数据

与标准SQL一样，Phoenix也是使用select语句来实现数据的查询。

##### 查询所有数据

**SELECT** **\*** **FROM** ORDER\_DTL;

##### 更新数据

在Phoenix中，更新数据也是使用UPSERT。语法格式如下：

UPSERT INTO 表名(列名, …) VALUES(对应的值, …);

需求：

将ID为'000001'的订单状态修改为已付款。

|  |
| --- |
| UPSERT INTO ORDER\_DTL("id", C1."status") VALUES ('000001', '已付款'); |

##### 根据ID查询数据

|  |
| --- |
| SELECT \* FROM ORDER\_DTL WHERE "id" = '000001'; |

#### 根据ID删除数据

|  |
| --- |
| DELETE FROM ORDER\_DTL WHERE "id" = '000001'; |

#### 导入测试数据

为了方便我们做更多的查询，将以下SQL语句复制到Phoenix客户端中执行。

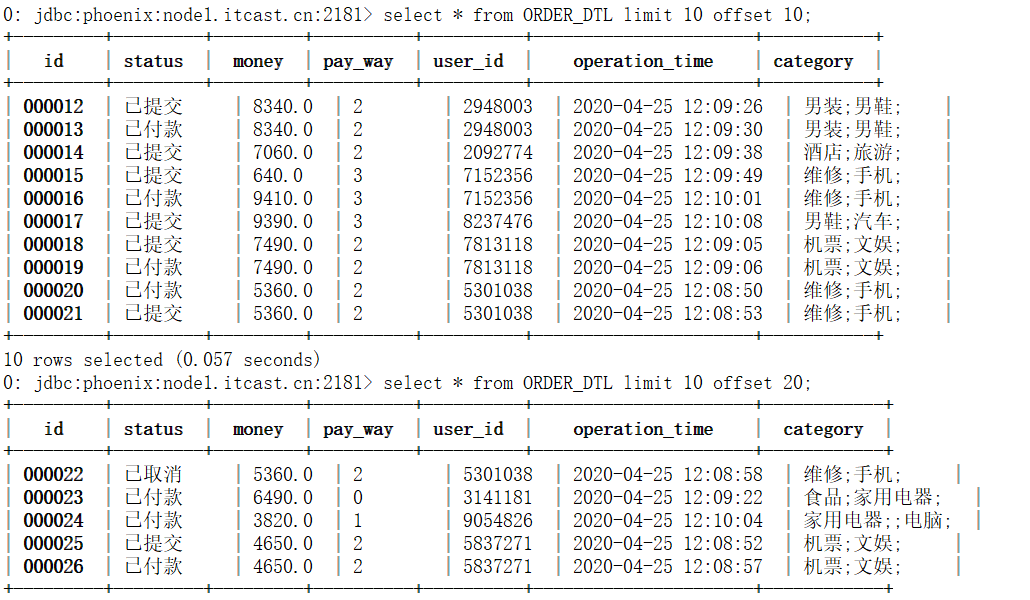
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | UPSERT INTO "ORDER\_DTL" VALUES('000002','已提交',4070,1,'4944191','2020-04-25 12:09:16','手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000003','已完成',4350,1,'1625615','2020-04-25 12:09:37','家用电器;;电脑;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000004','已提交',6370,3,'3919700','2020-04-25 12:09:39','男装;男鞋;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000005','已付款',6370,3,'3919700','2020-04-25 12:09:44','男装;男鞋;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000006','已提交',9380,1,'2993700','2020-04-25 12:09:41','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000007','已付款',9380,1,'2993700','2020-04-25 12:09:46','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000008','已完成',6400,2,'5037058','2020-04-25 12:10:13','数码;女装;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000009','已付款',280,1,'3018827','2020-04-25 12:09:53','男鞋;汽车;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000010','已完成',5600,1,'6489579','2020-04-25 12:08:55','食品;家用电器;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000011','已付款',5600,1,'6489579','2020-04-25 12:09:00','食品;家用电器;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000012','已提交',8340,2,'2948003','2020-04-25 12:09:26','男装;男鞋;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000013','已付款',8340,2,'2948003','2020-04-25 12:09:30','男装;男鞋;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000014','已提交',7060,2,'2092774','2020-04-25 12:09:38','酒店;旅游;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000015','已提交',640,3,'7152356','2020-04-25 12:09:49','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000016','已付款',9410,3,'7152356','2020-04-25 12:10:01','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000017','已提交',9390,3,'8237476','2020-04-25 12:10:08','男鞋;汽车;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000018','已提交',7490,2,'7813118','2020-04-25 12:09:05','机票;文娱;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000019','已付款',7490,2,'7813118','2020-04-25 12:09:06','机票;文娱;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000020','已付款',5360,2,'5301038','2020-04-25 12:08:50','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000021','已提交',5360,2,'5301038','2020-04-25 12:08:53','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000022','已取消',5360,2,'5301038','2020-04-25 12:08:58','维修;手机;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000023','已付款',6490,0,'3141181','2020-04-25 12:09:22','食品;家用电器;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000024','已付款',3820,1,'9054826','2020-04-25 12:10:04','家用电器;;电脑;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000025','已提交',4650,2,'5837271','2020-04-25 12:08:52','机票;文娱;'); | | UPSERT INTO "ORDER\_DTL" VALUES('000026','已付款',4650,2,'5837271','2020-04-25 12:08:57','机票;文娱;'); | |

#### 分页查询

使用limit和offset可以快速进行分页。

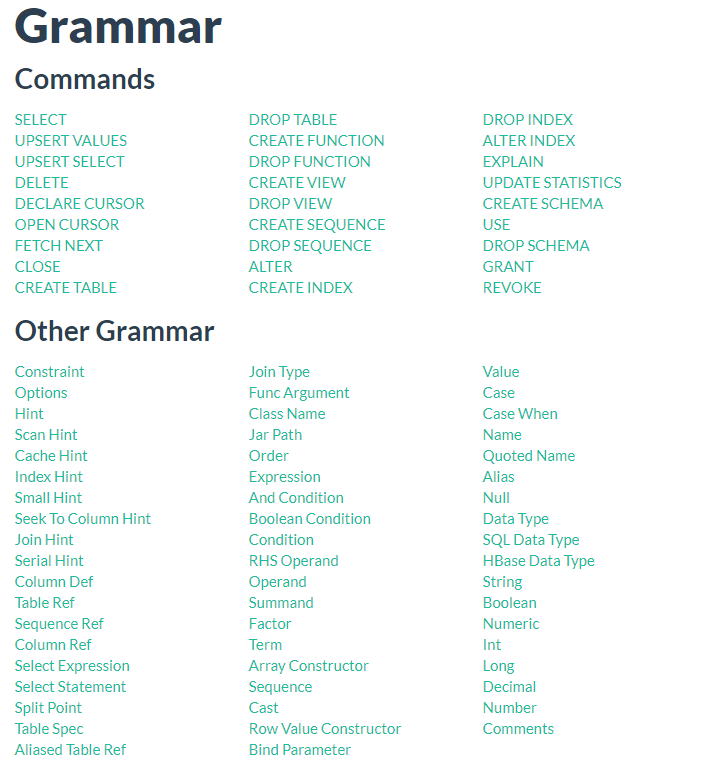
limit表示每页多少条记录，offset表示从第几条记录开始查起。

|  |
| --- |
| -- 第一页  select \* from ORDER\_DTL limit 10 offset 0;  -- 第二页  -- offset从10开始  select \* from ORDER\_DTL limit 10 offset 10;  -- 第三页  select \* from ORDER\_DTL limit 10 offset 20; |



#### 更多语法

<http://phoenix.apache.org/language/index.html#delete>



### 预分区表

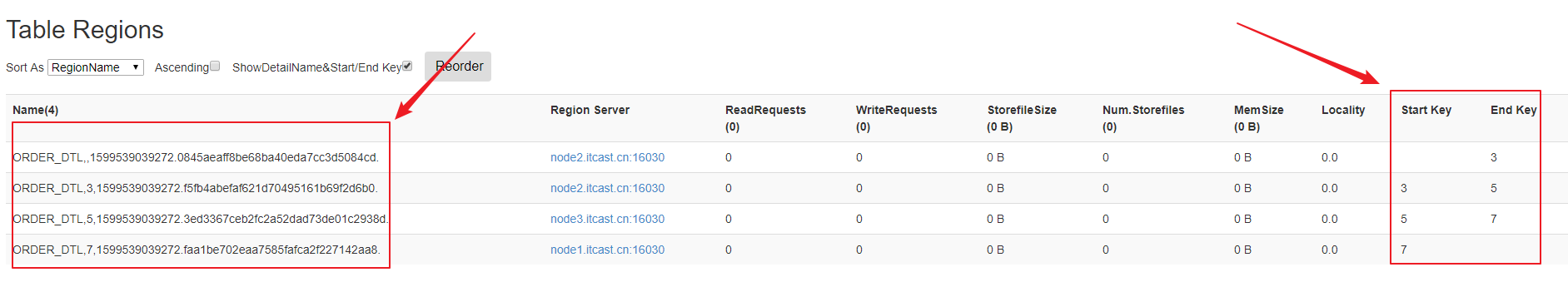
默认创建表的方式，则HBase顺序写入可能会受到RegionServer热点的影响。对行键进行加盐可以解决热点问题。在HBase中，可以使用两种方式：

1. ROWKEY预分区
2. 加盐指定数量分区

#### ROWKEY预分区

按照用户ID来分区，一共4个分区。并指定数据的压缩格式为GZ。

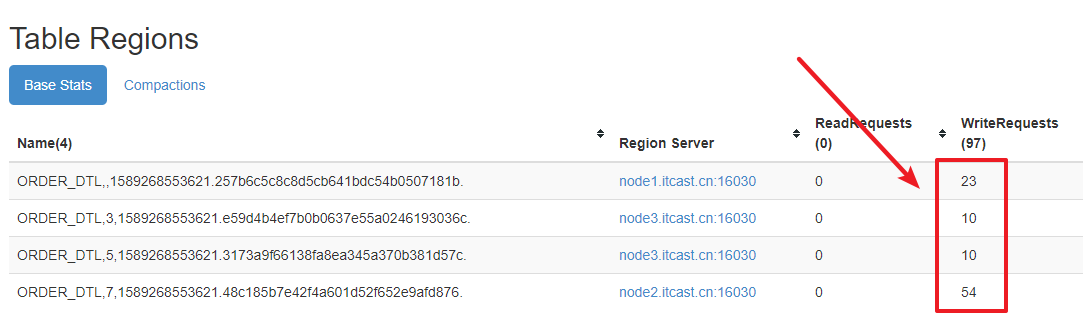
|  |
| --- |
| drop table if exists ORDER\_DTL;  create table if not exists ORDER\_DTL(      "id" varchar primary key,      C1."status" varchar,      C1."money" float,      C1."pay\_way" integer,      C1."user\_id" varchar,      C1."operation\_time" varchar,      C1."category" varchar  )  CONPRESSION='GZ'  SPLIT ON ('3','5','7'); |



我们尝试往表中插入一些数据，然后去HBase中查看数据的分布情况。

|  |
| --- |
| UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'02602f66-adc7-40d4-8485-76b5632b5b53'**,**'已提交'**,4070,1,**'4944191'**,**'2020-04-25 12:09:16'**,**'手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'0968a418-f2bc-49b4-b9a9-2157cf214cfd'**,**'已完成'**,4350,1,**'1625615'**,**'2020-04-25 12:09:37'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'0e01edba-5e55-425e-837a-7efb91c56630'**,**'已提交'**,6370,3,**'3919700'**,**'2020-04-25 12:09:39'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'0e01edba-5e55-425e-837a-7efb91c56630'**,**'已付款'**,6370,3,**'3919700'**,**'2020-04-25 12:09:44'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'0f46d542-34cb-4ef4-b7fe-6dcfa5f14751'**,**'已提交'**,9380,1,**'2993700'**,**'2020-04-25 12:09:41'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'0f46d542-34cb-4ef4-b7fe-6dcfa5f14751'**,**'已付款'**,9380,1,**'2993700'**,**'2020-04-25 12:09:46'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'1fb7c50f-9e26-4aa8-a140-a03d0de78729'**,**'已完成'**,6400,2,**'5037058'**,**'2020-04-25 12:10:13'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'23275016-996b-420c-8edc-3e3b41de1aee'**,**'已付款'**,280,1,**'3018827'**,**'2020-04-25 12:09:53'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2375a7cf-c206-4ac0-8de4-863e7ffae27b'**,**'已完成'**,5600,1,**'6489579'**,**'2020-04-25 12:08:55'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2375a7cf-c206-4ac0-8de4-863e7ffae27b'**,**'已付款'**,5600,1,**'6489579'**,**'2020-04-25 12:09:00'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'269fe10c-740b-4fdb-ad25-7939094073de'**,**'已提交'**,8340,2,**'2948003'**,**'2020-04-25 12:09:26'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'269fe10c-740b-4fdb-ad25-7939094073de'**,**'已付款'**,8340,2,**'2948003'**,**'2020-04-25 12:09:30'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2849fa34-6513-44d6-8f66-97bccb3a31a1'**,**'已提交'**,7060,2,**'2092774'**,**'2020-04-25 12:09:38'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'28b7e793-6d14-455b-91b3-0bd8b23b610c'**,**'已提交'**,640,3,**'7152356'**,**'2020-04-25 12:09:49'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'28b7e793-6d14-455b-91b3-0bd8b23b610c'**,**'已付款'**,9410,3,**'7152356'**,**'2020-04-25 12:10:01'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2909b28a-5085-4f1d-b01e-a34fbaf6ce37'**,**'已提交'**,9390,3,**'8237476'**,**'2020-04-25 12:10:08'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2a01dfe5-f5dc-4140-b31b-a6ee27a6e51e'**,**'已提交'**,7490,2,**'7813118'**,**'2020-04-25 12:09:05'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2a01dfe5-f5dc-4140-b31b-a6ee27a6e51e'**,**'已付款'**,7490,2,**'7813118'**,**'2020-04-25 12:09:06'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2b86ab90-3180-4940-b624-c936a1e7568d'**,**'已付款'**,5360,2,**'5301038'**,**'2020-04-25 12:08:50'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2b86ab90-3180-4940-b624-c936a1e7568d'**,**'已提交'**,5360,2,**'5301038'**,**'2020-04-25 12:08:53'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2b86ab90-3180-4940-b624-c936a1e7568d'**,**'已取消'**,5360,2,**'5301038'**,**'2020-04-25 12:08:58'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2e19fbe8-7970-4d62-8e8f-d364afc2dd41'**,**'已付款'**,6490,0,**'3141181'**,**'2020-04-25 12:09:22'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'2fc28d36-dca0-49e8-bad0-42d0602bdb40'**,**'已付款'**,3820,1,**'9054826'**,**'2020-04-25 12:10:04'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'31477850-8b15-4f1b-9ec3-939f7dc47241'**,**'已提交'**,4650,2,**'5837271'**,**'2020-04-25 12:08:52'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'31477850-8b15-4f1b-9ec3-939f7dc47241'**,**'已付款'**,4650,2,**'5837271'**,**'2020-04-25 12:08:57'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'39319322-2d80-41e7-a862-8b8858e63316'**,**'已提交'**,5000,1,**'5686435'**,**'2020-04-25 12:08:51'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'39319322-2d80-41e7-a862-8b8858e63316'**,**'已完成'**,5000,1,**'5686435'**,**'2020-04-25 12:08:56'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'3d2254bd-c25a-404f-8e42-2faa4929a629'**,**'已提交'**,5000,3,**'1274270'**,**'2020-04-25 12:08:41'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'3d2254bd-c25a-404f-8e42-2faa4929a629'**,**'已付款'**,5000,3,**'1274270'**,**'2020-04-25 12:08:42'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'3d2254bd-c25a-404f-8e42-2faa4929a629'**,**'已完成'**,5000,1,**'1274270'**,**'2020-04-25 12:08:43'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'42f7fe21-55a3-416f-9535-baa222cc0098'**,**'已完成'**,3600,2,**'2661641'**,**'2020-04-25 12:09:58'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'44231dbb-9e58-4f1a-8c83-be1aa814be83'**,**'已提交'**,3950,1,**'3855371'**,**'2020-04-25 12:08:39'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'44231dbb-9e58-4f1a-8c83-be1aa814be83'**,**'已付款'**,3950,1,**'3855371'**,**'2020-04-25 12:08:40'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'526e33d2-a095-4e19-b759-0017b13666ca'**,**'已完成'**,3280,0,**'5553283'**,**'2020-04-25 12:09:01'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'5a6932f4-b4a4-4a1a-b082-2475d13f9240'**,**'已提交'**,50,2,**'1764961'**,**'2020-04-25 12:10:07'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'5fc0093c-59a3-417b-a9ff-104b9789b530'**,**'已提交'**,6310,2,**'1292805'**,**'2020-04-25 12:09:36'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'605c6dd8-123b-4088-a047-e9f377fcd866'**,**'已完成'**,8980,2,**'6202324'**,**'2020-04-25 12:09:54'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'613cfd50-55c7-44d2-bb67-995f72c488ea'**,**'已完成'**,6830,3,**'6977236'**,**'2020-04-25 12:10:06'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'62246ac1-3dcb-4f2c-8943-800c9216c29f'**,**'已提交'**,8610,1,**'5264116'**,**'2020-04-25 12:09:14'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'62246ac1-3dcb-4f2c-8943-800c9216c29f'**,**'已付款'**,8610,1,**'5264116'**,**'2020-04-25 12:09:18'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'625c7fef-de87-428a-b581-a63c71059b14'**,**'已提交'**,5970,0,**'8051757'**,**'2020-04-25 12:09:07'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'625c7fef-de87-428a-b581-a63c71059b14'**,**'已付款'**,5970,0,**'8051757'**,**'2020-04-25 12:09:19'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'6d43c490-58ab-4e23-b399-dda862e06481'**,**'已提交'**,4570,0,**'5514248'**,**'2020-04-25 12:09:34'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'70fa0ae0-6c02-4cfa-91a9-6ad929fe6b1b'**,**'已付款'**,4100,1,**'8598963'**,**'2020-04-25 12:09:08'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'7170ce71-1fc0-4b6e-a339-67f525536dcd'**,**'已完成'**,9740,1,**'4816392'**,**'2020-04-25 12:09:51'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'7170ce71-1fc0-4b6e-a339-67f525536dcd'**,**'已提交'**,9740,1,**'4816392'**,**'2020-04-25 12:10:03'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'71961b06-290b-457d-bbe0-86acb013b0e3'**,**'已付款'**,6550,3,**'2393699'**,**'2020-04-25 12:08:47'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'71961b06-290b-457d-bbe0-86acb013b0e3'**,**'已付款'**,6550,3,**'2393699'**,**'2020-04-25 12:08:48'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'71961b06-290b-457d-bbe0-86acb013b0e3'**,**'已完成'**,6550,3,**'2393699'**,**'2020-04-25 12:08:49'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'72dc148e-ce64-432d-b99f-61c389cb82cd'**,**'已提交'**,4090,1,**'2536942'**,**'2020-04-25 12:10:12'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'72dc148e-ce64-432d-b99f-61c389cb82cd'**,**'已付款'**,4090,1,**'2536942'**,**'2020-04-25 12:10:14'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'7c0c1668-b783-413f-afc4-678a5a6d1033'**,**'已完成'**,3850,3,**'6803936'**,**'2020-04-25 12:09:20'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'7fa02f7a-10df-4247-9935-94c8b7d4dbc0'**,**'已提交'**,1060,0,**'6119810'**,**'2020-04-25 12:09:21'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'820c5e83-f2e0-42d4-b5f0-83802c75addc'**,**'已付款'**,9270,2,**'5818454'**,**'2020-04-25 12:10:09'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'83ed55ec-a439-44e0-8fe0-acb7703fb691'**,**'已完成'**,8380,2,**'6804703'**,**'2020-04-25 12:09:52'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'85287268-f139-4d59-8087-23fa6454de9d'**,**'已提交'**,9750,1,**'4382852'**,**'2020-04-25 12:09:43'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'85287268-f139-4d59-8087-23fa6454de9d'**,**'已付款'**,9750,1,**'4382852'**,**'2020-04-25 12:09:48'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'85287268-f139-4d59-8087-23fa6454de9d'**,**'已取消'**,9750,1,**'4382852'**,**'2020-04-25 12:10:00'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'8d32669e-327a-4802-89f4-2e91303aee59'**,**'已提交'**,9390,1,**'4182962'**,**'2020-04-25 12:09:57'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'8dadc2e4-63f1-490f-9182-793be64fed76'**,**'已付款'**,9350,1,**'5937549'**,**'2020-04-25 12:09:02'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'94ad8ee0-8898-442c-8cb1-083a4b609616'**,**'已提交'**,4370,0,**'4666456'**,**'2020-04-25 12:09:13'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'994cbb44-f0ee-45ff-a4f4-76c87bc2b972'**,**'已付款'**,3190,3,**'3200759'**,**'2020-04-25 12:09:25'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'9bf92519-6eb3-449a-853b-0e19f6005887'**,**'已提交'**,1100,0,**'3457528'**,**'2020-04-25 12:10:11'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'9ff3032c-8679-4247-9e6f-4caf2dc93aff'**,**'已提交'**,850,0,**'8835231'**,**'2020-04-25 12:09:40'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'9ff3032c-8679-4247-9e6f-4caf2dc93aff'**,**'已付款'**,850,0,**'8835231'**,**'2020-04-25 12:09:45'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'a467ba42-f91e-48a0-865e-1703aaa45e0e'**,**'已提交'**,8040,0,**'8206022'**,**'2020-04-25 12:09:50'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'a467ba42-f91e-48a0-865e-1703aaa45e0e'**,**'已付款'**,8040,0,**'8206022'**,**'2020-04-25 12:10:02'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'a5302f47-96d9-41b4-a14c-c7a508f59282'**,**'已付款'**,8570,2,**'5319315'**,**'2020-04-25 12:08:44'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'a5b57bec-6235-45f4-bd7e-6deb5cd1e008'**,**'已提交'**,5700,3,**'6486444'**,**'2020-04-25 12:09:27'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'a5b57bec-6235-45f4-bd7e-6deb5cd1e008'**,**'已付款'**,5700,3,**'6486444'**,**'2020-04-25 12:09:31'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ae5c3363-cf8f-48a9-9676-701a7b0a7ca5'**,**'已付款'**,7460,1,**'2379296'**,**'2020-04-25 12:09:23'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'b1fb2399-7cf2-4af5-960a-a4d77f4803b8'**,**'已提交'**,2690,3,**'6686018'**,**'2020-04-25 12:09:55'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'b21c7dbd-dabd-4610-94b9-d7039866a8eb'**,**'已提交'**,6310,2,**'1552851'**,**'2020-04-25 12:09:15'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'b4bfd4b7-51f5-480e-9e23-8b1579e36248'**,**'已提交'**,4000,1,**'3260372'**,**'2020-04-25 12:09:35'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'b63983cc-2b59-4992-84c6-9810526d0282'**,**'已提交'**,7370,3,**'3107867'**,**'2020-04-25 12:08:45'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'b63983cc-2b59-4992-84c6-9810526d0282'**,**'已付款'**,7370,3,**'3107867'**,**'2020-04-25 12:08:46'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'bf60b752-1ccc-43bf-9bc3-b2aeccacc0ed'**,**'已提交'**,720,2,**'5034117'**,**'2020-04-25 12:09:03'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'c808addc-8b8b-4d89-99b1-db2ed52e61b4'**,**'已提交'**,3630,1,**'6435854'**,**'2020-04-25 12:09:10'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'cc9dbd20-cf9f-4097-ae8b-4e73db1e4ba1'**,**'已付款'**,5000,0,**'2007322'**,**'2020-04-25 12:08:38'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ccceaf57-a5ab-44df-834a-e7b32c63efc1'**,**'已提交'**,2660,2,**'7928516'**,**'2020-04-25 12:09:42'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ccceaf57-a5ab-44df-834a-e7b32c63efc1'**,**'已付款'**,2660,2,**'7928516'**,**'2020-04-25 12:09:47'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ccceaf57-a5ab-44df-834a-e7b32c63efc1'**,**'已完成'**,2660,2,**'7928516'**,**'2020-04-25 12:09:59'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'d7be5c39-e07c-40e8-bf09-4922fbc6335c'**,**'已付款'**,8750,2,**'1250995'**,**'2020-04-25 12:09:09'**,**'食品;家用电器;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'dfe16df7-4a46-4b6f-9c6d-083ec215218e'**,**'已完成'**,410,0,**'1923817'**,**'2020-04-25 12:09:56'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e1241ad4-c9c1-4c17-93b9-ef2c26e7f2b2'**,**'已付款'**,6760,0,**'2457464'**,**'2020-04-25 12:08:54'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e1241ad4-c9c1-4c17-93b9-ef2c26e7f2b2'**,**'已提交'**,6760,0,**'2457464'**,**'2020-04-25 12:08:59'**,**'数码;女装;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e180a9f2-9f80-4b6d-99c8-452d6c037fc7'**,**'已付款'**,8120,2,**'7645270'**,**'2020-04-25 12:09:28'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e180a9f2-9f80-4b6d-99c8-452d6c037fc7'**,**'已完成'**,8120,2,**'7645270'**,**'2020-04-25 12:09:32'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e4418843-9ac0-47a7-bfd8-d61c4d296933'**,**'已付款'**,8170,2,**'7695668'**,**'2020-04-25 12:09:11'**,**'家用电器;;电脑;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'e8b3bb37-1019-4492-93c7-305177271a71'**,**'已完成'**,2560,2,**'4405460'**,**'2020-04-25 12:10:05'**,**'男装;男鞋;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'eb1a1a22-953a-42f1-b594-f5dfc8fb6262'**,**'已完成'**,2370,2,**'8233485'**,**'2020-04-25 12:09:24'**,**'机票;文娱;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ecfd18f5-45f2-4dcd-9c47-f2ad9b216bd0'**,**'已付款'**,8070,3,**'6387107'**,**'2020-04-25 12:09:04'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'ecfd18f5-45f2-4dcd-9c47-f2ad9b216bd0'**,**'已完成'**,8070,3,**'6387107'**,**'2020-04-25 12:09:17'**,**'酒店;旅游;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'f1226752-7be3-4702-a496-3ddba56f66ec'**,**'已付款'**,4410,3,**'1981968'**,**'2020-04-25 12:10:10'**,**'维修;手机;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'f642b16b-eade-4169-9eeb-4d5f294ec594'**,**'已提交'**,4010,1,**'6463215'**,**'2020-04-25 12:09:29'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'f642b16b-eade-4169-9eeb-4d5f294ec594'**,**'已付款'**,4010,1,**'6463215'**,**'2020-04-25 12:09:33'**,**'男鞋;汽车;'**);  UPSERT **INTO** **"ORDER\_DTL"** **VALUES**(**'f8f3ca6f-2f5c-44fd-9755-1792de183845'**,**'已付款'**,5950,3,**'4060214'**,**'2020-04-25 12:09:12'**,**'机票;文娱;'**); |

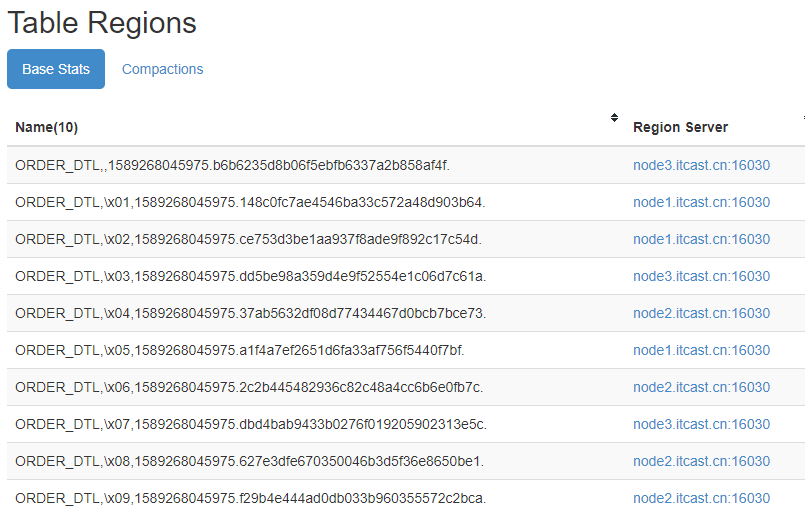
我们发现数据分布在每一个Region中。



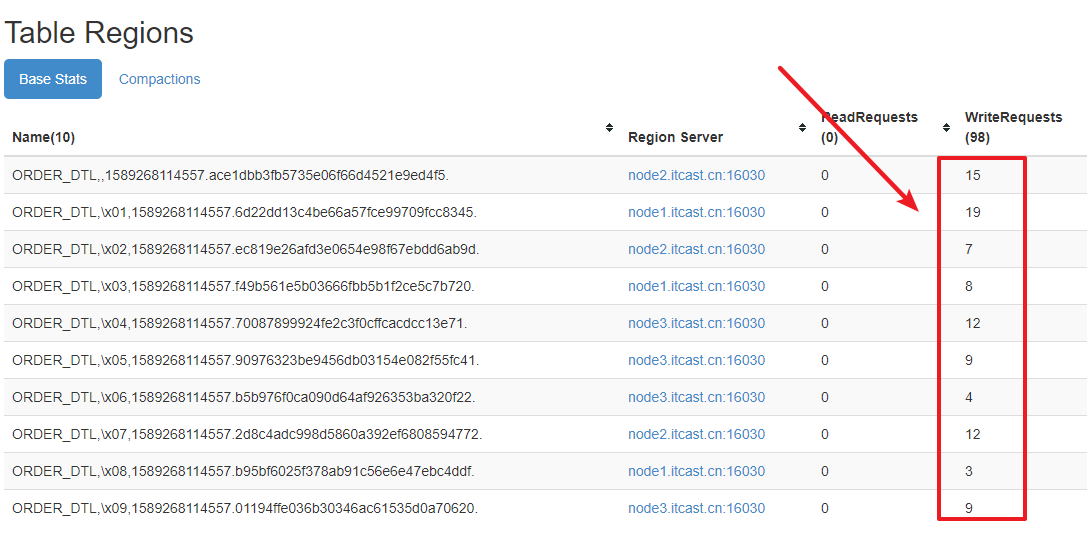
#### 加盐指定数量分区

|  |
| --- |
| drop table if exists ORDER\_DTL;  create table if not exists ORDER\_DTL(      "id" varchar primary key,      C1."status" varchar,      C1."money" float,      C1."pay\_way" integer,      C1."user\_id" varchar,      C1."operation\_time" varchar,      C1."category" varchar  )  CONPRESSION='GZ', SALT\_BUCKETS=10; |

我们在HBase的Web UI中可以查看到生成了10个Region



插入数据后，发现数据分部在每一个Region中。



查看HBase中的表，我们发现Phoenix在每个ID前，都添加了一个Hash值，用来将分布分布到不同的Region中。

|  |
| --- |
| hbase(main):018:0> scan "ORDER\_DTL", {LIMIT => 1}  ROW COLUMN+CELL  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x00\x00\x00\x00, timestamp=1589268724801, value=x  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0B, timestamp=1589268724801, value=\xE5\xB7\xB2\xE4\xBB\x98\xE6\xAC\xBE  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0C, timestamp=1589268724801, value=\xC6\x12\x90\x01  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0D, timestamp=1589268724801, value=\x80\x00\x00\x01  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0E, timestamp=1589268724801, value=2993700  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0F, timestamp=1589268724801, value=2020-04-25 12:09:46  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x10, timestamp=1589268724801, value=\xE7\xBB\xB4\xE4\xBF\xAE;\xE6\x89\x8B\xE6\x9C\xBA;  1 row(s) |

注意：

* CONPRESSION和SALT\_BUCKETS之间需要使用逗号分隔，否则会出现语法错误

## 基于Phoenix消息数据查询

### 建立视图

#### 应用场景

因为我们之前已经创建了CHAT:MSG 表，而且数据添加的方式都是以PUT方式原生API来添加的。故此时，我们不再需要再使用Phoenix创建新的表，而是使用Phoenix中的**视图**，通过视图来建立与HBase表之间的映射，从而实现数据快速查询。

#### 视图介绍

我们可以在现有的HBase或Phoenix表上创建一个视图。表、列蔟和列名必须与现有元数据完全匹配，否则会出现异常。当创建视图后，就可以使用SQL查询视图，和操作Table一样。

语法示例：

|  |
| --- |
| -- 映射HBase中的表  CREATE VIEW "my\_hbase\_table"     ( k VARCHAR primary key, "v" UNSIGNED\_LONG) default\_column\_family='a';  -- 映射Phoenix中的表  CREATE VIEW my\_view ( new\_col SMALLINT )     AS SELECT \* FROM my\_table WHERE k = 100;  -- 映射到一个SQL查询  CREATE VIEW my\_view\_on\_view     AS SELECT \* FROM my\_view WHERE new\_col > 70; |

#### 建立CHAT:MSG的视图

考虑以下几个问题：

1. 视图如何映射到HBase的表？

* 视图的名字必须是：命名空间.表名

1. 视图中的列如何映射到HBase的列蔟和列？

* 列名必须是：列蔟.列名

1. 视图中的类如何映射到HBase的ROWKEY？

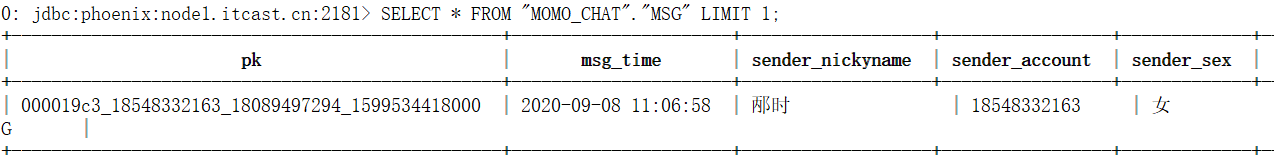
* 指定某个列为primary key，自动映射ROWKEY

参考创建语句：

|  |
| --- |
| *-- 创建CHAT:MSG视图*  **create** view **if** **not** **exists** **"** CHAT**"**. **"**MSG**"** (  **"pk"** **varchar** **primary** **key**, *-- 指定ROWKEY映射到主键*  **"C1"**.**"msg\_time"** **varchar**,  **"C1"**.**"sender\_nickyname"** **varchar**,  **"C1"**.**"sender\_account"** **varchar**,  **"C1"**.**"sender\_sex"** **varchar**,  **"C1"**.**"sender\_ip"** **varchar**,  **"C1"**.**"sender\_os"** **varchar**,  **"C1"**.**"sender\_phone\_type"** **varchar**,  **"C1"**.**"sender\_network"** **varchar**,  **"C1"**.**"sender\_gps"** **varchar**,  **"C1"**.**"receiver\_nickyname"** **varchar**,  **"C1"**.**"receiver\_ip"** **varchar**,  **"C1"**.**"receiver\_account"** **varchar**,  **"C1"**.**"receiver\_os"** **varchar**,  **"C1"**.**"receiver\_phone\_type"** **varchar**,  **"C1"**.**"receiver\_network"** **varchar**,  **"C1"**.**"receiver\_gps"** **varchar**,  **"C1"**.**"receiver\_sex"** **varchar**,  **"C1"**.**"msg\_type"** **varchar**,  **"C1"**.**"distance"** **varchar**  ); |

#### 尝试查询一条数据

|  |
| --- |
| SELECT \* FROM " CHAT"."MSG" LIMIT 1; |



如果发现数据能够正常展示，说明视图映射已经成功。

注意：

* 因为列名中有小写，需要用引号将字段名包含起来

### 开发基于SQL查询数据接口

#### 使用SQL语句查询数据

##### 需求

根据日期、发送人账号、接收人账号查询历史消息

##### 编写SQL语句

|  |
| --- |
| *-- 查询对应日期的数据（只展示出来5条）*  SELECT \* FROM "CHAT"."MSG" T  WHERE substr("msg\_time", 0, 10) = '2020-08-29'      AND T."sender\_account" = '13504113666'      AND T."receiver\_account" = '18182767005' LIMIT 100; |

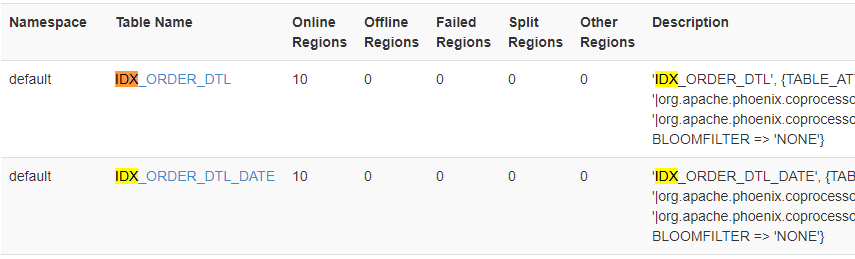
##### 编写Java代码

1. 编写PhoenixChatMessageService实现ChatMessageService接口
2. 在构造器中创建JDBC连接
   1. JDBC驱动为：PhoenixDriver.class.getName()
   2. JDBC连接URL为：jdbc:phoenix:node1:2181
3. 基于JDBC实现getMessage查询
4. 在close方法中

|  |
| --- |
| **public class** PhoenixChatMessageService **implements** ChatMessageService {  **private** Connection **connection**;   **public** PhoenixChatMessageService() {  **try** {  Class.*forName*(PhoenixDriver.**class**.getName());  **connection** = DriverManager.*getConnection*(**"jdbc:phoenix:node1:2181"**);  } **catch** (ClassNotFoundException e) {  **throw new** RuntimeException(**"加载Phoenix驱动失败!"**);  } **catch** (SQLException e) {  **throw new** RuntimeException(**"获取Phoenix JDBC连接失败!"**);  }  }   @Override  **public** List<Msg> getMessage(String date, String sender, String receiver) **throws** Exception {  PreparedStatement ps = **connection**.prepareStatement(  **"SELECT** *\** **FROM CHAT.MSG T WHERE substr(\"msg\_time\", 0, 10) = ? "** + **"AND T.\"sender\_account\" = ? "** + **"AND T.\"receiver\_account\" = ? "**);   ps.setString(1, date);  ps.setString(2, sender);  ps.setString(3, receiver);   ResultSet rs = ps.executeQuery();  List<Msg> msgList = **new** ArrayList<>();   **while**(rs.next()) {  Msg msg = **new** Msg();  msg.setMsg\_time(rs.getString(**"msg\_time"**));  msg.setSender\_nickyname(rs.getString(**"sender\_nickyname"**));  msg.setSender\_account(rs.getString(**"sender\_account"**));  msg.setSender\_sex(rs.getString(**"sender\_sex"**));  msg.setSender\_ip(rs.getString(**"sender\_ip"**));  msg.setSender\_os(rs.getString(**"sender\_os"**));  msg.setSender\_phone\_type(rs.getString(**"sender\_phone\_type"**));  msg.setSender\_network(rs.getString(**"sender\_network"**));  msg.setSender\_gps(rs.getString(**"sender\_gps"**));  msg.setReceiver\_nickyname(rs.getString(**"receiver\_nickyname"**));  msg.setReceiver\_ip(rs.getString(**"receiver\_ip"**));  msg.setReceiver\_account(rs.getString(**"receiver\_account"**));  msg.setReceiver\_os(rs.getString(**"receiver\_os"**));  msg.setReceiver\_phone\_type(rs.getString(**"receiver\_phone\_type"**));  msg.setReceiver\_network(rs.getString(**"receiver\_network"**));  msg.setReceiver\_gps(rs.getString(**"receiver\_gps"**));  msg.setReceiver\_sex(rs.getString(**"receiver\_sex"**));  msg.setMsg\_type(rs.getString(**"msg\_type"**));  msg.setDistance(rs.getString(**"distance"**));   msgList.add(msg);  }   **return** msgList;  }   @Override  **public void** close() {  **try** {  **connection**.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }   **public static void** main(String[] args) **throws** Exception {  ChatMessageService chatMessageService = **new** PhoenixChatMessageService();  List<Msg> message = chatMessageService.getMessage(**"2020-08-24"**, **"13504113666"**, **"18182767005"**);   **for** (Msg msg : message) {  System.***out***.println(msg);  }   chatMessageService.close();  } } |

### 二级索引

上面的查询，因为没有建立索引，组合条件查询效率较低，而通过使用Phoenix，我们可以非常方便地创建二级索引。Phoenix中的索引，其实底层还是表现为HBase中的表结构。这些索引表专门用来加快查询速度。



#### 索引分类

1. 全局索引
2. 本地索引
3. 覆盖索引
4. 函数索引

##### 全局索引

* 全局索引适用于读多写少业务
* 全局索引绝大多数负载都发生在写入时，当构建了全局索引时，Phoenix会拦截写入(DELETE、UPSERT值和UPSERT SELECT)上的数据表更新，构建索引更新，同时更新所有相关的索引表，开销较大
* 读取时，Phoenix将选择最快能够查询出数据的索引表。默认情况下，除非使用Hint，如果SELECT查询中引用了其他非索引列，该索引是不会生效的
* 全局索引一般和覆盖索引搭配使用，读的效率很高，但写入效率会受影响

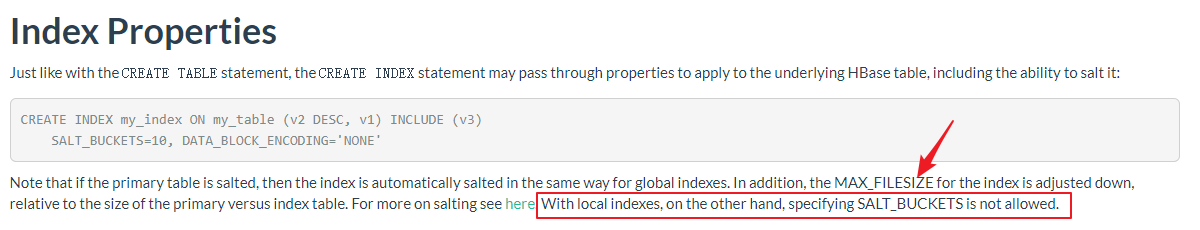
创建语法：

|  |
| --- |
| CREATE INDEX 索引名称 ON 表名 (列名1, 列名2, 列名3...) |

##### 本地索引

* 本地索引适合写操作频繁，读相对少的业务
* 当使用SQL查询数据时，Phoenix会自动选择是否使用本地索引查询数据
* 在本地索引中，索引数据和业务表数据存储在同一个服务器上，避免写入期间的其他网络开销
* 在Phoenix 4.8.0之前，本地索引保存在一个单独的表中，在Phoenix 4.8.1中，本地索引的数据是保存在一个影子列蔟中
* 本地索引查询即使SELECT引用了非索引中的字段，也会自动应用索引的

注意：创建表的时候指定了SALT\_BUCKETS，是不支持本地索引的。



创建语法：

|  |
| --- |
| CREATE local INDEX 索引名称 ON 表名 (列名1, 列名2, 列名3...) |

##### 覆盖索引

Phoenix提供了覆盖的索引，可以不需要在找到索引条目后返回到主表。Phoenix可以将关心的数据捆绑在索引行中，从而节省了读取时间的开销。

例如，以下语法将在v1和v2列上创建索引，并在索引中包括v3列，也就是通过v1、v2就可以直接把数据查询出来。

|  |
| --- |
| CREATE INDEX my\_index ON my\_table (v1,v2) INCLUDE(v3) |

##### 函数索引

函数索引(4.3和更高版本)可以支持在列上创建索引，还可以基于任意表达式上创建索引。然后，当查询使用该表达式时，可以使用索引来检索结果，而不是数据表。例如，可以在UPPER(FIRST\_NAME||‘ ’||LAST\_NAME)上创建一个索引，这样将来搜索两个名字拼接在一起时，索引依然可以生效。

|  |
| --- |
| -- 创建索引  CREATE INDEX UPPER\_NAME\_IDX ON EMP (UPPER(FIRST\_NAME||' '||LAST\_NAME))  -- 以下查询会走索引  SELECT EMP\_ID FROM EMP WHERE UPPER(FIRST\_NAME||' '||LAST\_NAME)='JOHN DOE' |

#### 索引示例一：创建全局索引 + 覆盖索引

##### 需求

我们需要根据用户ID来查询订单的ID以及对应的支付金额。例如：查询已付款的订单ID和支付金额

此时，就可以在USER\_ID列上创建索引，来加快查询

##### 创建索引

|  |
| --- |
| create index GBL\_IDX\_ORDER\_DTL on ORDER\_DTL(C1."user\_id") INCLUDE("id", C1."money"); |

可以在HBase shell中看到，Phoenix自动帮助我们创建了一张GBL\_IDX\_ORDER\_DTL的表。这种表就是一张索引表。它的数据如下：

|  |
| --- |
| hbase(main):005:0> **scan "GBL\_IDX\_ORDER\_DTL", { LIMIT => 1}**  ROW COLUMN+CELL  1250995\x00d7be5c39-e07c-40e8-bf09-492 column=C1:\x00\x00\x00\x00, timestamp=1589350330650, value=x  2fbc6335c  1250995\x00d7be5c39-e07c-40e8-bf09-492 column=C1:\x80\x0B, timestamp=1589350330650, value=\xC6\x08\xB8\x01  2fbc6335c  1 row(s)  Took 0.1253 seconds |

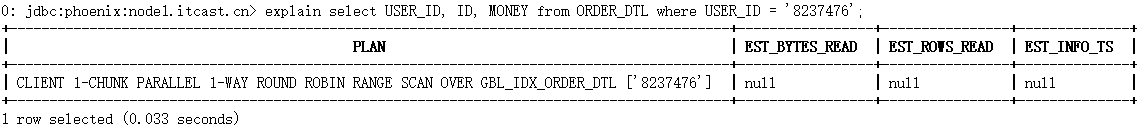
这张表的ROWKEY为：用户ID + \x00 + 原始表ROWKEY，列蔟对应的就是include中指定的两个字段。

##### 查询数据

|  |
| --- |
| select "user\_id", "id", "money" from ORDER\_DTL where "user\_id" = '8237476'; |

##### 查看执行计划

|  |
| --- |
| explain select "user\_id", "id", "money" from ORDER\_DTL where "user\_id" = '8237476'; |



我们发现，PLAN中能看到SCAN的是GBL\_IDX\_ORDER\_DTL，说明Phoenix是直接通过查询索引表获取到数据。

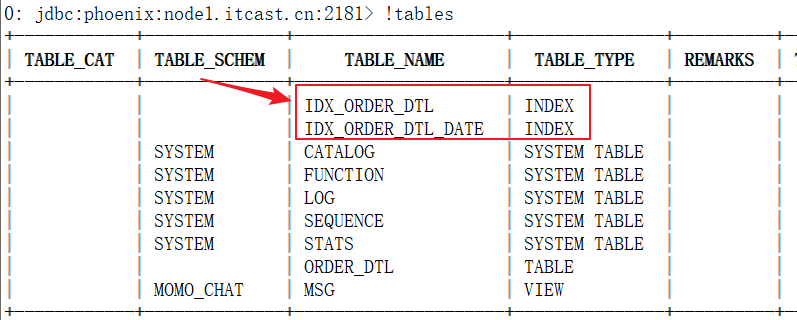
##### 删除索引

使用drop index 索引名 ON 表名

|  |
| --- |
| drop index IDX\_ORDER\_DTL\_DATE on ORDER\_DTL; |

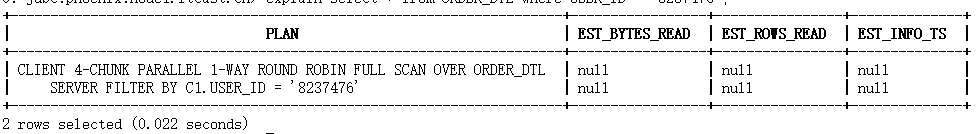
##### 查看索引

|  |
| --- |
| !table |



##### 测试查询所有列是否会使用索引

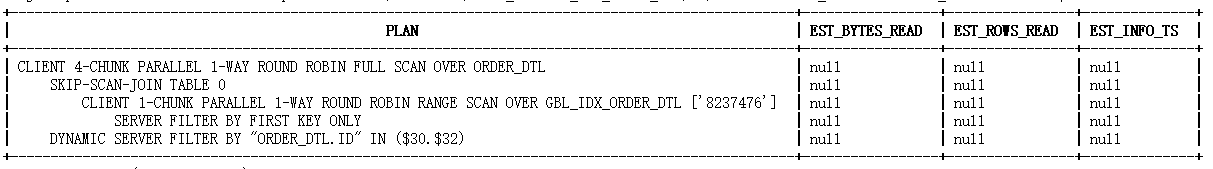
|  |
| --- |
| explain select \* from ORDER\_DTL where "user\_id" = '8237476'; |



通过查询结果发现，PLAN中是执行的FULL SCAN，说明索引并没有生效，进行的全表扫描。

##### 使用Hint强制使用索引

explain select /\*+ INDEX(ORDER\_DTL GBL\_IDX\_ORDER\_DTL) \*/ \* from ORDER\_DTL where USER\_ID = '8237476';



通过执行计划，我们可以观察到查看全局索引，找到ROWKEY，然后执行全表的JOIN，其实就是把对应ROWKEY去查询ORDER\_DTL表。

#### 索引示例二：创建本地索引

##### 需求

在程序中，我们可能会根据订单ID、订单状态、支付金额、支付方式、用户ID来查询订单。所以，我们需要在这些列上来查询订单。

针对这种场景，我们可以使用本地索引来提高查询效率。

##### 创建本地索引

|  |
| --- |
| create local index LOCAL\_IDX\_ORDER\_DTL on ORDER\_DTL("id", "status", "money", "pay\_way", "user\_id") ; |

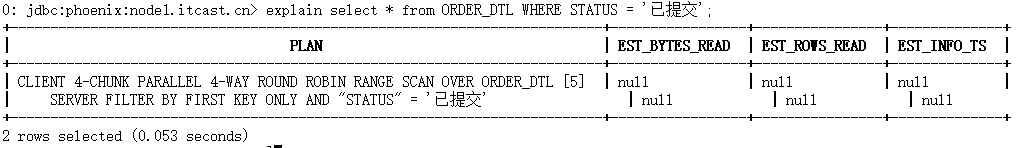
通过查看WebUI，我们并没有发现创建名为：LOCAL\_IDX\_ORDER\_DTL 的表。那索引数据是存储在哪儿呢？我们可以通过HBase shell

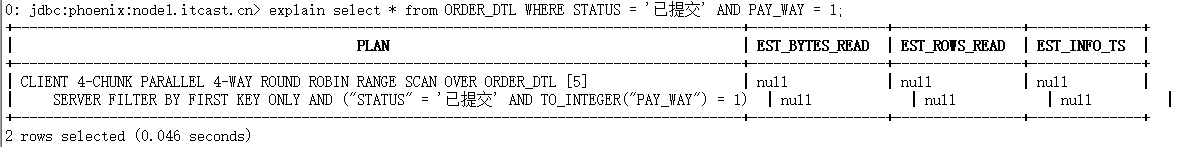
|  |
| --- |
| hbase(main):031:0> **scan "ORDER\_DTL", {LIMIT => 1}**  ROW COLUMN+CELL  \x00\x00\x0402602f66-adc7-40d4-8485-76 column=L#0:\x00\x00\x00\x00, timestamp=1589350314539, value=\x00\x00\x00\x00  b5632b5b53\x00\xE5\xB7\xB2\xE6\x8F\x90  \xE4\xBA\xA4\x00\xC2)G\x00\xC1\x02\x00  4944191  1 row(s)  Took 0.0155 seconds |

可以看到Phoenix对数据进行处理，原有的数据发生了变化。建立了本地二级索引表，不能再使用Hbase的Java API查询，只能通过JDBC来查询。

##### 查看数据

|  |
| --- |
| explain select \* from ORDER\_DTL WHERE "status" = '已提交';  explain select \* from ORDER\_DTL WHERE "status" = '已提交' AND "pay\_way" = 1; |





通过观察上面的两个执行计划发现，两个查询都是通过RANGE SCAN来实现的。说明本地索引生效。

##### 删除本地索引

|  |
| --- |
| drop index LOCAL\_IDX\_ORDER\_DTL on ORDER\_DTL; |

重新执行一次扫描，你会发现数据变魔术般的恢复出来了。

|  |
| --- |
| hbase(main):007:0> **scan "ORDER\_DTL", {LIMIT => 1}**  ROW COLUMN+CELL  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x00\x00\x00\x00, timestamp=1599542260011, value=x  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0B, timestamp=1599542260011, value=\xE5\xB7\xB2\xE4\xBB\x98\xE6\xAC\xBE  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0C, timestamp=1599542260011, value=\xC6\x12\x90\x01  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0D, timestamp=1599542260011, value=\x80\x00\x00\x01  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0E, timestamp=1599542260011, value=2993700  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x0F, timestamp=1599542260011, value=2020-04-25 12:09:46  \x000f46d542-34cb-4ef4-b7fe-6dcfa5f14751 column=C1:\x80\x10, timestamp=1599542260011, value=\xE7\xBB\xB4\xE4\xBF\xAE;\xE6\x89\x8B\xE6\x9C\xBA;  1 row(s)  Took 0.0266 seconds |

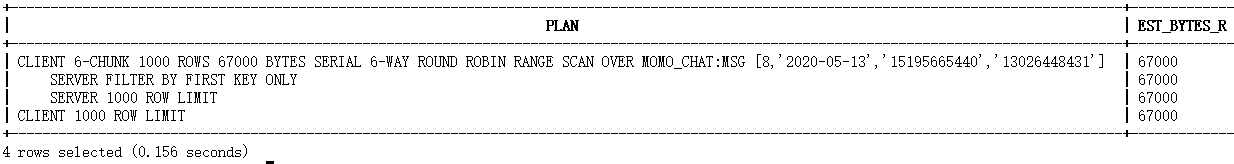
#### 使用Phoenix建立二级索引高效查询

##### 创建本地函数索引

|  |
| --- |
| CREATE LOCAL INDEX LOCAL\_IDX\_MSG ON CHAT.MSG(substr("msg\_time", 0, 10), "sender\_account", "receiver\_account"); |

##### 执行数据查询

|  |
| --- |
| SELECT \* FROM "CHAT"."MSG" T  WHERE substr("msg\_time", 0, 10) = '2020-08-29'  AND T."sender\_account" = '13504113666'  AND T."receiver\_account" = '18182767005' LIMIT 100; |

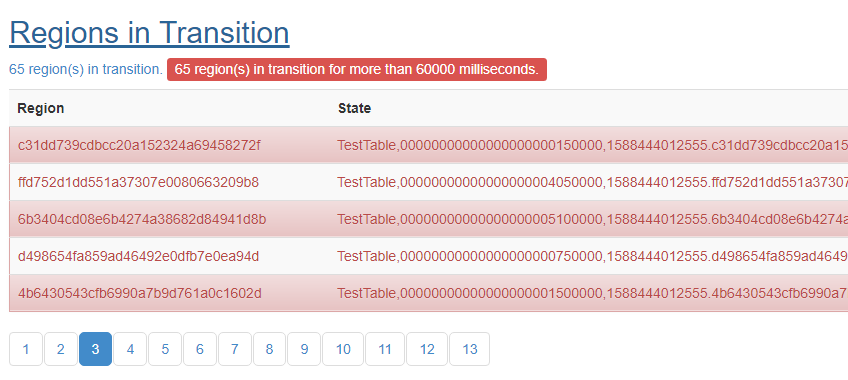


可以看到，查询速度非常快，0.1秒就查询出来了数据。

#### 再次测试Java接口

## 常见问题

### Regions In Transition



错误信息如下：

|  |
| --- |
| 2020-05-09 12:14:22,760 WARN [RS\_OPEN\_REGION-regionserver/node1:16020-2] handler.AssignRegionHandler: Failed to open region TestTable,00000000000000000006900000,1588444012555.8a72d1ccdadd3b14284a24ec01918023., will report to master  java.io.IOException: Missing table descriptor for TestTable,00000000000000000006900000,1588444012555.8a72d1ccdadd3b14284a24ec01918023.  at org.apache.hadoop.hbase.regionserver.handler.AssignRegionHandler.process(AssignRegionHandler.java:129)  at org.apache.hadoop.hbase.executor.EventHandler.run(EventHandler.java:104)  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)  at java.lang.Thread.run(Thread.java:748) |

#### 问题解析

在执行Region Split时，因为系统中断或者HDFS中的Region文件已经被删除。

Region的状态由master跟踪，包括以下状态：

| State | Description |
| --- | --- |
| Offline | Region is offline |
| Pending Open | A request to open the region was sent to the server |
| Opening | The server has started opening the region |
| Open | The region is open and is fully operational |
| Pending Close | A request to close the region has been sent to the server |
| Closing | The server has started closing the region |
| Closed | The region is closed |
| Splitting | The server started splitting the region |
| Split | The region has been split by the server |

Region在这些状态之间的迁移（transition）可以由master引发，也可以由region server引发。

#### 解决方案

* 使用 hbase hbck 找到哪些Region出现Error
* 使用以下命令将失效的Region删除

deleteall "hbase:meta","TestTable,00000000000000000005850000,1588444012555.89e1c07384a56c77761e490ae3f34a8d."

* 重启hbase即可

### Phoenix: Table is read only

|  |
| --- |
| Error: ERROR 505 (42000): Table is read only. (state=42000,code=505)  org.apache.phoenix.schema.ReadOnlyTableException: ERROR 505 (42000): Table is read only.  at org.apache.phoenix.query.ConnectionQueryServicesImpl.ensureTableCreated(ConnectionQueryServicesImpl.java:1126)  at org.apache.phoenix.query.ConnectionQueryServicesImpl.createTable(ConnectionQueryServicesImpl.java:1501)  at org.apache.phoenix.schema.MetaDataClient.createTableInternal(MetaDataClient.java:2721)  at org.apache.phoenix.schema.MetaDataClient.createTable(MetaDataClient.java:1114)  at org.apache.phoenix.compile.CreateTableCompiler$1.execute(CreateTableCompiler.java:192)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:408)  at org.apache.phoenix.jdbc.PhoenixStatement$2.call(PhoenixStatement.java:391)  at org.apache.phoenix.call.CallRunner.run(CallRunner.java:53)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:390)  at org.apache.phoenix.jdbc.PhoenixStatement.executeMutation(PhoenixStatement.java:378)  at org.apache.phoenix.jdbc.PhoenixStatement.execute(PhoenixStatement.java:1825)  at sqlline.Commands.execute(Commands.java:822)  at sqlline.Commands.sql(Commands.java:732)  at sqlline.SqlLine.dispatch(SqlLine.java:813)  at sqlline.SqlLine.begin(SqlLine.java:686)  at sqlline.SqlLine.start(SqlLine.java:398)  at sqlline.SqlLine.main(SqlLine.java:291) |

### [phoenix连接hbase数据库，创建二级索引报错：Error: org.apache.phoenix.exception.PhoenixIOException: Failed after attempts=36, exceptions: Tue Mar 06 10:32:02 CST 2018, null, java.net.SocketTimeoutException: callTimeou](https://www.cnblogs.com/chuanzhang053/p/8514477.html)t

1. 修改phoenix的hbase-site.xml配置文件为

|  |
| --- |
| <property>      <name>phoenix.query.timeoutMs</name>      <value>1800000</value>  </property>  <property>      <name>hbase.regionserver.lease.period</name>      <value>1200000</value>  </property>  <property>      <name>hbase.rpc.timeout</name>      <value>1200000</value>  </property>  <property>      <name>hbase.client.scanner.caching</name>      <value>1000</value>  </property>  <property>      <name>hbase.client.scanner.timeout.period</name>      <value>1200000</value>  </property> |

1. 设置完以上内容后，重新通过sqlline.py连接hbase