# Phoenix特性

目录

[Phoenix特性 1](#_Toc531355699)

[概述 2](#_Toc531355700)

[Phoenix安装 2](#_Toc531355701)

[Phoenix二级索引 3](#_Toc531355702)

[为什么需要二级索引？ 3](#_Toc531355703)

[配置Hbase支持Phoenix二级索引 3](#_Toc531355704)

[索引类型 4](#_Toc531355705)

[同步创建索引与异步创建索引 5](#_Toc531355706)

[二级索引的正确创建姿势是: 6](#_Toc531355707)

[测试索引效率 6](#_Toc531355708)

[Phoenix导入数据以及HBase中已有表与Phoenix映射 7](#_Toc531355709)

[Phoenix导入数据 7](#_Toc531355710)

[HBase中已有表与Phoenix映射 9](#_Toc531355711)

[开启schema与namespace的对应关系 10](#_Toc531355712)

[原子的Upsert 10](#_Toc531355713)

[提升效率的方法 11](#_Toc531355714)

[mutable 和 immutable 表区别 13](#_Toc531355715)

[如何使用自增ID 14](#_Toc531355716)

[动态列 14](#_Toc531355717)

[分页查询 15](#_Toc531355718)

[Skip Scan（跳跃扫描） 16](#_Toc531355719)

[事物（Transactions） 17](#_Toc531355720)

[用户自定义方法（UDFs） 17](#_Toc531355721)

[Tracing 追踪 19](#_Toc531355722)

[Matrics度量 20](#_Toc531355723)

[Phoenix数据类型 20](#_Toc531355724)

[Phoenix语法 21](#_Toc531355725)

[Phoenix函数 21](#_Toc531355726)

[使用Phoenix过程中出现的问题 21](#_Toc531355727)

## 概述

Phoenix最早是saleforce的一个开源项目，后来成为Apache基金的顶级项目。

Phoenix是构建在HBase上的一个SQL层，能让我们用标准的JDBC APIs而不是HBase客户端APIs来创建表，插入数据和对HBase数据进行查询。

Phoenix完全使用Java编写，作为HBase内嵌的JDBC驱动。Phoenix查询引擎会将SQL查询转换为一个或多个HBase扫描，并编排执行以生成标准的JDBC结果集。直接使用HBase API、协同处理器与自定义过滤器，对于简单查询来说，其性能量级是毫秒，对于百万级别的行数来说，其性能量级是秒。

## Phoenix安装

#### 下载Phoenix

下载地址:http://phoenix.apache.org/download.html(hbase的版本一定要与phoenix的版本保持一致)

#### 配置Phoenix

\* 解压缩文件

\* 配置Phoenix环境变量

\* 将 Phoenix 目录下的phoenix-core-4.14.1-HBase-1.2.jar、phoenix-4.14.1-HBase-1.2-client.jar 拷贝到 hbase 集群各个节点 hbase 安装目录 lib 中。

\* 将 hbase 集群中的配置文件 hbase-site.xml 拷贝到 Phoenix 的 bin 目录下，覆盖原有的配置文件。

\* 将 hdfs 集群中的配置文件 core-site.xml、 hdfs-site.xml 拷贝到 Phoenix 的 bin 目录下。

\* 重启hbase

#### 启动Phoenix

sqlline.py <hbase.zookeeper.quorum>

## Phoenix二级索引

### 为什么需要二级索引？

对于HBase而言，如果想精确地定位到某行记录，唯一的办法是通过rowkey来查询。如果不通过rowkey来查找数据，就必须逐行地比较每一列的值，即全表扫瞄。对于较大的表，全表扫描的代价是不可接受的。但是，很多情况下，需要从多个角度查询数据。

所以，需要secondary index（二级索引）来完成这件事。

### 配置Hbase支持Phoenix二级索引

如果要启用phoenix的二级索引功能，需要对HMaster以及每一个RegionServer上的hbase-site.xml进行额外的配置。首先，在每一个RegionServer的hbase-site.xml里加入如下属性：

<property>

<name>hbase.regionserver.wal.codec</name>

<value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>

</property>

<property>

<name>hbase.region.server.rpc.scheduler.factory.class</name>

<value>org.apache.hadoop.hbase.ipc.PhoenixRpcSchedulerFactory</value>

<description>Factory to create the Phoenix RPC Scheduler that uses separate queues for index and metadata updates</description>

</property>

<property>

<name>hbase.rpc.controllerfactory.class</name>

<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>

<description>Factory to create the Phoenix RPC Scheduler that uses separate queues for index and metadata updates</description>

</property>

<property>

<name>hbase.coprocessor.regionserver.classes</name>

<value>org.apache.hadoop.hbase.regionserver.LocalIndexMerger</value>

</property>

注：如果没有在每个regionserver上的hbase-site.xml里面配置如上属性，那么使用create index语句创建二级索引将会抛出如下异常：

Error:ERROR 1029 (42Y88): Mutable secondary indexes must have the

hbase.regionserver.wal.codec property set to

org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec in the hbase-sites.xml of every region server tableName=TEST\_INDEXES (state=42Y88,code=1029)

然后在每一个master的hbase-site.xml里加入如下属性：

<property>

<name>hbase.master.loadbalancer.class</name>

<value>org.apache.phoenix.hbase.index.balancer.IndexLoadBalancer</value>

</property>

<property>

<name>hbase.coprocessor.master.classes</name>

<value>org.apache.phoenix.hbase.index.master.IndexMasterObserver</value>

</property>

完成上述修改，重启HBase。

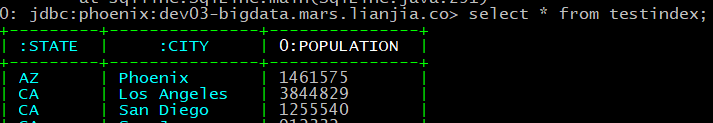
### 索引类型

#### 覆盖索引（Covered Indexes）

覆盖索引：只需要通过索引就能返回所要查询的数据，所以索引的列必须包含所需查询的列(SELECT的列和WHRER的列)

Create index testindex on us\_population(state) include(population);

注意关键字INCLUDE，就是包含需要返回数据结果的列。这种索引方式的最大好处就是速度快，而我们也知道，索引就是空间换时间，所以缺点也很明显，存储空间耗费较多。Phoenix的索引其实就是建了一张HBase的表。你可以通过HBase Shell的list命令看到。查看表testindex，你会发现，这张表一共三列，一列就是索引，第二列是RowKey，最后一列就是population的值。很明显在这里记录的RowKey，就是为了快速查找HBase中的数据。只是这里用不到，population已经被缓存在这张索引里面了，直接返回。



#### 函数索引（Functional Indexes）

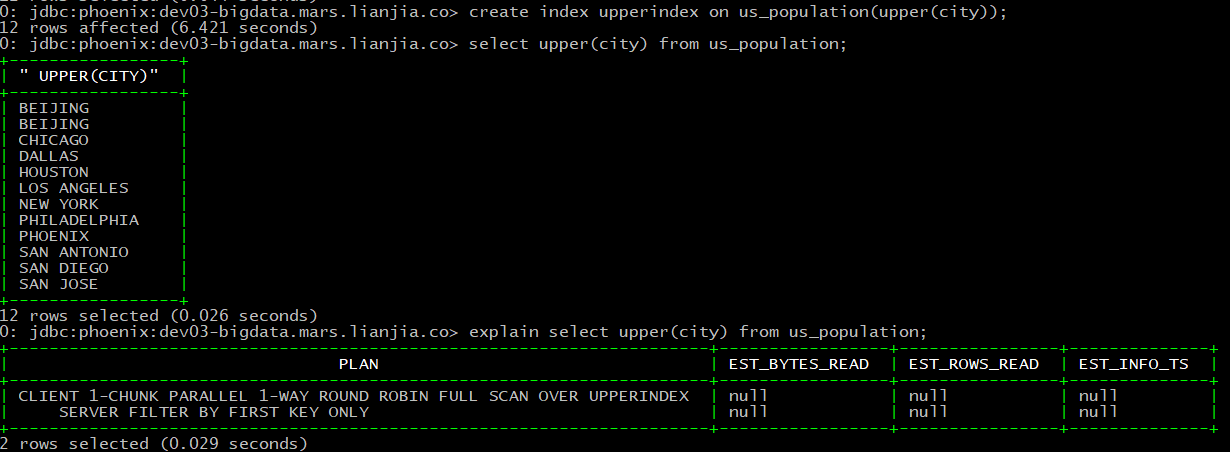
从Phoeinx4.3以上就支持函数索引，其索引不局限于列，可以合适任意的表达式来创建索引，当在查询时用到了这些表达式时就直接返回表达式结果。

CREATE INDEX UPPER\_NAME\_IDX ON EMP (UPPER(FIRST\_NAME||' '||LAST\_NAME))

查询命令：

SELECT EMP\_ID FROM EMP WHERE UPPER(FIRST\_NAME||' '||LAST\_NAME)='JOHN DOE'

例如：



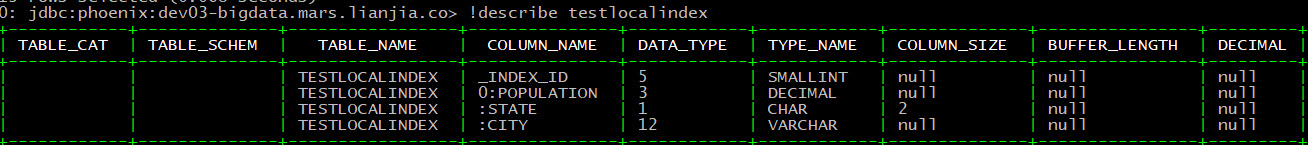
#### 全局索引（Global Indexes）

全局索引适用于多读少写的场景，在写操作上会给性能带来极大的开销，因为所有的更新和写操作（DELETE,UPSERT VALUES和UPSERT SELECT）都会引起索引的更新,在读数据时，Phoenix将通过索引表来达到快速查询的目的。默认为全局索引。

#### 本地索引（Local Indexes）

本地索引适用于写多读少，空间有限的场景，和全局索引一样，Phoneix在查询时会自动选择是否使用本地索引，使用本地索引，为避免进行写操作所带来的网络开销，索引数据和表数据都存放在相同的服务器中，当查询的字段不完全是索引字段时本地索引也会被使用，与全局索引不同的是，所有的本地索引都单独存储在同一张共享表中，由于无法预先确定region的位置，所以在读取数据时会检查每个region上的数据因而带来一定性能开销。

Create local index testlocalindex on us\_population(state);



### 同步创建索引与异步创建索引

当执行create index的时候，索引表会直接与源数据表进行同步。但是，有时候我们的源表数据量很大，同步创建索引会抛出异常。

这个时候，我们可以采用异步创建索引，方式如下：

CREATE INDEX async\_index ON my\_schema.my\_table (v) ASYNC

通过create index的时候指定 ASYNC 关键字来指定异步创建索引。执行这个命令之后并不会引起索引表与源表的直接同步。

必须通过HBase命令行单独启动填充索引表的map reduce作业，如下所示：

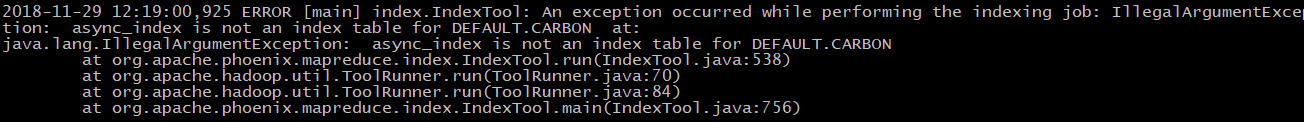
${HBASE\_HOME}/bin/hbase org.apache.phoenix.mapreduce.index.IndexTool

--schema default --data-table "carbon" --index-table "async\_index"

--output-path ASYNC\_IDX\_HFILES

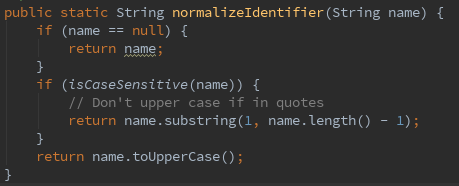
出现问题（未解决）：

发现该命令自动把加了引号的表名转为大写，于是报了该索引不是该表的错误：



查看源码发现下面代码把已为小写的表名转为了大写：





### 二级索引的正确创建姿势是:

create index 二级索引表名 on 原始数据表名(二级索引主键, where条件中的字段) include(普通字段, select 后面跟的需要显示的字段);

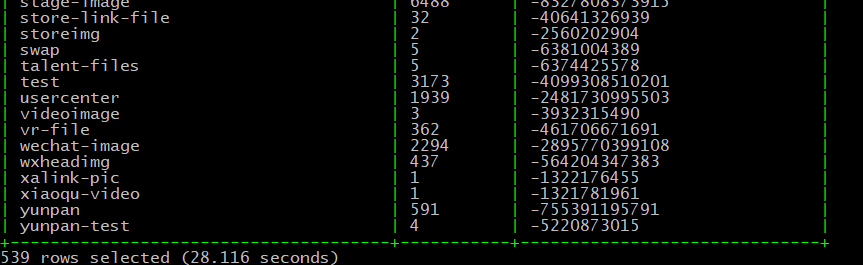
### 测试索引效率

创建视图：  
create view "carbon" ("ROW" varchar primary key, "header"."Bucket" varchar, "header"." ContentLength" varchar);

注：“ContentLength”列类型设为varchar避免整型出错。

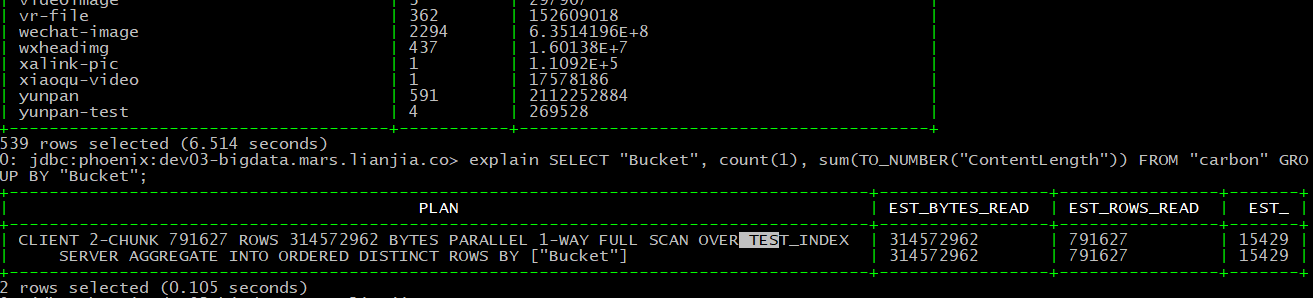
SELECT "Bucket", count(1), sum("ContentLength") FROM "carbon" GROUP BY "Bucket";

没加索引：



加索引：

Create index test\_index on "carbon" ("header". "Bucket") include("header". "ContentLength");

SELECT "Bucket", count(1), sum(TO\_NUMBER("ContentLength")) FROM "carbon" GROUP BY "Bucket";

## Phoenix导入数据以及HBase中已有表与Phoenix映射

phoenix有两种方式供批量写数据。一种是单线程psql方式，另一种是mr分布式。单线程适合一次写入十来兆的文件，mr方式更加适合写大批量数据。

### Phoenix导入数据

#### 准备阶段

1. 创建表

CREATE TABLE IF NOT EXISTS us\_population (

state CHAR(2) NOT NULL,

city VARCHAR NOT NULL,

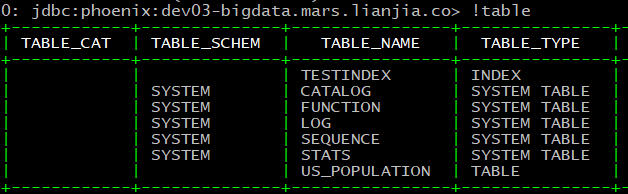
population BIGINT

CONSTRAINT my\_pk PRIMARY KEY (state, city));

1. 创建二级索引

CREATE INDEX TESTINDEX ON us\_population(state) INCLUDE(population);

创建索引时，会在hbase中创建对应的索引表第一列为state，最后一列为population。



1. 准备数据

psql方式，创建本地文件us\_population.csv

mr分布式，把文件上传到hdfs

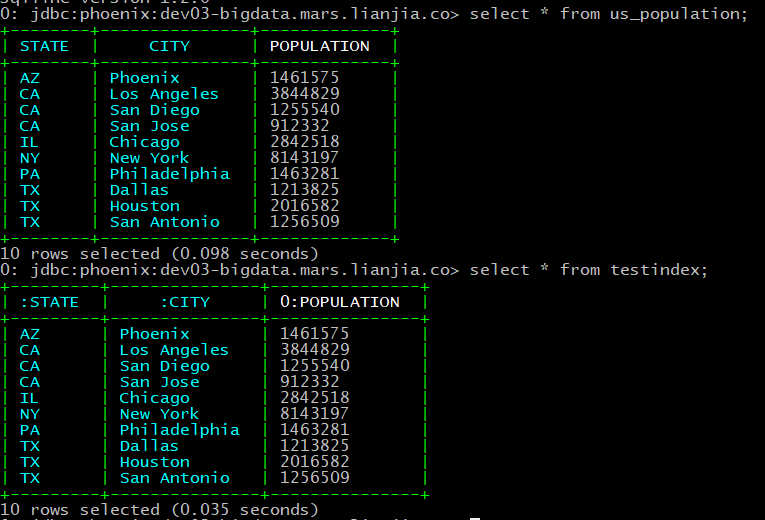
#### 批量导入数据

单线程psql方式：

psql.py –t us\_population <hbase.zookeeper.quorum>:2181 us\_population.csv

<hbase.zookeeper.quorum>2181为zookeeper对应的主机以及端口

验证数据是否录入正常以及索引表是否同步更新：



通过上述结果可知批量导入数据正常以及批量导入数据是会自动更新索引表的。

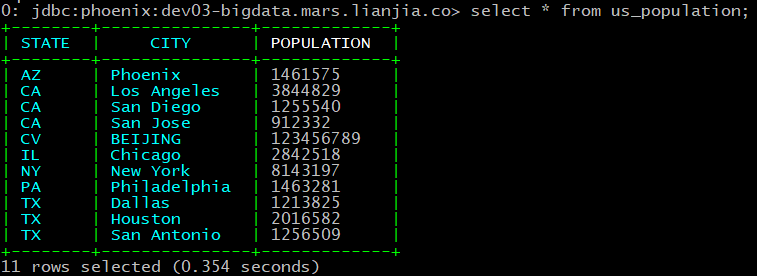
mr分布式方式：

官网指出如果为phoenix4.0及以上要用如下方式：

HADOOP\_CLASSPATH=/home/bigdata/bin/lib/hbase-protocol-1.2.6.1.jar:/home/bigdata/bin/hbase/conf hadoop jar /home/bigdata/bin/phoenix/phoenix-4.14.1-HBase-1.2-client.jar org.apache.phoenix.mapreduce.CsvBulkLoadTool –t TABLE\_NAME –input /data/example.csv



导入数据后结果：



小结：  
1、速度：

通过bulk loaded 的方式批量写数据速度大概能达到20K-50K每秒。

2、 通过测试确认批量导入会自动更新phoenix二级索引（这个结果不受是否先有hbase表的影响）。

3、导入文件编码默认是utf-8格式。

4、mr方式支持的参数还有其他的具体如下：

* mr方式导入数据默认会自动更新指定表的所有索引表，如果只需要更新指定的索引表可用-it 参数指定更新的索引表。对文件默认支持的分割符是逗号，参数为-d。
* 如果是想通过代码方式批量导入数据，可以通过代码先将数据写到hdfs中，将mr批量导入方式写到shell脚本中，再通过代码调用shell脚本。

### HBase中已有表与Phoenix映射

只需在phoenix中添加同名表即可映射到hbase的同名表

Create table “<table name>”("row key" varchar primary key, "column family"."column name" varchar)

问题：创建映射后发现没有数据

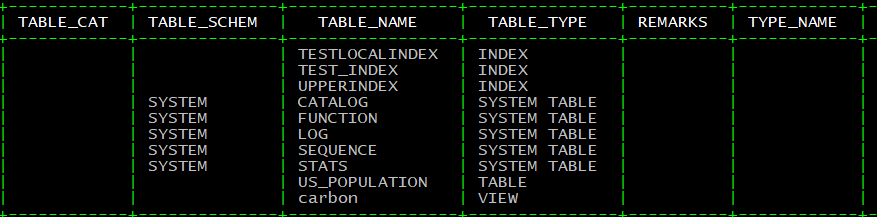
查看官方文档，phoenix 4.10 版本后，对列映射做了优化，采用一套新的机制，不在基于列名方式映射到 hbase。需要禁用列映射规则（会降低查询性能），如下：

Create table “<table name>”("row key" varchar primary key, "column family"."column name" varchar) column\_encoded\_bytes=0;

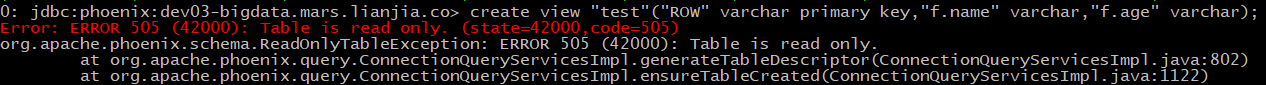
创建视图：

Create view “<table name>”(row key” varchar primary key,”相关列” datatype);

表名和列族以及列名需要用双引号括起来，因为HBase是区分大小写的，如果不用双引号括起来的话Phoenix在创建表的时候会自动将小写转换为大写字母。



注意：不能列族和列名写成”f.name”，而要写成”f”.”name”，不然会报以下错误（HBase表没找到）：



删除映射表时，应注意如果要保留原表，应使用以下命令：

DELETE from SYSTEM.CATALOG where TABLE\_NAME ='mapped\_table\_name';

而不应该使用drop table命令，会把hbase中的表以及数据删除。

### 开启schema与namespace的对应关系

如果使用了hbase中的自定义namespace，不仅仅使用default，那么在phoenix中与之对应的是schema的概念，但是默认并没有开启，需要在hbase-site.xml中增加以下配置项：

<property>

<name>phoenix.schema.isNamespaceMappingEnabled</name>

<value>true</value>

</property>

<property>

<name>phoenix.schema.mapSystemTablesToNamespace</name>

<value>true</value>

</property>

重启HBase。

## 原子的Upsert

Phoenix的插入语句是Upsert，Update和Insert的组合语义。即，如果数据表中没有这条记录那么插入这条记录，如果有则更新。判断是否存在相同的数据是使用ON DUPLICATE KEY来验证的，这里的KEY就是建表时候的主键（PRIMARY KEY）。和Oracle的Merge Into以及MySQL的Insert …on duplicate key 是类似的功能。ON DUPLICATE KEY这个语法要到Phoenix 4.9之后才有。这个功能把HBase的Increment和CheckAndPut两个原子操作合在了一起。当Upsert语句被提交到服务端的时候，所要更新的行会被lock住，同时相关的列会被读取，ON DUPLICATE KEY语句会被执行。由于会锁住所在的行，所以会有一些性能损耗，但是这个损耗比较小，类似于HBase中Put和CheckAndPut的之间的性能差异。

由于有了ON DUPLICATE KEY分句，如果行存在，那么VALUES后面的值将会为忽略。在ON DUPLICATE KEY后面还有其他的语法来实现不同的功能：

如果分句是ON DUPLICATE KEY IGNORE 的话，那么这行就不会被Update。

如果分句是ON DUPLICATE KEY UPDATE 的话，行就会被后面的Update语句更新，这个更新是获取了行级锁的。

如果操作同一行的多个Upsert在同一时间被批量提交的话，服务端将会按顺序执行它们。所以无论自动提交是开还是关，都会得到一样的结果。

下面列出了UPSERT的一些常用语法：

UPSERT INTO TEST VALUES('foo','bar',3);

UPSERT INTO TEST(NAME,ID) VALUES('foo',123);

UPSERT INTO TEST(ID, COUNTER) VALUES(123, 0) ON DUPLICATE KEY UPDATE COUNTER = COUNTER + 1;

UPSERT INTO TEST(ID, MY\_COL) VALUES(123, 0) ON DUPLICATE KEY IGNORE;

如果是4.9之前的版本的话，ON DUPLICATE KEY 分句是不存在的。只有上例的第一和第二种语法。如果原表中有数据重复的话，就会直接更新。

Phoenix也支持upsert select语法，插入或更新的数据是另外一个查询结果集。插入或更新的列和查询结果集的列要一一匹配，如果不显式的列出列，那么查询结果集的列要和目标表的列的元数据保持顺序一致。

常用示例：

UPSERT INTO test.targetTable(col1, col2) SELECT col3, col4 FROM test.sourceTable WHERE col5 < 100 ;

UPSERT INTO foo SELECT \* FROM bar;

ON DUPLICATE KEY的一些限制：  
1. 主键不会被更新，它是创建新行的基础。

会报错：



2. 事务表不应该使用这个功能，因为当冲突发生时，原子的Upsert操作有可能已经被异常处理掉了。

3. 不可变表不应该使用这个功能，因为不应该有数据更新不可变表。

4. 如果要使用这个子句，在连接时，就不能设置CURRENT\_SCN属性，因为除非最新的值正在被更新，否则HBase的操作不是原子的。

5. 同一列在同一句语句中不应该被更新超过一次。

6. ON DUPLICATE KEY 子句中不能有聚合或者序列。

7. 虽然在列上的全局索引是支持原子化的更细你的，但是还是不推荐在全局索引表上使用这个子语句。因为当维护二级索引的时候，行会被锁定，这个子句会产生一个RPC调用。

## 提升效率的方法

#### 1、加盐：

加盐可以将数据存入多个region里，从而提升读写性能。

CREATE TABLE TEST (HOST VARCHAR NOT NULL PRIMARY KEY, DESCRIPTION VARCHAR) SALT\_BUCKETS=42

如果有16台region server，每台server有4核CPU，则SALT\_BUCKETS 设置为32~64之间。

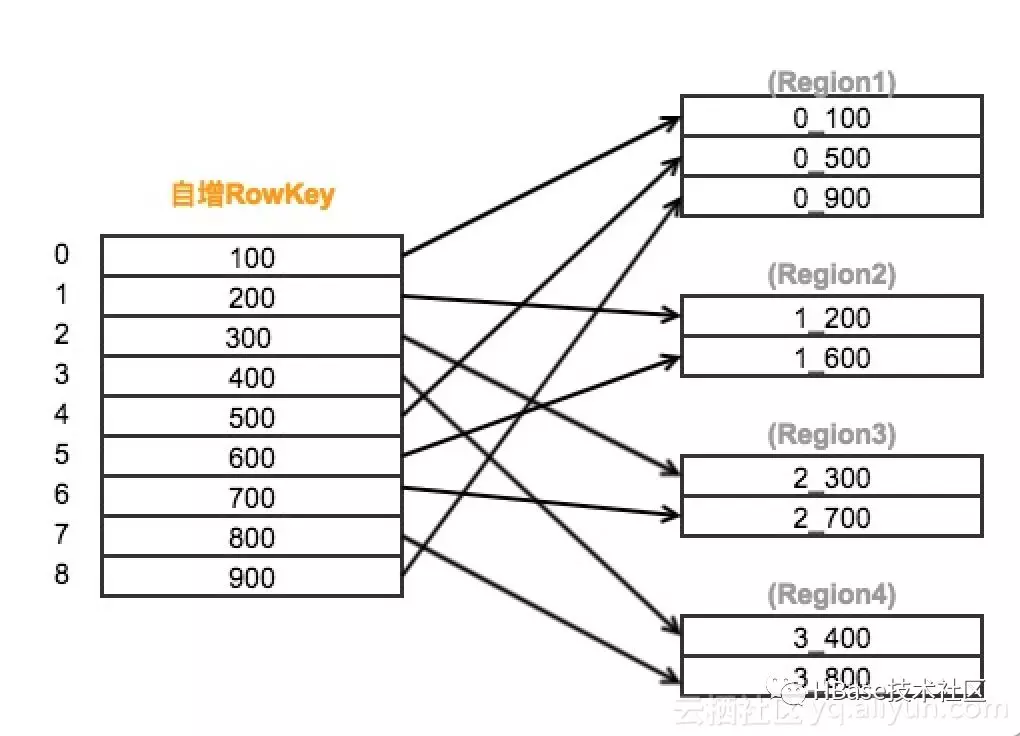
即如果集群总的CPU核数为N，则SALT\_BUCKETS为 0.5N ~ N 之间。

##### 加盐的原理是什么？

加盐的过程就是在原来key的基础上增加一个byte作为前缀,计算公式如下：

new\_row\_key = (++index % BUCKETS\_NUMBER) + original\_key

下图展示了自增rowkey通过加盐被打散写入到各个region中的过程：



#### 2、split

如果不想通过加盐来分区，可以自己手动设置分区的方法。这样可以不引入额外的byte，或者改变row key的顺序，例子

CREATE TABLE TEST (HOST VARCHAR NOT NULL PRIMARY KEY, DESCRIPTION VARCHAR) SPLIT ON ('CS','EU','NA');

#### 3、使用多个列族

CREATE TABLE TEST (MYKEY VARCHAR NOT NULL PRIMARY KEY, A.COL1 VARCHAR, A.COL2 VARCHAR, B.COL3 VARCHAR);

#### 4、使用压缩

CREATE TABLE TEST (HOST VARCHAR NOT NULL PRIMARY KEY, DESCRIPTION VARCHAR) COMPRESSION='GZ'

#### 5、使用二级索引

[见上文。](#_Phoenix二级索引)

## mutable 和 immutable 表区别

分别创建表：

create table my\_mutable (id varchar not null primary key, cf1.a varchar , cf1.b varchar, cf2.c varchar, cf2.d varchar) ;

create table my\_immutable (id varchar not null primary key, cf1.a varchar , cf1.b varchar, cf2.c varchar, cf2.d varchar) immutable\_rows=true ;

分别创建索引：

create index index\_my\_mutable on my\_mutable(a,c) include (b,d);

create index index\_my\_immutable on my\_immutable(a,c) include (b,d);

分别插入数据：

upsert into my\_mutable values ('1000001','a1','b1','c1','d1');

upsert into my\_mutable values ('1000001','a2','b2','c2','d2');

upsert into my\_mutable values ('1000001','a3','b3','c3','d3');

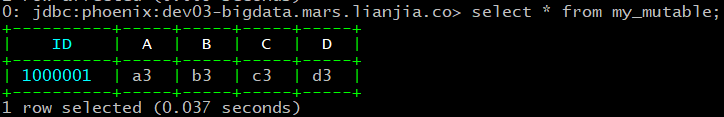
upsert into my\_immutable values ('1000001','a1','b1','c1','d1');

upsert into my\_immutable values ('1000001','a2','b2','c2','d2');

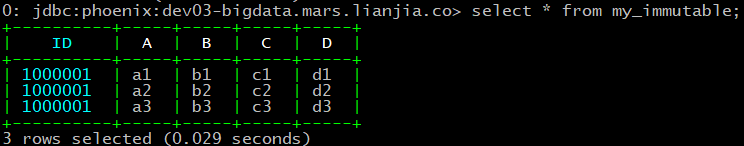
upsert into my\_immutable values ('1000001','a3','b3','c3','d3');

查看数据：

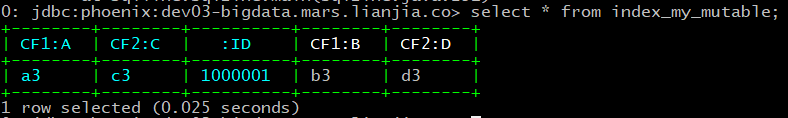
select \* from my\_mutable;



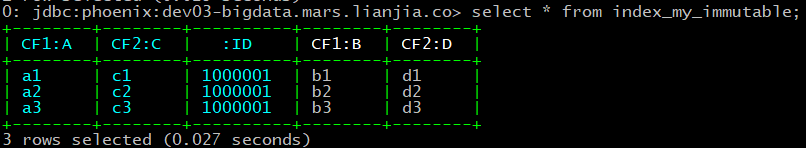
select \* from my\_immutable;



select \* from index\_my\_mutable;



select \* from index\_my\_immutable;



## 如何使用自增ID

在传统关系型数据库中设计主键时，自增ID经常被使用。不仅能够保证主键的唯一，同时也能简化业务层实现。

语法：

CREATE SEQUENCE [IF NOT EXISTS] SCHEMA.SEQUENCE\_NAME

[START WITH number]

[INCREMENT BY number]

[MINVALUE number]

[MAXVALUE number]

[CYCLE]

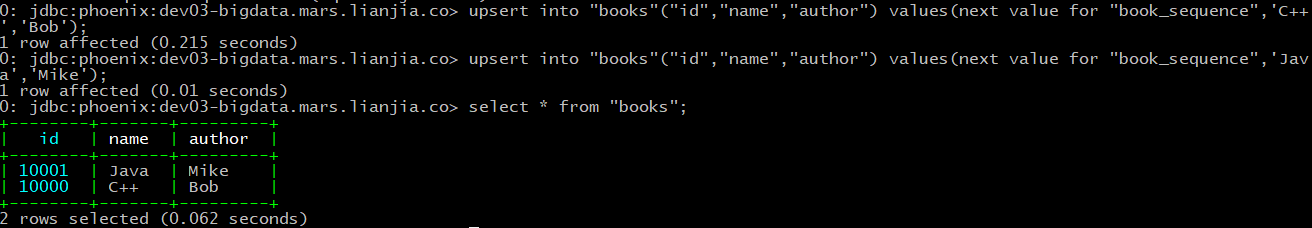
[CACHE number]

* start用于指定第一个值。如果不指定默认为1.
* increment指定每次调用next value for后自增大小。 如果不指定默认为1。
* minvalue和maxvalue一般与cycle连用, 让自增数据形成一个环，从最小值到最大值，再从最大值到最小值。
* cache默认为100, 表示server端生成100个自增序列缓存在客户端，可以减少rpc次数。此值也可以通过phoenix.sequence.cacheSize来配置。

由于自增ID作为rowkey, 容易造成集群热点问题，所以在创建表时最好通过加盐的方式解决这个问题。

create table "books"("id" integer not null primary key,"name" varchar,"author" varchar) SALT\_BUCKETS = 8;

create sequence "book\_sequence" start with 10000 increment by 1 cache 1000;



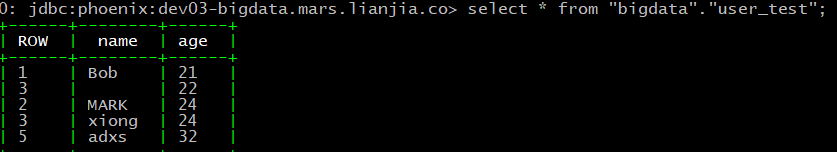
## 动态列

动态列是指在查询中新增字段，操作创建表时未指定的列。传统关系型数据要实现动态列目前常用的方法有：设计表结构时预留新增字段位置、设计更通用的字段、列映射为行和利用json/xml存储字段扩展字段信息等，这些方法多少都存在一些缺陷，动态列的实现只能依赖逻辑层的设计实现。由于Phoenix是HBase上的SQL层，借助HBase特性实现的动态列，避免了传统关系型数据库动态列实现存在的问题。

1. Upsert，动态插入列。

upsert into "bigdata"."user\_test"("ROW","f"."name","f"."age","f"."currentTime" varchar) values('5','adxs','32',TO\_CHAR(CURRENT\_TIME()));

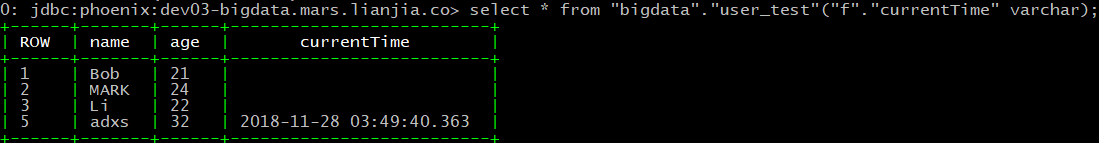
查询一下：



发现没有动态列，实际上HBase表中已经新增列以及数据。那通过动态列添加的数据怎么查询呢？

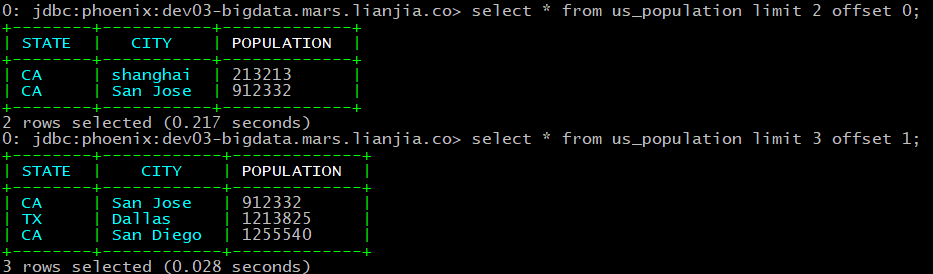
1. Select，查询动态列。

select \* from "bigdata"."user\_test"("f"."currentTime" varchar);



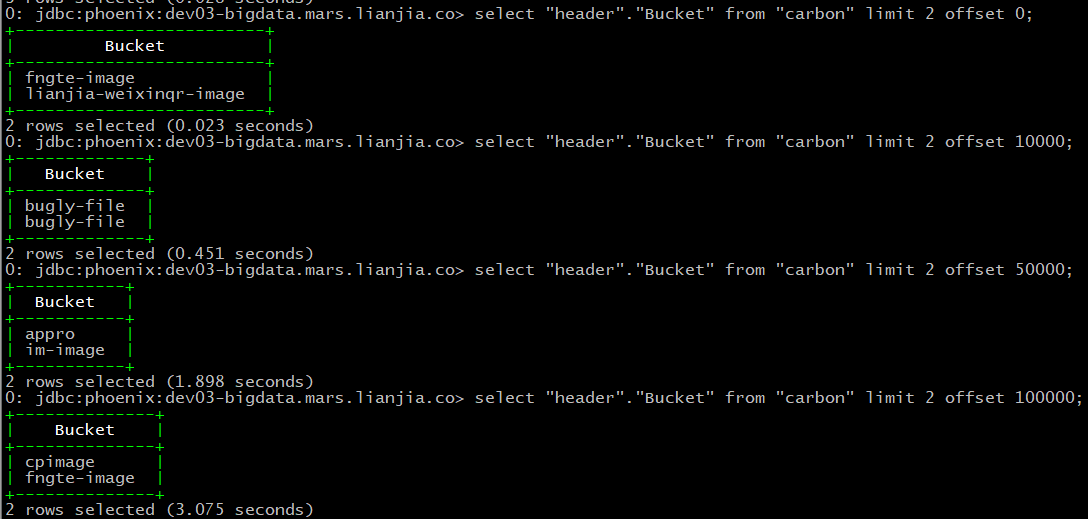
## 分页查询

所谓分页查询就是从符合条件的起始记录，往后遍历“页大小”的行。数据库的分页是在server端完成的，避免客户端一次性查询到大量的数据，让查询数据数据分段展示在客户端。

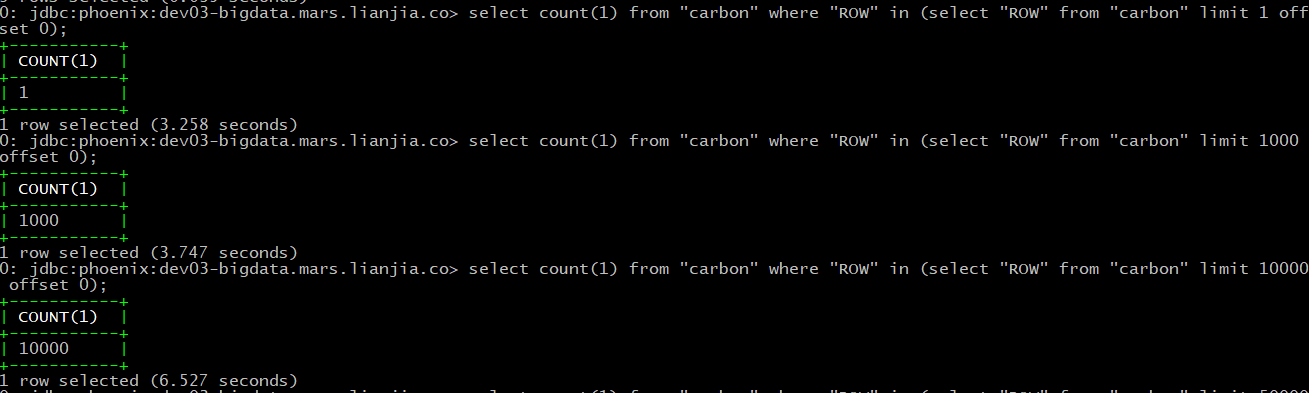


Limit表示查询数据条数，offset表示从哪一行开始查询。

性能：



结论1：当limit的值一定时，随着offset N的值越大，查询性基本会线性下降。



结论2：当offset的值一定时，随着Limit的值越大，查询性能逐步下降。当limit的值相差一个数量级时，查询性能也会有几十倍的差距。

大多数场景中分页查询都是和order by子句一起使用的, 在这里需要注意的是，order by的排序字段最好是主键，否则查询性能会比较差。

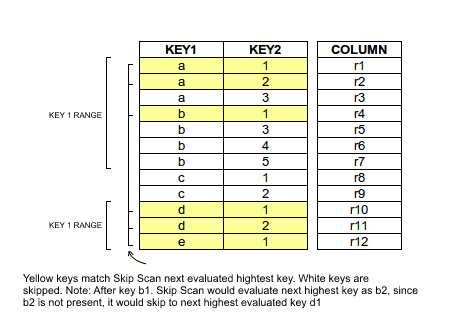
## Skip Scan（跳跃扫描）

例如：

SELECT \* from T

WHERE ((KEY1 >='a' AND KEY1 <= 'b') OR (KEY1 > 'c' AND KEY1 <= 'e'))

AND KEY2 IN (1, 2)



## 事物（Transactions）

phoenix除了有hbase的行级事务外，通过和Tephra集成，添加了完整的ACID事务支持。

在phoenix中启动事务需要两个步骤：

1. 在每个regionserver里修改配置文件并启动hbase，配置hbase-site.xml

<property>

<name>phoenix.transactions.enabled</name>

<value>true</value>

</property>

在每一个Master修改配置文件hbase-site.xml

<property>

<name>data.tx.snapshot.dir</name>

<value>/tmp/tephra/snapshots</value>

</property>

<property>

<name>data.tx.timeout</name>

<value>60</value>

</property>

1. 启动tephra

./bin/tephra

在创建表的时候要加上一个使用事务的属性 TRANSACTIONAL=TRUE的属性：

CREATE TABLE my\_table (k BIGINT PRIMARY KEY, v VARCHAR) TRANSACTIONAL=true;

现有的表也可以通过alter语句修改成为事务表

ALTER TABLE my\_table SET TRANSACTIONAL = TRUE;

事物语句：

SELECT \* FROM my\_table; - 这将启动一个事务

UPSERT INTO my\_table VALUES（1，'A'）;

SELECT count（\*）FROM my\_table WHERE k = 1; - 将看到未提交的行

DELETE FROM my\_other\_table WHERE k = 2;

！commit - 其他交易现在会看到您的更新，您将看到他们的更新

测试事物语句：

1.插入输入不提交

UPSERT INTO my\_table VALUES（1，'A'）;

2.在新的客户端插入数据并且主键一致试图提交，这样会抛出异常错误。

UPSERT INTO my\_table VALUES（1，'B'）;

！commit

## 用户自定义方法（UDFs）

用户可以创建临时/永久的函数。UDF可以与查询中的内置函数一样使用，例如select，upsert，delete，create functions index。临时函数在特定的回话和连接中生效，但是不能在其他的回话和连接中使用。永久函数的元信息将保存在SYSTEM.FUNCION的系统表中。支持租户特定函数，在特定的租户的连接中创建的函数对于其他的租户是不可见的。只有全局租户的UDF才可以在所有的连接中可见。

pheonix使用hbase的动态类加载jar，在客户端和regionserver上动态加载jar。不需要重启服务

1. 配置

在Phoenix的客户端中配置hbase-site.xml文件：

<property>

<name>phoenix.functions.allowUserDefinedFunctions</name>

<value>true</value>

</property>

<property>

<name>fs.hdfs.impl</name>

<value>org.apache.hadoop.hdfs.DistributedFileSystem</value>

</property>

<property>

<name>hbase.rootdir</name>

<value>${hbase.tmp.dir}/hbase</value>

</property>

<property>

<name>hbase.dynamic.jars.dir</name>

<value>${hbase.rootdir}/lib</value>

</property>

hbase.rootdir 。这是一个共享目录。默认在 临时/tmp下，需要自己配置修改，因为重启服务器未见回丢失，也可以配置hdfs 例如：hdfs：//namenode.ehl.com:9000/hbase

hbase.dynamic.jars.dir 这个目录是phoenix 客户端/regionserver获取自定义udf的jar的地方。不需要重启服务。

还有两个在服务器端的配置：

在使用JDBC时候的配置：

Properties props = new Properties();

props.setProperty("phoenix.functions.allowUserDefinedFunctions", "true");

Connection conn = DriverManager.getConnection("jdbc:phoenix:localhost", props);

以下配置用于UDF的jar动态从hdfs加载到本地文件系统：

<property>

<name>hbase.local.dir</name>

<value>${hbase.tmp.dir}/local/</value>

<description>Directory on the local filesystem to be used

as a local storage.</description>

</property>

1. 创建用户自定义函数

 1.实现一个自定义的udf的jar

 2.上传jar到hdfs目录的 hbase.dynamic.jars.dir中

 3.执行 create function的语句

 怎么样创建udf的jar呢？

1.创建一个class 实现这个接口 org.apache.phoenix.expression.function.ScalarFunction

 2.实现gteDataType的方法，这个方法将返回一个function类型。

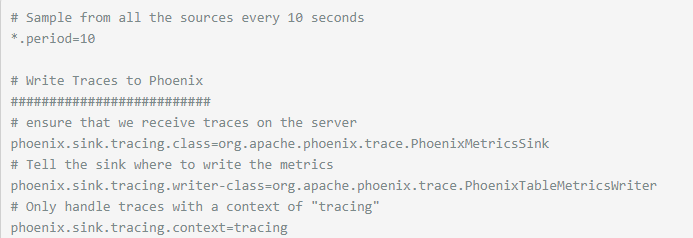
1. 实现evaluate方法，这个是用于计算结果中的row的。

## Tracing 追踪

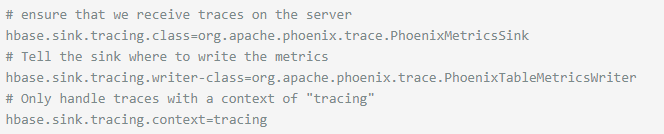
从4.1版本开始Phoenix增加这个特性来追踪每条查询的踪迹，这使用户能够看到每一条查询或插入操作背后从客户端到HBase端执行的每一步。目前只支持Hadoop2。

###### 配置：

Phoenix/bin/hadoop-metrics2-phoenix.properties



Phoenix/bin/ hadoop-metrics2-hbase.properties



hbase-site.xml

<property>

<name>phoenix.trace.frequency</name>

<value>always</value>

</property>

<property>

<name>phoenix.trace.statsTableName</name>

<value><your custom tracing table name></value>

</property>

追踪表的创建：

CREATE TABLE SYSTEM.TRACING\_STATS (

trace\_id BIGINT NOT NULL,

parent\_id BIGINT NOT NULL,

span\_id BIGINT NOT NULL,

description VARCHAR,

start\_time BIGINT,

end\_time BIGINT,

hostname VARCHAR,

tags.count SMALLINT,

annotations.count SMALLINT,

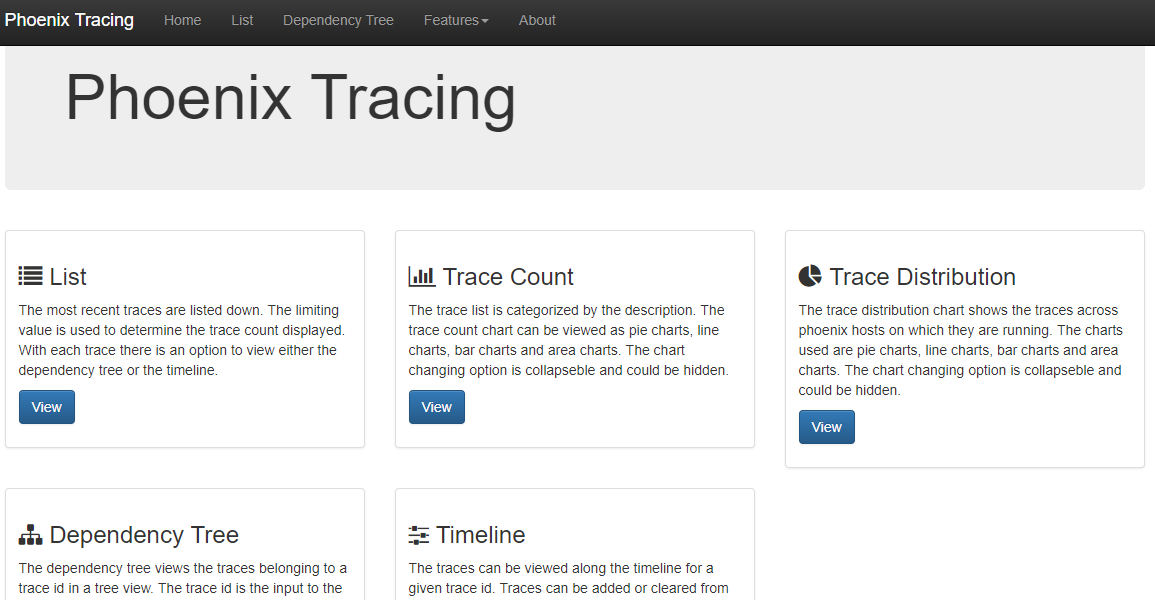
CONSTRAINT pk PRIMARY KEY (trace\_id, parent\_id, span\_id)

)

使用网页查看：

开启web app：./bin/traceserver.py start

打开网页：http://<phoenix master>:8864/



关闭web app：./bin/traceserver.py stop

## Matrics度量

Phoenix提供了各种指标，可以在执行各种SQL语句时深入了解Phoenix客户端内发生的情况。这些度量标准以两种方式在客户端JVM中收集：

Request level metrics - collected at an individual SQL statement level（在单个SQL语句级别收集）

Global metrics - collected at the client JVM level（在客户端JVM级别收集）

## Phoenix数据类型

见官网：http://phoenix.apache.org/language/datatypes.html

## Phoenix语法

见官网：<http://phoenix.apache.org/language/index.html>

## Phoenix函数

见官网：<http://phoenix.apache.org/language/functions.html>

## 使用Phoenix过程中出现的问题

1. 安装问题，拷贝Phoenix下的core.jar和client.jar两个包到hbase/lib下，注意不要漏掉clien.jar，不然会报错。
2. 执行命令超时报连接超时错误，Phoenix执行命令时，默认超过60s会报异常；当查询的表数据过大时，很容易报错，连接超时。更改hbase和Phoenix下的hbase-site.xml,重启hbase,添加内容如下：

<property>

<name>phoenix.query.timeoutMs</name>

<value>3600000</value>

</property>

<property>

<name>hbase.rpc.timeout</name>

<value>3600000</value>

</property>

1. 创建索引时列的数据类型报错，在Phoenix映射HBase表时，把要出错的列申明为varchar，然后通过相关函数转换为需要的类型。比如：TO\_NUMBER，转换为数字。
2. 大小写问题，Phoenix中区分大小写，会自动把小写转换为大写，因此，需要加上引号禁止转换。
3. 创建视图应注意的问题。首先，在映射到HBase表时，视图名字应与HBase表的名字一致。其次，要注意相关列的数据类型。
4. 删除映射表时，应注意如果要保留原表，应使用以下命令：

DELETE from SYSTEM.CATALOG where TABLE\_NAME ='mapped\_table\_name';

而不应该使用drop table命令，会把HBase中的表以及数据删除。

1. Phoenix插入数据出错：Values in UPSERT must evaluate to a constant

注意：

（1）插入值如果是字符串要用单引号引起来，切记不能用双引号！！！

（2）表名如果要体现小写效果，必须要用双引号！！！

1. Phoenix二级索引表与源数据表保持同步的前提，源数据表的增、删、改操作必须通过Phoenix命令行或者客户端，才能保证二级索引表与源表保持同步。Phoenix提供的Bulk Loading同样也能保证二级索引表与源数据表保持同步。