

[\[关闭\]](#)

@xtccc 2015-12-17 09:55 字数 6225 阅读 1336

Table and View

[给我写信](#)[GitHub](#)肖韬
江苏 南京

扫一扫上面的二维码图案，加我微信

Phoenix

Phoenix创建table

在sqlline.py中创建一个table，表名为TA1，主键名为MYKEY，两列分别为C1和C2，然后插入一条记录：

```
1. create table tal(mykey varchar not null primary key, c1 varchar, c2 integer);
2.
3. upsert into tal values ('row-1', 'c1-val', 100);
```

通过HBase shell来扫描该表：

```
hbase(main):008:0> scan 'TA1'
ROW                                COLUMN+CELL
row-1                             column=0:C1, timestamp=1445003490381,
row-1                             column=0:C2, timestamp=1445003490381,
row-1                             column=0:_0, timestamp=1445003490381,
1 row(s) in 0.0240 seconds
```

可见：在Phoenix SQL中，如果列名或者表名没有加上双引号，则将被自动转换为全大写字母。

可以注意到针对每一个row，HBase中都存在一个column name为_0的列，且它的value为空值。

Empty cell is created for each row. It's used to enforce PRIMARY KEY constraints because HBase doesn't store cells with NULL values.

Phoenix的主键 & HBase的rowkey

在上例中，可以看到，Phoenix通过mykey varchar not null primary key指定的主键mykey就是HBase中的rowkey。

还有另外一种对Phoenix table进行主键约束（Constraint PK）的方法，如下：

```
1. > create table "test" ("row" varchar not null, "c1" varchar, "c2" varchar CONSTRAINT PK primary key ("row"));
```

这里创建了一个含有3个column（rowkey, c1, c2）的表，并且通过CONSTRAINT PK指定了列rowkey作为该表的主键。

下面通过Phoenix SQL插入两条记录，然后查看该表内容：

```
1. > upsert into "test" values ('row-1', 'c1-1', 'c2-1');
2.
3. > upsert into "test" values ('row-2', 'c1-2', 'c2-2');
4.
5. > select * from "test";
```

6.			
7.	row	c1	c2
8.			
9.	row-1	c1-1	c2-1
10.	row-2	c1-2	c2-2
11.			

现在再通过HBase shell看看HBase table中的实际内容：

```

1. hbase(main):002:0> scan 'test'
2. ROW COLUMN+CELL
3. row-1 column=0:_0, timestamp=1445235997184, value=
4. row-1 column=0:c1, timestamp=1445235997184, value=c1-1
5. row-1 column=0:c2, timestamp=1445235997184, value=c2-1
6. row-2 column=0:_0, timestamp=1445236002241, value=
7. row-2 column=0:c1, timestamp=1445236002241, value=c1-2
8. row-2 column=0:c2, timestamp=1445236002241, value=c2-2
9. 2 row(s) in 0.0600 seconds

```

Namespace

HBase中的table有一个相关联的namespace（默认的namespace为default），我们在Phoenix中在创建table时也可以指定namespace，例如，下面我们在namespace spark 中创建 table test：

```

1. create table "spark:test" ("pk" varchar not null primary key, "c" varchar);

```

Family

Phoenix中只指定了列名，那么HBase中的family name就默认是0。我们也可以在Phoenix中指定一个名为f的family name，如下：

```

1. create table ta2(mykey integer not null primary key, f.c1 varchar);
2.
3. upsert into ta2 values (101, 'hello');

```

```

hbase(main):011:0> scan 'TA2'
ROW COLUMN+CELL
\x80\x00\x00e column=F:C1, timestamp=1445004941443,
\x80\x00\x00e column=F:_0, timestamp=1445004941443,
1 row(s) in 0.0090 seconds

```

所以，Phoenix在创建table时，table name中不要含有字符.，否则.前面的字符串会被当做family name。

也可以指定多个family name，如下：

```

1. create table ta3(mykey integer not null primary key, f1.c1 varchar, f1.c2 varchar, f2.c1 varchar, f2.c2 varchar);
2.
3. upsert into ta3 values (1, 'hello', 'hbase', 'hi', 'phoenix');

```

```

hbase(main):012:0> scan 'TA3'
ROW COLUMN+CELL
\x80\x00\x00\x01 column=F1:C1, timestamp=1445005839448,
\x80\x00\x00\x01 column=F1:C2, timestamp=1445005839448,
\x80\x00\x00\x01 column=F1:_0, timestamp=1445005839448,
\x80\x00\x00\x01 column=F2:C1, timestamp=1445005839448,
\x80\x00\x00\x01 column=F2:C2, timestamp=1445005839448,
1 row(s) in 0.0280 seconds

```

Compression

开启压缩后，可以提高大表IO的效率。

```

1. > create table "ta1" ("pk" varchar not null primary key, "f1"."c" varchar, "f2"."c" varchar) COMPRESSION='SNAPPY';

```

```
hbase(main):014:0> describe 'ta1'
DESCRIPTION
'ta1', {TABLE_ATTRIBUTES => {coprocessor$1 => '|org.apache.phoenix.coproc
server|805306366|', coprocessor$2 => '|org.apache.phoenix.coprocessor.Ungr
n0bserver|805306366|', coprocessor$3 => '|org.apache.phoenix.coprocessor.Gr
ion0bserver|805306366|', coprocessor$4 => '|org.apache.phoenix.coprocessor.
ointImpl|805306366|', coprocessor$5 => '|org.apache.phoenix.hbase.index.Ind
dex.builder=org.apache.phoenix.index.PhoenixIndexBuilder,org.apache.hadoop.
.class=org.apache.phoenix.index.PhoenixIndexCodec', coprocessor$6 => '|org.
se.regionserver.LocalIndexSplitter|805306366|'}, {NAME => 'f1', DATA_BLOCK
_DIFF', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', CO
PPY', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}, {NAME => 'f2', DATA_BLOCK
ST_DIFF', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1',
NAPPY', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE
65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0380 seconds
```

为已有的HBase Table创建Phoenix View

如果HBase中已经存在了某个table，则可以在Phoenix中为该HBase table建立一个view。

这里要注意：HBase table中的数据的序列化方式，必须与Phoenix table/view中的数据的序列化方式是一致的。对于varchar、char和unsigned_*类型的数据，使用了HBase Bytes方法。

例子：

在HBase shell中创建table并插入数据

```
hbase(main):005:0> create 'testtable', {NAME => 'f1', VERSIONS => 5}
0 row(s) in 0.3670 seconds

=> Hbase::Table - testtable
hbase(main):006:0> put 'testtable', 'row-1', 'f1:val', 'value-1'
0 row(s) in 0.0170 seconds

hbase(main):007:0> put 'testtable', 'row-2', 'f1:val', 'value-2'
0 row(s) in 0.0100 seconds

hbase(main):008:0> put 'testtable', 'row-3', 'f1:c', 'value-3'
0 row(s) in 0.0080 seconds

hbase(main):009:0> put 'testtable', 'row-4', 'f1:c', 'value-4'
0 row(s) in 0.0100 seconds
```

在Phoenix shell中创建映射到testtable的view

```
1. > create view "testtable" ( pk varchar primary key, "f1"."val" varchar);
2.
3. > select * from "testtable";
4.
5. +-----+-----+
6. |          PK          |          val          |
7. +-----+-----+
8. | row-1                | value-1               |
9. | row-2                | value-2               |
10. | row-3                |                       |
11. | row-4                |                       |
12. +-----+-----+
13. > upsert into "testtable" values("r1", "v1");
```

```
14.  Error: ERROR 505 (42000): Table is read only. (state=42000,code=505)
```

可以看到：

1. Phoenix view的名字必须与HBase table的名字相同
2. 对Phoenix view使用select * from {view} 只能查询出view中已经映射的列
3. Phoenix view中的primary key 就是HBase table中的row key
4. View是只读的，不能修改（增加、删除、修改数据）

Salted Table

创建salted table时，可以指定buckets的数量，也可以指定split points。

通过指定buckets的数量来创建salted table

用关键字 SALT_BUCKETS来指定region的数量。

下面创建一个名为test的salted table，并要求其拥有10个pre-split region。

```
1.  create table "test" ("pk" varchar not null primary key, "c" varchar) SALT_BUCKETS=10;
```

从HBase的web中可以看出，表test在创建后拥有了10个pre-split region，并且各个region之间的分隔key是\0x01 ~ \0x09。

Table Regions

Name	Region Server
test,,1445157946223.9d6973c452ea9c232b746fd49c7025b5.	ecs5.njzd.com:60020
test,\x01,1445157946223.6232e1448749f79c5930eaadef665c33.	ecs3.njzd.com:60020
test,\x02,1445157946223.2912744d481542b888d753d91a0fd52a.	ecs4.njzd.com:60020
test,\x03,1445157946223.6fc7214f52a1999a411ce9764a532621.	ecs4.njzd.com:60020
test,\x04,1445157946223.c08f2b67ec54d8f013de0c46de6e6aff.	ecs1.njzd.com:60020
test,\x05,1445157946223.b6b9cefb5c5a2c401ba4082223d0e3ee.	ecs2.njzd.com:60020
test,\x06,1445157946223.44c8bd9c59641ce2c73e05b99886b7e9.	ecs1.njzd.com:60020
test,\x07,1445157946223.64fd0243c1a9283b4bf6968206207073.	ecs2.njzd.com:60020
test,\x08,1445157946223.7d803892b3f3ba91ab60cf2ffd2f8e82.	ecs3.njzd.com:60020
test,\x09,1445157946223.2c728db0ea1dc11382ec602bec82081f.	ecs5.njzd.com:60020

Regions by Region Server

Region Server	Region Count
ecs1.njzd.com:60020	2
ecs2.njzd.com:60020	2
ecs3.njzd.com:60020	2
ecs4.njzd.com:60020	2
ecs5.njzd.com:60020	2

对于使用SALT_BUCKETS创建的salted table，当使用Phoenix的接口向表中写入数据时，rowkey的前面会自动地被加上1个随机字节。由于只会加上1个字节，因此，buckets 的数量最多为256，并且，被创建的表的每个region的start key和end key都只是1个字节。

例：向表中插入数据

首先，向表中插入数据：

```
1. > create table "test" ("pk" varchar not null primary key, "c" varchar) SALT_BUCKETS=10;
2.
3. > upsert into "test" values('EF123', 'hello');
4.
5. > upsert into "test" values('EGC', 'phoenix');
6.
7. > upsert into "test" values('AVE', 'hi');
8.
9. > upsert into "test" values('XZCC', 'hbase');
```

然后，通Phoenix SQL来查询上面插入的数据:

```
1. > select * from "test";
2.
3. +-----+-----+
4. | pk | c |
5. +-----+-----+
6. | AVE | hi |
```

7.	EF123	hello
8.	EGC	phoenix
9.	XZCC	hbase
10.		

再通过HBase shell来查询这些数据：

1.	hbase(main):010:0> scan 'test'	
2.	ROW	COLUMN+CELL
3.	\x00EF123	column=0:_0, timestamp=1445160796559, value=
4.	\x00EF123	column=0:c, timestamp=1445160796559, value=hello
5.	\x01AVE	column=0:_0, timestamp=1445160809493, value=
6.	\x01AVE	column=0:c, timestamp=1445160809493, value=hi
7.	\x03XZCC	column=0:_0, timestamp=1445160816462, value=
8.	\x03XZCC	column=0:c, timestamp=1445160816462, value=hbase
9.	\x08EGC	column=0:_0, timestamp=1445160802829, value=
10.	\x08EGC	column=0:c, timestamp=1445160802829, value=phoenix
11.	4 row(s) in 0.3840 seconds	

通过指定split points创建salted table

用关键字SPLIT ON来指定split points。

下面创建一个名为test的表，并且要求以'CS'、'ED'和'XYZ'来作为各个region之间的split point：

```
1. create table "test" ("pk" varchar not null primary key, "c" varchar) SPLIT ON ('CS', 'ED', 'XYZ');
```

从HBase的web可以看出，表test在创建后拥有了4个pre-split region，并且各个region之间就是按照预期来分隔的。

Table Regions

Name	Region Server
test,,1445158946545.3a4ce0c6469c7be02aa9db75ee326de9.	ecs3.njzd.com:60020
test,CS,1445158946545.9c60d39ee050975452b7196044b89832.	ecs2.njzd.com:60020
test,ED,1445158946545.6345a13b14e5a84cb8ddddc90896e146.	ecs5.njzd.com:60020
test,XYZ,1445158946545.e69195903cf7490adc1659bce349f2ea.	ecs4.njzd.com:60020

Regions by Region Server

Region Server	Region Coun
ecs2.njzd.com:60020	1
ecs3.njzd.com:60020	1
ecs4.njzd.com:60020	1
ecs5.njzd.com:60020	1

对于使用SPLIT ON创建的salted table，当使用Phoenix的接口向表中写入数据时，rowkey的前面不会自动地被加上1个随机字节。

例：向表中插入数据

首先，通过Phoenix插入数据：

```
1. > create table "test" ("pk" varchar not null primary key, "c" varchar) SPLIT ON ('CS', 'ED', 'XYZ');
2.
3. > upsert into "test" values('EF123', 'hello');
4.
5. > upsert into "test" values('EGC', 'phoenix');
6.
7. > upsert into "test" values('AVE', 'hi');
8.
```

```
9. > upsert into "test" values('XZCC', 'hbase');
```

然后，通Phoenix SQL来查询上面插入的数据：

```
1. > select * from "test";
2.
3.
4. +-----+-----+
5. |                pk                |                c                |
6. +-----+-----+
7. | AVE                               | hi                               |
8. | EF123                             | hello                           |
9. | EGC                               | phoenix                         |
10. | XZCC                              | hbase                           |
11. +-----+-----+
```

再通过HBase shell来查询这些数据：

```
1. hbase(main):009:0> scan 'test'
2. ROW COLUMN+CELL
3. AVE column=0:_0, timestamp=1445160002033, value=
4. AVE column=0:c, timestamp=1445160002033, value=hi
5. EF123 column=0:_0, timestamp=1445159966190, value=
6. EF123 column=0:c, timestamp=1445159966190, value=hello
7. EGC column=0:_0, timestamp=1445159983411, value=
8. EGC column=0:c, timestamp=1445159983411, value=phoenix
9. XZCC column=0:_0, timestamp=1445160025898, value=
10. XZCC column=0:c, timestamp=1445160025898, value=hbase
11. 4 row(s) in 0.0490 seconds
```

• 内容目录

- [Table and View](#)
- [Phoenix创建table](#)
 - [Phoenix的主键 & HBase的rowkey](#)
 - [Namespace](#)
 - [Family](#)
 - [Compression](#)
- [为已有的HBase Table创建Phoenix View](#)
- [Salted Table](#)
 - [通过指定buckets的数量来创建salted table](#)
 - [例：向表中插入数据](#)
 - [通过指定split points创建salted table](#)
 - [例：向表中插入数据](#)

•

- - AWS 2
 - Akka 10
 - Boot 1
 - Cassandra 6
 - Cloudera 3
 - Database 3
 - ElasticSearch 5
 - English 5
 - Gradle 11
 - HBase 3
 - HDFS 3
 - Java 8
 - Kafka 4
 - Kerberos 6
 - Kryo 1
 - Linux 6
 - Maven 2
 - NLP 1
 - Oozie 5
 - Phoenix 8


搜索 xtccc 的文稿标题，* 显

- [Build Phoenix Against HBase 1.0 \(CDH 5.4.7\)](#)
- [JDBC SQL](#)
- [Bulk CSV Data Loading](#)
- [下载客户端](#)
- [Index](#)
- [关注开源](#)
- [报告问题](#)
- [Introduction to Apache Phoenix](#)
- [联系我们](#)
- [RabbitMQ 3](#)

•

- - Scala 11
 - Shell 5
 - Spark 4
 - SparkStreaming 1


添加新批注



- Spring 1
- YARN 2
- ZooKeeper 3

保存 取消


在作者公开此批注前,只有你和作者可见。



- 推荐系统 3
- 数据挖掘&机器学习 1

保存 取消

算法 1- 未分类 1



修改 保存 取消 删除



- 私有
- 公开
- 删除

查看更早的 5 条回复

回复批注

通知

取消 确认

- 
- 

<https://www.zybuluo.com/xtccc/note/195866>

8/8