一. 安装

准备好一个编译好的phoenix二进制包， 按照下面的步骤：
（1）、解压phoenix-[version]-bin.tar
（2）、将phoenix- [version] -server.jar添加到所有HBase region server 的classpath路径中，并删除任何以前的版本。最简单的方法是将其复制到HBase lib目录中。
（3）、重启hbase

如果需要phoenix支持index、事务等功能， 需要修改配置文件， 具体可参见官网。

下面是我单机版的hbase， 使用phoenix支持索引， 修改了hbase-site.xml配置文件：
增加如下配置：

```
<property>
    <name>hbase.regionserver.wal.codec</name>
    <value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>
</property>
```

二 使用Phoenix

1. Getting Started

进入sql的交互界面：
第一次进入会比较慢， 因为会初始化元数据：

```
./bin/sqlline.py spark1234:12181
```

第一次进入， 使用!table命令查看， 可以看到元数据相关的4张表：

```
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
18/02/20 08:16:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bu
java classes where applicable
Connected to: Phoenix (version 4.13)
Driver: PhoenixEmbeddedDriver (version 4.13)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
Building list of tables and columns for tab-completion (set fastconnect to true to skip)...
92/92 (100%) Done
Done
sqlline version 1.2.0
0: jdbc:phoenix:spark1234:12181> !table
+------------+--------------+-------------+---------------+----------+------------+----------------------------+
| TABLE_CAT  | TABLE_SCHEM  | TABLE_NAME  |  TABLE_TYPE   | REMARKS  | TYPE_NAME  | SELF_REFERENCING_COL_NAME  |
+------------+--------------+-------------+---------------+----------+------------+----------------------------+
|            | SYSTEM       | CATALOG     | SYSTEM TABLE  |          |            |                            |
|            | SYSTEM       | FUNCTION    | SYSTEM TABLE  |          |            |                            |
|            | SYSTEM       | SEQUENCE    | SYSTEM TABLE  |          |            |                            |
|            | SYSTEM       | STATS       | SYSTEM TABLE  |          |            |                            |
+------------+--------------+-------------+---------------+----------+------------+----------------------------+
0: jdbc:phoenix:spark1234:12181>
```

执行sql脚本：

```
./bin/sqlline.py spark1234:12181 ./examples/STOCK_SYMBOL.sql
```

加载数据：
可以使用bin/psql.py加载csv数据或者执行sql脚本插入数据到phoenix：

```
./bin/psql.py spark1234:12181 -t STOCK_SYMBOL ./examples/STOCK_SYMBOL.csv
```

```
[hadoop@spark1234 phoenix-4.13.2-cdh5.7.0]$ ./bin/sqlline.py spark1234:12181
Traceback (most recent call last):
  File "./bin/sqlline.py", line 27, in <module>
    import argparse
ImportError: No module named argparse
[hadoop@spark1234 phoenix-4.13.2-cdh5.7.0]$
```

缺少argparse模块， 从网上down一个argparse.py文件， 复制到bin目录下即可。

创建表：CREATE TABLE IF NOT EXISTS STOCK_SYMBOL (SYMBOL VARCHAR NOT NULL PRIMARY KEY, COMPANY VARCHAR);

插入数据：
UPSERT INTO STOCK_SYMBOL VALUES ('CRM','SalesForce.com');

查看数据：
SELECT * FROM STOCK_SYMBOL;

导入数据：

./bin/psql.py spark1234:12181 -t STOCK_SYMBOL ./examples/STOCK_SYMBOL.csv

-t是指定表名

```
[hadoop@spark1234 phoenix-4.13.2-cdh5.7.0]$ ./bin/psql.py spark1234:12181 -t STOCK_SYMBOL ./examples/STOCK_SYMBOL.csv
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/soft/phoenix-4.13.2-cdh5.7.0/phoenix-4.13.2-cdh5.7.0-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/app/hadoop-2.6.0-cdh5.7.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
18/02/20 02:24:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
csv columns from database.
CSV Upsert complete. 9 rows upserted
Time: 0.173 sec(s)
```

查看导入的数据：

```
0: jdbc:phoenix:spark1234:12181> select * from STOCK_SYMBOL;
+---------+-----------------------+
| SYMBOL  |        COMPANY        |
+---------+-----------------------+
| AAPL    | APPLE Inc.            |
| GOOG    | Google                |
| HOG     | Harlet-Davidson Inc.  |
| HPQ     | Hewlett Packard       |
| INTC    | Intel                 |
| MSFT    | Microsoft             |
| CRM     | SALESFORCE            |
| WAG     | Walgreens             |
| WMT     | Walmart               |
+---------+-----------------------+
9 rows selected (0.051 seconds)
```

```
./bin/psql.py spark1234:12181 ./examples/WEB_STAT.sql ./examples/WEB_STAT.csv ./examples/WEB_STAT_QUERIES.sql
```

WEB_STAT.sql是建表语句， WEB_STAT.csv是数据， WEB_STAT_QUERIES.sql是查询语句。

2. Tuning Guide

对Phoenix调优可能会很复杂， 但是对于Phoenix的工作原理有一点了解会对其读写性能有重大改进。 对性能影响最重要的因素是设计schema， 特别是当schema的设计影响

Hbase底层的rowkey。

需要注意的是， 当你的应用程序执行点查询或者小的范围扫描时， Phonix和Hbase可以很好的工作， 这个可以通过良好的主键设计来实现。 但是如果你的应用程序需要做大量的全表扫描， Phonix和Hbase就不是胜任这项工作最好的工具了。 相反，比如使用Parquet等直接将数据写入hdfs的工具。

3. Primary Keys

底层的rowkey设计是Phonix性能中唯一最重要的因素， 在设计时候正确的设置row key非常重要， 因为在后期， 如果不重写数据和索引表， 你是无法更高row key的。

Phoenix主键将关联创建Hbase底层的row key。 主键约束应该按照常用的查询模式对齐的方式来选择和排序（比如复合主键， 主键列的顺序选择）-使用最频繁查询的列作为主键。 例如， 当你使用包含组织id的列作为主键的引导列， 则可以轻松选择与特定的组织有关所有行。 你也可以使用时间戳作为主键， 这样可以过滤时间范围之外的列， 提升范围扫描效率。

每个主键都会产生一定的成本， 因为整个行健会被追加到内存和磁盘中的每一条数据上。 行健越大， 存储的开销就越大。 因此， 尽可能将信息紧凑地存储在用于主键的列中，比如存储增量数据而不是完整的时间戳。

总而言之， 最佳的做法是设计的组成row key的主键可以扫描最小量的数据。

Tips:选择主键时， 将最频繁过滤查询的列作为前置列。 如果使用Order BY子句， 确保主键列匹配ORDER BY子句的列。

单调递增的主键：

如果发现主键单调递增， 使用salting来帮助将写分布到集群中并提高并行度。

```
CREATE TABLE … ( … ) SALT_BUCKETS = N
```

为了获得最佳性能， salt buckets的数量应该近似等于region servers的数量。 不要自动地salt， 只有在遇到热点时才使用salt。 salt的缺点是增加了数据读取的成本， 因为当你要查询数据时， 必须需要运行多个范围扫描查询。

4. General Tips

Todo...

三、Phoenix整合JDBC、Spark

1. Phoenix整合Spark

手工将phoenix的jar包（phoenix-4.13.2-cdh5.7.0-client.jar）import到项目中。

2. Phoenix整合JDBC

启动服务：bin/queryserver.py start

端口号：8765

手工将phoenix的jar包（phoenix-4.13.2-cdh5.7.0-client.jar 和phoenix-4.13.2-cdh5.7.0-thin-client.jar）import到项目中。

注意： phoenix-4.13.2-cdh5.7.0-thin-client.jar和spark项目中jar会有冲突， 因此最好不要在spark项目中创建Phoenix的jdbc程序。