Cloudera Runtime 7.2.2

Using Apache Phoenix to Store and Access Data

Date published: 2020-02-29
Date modified: 2020-10-07

https://docs.cloudera.com/

Legal Notice

Contents

Mapping Phoenix Schemas to HBase Namespaces

You can map a Phoenix schema to an HBase namespace to gain multitenancy features in Phoenix.

Important: You must use this feature only in a CDP Private Cloud Base deployment. This feature is configured automatically in a CDP Public Cloud deployment.

HBase, the underlying storage engine for Phoenix, has namespaces to support multi-tenancy features. Multitenancy helps an HBase user or administrator to perform access control and quota management tasks. Also, namespaces enable tighter control of where a particular data set is stored on RegionsServers.

Enable namespace mapping

You can enable namespace mapping by configuring a set of properties using Cloudera Manager.

About this task

After you set the properties to enable the mapping of Phoenix schemas to HBase namespaces, reverting the property settings renders the Phoenix database unusable. Test or carefully plan the Phoenix to HBase namespace mappings before implementing them.

Important: Cloudera recommends that you enable namespace mapping. If you decide not to enable this feature, you can skip the following steps.

To enable Phoenix schema mapping to a non-default HBase namespace:

Procedure

1. Go to the HBase service.
2. Click the Configuration tab.
3. Select Scope (Service-Wide) .
4. Locate the HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml property or search for it by typing its name in the Search box.
5. Add the following property values:

   Name: phoenix.schema.isNamespaceMappingEnabled

   Description: Enables mapping of tables of a Phoenix schema to a non-default HBase namespace. To enable mapping of a schema to a non-default namespace, set the value of this property to true. The default setting for this property is false.

   Value: true

   Name: phoenix.schema.mapSystemTablesToNamespace

   Description: With true setting (default): After namespace mapping is enabled with the other property, all system tables, if any, are migrated to a namespace called system. With false setting: System tables are associated with the default namespace.

   Value: true
6. Select Scope Gateway .
7. Locate the HBase Client Advanced Configuration Snippet (Safety Valve) for hbase-site.xml property or search for it by typing its name in the Search box.

8. Add the following property values:

   Name: phoenix.schema.isNamespaceMappingEnabled

   Description: Enables mapping of tables of a Phoenix schema to a non-default HBase namespace. To enable mapping of the schema to a non-default namespace, set the value of this property to true. The default setting for this property is false.

   Value: true

   Name: phoenix.schema.mapSystemTablesToNamespace

   Description: With true setting (default): After namespace mapping is enabled with the other property, all system tables, if any, are migrated to a namespace called system.With false setting: System tables are associated with the default namespace.

   Value: true
9. Enter a Reason for change, and then click Save Changes to commit the changes.
10. Restart the role and service when Cloudera Manager prompts you to restart.

Note: If you do not want to map Phoenix system tables to namespaces because of compatibility issues with your current applications, set the phoenix.schema.mapSystemTablesToNamespace property to false.

Associating Tables of a Schema to a Namespace

Important: You must use this feature only in a CDP Private Cloud Base deployment. This feature is configured automatically in a CDP Public Cloud deployment.

After you enable namespace mapping on a Phoenix schema that already has tables, you can migrate the tables to an HBase namespace. The namespace directory that contains the migrated tables inherits the schema name.

For example, if the schema name is store1, then the full path to the namespace is $hbase.rootdir/data/store1. System tables are migrated to the namespace automatically during the first connection after enabling namespace properties.

Associate table in a customized Kerberos environment

You can run a command to associate a table in a customized environment without Kerberos.

Before you begin

In a Kerberos-secured environment, you must have admin privileges (user hbase) to complete the following task.

Procedure

- Run a command to migrate a table of a schema to a namespace, using the following command syntax for the options that apply to your environment:

```
phoenix-psql
ZooKeeper_hostnames:2181
:zookeeper.znode.parent
:principal_name
:HBase_headless_keytab_location
;TenantId=tenant_Id
;CurrentSCN=current_SCN
-m
schema_name.table_name
```

Associate a table in a non-customized environment without Kerberos

You can run a command to associate a table in a non-customized environment without Kerberos.

Procedure

- Run the following command to associate a table:

```
phoenix-psql ZooKeeper_hostname -m Schema_name.table_name
```

Using JDBC API with Apache Phoenix

You can create and interact with Apache HBase tables using Phoenix DDL/DML statements through its standard JDBC API. Apache Phoenix JDBC driver can be easily embedded in any application that supports JDBC.

Apache Phoenix enables you to use the standard JDBC API to create and access Apache HBase tables. You can use JDBC APIs with Apache Phoenix instead of native Apache HBase client APIs to create tables, insert, and query data.

Apache Phoenix tables have a 1:1 relationship with Apache HBase tables. You can choose to create a new table using an Apache Phoenix DDL statement such as CREATE TABLE, or create a view on an existing Apache HBase table using the VIEW statement.

Important: Modifying an Apache Phoenix table using Apache HBase native API is not supported. Doing this leads to errors, inconsistent indexes, incorrect query results, or sometimes to corrupt data.

To use the Apache Phoenix JDBC driver, you must embed the driver in your application that supports JDBC. Apache Phoenix has two kinds of JDBC drivers.

- A *thick driver* communicates directly with Apache HBase. The thick driver, therefore, needs access to all the nodes in the Apache HBase cluster.
- A *thin driver* communicates with Apache HBase through Phoenix Query Server (PQS) and requires access only to PQS. Use the thin driver to connect to PQS through Apache Knox or connect to PQS directly.

In an operational database Data Hub cluster, Data Lake (SDX cluster) provides security dependencies such as Apache Knox. Your JDBC URL string would depend on whether you want to connect directly or through Apache Knox. Before you try connecting to Apache Phoenix, check if you are in the list of allowed users in Apache Ranger allowed to access Apache Phoenix and Apache HBase.

Based on whether you want to use the thick or thin driver, you need the JAR files for the Apache HBase client, the Apache Phoenix client, and the PQS client.

For the thick driver, you need:

- hbase-client-[***VERSION***].jar
- hbase-site.xml

Note: You must add the cluster's current hbase-site.xml to the application classpath. You can get the hbase-site.xml by doing an SSH to the cluster node with the hbase-gateway role. You can copy the hbase-site.xml file from the following path /etc/hbase/hbase-site.xml or /etc/hbase/conf/hbase-site.xml.

For the thin driver, you need:

- phoenix-queryserver-client-[***VERSION***].jar

You can get these JAR files from the following locations:

- Go to /opt/cloudera/parcels/CDH/lib/phoenix/ on an operational database cluster node with the phoenix-gateway role.

  or

- Download the JAR files from the Cloudera Repository

When using the thin driver, your applications interact with the Phoenix Query Server using the Avatica API and Google Protocol Buffers serialization format.

JDBC driver location

Use the /opt/cloudera/parcels/CDH/lib/phoenix/[***PHOENIX VERSION***].jar file present in your deployment location. For example, /opt/cloudera/parcels/CDH/lib/phoenix/phoenix-5.0.0.7.2.0.0-128-client.jar

URL syntax for the thick JDBC driver

To connect to Apache Phoenix using the thick JDBC driver, you must use the following JDBC URL syntax:

```
jdbc:phoenix:[***zookeeper_quorum***]:[***zookeeper_port***]:
[***zookeeper_hbase_path***]
```

The zookeeper_quorum and zookeeper_port parameters are optional if you have added the operational database Apache HBase cluster's current hbase-site.xml to the application classpath.

Apart from the JDBC driver, the following drivers are supported:

- ODBC driver
- Python driver for Phoenix

Connecting to Apache Phoenix Query Server using the JDBC client

You can interact with Apache Phoenix using your client and Apache Phoenix Query Server (PQS).

PQS is automatically configured when you create an Operational Database Data Hub cluster. There are two ways in which you can use the thin client to interact with Phoenix:

- Connect to PQS directly
- Connect to PQS using the Apache Knox Gateway

Connect to Phoenix Query Server

You can connect to Phoenix Query Server (PQS) using the JDBC thin client without using a gateway such as Apache Knox. You must use the JDBC URL syntax to form the URL and connect to PQS.

About this task

You can use the JDBC URL syntax to form the URL and connect to PQS.

Procedure

- To connect to the PQS directly, you must use the JDBC URL syntax as shown here: jdbc:phoenix:thin:[key=value [;key=value...]]

  You must provide the correct URL, serialization, and authentication key-values to interact with the Phoenix Query Server.

  Important: Before you can connect to PQS, you must set the authentication, avatica-user, avatica-password, truststore, and truststore-password parameters in your client URL as described here: Client ReferenceEnsure that you use the certificate configured in PQS.

Example

```
jdbc:phoenix:thin:url=http://localhost:8765;serialization=PROTOBUF; authenti
cation=SPENGO;
principal=[***Prinicpal@EXAMPLE.com***];keytab=[***PATH TO THE KEYTAB
 FILE***]
```

Connect to Phoenix Query Server through Apache Knox

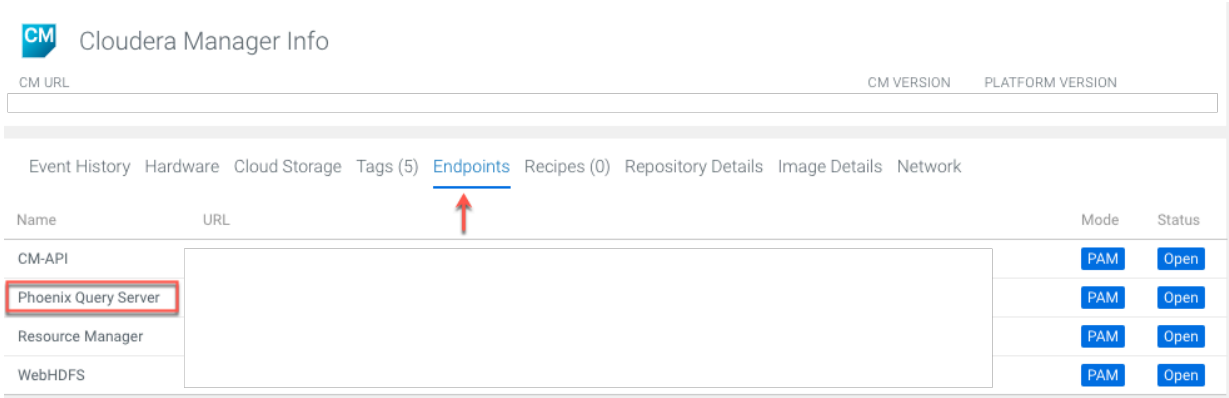
You can connect to Phoenix Query Server (PQS) using the JDBC thin client through the Apache Knox gateway. Apache Knox requires your thin client connection to be over HTTPS.

Before you begin

Ensure that you have access to Apache Phoenix and Apache HBase, and you have the required permissions configured in Ranger to connect to PQS.

Procedure

- Get the PQS Knox endpoint to connect to PQS from the CDP Data Hub user interface. Go to Data Hub cluster Endpoints Cloudera Manager Info Endpoints Phoenix Query Server .



- Use the JDBC URL in the following sytax to connect to PQS through the Apache Knox gateway:

  jdbc:phoenix:thin:url=https://[***KNOX ENDPOINT***]:[***PORT***]/[***CLUSTER NAME***]/cdp-proxy-api/avatica/;serialization=PROTOBUF;authentication=BASIC;avatica_user=[***WORKLOAD USERNAME*];avatica_password=[***WORKLOAD PASSWORD***];truststore=[***PATH TO THE KNOX TRUSTSTORE .jks FILE***];truststore_password=[***TRUSTSTORE PASSWORD***]

  The standard Oracle Java JDK distribution includes a default truststore (cacerts) that contains root certificates for many well-known CAs, including Symantec. Rather than using the default truststore, Cloudera recommends using the alternative truststore, jssecacerts. The alternative truststore is created by copying cacerts to that filename (jssecacerts). Certificates can be added to this truststore when needed for additional roles or services. This alternative truststore is loaded by Hadoop daemons at startup. Password (if there is one for the truststore) stored in a plaintext file readable by all (OS filesystem permissions set to 0440). For more information about truststores, see Understanding Keystores and Truststores.

  Note: The user name and password are for the Apache Knox gateway, and the authentication must always be set to BASIC. The truststore and truststore_password the Knox public certificate.

Example

```
jdbc:phoenix:thin:url=https://[***https://pqs.knox.endpoint:8443/gateway/
cdp-proxy-api/avatica/***]:[***8765***]
```

```
/[***clusteropdb***]/cdp-proxy-api/avatica/;serialization=PROTOBUF;authentic
ation=BASIC;avatica_user=[***WORKLOADUSERNAME***]
;avatica_password=[***WORKLOADPASSWORD***];truststore=[***/home/path/
truststore.jks***];truststore_password=[***TRUSTSTOREPASSWORD***]
```

Using Apache Phoenix-Spark connector

You can use the Apache Phoenix-Spark connector on your secure clusters to perform READ and WRITE operations. The Phoenix-Spark connector allows Spark to load Phoenix tables as Resilient Distributed Datasets (RDDs) or DataFrames and lets you save them back to Phoenix.

Connect to a secure cluster

You can connect to a secured cluster using the Phoenix JDBC connector. Enter the following syntax in the shell:

```
jdbc:phoenix:<ZK hostnames>:<ZK port>:<root znode>:<principal name>:<keytab
file location>
jdbc:phoenix:h1.cdh.local,h2.cdh.local,h3.cdh.local:2181:/hbase-secure:us
er1@cdh.LOCAL:/Users/user1/keytabs/myuser.headless.keytab
```

You need Principal and keytab parameters only if you have not run the kinit command before starting the job and want Phoenix to log you in automatically.

Considerations for setting up Spark

- Before you can use Phoenix-Spark connector for your Spark programs, you must configure your maven settings to have a repository that points to the password protected repository at https://repository.cloudera.com/artifactory/public/org/apache/phoenix/phoenix-spark/ and use the dependency:

  ```
  <dependency>
     <groupId>org.apache.phoenix</groupId>
     <artifactId>phoenix-spark</artifactId>
     <version>5.0.0-cdh7</version>
     <scope>provided</scope>
  </dependency>
  ```

You can access the Maven repository using your Enterprise Support Subscription credentials.

Configure Phoenix-Spark Connector using Cloudera Manager

Procedure

1. Go to the Spark service.
2. Click the Configuration tab.
3. Select Scope Gateway .
4. Select Category Advanced .
5. Locate the Spark Client Advanced Configuration Snippet (Safety Valve) for spark-conf/spark-defaults.conf property or search for it by typing its name in the Search box.
6. Add the following properties to ensure that all required Phoenix and HBase platform dependencies are available on the classpath for the Spark executors and drivers:

   Phoenix client JARs:

   ```
   spark.executor.extraClassPath=phoenix-client-[***VERSION***].jar
   spark.driver.extraClassPath=phoenix-client-[***VERSION***].jar
   ```

7. Enter a Reason for change, and then click Save Changes to commit the changes.
8. Restart the role and service when Cloudera Manager prompts you to restart.

What to do next

Note: You can enable your IDE by adding the following provided dependency to your build:

```
<dependency>
    <groupId>org.apache.phoenix</groupId>
    <artifactId>phoenix-spark</artifactId>
    <version>${phoenix.version}</version>
    <scope>provided</scope>
</dependency>
```

Phoenix-Spark Connector usage examples

You can refer to the following Phoenix-Spark connector examples:

- Reading Phoenix tables
- Saving Phoenix tables
- Using PySpark to READ and WRITE tables

Reading Phoenix tables

For example, you have a Phoenix table with the following DDL, you can use one of the following methods to load the table:

- As a DataFrame using the Data Source API
- As a DataFrame using a configuration object
- As an RDD using a Zookeeper URL

```
CREATE TABLE TABLE1 (ID BIGINT NOT NULL PRIMARY KEY, COL1 VARCHAR);
UPSERT INTO TABLE1 (ID, COL1) VALUES (1, 'test_row_1');
UPSERT INTO TABLE1 (ID, COL1) VALUES (2, 'test_row_2');
```

Example: Load a DataFrame using the Data Source API

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.phoenix.spark._

val sc = new SparkContext("local", "phoenix-test")
val sqlContext = new SQLContext(sc)

val df = sqlContext.load(
  "org.apache.phoenix.spark",
  Map("table" -> "TABLE1", "zkUrl" -> "phoenix-server:2181")
)

df
  .filter(df("COL1") === "test_row_1" && df("ID") === 1L)
  .select(df("ID"))
  .show
```

Example: Load as a DataFrame directly using a Configuration object

```
import org.apache.hadoop.conf.Configuration
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
```

```
import org.apache.phoenix.spark._

val configuration = new Configuration()
// Can set Phoenix-specific settings, requires 'hbase.zookeeper.quorum'

val sc = new SparkContext("local", "phoenix-test")
val sqlContext = new SQLContext(sc)

// Loads the columns 'ID' and 'COL1' from TABLE1 as a DataFrame
val df = sqlContext.phoenixTableAsDataFrame(
  "TABLE1", Array("ID", "COL1"), conf = configuration
)

df.show
```

Example: Load as an RDD using a Zookeeper URL

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.phoenix.spark._

val sc = new SparkContext("local", "phoenix-test")

// Loads the columns 'ID' and 'COL1' from TABLE1 as an RDD
val rdd: RDD[Map[String, AnyRef]] = sc.phoenixTableAsRDD(
  "TABLE1", Seq("ID", "COL1"), zkUrl = Some("phoenix-server:2181")
)

rdd.count()

val firstId = rdd1.first()("ID").asInstanceOf[Long]
val firstCol = rdd1.first()("COL1").asInstanceOf[String]
```

Saving Phoenix tables

You can refer to the following examples for saving RDDs and DataFrames.

Example: Saving RDDs

For example, you have a Phoenix table with the following DDL, you can save it as an RDD.

```
CREATE TABLE OUTPUT_TEST_TABLE (id BIGINT NOT NULL PRIMARY KEY, col1 VARCHAR
, col2 INTEGER);
```

The saveToPhoenix method is an implicit method on RDD[Product], or an RDD of Tuples. The data types must correspond to one of the Java types supported by Phoenix.

```
import org.apache.spark.SparkContext
import org.apache.phoenix.spark._

val sc = new SparkContext("local", "phoenix-test")
val dataSet = List((1L, "1", 1), (2L, "2", 2), (3L, "3", 3))
sc
  .parallelize(dataSet)
  .saveToPhoenix(
    "OUTPUT_TEST_TABLE",
    Seq("ID","COL1","COL2"),
    zkUrl = Some("phoenix-server:2181")
  )
```

Example: Saving DataFrames

The save is method on DataFrame allows passing in a data source type. You can use org.apache.phoenix.spark, and must also pass in a table and zkUrl parameter to specify which table and server to persist the DataFrame to. The column names are derived from the DataFrame's schema field names, and must match the Phoenix column names. The save method also takes a SaveMode option, for which only SaveMode.Overwrite is supported. For example, you have a two Phoenix tables with the following DDL, you can save it as a DataFrames.

Using PySpark to READ and WRITE tables

With Spark's DataFrame support, you can use pyspark to READ and WRITE from Phoenix tables.

Example: Load a DataFrame

Given a table TABLE1 and a Zookeeper url of localhost:2181, you can load the table as a DataFrame using the following Python code in pyspark:

```
df = sqlContext.read \
  .format("org.apache.phoenix.spark") \
  .option("table", "TABLE1") \
  .option("zkUrl", "localhost:2181") \
  .load()
```

Example: Save a DataFrame

Given the same table and Zookeeper URLs above, you can save a DataFrame to a Phoenix table using the following code:

```
df.write \
  .format("org.apache.phoenix.spark") \
  .mode("overwrite") \
  .option("table", "TABLE1") \
  .option("zkUrl", "localhost:2181") \
  .save()
```

Note: The functions phoenixTableAsDataFrame, phoenixTableAsRDD and saveToPhoenix all support optionally specifying a conf Hadoop configuration parameter with custom Phoenix client settings, as well as an optional zkUrl parameter for the Phoenix connection URL. If zkUrl isn't specified, it's assumed that the hbase.zookeeper.quorum property has been set in the conf parameter. Similarly, if no configuration is passed in, zkUrl must be specified.

Using Apache Phoenix-Hive connector

This connector enables you to access the Phoenix data from Hive without any data transfer. So the Business Intelligence (BI) logic in Hive can access the operational data available in Phoenix.

Using this connector, you can run a certain type of queries in Phoenix more efficiently than using Hive or other applications, however, this is not a universal tool that can run all types of queries. In some cases, Phoenix can run queries faster than the Phoenix Hive integration and vice versa. In others, you can run this tool to perform operations like many to many joins and aggregations which Phoenix would otherwise struggle to effectively run on its own. This integration is better suited for performing online analytical query processing (OLAP) operations than Phoenix.

Another use case for this connector is transferring the data between these two systems. You can use this connector to simplify the data movement between Hive and Phoenix, since an intermediate form of the data (for example, a .CSV file) is not required. The automatic movement of structured data between these two systems is the major advantage of using this tool. You should be aware that for moving a large amount of data from Hive to Phoenix CSV bulk load is preferable due to performance reasons.

Configure Phoenix-Hive Connector using Cloudera Manager

Before you begin

You must configure Phoenix-Hive connector before you can access Phoenix data from Hive. To configure the Phoenix-Hive connector using Cloudera Manager:

Procedure

1. Go to the Hive service.
2. Click the Configuration tab.
3. Select Scope Hive Cluster (Service-Wide) .
4. Select Category Advanced .
5. Locate the Hive Auxiliary JARs Directory property or search for it by typing its name in the Search box.
6. Add the following auxiliary path directory: */usr/local/phoenix-hive*.

   Important: You must manually create the */usr/local/phoenix-hive* directory, and copy the opt/cloudera/parcels/PHOENIX/lib/phoenix/phoenix-<*version*>-hive.jar on every node in the cluster that runs Hive.

   Note: You can use any directory instead of */usr/local/phoenix-hive* that Hive can read.

7. Locate the Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml property or search for it by typing its name in the Search box.
8. Add the following property values:

   Name: hive.aux.jars.path

   Value: file:///opt/cloudera/parcels/PHOENIX/lib/phoenix/phoenix-*version*-hive.jar
9. Select Scope Gateway .
10. Select Category Advanced .
11. Locate the Hive Client Advanced Configuration Snippet (Safety Valve) for hive-site.xml property or search for it by typing its name in the Search box.
12. Add the following property values:

    Name: hive.aux.jars.path

    Value: file:///opt/cloudera/parcels/PHOENIX/lib/phoenix/phoenix-*version*-hive.jar
13. Enter a Reason for change, and then click Save Changes to commit the changes.
14. Restart the role and service when Cloudera Manager prompts you to restart.

Apache Phoenix-Hive Usage Examples

You can refer to the following Phoenix-Hive connector examples:

- Creating a table
- Loading data
- Querying data

Creating a table

Creating an external Hive table requires an existing table in Phoenix. Hive manages only the Hive metadata. Dropping an external table from Hive deletes only the Hive metadata, but the Phoenix table is not deleted.

Use the create external table command to create an EXTERNAL Hive table.

```
create external table ext_table (
  i1 int,
  s1 string,
  f1 float,
  d1 decimal
)
STORED BY 'org.apache.phoenix.hive.PhoenixStorageHandler'
TBLPROPERTIES (
  "phoenix.table.name" = "ext_table",
  "phoenix.zookeeper.quorum" = "localhost",
  "phoenix.zookeeper.znode.parent" = "/hbase",
  "phoenix.zookeeper.client.port" = "2181",
  "phoenix.rowkeys" = "i1",
  "phoenix.column.mapping" = "i1:i1, s1:s1, f1:f1, d1:d1"
);
```

Following are the parameters that you could use when creating an external table:

Parameter	Default Value	Description
phoenix.table.name	The same name as the Hive table	Name of the existing Phoenix table
phoenix.zookeeper.quorum	localhost	Specifies the ZooKeeper quorum for HBase
phoenix.zookeeper.znode.parent	/hbase	Specifies the ZooKeeper parent node for HBase
phoenix.zookeeper.client.port	2181	Specifies the ZooKeeper port
phoenix.rowkeys	N/A	The list of columns to be the primary key in a Phoenix table
phoenix.column.mapping	N/A	Mappings between column names for Hive and Phoenix

Loading data

Use insert statement to load data to the Phoenix table through Hive.

```
insert into table T values (....);
insert into table T select c1,c2,c3 from source_table;
```

Querying data

You can use HiveQL for querying data in a Phoenix table. A Hive query on a single table can be as fast as running the query in the Phoenix CLI with the following property settings:

```
hive.fetch.task.conversion=more and hive.exec.parallel=true
```

Following are some of the parameters that you could use when querying the data:

Parameter	Default Value	Description
hbase.scan.cache	100	Read row size for a unit request
hbase.scan.cacheblock	false	Whether or not cache block
split.by.stats	false	If true, mappers use table statistics. One mapper per guide post.
[hive-table-name].reducer.count	1	Number of reducers. In Tez mode, this affects only single-table queries. See Limitations.

Parameter	Default Value	Description
[phoenix-table-name].query.hint	N/A	Hint for Phoenix query (for example, NO_INDEX)

Limitations of Phoenix-Hive Connector

Following are some of the limitations of Phoenix-Hive connector:

- Only 4K character specification is allowed to specify a full table. If the volume of the data is huge, then there is a possibility to lose the metadata information.
- There is a difference in the way timestamp is saved in Phoenix and Hive. Phoenix uses binary format, whereas Hive uses a text format to store data.
- Hive LLAP is not supported in this integration.