cloudera®

Apache Phoenix Guide

Important Notice

Cloudera, Inc.
395 Page Mill Road
Palo Alto, CA 94306
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: Cloudera Enterprise 5.16.x
Date: August 3, 2021

Table of Contents

Release Notes

Apache Phoenix 4.14.1 is available with CDH 5.16.2.

Starting with CDH 5.16.2, Apache Phoenix 4.14.1 parcel can be installed and used with CDH. You can now download and install the Phoenix parcel on CDH 5.16.2.

Known Issues

Potential deadlock on region opening with secondary index recovery

Distributed deadlock happens in clusters with a moderate number of regions for the data tables, secondary index tables.

Products affected: Phoenix

Releases affected: Phoenix 4.14.1 parcel

User affected: Users of Phoenix with secondary indexes

Severity (Low/Medium/High): Medium

Impact: Deadlock on region opening

Immediate action required/workaround:

Make the following configuration changes in HBase:

1) Set `hbase.master.startup.retainassign` to `false`

2) Increase the value of `hbase.regionserver.executor.openregion.threads` and restart cluster should bring up cluster.

If you continue to encounter issues, tune the following assignment manager parameters to match the count of regions for faster assignments:

- `hbase.assignment.threads.max`
- `hbase.master.namespace.init.timeout`
- `hbase.master.wait.on.regionservers.mintostart`
- `hbase.bulk.assignment.threshold.regions`
- `hbase.bulk.assignment.threshold.servers`

Unsupported Features

The following upstream Apache Phoenix 4.14.1 features are currently not supported in the Cloudera parcel of Phoenix:

- Hive, Pig, Flume, Kafka and MapReduce integration
- Phoenix Query Server
- Cross-row and cross-table transaction support

Prerequisites

- Phoenix requires HBase services running in your CDH cluster where you want to install Phoenix..
- The current release of Cloudera's parcel of Apache Phoenix 4.14.1 is supported on CDH 5.16.2.

For more information about the hardware and software requirements, see the Cloudera Enterprise 5.x Release Notes.

Installing Apache Phoenix using Cloudera Manager

You can install Apache Phoenix using Cloudera Manager 5.16.2. To install Apache Phoenix:

- Download and install the Phoenix parcel
- Configure HBase for use with Phoenix
- Validate the Phoenix installation

Downloading and Installing the Phoenix Parcel

Important: As of February 1, 2021, all downloads of CDH and Cloudera Manager require a username and password and use a modified URL. You must use the modified URL, including the username and password when downloading the repository contents described below. You may need to upgrade Cloudera Manager to a newer version that uses the modified URLs.

This can affect new installations, upgrades, adding new hosts to a cluster, downloading a new parcel, and adding a new cluster.

For more information, see Updating an existing CDH/Cloudera Manager deployment to access downloads with authentication.

Minimum Required Role: **Cluster Administrator** (also provided by **Full Administrator**)

1. In the Parcels page, click **Configuration**.
2. In the Remote Parcel Repository URLs section, add a new entry and type the correct Phoenix parcel repository URL in the format:

```
https://username:password@archive.cloudera.com/p/phoenix/parcel_version/parcels/
```

For example,

```
https://username:password@archive.cloudera.com/p/phoenix/5.16.2/parcels/
```

Note:

- The username and password in the URL are your Phoenix repository authentication credentials. You must have an Enterprise Support Subscription account to download and install this product.
- If your authentication credentials for the main parcel repository and the Phoenix parcel repository are the same. You can use **HTTP authentication username override for Cloudera Repositories** and **HTTP authentication password override for Cloudera Repositories** to enter your user name and password, respectively.

3. Click **Save Changes**.

   The Phoenix parcel will now be available to download.

4. In the Location selector, click the ***cluster name*** in which you want to install Phoenix.

   The Phoenix parcel is listed as **Available Remotely** in the Status column.

5. Click **Download** to download the Phoenix parcel to your local repository. The status changes to Downloading.
6. Click **Distribute**. The status changes to Distributing. During distribution, you can:

   - Click the Details link in the Status column to view the Parcel Distribution Status page.

- Click Cancel to cancel the distribution. When the Distribute action completes, the button changes to Activate, and you can click the Distributed status link to view the status page.

7. Click **Activate** to activate the parcel.

   You will see Distributed, Activated in the Status column.

Configuring HBase for use with Phoenix

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

Before you start using Phoenix, you must configure the following HBase properties using Cloudera Manager:

1. Go to the HBase service.
2. Click the **Configuration** tab.
3. Select **Scope > HBase Cluster (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. Click View as XML, and add the following properties:
   - Set `hbase.regionserver.wal.codec` to enable custom Write Ahead Log ("WAL") edits to be written as follows:

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

   - Set the following property to enable user-defined functions:

```
<property>
 <name>phoenix.functions.allowUserDefinedFunctions</name>
 <value>true</value>
 <description>enable UDF functions</description>
</property>
```

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

Validating the Phoenix Installation

Validating a Native Phoenix Installation on an Unsecured Cluster

To validate your installation, log in as the hbase user, and run the following smoke tests from the command prompt:
:

```
phoenix-psql /opt/cloudera/parcels/phoenix parcel/lib/phoenix/examples/WEB_STAT.sql
/opt/cloudera/parcels/<phoenix parcel>/lib/phoenix/examples/WEB_STAT.csv
/opt/cloudera/parcels/<phoenix parcel>/lib/phoenix/examples/WEB_STAT_QUERIES.sql
```

Validating a Native Phoenix Installation on a Cluster Secured with Kerberos

To validate your installation, log in as the hbase user, and perform the following actions:

1. Obtain a valid Kerberos ticket by running `kinit`. For example:

```
kinit var/run/cloudera-scm-agent/process/<latest-process-id>-HBASE/hbase.keytab
hbase/<local.node.host.name>
```

Note: You can find the process directory using the following command: `ls -lrt var/run/cloudera-scm-agent/process/ | awk '{print $9}' |grep -i hbase | tail -1 360-ks_indexer-HBASE_INDEXER`

2. Run the following smoke tests from your command line in your cluster:

```
phoenix-psql
```

You will see the help displayed for the phoenix-psql script. You can test Phoenix using one of the examples that you can find here: `/opt/cloudera/parcels/Phoenix parcel/lib/phoenix/examples`

For example,

```
phoenix-psql
/opt/cloudera/parcels/PHOENIX-4.14.1-cdh5.16.2/lib/phoenix/examples/WEB_STAT.sql
```

Using Phoenix to Store and Access Data

Phoenix lets you create and interact with tables in the form of typical DDL/DML statements through its standard JDBC API. With the driver APIs, Phoenix translates SQL to native HBase API calls.

Orchestrating SQL and APIs with Apache Phoenix

You can use the standard JDBC APIs instead of the regular HBase client APIs to create tables, insert data, and query your HBase data.

Obtaining a driver for application development

Based on your application development requirements, you can obtain one of the following drivers:

JDBC driver

Use the `/opt/cloudera/parcels/phoenix parcel/lib/phoenix/phoenix parcel-client.jar` file in the Phoenix server-client repository. If you use the repository, download the JAR file corresponding to your installed CDH version.

JDBC driver as a Maven dependency

You can pull the CDH 5 artifacts from the Cloudera Maven repository from here: https://archive.cloudera.com/phoenix/5.16.2/maven-repository/. You can access the Maven repository using your Enterprise Support Subscription credentials.

Creating and Using User-Defined Functions (UDFs) in Phoenix

With a user-defined function (UDF), you can extend the functionality of your SQL statements by creating scalar functions that operate on a specific tenant. For details about creating, dropping, and how to use UDFs for Phoenix, see User-defined functions on the Apache website.

For more information, see https://phoenix.apache.org/udf.html.

Mapping Phoenix Schemas to HBase Namespaces

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

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.

Enabling Namespace Mapping

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

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

If namespace mapping is enabled, then the tables created with the schema will be mapped to the namespace. The old clients will not work after this property is enabled. 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:

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.
11. Restart the service.

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.mapSystemTablesToNamespaceproperty` to false.

Associating Tables of a Schema to a Namespace

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.

Associating table in a non-customized environment without Kerberos

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

Run the following command to associate a table:

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

Associating table in a customized Kerberos environment

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

1. 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
:HBase_headless_keytab_location
:principal_name
;TenantId=tenant_Id
;CurrentSCN=current_SCN
-m
schema_name.table_name
```

Additional information for valid command parameters:

- `ZooKeeper_hostnames`

  Enter the ZooKeeper hostname or hostnames that compose the ZooKeeper quorum. If you enter multiple hostnames, enter them as comma-separated values. This parameter is required. You must append the colon and ZooKeeper port number if you invoke the other security parameters in the command. The default port number is 2181.

- `zookeeper.znode.parent`

  This setting is defined in the `hbase-site.xml` file.

- **-m** `schema_name.table_name`

  The `-m` argument is required. There is a space before and after the `-m` option.

Using Phoenix Client to Load Data

You must use the Phoenix client to load data into the HBase database and also to write to the Phoenix tables.

Index updates are automatically generated by the Phoenix client and there is no user intervention or effort required. Whenever a record is written to the Phoenix tables, the client generates the updates for the indexes automatically.

Note: If the Phoenix table has indexes, you can use the JDBC driver or CSV bulk load table to update or ingest data.

It is highly recommended that you use the Phoenix client to load data into the HBase database and also to write to the Phoenix tables. If you use the HBase APIs to write data to a Phoenix data table, the indexes for that Phoenix data table will not be updated.

Using the Index in Phoenix

Apache Phoenix automatically uses indexes to service a query. Phoenix supports global and local indexes. Each is useful in specific scenarios and has its own performance characteristics.

Global indexes in Phoenix

You can use global indexes for READ-heavy use cases. Each global index is stored in its own table and thus is not co-located with the data table. With global indexes, you can disperse the READ load between the main and secondary index table on different RegionServers serving different sets of access patterns. A Global index is a covered index. It is used for queries only when all columns in that query are included in that index.

Local indexes in Phoenix

You can use local indexes for WRITE-heavy use cases. Each local index is stored within the data table. With global indexes, you can use local indexes even when all columns referenced in a query are not contained in the index.

This is done by default for local indexes because the table and index data reside on the same region server and hence it ensures that the lookup is local.

Understanding Apache Phoenix-Spark Connector

You can use Apache Phoenix-spark plugin on your secured clusters to perform READ and WRITE operations.

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:user1@cdh.LOCAL:/Users/user1/keytabs/myuser.headless.keytab
```

You need Principal and keytab parameters only if you have not done the kinit 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 1.6 programs, you must configure your maven settings to have a repository that points to the password protected repository at https://archive.cloudera.com/phoenix/5.16.2/maven-repository/ and use the dependency:

```
<dependency>
   <groupId>org.apache.phoenix</groupId>
   <artifactId>phoenix-spark</artifactId>
   <version>4.14.1-cdh5.16.2</version>
   <scope>provided</scope>
</dependency>
```

> **Note:** You can access the Maven repository using your Enterprise Support Subscription credentials. The Phoenix-Spark connector works with Spark 1.6, but it does not support Spark 2 that can be installed on top of CDH 5.16.

Configure Phoenix-Spark Connector using Cloudera Manager

Minimum Required Role: Configurator (also provided by **Cluster Administrator, Full Administrator**)

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-<version>-client.jar
spark.driver.extraClassPath=phoenix-<version>-client.jar
```

   Phoenix-Spark JARs:

```
spark.executor.extraClassPath=phoenix-spark-<version>.jar
spark.driver.extraClassPath=phoenix-spark-<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.

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.

Performance Tuning

You can use the following configuration properties to tune Phoenix to work optimally on your cluster. You can tune your Phoenix deployment by configuring certain Phoenix specific properties that are configured both on the client and server side hbase-site.xml files. For a full list of Phoenix Tuning properties that are available, see the Apache Phoenix tuning guide.

Table 1:

Property	Description	Default
phoenix.query.threadPoolSize	The number of threads in client-side thread pool executor. As the number of machines/cores in the cluster grows, this value should be increased.	128
phoenix.query.queueSize	Max queue depth of the bounded round robin backing the client side thread pool executor, beyond which an attempt to queue additional work is rejected. If zero, a SynchronousQueue is used instead of the bounded round-robin queue. The default value is 5000.	5000
phoenix.stats.guidepost.width	The server-side parameter that specifies the number of bytes between guideposts. A smaller amount increases parallelization, but also increases the number of chunks which must be merged on the client side. The default value is 100 MB.	104857600
phoenix.stats.guidepost.per.region	The server-side parameter that specifies the number of guideposts per region. If set to a value greater than zero, then the guidepost width is determined by MAX_FILE_SIZE of table/phoenix .stats.guidepost.per.region. Otherwise, if not set, then the phoenix.stats.guidepost.width parameter is used. No default value.	None
phoenix.stats.updateFrequency	The server-side parameter that determines the frequency in milliseconds for which statistics will be refreshed from the statistics table and subsequently used by the client. The default value is 15 min.	900000
phoenix.query.spoolThresholdBytes	Threshold size in bytes after which results from parallelly executed query results are spooled to disk. Default is 20 mb.	20971520

Property	Description	Default
phoenix.query.maxSpoolToDiskBytes	Threshold size in bytes up to which results from parallelly executed query results are spooled to disk above which the query will fail. Default is 1 GB.	1024000000
phoenix.query.maxGlobalMemoryPercentage	Percentage of total heap memory (i.e. Runtime.getRuntime().maxMemory()) that all threads may use. Only course grain memory usage is tracked, mainly accounting for memory usage in the intermediate map built during group by aggregation. When this limit is reached, the clients' block attempting to get more memory, essentially throttling memory usage. Defaults to 15%	15
phoenix.query.maxGlobalMemorySize	Max size in bytes of total tracked memory usage. By default it is not specified, however, if present, the lower of this parameter and the phoenix.query.maxGlobalMemoryPercentage will be used.	
phoenix.query.maxGlobalMemoryWaitMs	The maximum amount of time that a client will block while waiting for more memory to become available. After this amount of time, an InsufficientMemoryException error is displayed. The default value is 10 seconds.	10000
phoenix.query.maxTenantMemoryPercentage	The maximum percentage of phoenix.query.maxGlobalMemoryPercentage that a tenant is allowed to consume. After this percentage, an InsufficientMemoryException error is displayed. Default is 100%	100
phoenix.mutate.maxSize	The maximum number of rows that may be batched on the client before a commit or rollback must be called.	500000
phoenix.mutate.batchSize	The number of rows that are batched together and automatically committed during the execution of an UPSERT SELECT or DELETE statement. This property may be overridden at connection time by specifying the UpsertBatchSize property value. Note that the connection property value does not affect the batch size used by the coprocessor when these statements are executed completely on the server side.	1000

Property	Description	Default
phoenix.query.maxServerCacheBytes	Maximum size (in bytes) of a single subquery result (usually the filtered result of a table) before compression and conversion to a hash map. Attempting to hash an intermediate subquery result of a size bigger than this setting will result in a MaxServerCacheSizeExceededException. Default 100MB.	104857600
phoenix.coprocessor. maxServerCacheTimeToLiveMs	Maximum living time (in milliseconds) of server caches. A cache entry expires after this amount of time has passed since last access. Consider adjusting this parameter when a server-side IOException("Could not find a hash cache for joinId") happens. Getting warnings like "Earlier hash cache(s) might have expired on servers" might also be a sign that this number should be increased.	30000
phoenix.query.useIndexes	Client-side property determining whether or not indexes are considered by the optimizer to satisfy a query. Default is true	true
phoenix.index.failure.handling.rebuild	Server-side property determining whether or not a mutable index is rebuilt in the background in the event of a commit failure. Only applicable for indexes on mutable, non-transactional tables. Default is true	true
phoenix.groupby.maxCacheSize	Size in bytes of pages cached during GROUP BY spilling. Default is 100Mb	102400000
phoenix.groupby. estimatedDistinctValues	Number of estimated distinct values when a GROUP BY is performed. Used to perform initial sizing with the growth of 1.5x each time reallocation is required. Default is 1000	1000
phoenix.distinct.value.compress. threshold	Size in bytes beyond which aggregate operations which require tracking distinct value counts (such as COUNT DISTINCT) will use Snappy compression. Default is 1Mb	1024000
phoenix.index.maxDataFileSizePerc	The percentage used to determine the MAX_FILESIZE for the shared index table for views relative to the data table MAX_FILESIZE. The percentage should be estimated based on the anticipated average size of a view	50

Property	Description	Default
	index row versus the data row. Default is 50%.	
phoenix.coprocessor. maxMetaDataCacheTimeToLiveMs	Time in milliseconds after which the server-side metadata cache for a tenant will expire if not accessed. Default is 30mins	180000
phoenix.coprocessor. maxMetaDataCacheSize	Max size in bytes of total server-side metadata cache after which evictions will begin to occur based on least recent access time. Default is 20Mb	20480000
phoenix.client.maxMetaDataCacheSize	Max size in bytes of total client-side metadata cache after which evictions will begin to occur based on least recent access time. Default is 10Mb	10240000
phoenix.sequence.cacheSize	Number of sequence values to reserve from the server and cache on the client when the next sequence value is allocated. Only used if not defined by the sequence itself. Default is 100	100

Frequently Asked Questions

You can refer to this to this list of questions and answers to understand Apache Phoenix and its deployment.

Can Phoenix be used for ETL use cases?

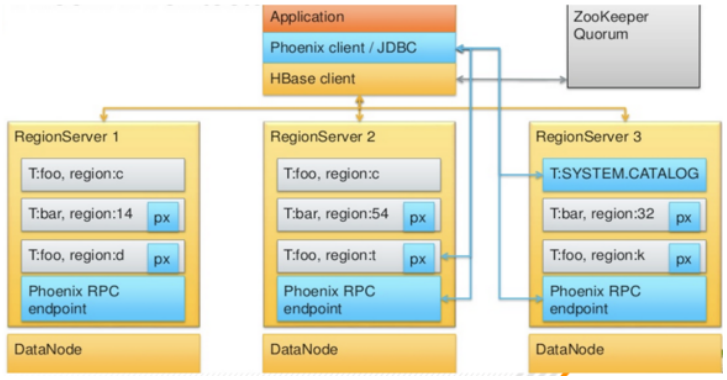
Yes. Apache Phoenix is used for OLTP (Online Transactional Processing) use cases and not OLAP (Online Analytical Processing) use cases. Although, you can use Phoenix for real-time data ingest as a primary use case.

What is the typical architecture for a Phoenix deployment?

A typical Phoenix deployment has the following:

- Application
- Phoenix Client/JDBC driver
- HBase client

A Phoenix client/JDBC driver is essentially a Java library that you should include in your Java code. Phoenix uses HBase as storage similar to how HBase uses HDFS as storage. However, the abstraction for Phoenix is not yet complete, for example, for implementing access controls, you need to set ACLs on the underlying HBase tables that contain the Phoenix data.



Are there sizing guidelines for Phoenix JDBC servers?

For Phoenix applications, you must follow the same sizing guidelines that you follow for HBase. For more information about Phoenix performance tuning, see Performance Tuning on page 17 and https://phoenix.apache.org/tuning_guide.html.

Can I govern access to the Phoenix servers?

Yes, you can use Kerberos for authentication. You can configure authorization using HBase authorization.

Can I see the individual cell timestamps in Phoenix tables? Is this something that's commonly used?

You can map HBase's native row timestamp to a Phoenix column. By doing this, you can take advantage of the various optimizations that HBase provides for time ranges on the store files as well as various query optimization capabilities built within Phoenix.

For more information, see https://phoenix.apache.org/rowtimestamp.html

What if the Phoenix index is being built asynchronously and data is added to a table during indexing?

Phoenix does local Indexing for deadlock prevention during global index maintenance.: Phoenix also atomically rebuild index partially when index update fails (PHOENIX-1112).

How do sequences work in Phoenix?

Sequences are a standard SQL feature that allows for generating monotonically increasing numbers typically used to form an ID.

For more information, see https://phoenix.apache.org/sequences.html.

What happens to a Phoenix write when a RegionServer fails?

Writes are durable and durability is defined by a WRITE that is committed to disk (in the Write Ahead Log). So in case of a RegionServer failure, the write is recoverable by replaying the WAL. A “complete” write is one that has been flushed from the WAL to an HFile. Any failures will be represented as exceptions.

Can I do bulk data loads in Phoenix?

Yes, you can do bulk inserts in Phoenix. For more information see https://phoenix.apache.org/bulk_dataload.html.

Can I access a table created by Phoenix using the standard HBase API?

Yes, but is it is not recommended or supported. Data is encoded by Phoenix, so you have to decode the data for reading. Writing to the HBase tables directly would result in corruption in Phoenix.

Can I map a Phoenix table over an existing HBase table?

Yes, as long as Phoenix data types are used. You have to use asynchronous indexes and manually update them since Phoenix won’t be aware of any updates.

What are guideposts?

For information about guideposts, see https://phoenix.apache.org/update_statistics.html.

Uninstalling Phoenix Parcel

Minimum Required Role: Cluster Administrator (also provided by **Full Administrator**)

If you want to stop using Phoenix and want to remove the Phoenix parcel completely, you must follow these steps in Cloudera Manager:

> **Important:** Before you uninstall the Phoenix parcel, you must disable all your Phoenix system tables, and tables created using Phoenix.

1. Revert the configurations that you set in the *Configure HBase for use with Phoenix* section.
2. Restart the HBase service.
3. From the Parcels page, in the Location selector, choose the cluster from which you want to uninstall Phoenix. For example, Cluster 1.
4. In the row that has the parcel name Phoenix, click Deactivate. The Deactivate confirmation window appears.
5. Click **Deactivate**.
6. Restart the HBase service.

Appendix: Apache License, Version 2.0

SPDX short identifier: Apache-2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims

licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```