

Hortonworks Data Platform

Data Integration Services with HDP

(Oct 28, 2014)

Hortonworks Data Platform: Data Integration Services with HDP

Copyright © 2012-2014 Hortonworks, Inc. Some rights reserved.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including YARN, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, [training](#) and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the [Hortonworks Data Platform](#) page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the [Support](#) or [Training](#) page. Feel free to [Contact Us](#) directly to discuss your specific needs.



Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 3.0 License.
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

Table of Contents

1. Using Data Integration Services Powered by Talend	1
1.1. Prerequisites	1
1.2. Instructions	2
1.2.1. Deploying Talend Open Studio	2
1.2.2. Writing a Talend Job for Data Import	3
1.2.3. Modifying the Job to Perform Data Analysis	4
2. Using Apache Hive	8
2.1. Hive Documentation	8
2.2. New Feature: Cost-based SQL Optimization	9
2.3. New Feature: Streaming Data Ingestion	10
2.4. Vectorization	11
2.4.1. Enable Vectorization in Hive	11
2.4.2. Log Information about Vectorized Execution of Queries	11
2.4.3. Supported Functionality	11
2.4.4. Unsupported Functionality	12
2.5. Comparing Beeline to the Hive CLI	12
2.5.1. Beeline Operating Modes and HiveServer2 Transport Modes	12
2.5.2. Connecting to Hive with Beeline	14
2.6. Hive ODBC and JDBC Drivers	15
2.7. Storage-based Authorization in Hive	21
2.8. Troubleshooting Hive	21
2.9. Hive JIRAs	21
3. SQL Compliance	22
3.1. New Feature: Authorization with Grant And Revoke	22
3.2. New Feature: Transactions	23
3.3. New Feature: Subqueries in WHERE Clauses	30
3.3.1. Understanding Subqueries in SQL	30
3.3.2. Restrictions on Subqueries in WHERE Clauses	31
3.4. New Feature: Common Table Expressions	32
3.5. New Feature: Quoted Identifiers in Column Names	33
3.6. New Feature: CHAR Data Type Support	33
4. Using HDP for Metadata Services (HCatalog)	35
4.1. Using HCatalog	35
4.2. Using WebHCat	36
4.2.1. Technical Update: WebHCat Standard Parameters	37
5. Using Apache HBase	39
5.1. New Feature: Cell-level ACLs	39
5.2. New Feature: Column Family Encryption	39
6. Using HDP for Workflow and Scheduling (Oozie)	40
7. Using Apache Sqoop	42
7.1. Apache Sqoop Connectors	42
7.2. Sqoop Import Table Commands	43
7.3. Netezza Connector	43
7.3.1. Extra Arguments	43
7.3.2. Direct Mode	43
7.3.3. Null String Handling	44
7.4. Sqoop-HCatalog Integration	45
7.4.1. HCatalog Background	45

7.4.2. Exposing HCatalog Tables to Sqoop	46
7.4.3. Controlling Transaction Isolation	48
7.4.4. Automatic Table Creation	48
7.4.5. Delimited Text Formats and Field and Line Delimiter Characters	48
7.4.6. HCatalog Table Requirements	49
7.4.7. Support for Partitioning	49
7.4.8. Schema Mapping	49
7.4.9. Support for HCatalog Data Types	50
7.4.10. Providing Hive and HCatalog Libraries for the Sqoop Job	50
7.4.11. Examples	50

List of Tables

2.1. CBO Configuration Parameters	10
2.2. Beeline Modes of Operation	12
2.3. HiveServer2 Transport Modes	13
2.4. Authentication Schemes with TCP Transport Mode	13
2.5. JDBC Connection Parameters	15
2.6.	18
2.7.	19
2.8.	20
3.1. Configuration Parameters for Standard SQL Authorization	22
3.2. HiveServer2 Command-Line Options	22
3.3. Hive Compaction Types	24
3.4. Hive Transaction Configuration Parameters	24
3.5. Trailing Whitespace Characters on Various Databases	34
7.1. Supported Netezza Extra Arguments	43
7.2. Supported Export Control Arguments	44
7.3. Supported Import Control Arguments	44

1. Using Data Integration Services Powered by Talend

Talend Open Studio for Big Data is a powerful and versatile open-source data integration solution. It enables an enterprise to work with existing data and systems, and use Hadoop to power large-scale data analysis across the enterprise.

Talend Open Studio (TOS) is distributed as an add-on for the Hortonworks Data Platform (HDP). TOS uses the following HDP components:

- Enables users to read/write from/to Hadoop as a data source/sink.
- HCatalog Metadata services enables users to import raw data into Hadoop (HBase and HDFS), and to create and manage schemas.
- Pig and Hive are used to analyze these data sets.
- Enables users to schedule these ETL jobs on a recurring basis on a Hadoop Cluster using Oozie.

This document includes the following sections:

- [Prerequisites](#)
- [Instructions](#)
 - [Deploying Talend Open Studio](#)
 - [Writing a Talend Job for Data Import](#)
 - [Modifying the Job to Perform Data Analysis](#)

For more information on Talend Open Studio, see [Talend Open Studio v5.3 Documentation](#).

1.1. Prerequisites

- Ensure that you have deployed HDP on all the nodes in your cluster. For instructions on deploying HDP, see "Getting Ready to Install" in *Installing HDP Using Apache Ambari* [here](#).
- Ensure that you create a home directory for the user launching the TOS in the HDFS cluster.

EXAMPLE: If `hdptestuser` is responsible for launching TOS, execute the following command on the gateway machine as the administrator user (HDFS user) to create a home directory:

```
% hadoop dfs -mkdir /user/hdptestuser
```

- Ensure that the user launching the TOS has appropriate permissions on the HDP cluster.

EXAMPLE: If `hdptestuser` is responsible for launching TOS, execute the following command on the gateway machine as the administrator user (HDFS user) to provide the required permissions:

```
% hadoop dfs -chown hdptestuser:hdptestuser /user/hdptestuser
```

1.2. Instructions

This section provides you instructions on the following:

- [Deploying Talend Open Studio](#)
- [Writing Talend job for data import](#)
- [Modifying the Job to Perform Data Analysis](#)

1.2.1. Deploying Talend Open Studio

Use the following instructions to set up Talend Open Studio:

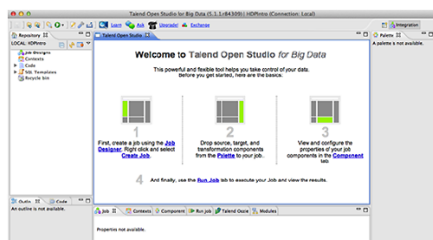
1. Download and launch the application.
 - a. Download the Talend Open Studio add-on for HDP from [here](#).
 - b. After the download is complete, unzip the contents in an install location.
 - c. Invoke the executable file corresponding to your operating system.
 - d. Read and accept the end-user license agreement.

Talend displays the "Welcome to Talend Open Studio" dialog.
2. Create a new project.
 - a. Ignore the Select a Demo Project field. Instead, create a new project. Provide a project name (for example, HDPIntro), then click **Create**.

Talend displays the **New Project** dialog.
 - b. Click **Finish** on the **New Project** dialog.
 - c. Select the newly-created project, then click **Open**.

The **Connect To TalendForge** dialog displays.
 - d. Either choose to register now, or click **Skip** to continue.

Talend displays the progress information bar. When progress is complete, Talend displays a Welcome window.
 - e. Wait for the application to initialize, then click **Start now!**. Talend displays the main Talend Open Studio (TOS) window.

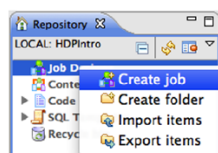


1.2.2. Writing a Talend Job for Data Import

This section describes how to design a simple job for importing a file into the Hadoop cluster.

1. Create a new job.

- a. In the Repository tree view, right-click the **Job Designs** node, then select **Create job**.



- b. In the New Job wizard, provide a name for the new job (for example, HDPJob), then click **Finish**.
- c. An empty design workspace, named after the new job, opens.

2. Create a sample input file.

- a. At your master deployment machine's the `/tmp` directory, create a text file (for example: `input.txt`) with the following contents:

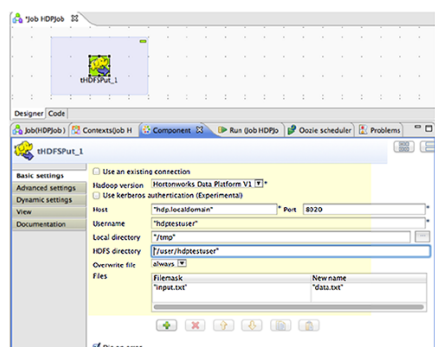
```
101;Adam;Wiley;Sales
102;Brian;Chester;Service
103;Julian;Cross;Sales
104;Dylan;Moore;Marketing
105;Chris;Murphy;Service
106;Brian;Collingwood;Service
107;Michael;Muster;Marketing
108;Miley;Rhodes;Sales
109;Chris;Coughlan;Sales
110;Aaron;King;Marketing
```

3. Build the job.

Jobs are composed of components that are available in the Palette.

- a. Expand the **Big Data** tab in the Palette.
- b. Click the `tHDFSput` component, then select the design workspace to drop this component.

- c. To define the component in its **Basic Settings** view, double-click **tHDFSput**.
- d. Set the values in the Basic Settings corresponding to your HDP cluster:



4. You now have a working job.

To run the job, click **Play**. Talend displays something similar to the following:

```
Starting job HDPJob at 12:56 12/06/2012.

[statistics] connecting to socket on port 3509
2012-06-12 12:56:38 java[10933:c07] Unable to
load realm info from SCDynamicStore
[statistics] connected
[statistics] disconnected
Job HDPJob ended at 12:56 12/06/2012. [exit code=0]
```

5. Verify the import operation. From the gateway machine or the HDFS client, open a console window and execute:

```
hadoop dfs -ls /user/testuser/data.txt
```

At your terminal window, Talend displays:

```
Found 1 items
-rw-r--r-- 3 testuser testuser
252 2012-06-12 12:52 /user/
testuser/data.txt
```

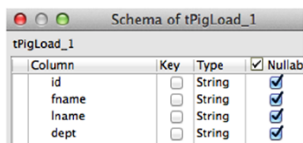
This message indicates that the local file was successfully created in your Hadoop cluster.

1.2.3. Modifying the Job to Perform Data Analysis

Use the following instructions to aggregate data using Apache Pig.

1. Add the Pig component from the Big Data Palette.
 - a. In the Big Data palette, select the **Pig** tab.
 - b. Click the **tPigLoad** component and drag it into the design workspace.
2. Define basic settings for the Pig component.
 - a. To define the component's basic settings, double-click **tPigLoad**.

- b. Click **Edit Schema**. Define the schema of the input data as shown below, then click **OK**:



Schema of tPigLoad_1

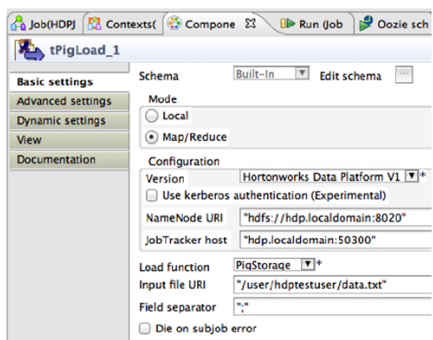
Column	Key	Type	Nullab
id	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>
fname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>
lname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>
dept	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>

- c. Provide the values for the mode, configuration, NameNode URI, JobTracker host, load function, and input file URI fields as shown.



Important

Ensure that the NameNode URI and the JobTracker hosts correspond to accurate values available in your HDP cluster. (The Input File URI corresponds to the path of the previously imported `input.txt` file.)



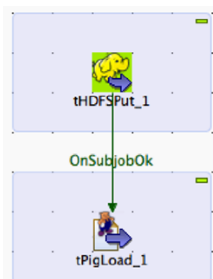
tPigLoad_1

Schema: Built-In Edit schema

Mode:
☐ Local
☒ Map/Reduce

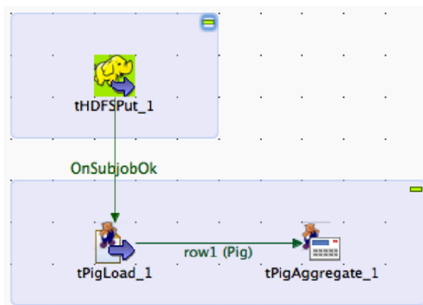
Configuration:
 Version: Hortonworks Data Platform V1
☐ Use kerberos authentication (Experimental)
 NameNode URI: hdfs://hdp.localdomain:8020
 JobTracker host: hdp.localdomain:50300
 Load function: PigStorage
 Input file URI: /user/hdptestuser/data.txt
 Field separator: *,*
☐ Die on subjob error

3. Connect the Pig and HDFS components to define the workflow.
- Right-click the source component (**tHDFSPut**) on your design workspace.
 - From the contextual menu, select **Trigger -> On Subjob Ok**.
 - Click the target component (**tPigLoad**).



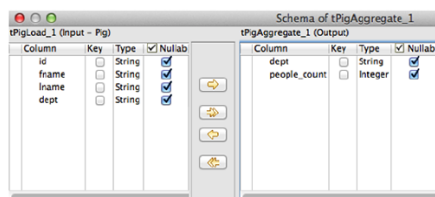
4. Add and connect Pig aggregate component.
- Add the component **tPigAggregate** next to **tPigLoad**.
 - From the contextual menu, right-click on **tPigLoad** and select **-> Pig Combine**.

- c. Click on **tPigAggregate**.



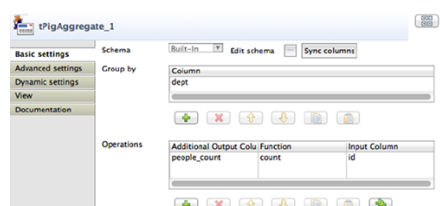
5. Define basic settings for the Pig Aggregate component.

- a. Double-click **tPigAggregate** to define the component in its Basic Settings.
- b. Click on the “Edit schema” button and define the output schema as shown below:



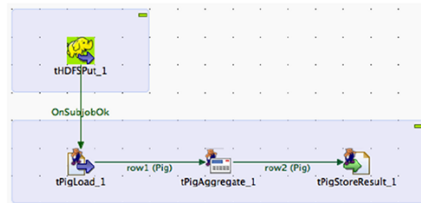
6. Define aggregation function for the data.

- a. For Group by add a Column and select dept.
- b. In the Operations table, choose the people_count in the Additional Output column, function as count and input column id as shown:



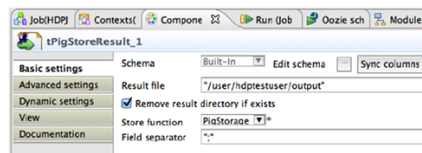
7. Add and connect Pig data storage component

- a. Add the component **tPigStoreResult** next to **tPigAggregate**.
- b. From the contextual menu, right-click on **tPigLoad**, select Row -> Pig Combine and click on **tPigStoreResult**.



8. Define basic settings for the data storage component.

- Double-click **tPigStoreResult** to define the component in its Basic Settings view.
- Specify the result directory on HDFS as shown:



9. Run the modified Talend job. The modified Talend job is ready for execution.

Save the job and click the play icon to run as instructed in Step 4.

10. Verify the results.

- From the gateway machine or the HDFS client, open a console window and execute the following command:

```
hadoop dfs -cat /user/testuser/output/part-r-00000
```

- You should see the following output:

```
Sales;4
Service;3
Marketing;3
```

2. Using Apache Hive

Hortonworks Data Platform deploys Apache Hive for your Hadoop cluster.

Hive is a data warehouse infrastructure built on top of Hadoop. It provides tools to enable easy data ETL, a mechanism to put structures on the data, and the capability for querying and analysis of large data sets stored in Hadoop files.

Hive defines a simple SQL query language, called QL, that enables users familiar with SQL to query the data. At the same time, this language also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language.

In this document:

- [Hive Documentation](#)
- [New Feature: Cost-based SQL Optimizer](#)
- [New Feature: Streaming Data Ingestion](#)
- [Vectorization](#)
- [Comparing Beeline to the Hive CLI](#)
- [Hive JDBC Driver](#)
- [Storage-based Authorization in Hive](#)
- [Hive Troubleshooting](#)
- [Hive JIRAs](#)

2.1. Hive Documentation

Documentation for Hive can be found in wiki docs and javadocs.

1. [Javadocs](#) describe the Hive API.
2. The [Hive wiki](#) is organized in four major sections:
 - General Information about Hive
 - [Getting Started](#)
 - [Presentations and Papers about Hive](#)
 - [Hive Mailing Lists](#)
 - User Documentation
 - [Hive Tutorial](#)
 - [SQL Language Manual](#)

- [Hive Operators and Functions](#)
- [Hive Web Interface](#)
- [Hive Client](#)
- [Beeline: HiveServer2 Client](#)
- [Avro SerDe](#)
- Administrator Documentation
 - [Installing Hive](#)
 - [Configuring Hive](#)
 - [Setting Up the Metastore](#)
 - [Setting Up Hive Server](#)
 - [Hive on Amazon Web Services](#)
 - [Hive on Amazon Elastic MapReduce](#)
- Resources for Contributors
 - [Hive Developer FAQ](#)
 - [How to Contribute](#)
 - [Hive Developer Guide](#)
 - [Plugin Developer Kit](#)
 - [Unit Test Parallel Execution](#)
 - [Hive Architecture Overview](#)
 - [Hive Design Docs](#)
 - [Full-Text Search over All Hive Resources](#)
 - [Project Bylaws](#)

2.2. New Feature: Cost-based SQL Optimization



Note

This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at <http://support.hortonworks.com>.

Hive 0.13.0 introduces Cost-based optimization of SQL queries. Cost-based optimization may reduce query latency by reducing the cardinality of the intermediate result set. The Cost-based Optimizer improves usability because it removes the need for query hints for good query execution plans. The Cost-based Optimizer is useful in the following scenarios:

- Ad hoc queries containing multiple views
- View chaining
- Business Intelligence tools that act as a front end to Hive

Enabling Cost-based Optimization

Set the following configuration parameters in `hive-site.xml` to enable cost-based optimization of SQL:

Table 2.1. CBO Configuration Parameters

Configuration Parameter	Description	Default Value
<code>hive.optimize.enabled</code>	Indicates whether to use cost-based query optimization.	<code>false</code>
<code>hive.optimize.max.joins</code>	Specifies the maximum number of joins that may be included in a query that is submitted for cost-based optimization.	<code>10</code>

2.3. New Feature: Streaming Data Ingestion



Note

This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at <http://support.hortonworks.com>.

Limitations

Hive 0.13 has the following limitations to ingesting streaming data:

- Only ORC files are supported
- Destination tables must be either bucketed or unpartitioned
- Only Apache Flume may be used as streaming source

2.4. Vectorization

Vectorization allows Hive to process a batch of rows together instead of processing one row at a time. Each batch consists of a *column vector* which is usually an array of primitive types. Operations are performed on the entire column vector, which improves the instruction pipelines and cache usage. [HIVE-4160](#) has the design document for vectorization and tracks the implementation of many subtasks.

2.4.1. Enable Vectorization in Hive

To enable vectorization, set this configuration parameter:

- `hive.vectorized.execution.enabled=true`

When vectorization is enabled, Hive examines the query and the data to determine whether vectorization can be supported. If it cannot be supported, Hive will execute the query with vectorization turned off.

2.4.2. Log Information about Vectorized Execution of Queries

The Hive client will log, at the `info` level, whether a query's execution is being vectorized. More detailed logs are printed at the `debug` level.

The client logs can also be configured to show up on the console.

2.4.3. Supported Functionality

The current implementation supports only single table read-only queries. DDL queries or DML queries are not supported.

The supported operators are selection, filter and group by.

Partitioned tables are supported.

These data types are supported:

- `tinyint`
- `smallint`
- `int`
- `bigint`
- `boolean`
- `float`
- `double`

- timestamp
- string
- char

These expressions are supported:

- Comparison: >, >=, <, <=, =, !=
- Arithmetic: plus, minus, multiply, divide, modulo
- Logical: AND, OR
- Aggregates: sum, avg, count, min, max

Only the ORC file format is supported in the current implementation.

2.4.4. Unsupported Functionality

All datatypes, file formats, and functionality *not listed in the previous section* are currently unsupported.

Two unsupported features of particular interest are the logical expression NOT and the cast operator. For example, a query such as `select x,y from T where a = b` will not vectorize if `a` is integer and `b` is double. Although both int and double are supported, casting of one to another is not supported.

2.5. Comparing Beeline to the Hive CLI

HDP supports two Hive clients: the Hive CLI and Beeline. The primary difference between the two involves how the clients connect to Hive. The Hive CLI connects directly to the Hive Driver and requires that Hive be installed on the same machine as the client. However, Beeline connects to HiveServer2 and does not require the installation of Hive libraries on the same machine as the client. Beeline is a thin client that also uses the Hive JDBC driver but instead executes queries through HiveServer2, which allows multiple concurrent client connections and supports authentication.

2.5.1. Beeline Operating Modes and HiveServer2 Transport Modes

Beeline supports the following modes of operation:

Table 2.2. Beeline Modes of Operation

Operation Mode	Description
Embedded	The Beeline client and the Hive installation both reside

Open Mode	Description
Open Mode	on the same host machine. No TCP connectivity is required.
Remote Mode	Remote mode to support multiple, concurrent clients executing queries against the same remote Hive installation. Remote transport mode supports authentication with LDAP and Kerberos. It also supports encryption with SSL. TCP connectivity is required.

Administrators may start HiveServer2 in one of the following transport modes:

Table 2.3. HiveServer2 Transport Modes

Transport Mode	Description
TCP	HiveServer2 uses TCP transport for sending and receiving Thrift RPC messages.
HTTP	HiveServer2 uses HTTP transport for sending and receiving Thrift RPC messages.

While running in TCP transport mode, HiveServer2 supports the following authentication schemes:

Table 2.4. Authentication Schemes with TCP Transport Mode

Authentication Scheme	Description
Kerberos	A network authentication protocol which operates that uses the concept of 'tickets' to allow nodes in a network to securely identify themselves. Administrators must specify <code>hive.server2.authentication=kerberos</code> in the <code>hive-site.xml</code> configuration file to use this authentication scheme.
LDAP	The Lightweight Directory Access Protocol, an application-layer protocol that uses the concept of 'directory services' to share information across a network. Administrators must specify

Authentication Scheme	Description
	<code>hive.server2.authentication=ldap</code> in the <code>hive-site.xml</code> configuration file to use this type of authentication.
PAM	Pluggable Authentication Modules, or PAM, allow administrators to integrate multiple authentication schemes into a single API. Administrators must specify <code>hive.server2.authentication=pam</code> in the <code>hive-site.xml</code> configuration file to use this authentication scheme.
Custom	Authentication provided by a custom implementation of the <code>org.apache.hive.service.auth.PasswdAuthenticationProvider</code> interface. The implementing class must be available in the classpath for HiveServer2 and its name provided as the value of the <code>hive.server2.custom.authentication.class</code> property in the <code>hive-site.xml</code> configuration property file.
None	The Beeline client performs no authentication with HiveServer2. Administrators must specify <code>hive.server2.authentication=none</code> in the <code>hive-site.xml</code> configuration file to use this authentication scheme.
NoSASL	While running in TCP transport mode, HiveServer2 uses the Java Simple Authentication and Security Layer (SASL) protocol to establish a security layer between the client and server. However, HiveServer2 also supports connections in TCP transfer mode that do not use the SASL protocol. Administrators must specify <code>hive.server2.authentication=nosasl</code> in the <code>hive-site.xml</code> configuration file to use this authentication scheme.

The next section describes the connection strings used to connect to HiveServer2 for all possible combinations of these modes, as well as the connection string required to connect to HiveServer2 in a secure cluster.

2.5.2. Connecting to Hive with Beeline

The following examples demonstrate how to use Beeline to connect to Hive for all possible variations of these modes:



Note

Documentation for the `connect` and other Beeline commands may be found at sqlline.sourceforge.net. Beeline is based on the SQLLine open source project.

Embedded Client

Use the following syntax to connect to Hive from Beeline in embedded mode:

```
!connect jdbc:hive2://
```

Remote Client with HiveServer2 TCP Transport Mode and SASL Authentication

Use the following syntax to connect to HiveServer2 in TCP mode from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>
```

The default port for HiveServer2 in TCP mode is 10000. `db` is the name of the database to which you want to connect.

Remote Client with HiveServer2 TCP Transport Mode and NoSASL Authentication

Clients must explicitly specify the authentication mode in their connection string when HiveServer2 runs in NoSASL mode:

```
!connect jdbc:hive2://<host>:<port>/<db>;auth=noSasl hiveuser pass org.apache.
hive.jdbc.HiveDriver
```



Important

If users forget to include `auth=noSasl` in the JDBC connection string, the JDBC client API attempts to make an SASL connection to HiveServer2. This causes an open connection that eventually results in the client crashing with an Out Of Memory error.

Remote Client with HiveServer2 HTTP Transport Mode

Use the following syntax to connect to HiveServer2 in HTTP mode from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>?hive.server2.transport.mode=
http;hive.server2.thrift.http.path=<http_endpoint>
```

Remote Client with HiveServer2 in Secure Cluster

Use the following syntax to connect to HiveServer2 in a secure cluster from a remote Beeline client:

```
!connect jdbc:hive2://<host>:<port>/<db>;principal=
<Server_Principal_of_HiveServer2>
```



Note

The Beeline client must have a valid Kerberos ticket in the ticket cache before attempting to connect.

2.6. Hive ODBC and JDBC Drivers

Hortonworks provides Hive JDBC and ODBC drivers that allow you to connect popular Business Intelligence (BI) tools to query, analyze and visualize data stored within the Hortonworks Data Platform. JDBC URLs have the following format:

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionConfs>?
<hiveConfs>#<hiveVars>
```

Table 2.5. JDBC Connection Parameters

JDBC Connection Parameter	Description
host	The cluster node hosting HiveServer2.
port	The port number to which HiveServer2 listens.
dbName	The name of the Hive

JDBC	Description Connection Parameter
	database to run the query against.
session	Optional configuration parameters for the JDBC/ODBC driver in the following format: <code><key1>=<value1>;<key2>=<key2>; ...</code> The configurations last for the duration of the user session.
hive	Optional configuration parameters for Hive on the server in the following format: <code><key1>=<value1>;<key2>=<key2>; ...</code> The configurations last for the duration of the user session.
hive	Optional configuration parameters for Hive variables in the following format: <code><key1>=<value1>;<key2>=<key2>; ...</code> The configurations last for the duration of the user session.

The specific JDBC connection URL for a HiveServe2 client depends on several factors:

- How is HiveServer2 deployed on the cluster?
- What type of transport does HiveServer2 use?
- Does HiveServer2 use transport layer security?
- Is HiveServer2 configured to authenticate users?

The rest of this topic describes how to use session configuration variables to format the JDBC connection URLs for all of these scenarios. In addition, the topic provides links to download the Hive ODBC driver and instructions for using it.

Some HiveServer2 clients may need to run on a host outside the Hadoop cluster. These clients require all of the following .jar files to successfully connect to the Hive JDBC driver in both HTTP and HTTPS modes:

Off-cluster Jars Without Kerberos

- hive-jdbc.jar
- hive-service.jar
- hive-common.jar
- hadoop-common.jar
- libthrift-0.9.0.jar
- httpclient-4.2.5.jar
- httpcore-4.2.5.jar
- commons-logging-1.1.3.jar
- commons-codec-1.4.jar
- slf4j-api-1.7.5.jar

Off-cluster Jars With Kerberos

- hive-jdbc.jar
- hive-service.jar
- hive-common.jar
- hive-shims-common.jar
- hive-shims-common-secure.jar
- hive-shims-0.23-*.jar
- hadoop-common.jar
- hadoop-auth.jar
- hadoop-mapreduce-client-core.jar
- libthrift-0.9.0.jar
- guava-11.0.2.jar
- httpclient-4.2.5.jar
- httpcore-4.2.5.jar
- commons-logging-1.1.3.jar
- commons-codec-1.4.jar
- commons-collections-3.1.jar

- commons-configuration-1.6.jar
- commons-lang-2.4.jar
- log4j-1.2.16.jar
- slf4j-api-1.7.5.jar

Embedded and Remote Modes

In embedded mode, HiveServer2 runs within the Hive client rather than in a separate process. No host or port number is necessary for the JDBC connection. In remote mode, HiveServer2 runs as a separate daemon on a specified host and port, and the JDBC client and HiveServer2 interact using the Thrift protocol.

Embedded Mode

```
jdbc://hive2://
```

Remote Mode

```
jdbc://hive2://<host>:<port>/<dbName>;<sessionConfs>?
<hiveConfs>#<hiveVars>
```



Note

The rest of the example JDBC connection URLs in this topic are valid only if HiveServer2 is configured in remote mode.

TCP and HTTP Transport

The JDBC client and HiveServer2 can use either HTTP or TCP-based transport to exchange RPC messages. Specify the transport used by HiveServer2 with the `transportMode` and `httpPath` session configuration variables. The default transport is TCP.

Table 2.6.

transportMode Variable Value	Description
http	Connect to HiveServer2 using HTTP transport.
binary	Connect to HiveServer2 using TCP transport.

HTTP Transport

Use the following JDBC connection URL for HTTP transport:

```
jdbc:hive2://<host>:<port>/
<dbName>;transportMode=http;httpPath=<http_endpoint>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```



Note

The JDBC driver assume a value of `cliservice` if the `httpPath` configuration variable is not specified.

TCP Transport

Use the following JDBC connection URL for TCP transport:

```
jdbc:hive2://<host>:<port>/<dbName>;<otherSessionConfs>?
<hiveConfs>#hiveVars
```

There is no need to specify `transportMode=binary` because the default transport is TCP.

User Authentication

HiveServer2 supports Kerberos, LDAP, Pluggable Authentication Modules (PAM), and custom plugins for authenticating JDBC users connecting to HiveServer2. The format of the JDBC connection URL for authentication with Kerberos differs from the format for other authentication models.

Table 2.7.

User Authentication Variable	Description
principal	String that uniquely identifies a Kerberos user.
sasl.qop	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
username	Username for non-Kerberos authentication model.
password	Password for non-Kerberos authentication model.

Kerberos Authentication

Use the following JDBC connection URL to authenticate the connecting user with Kerberos:


```
jdbc:hive2://<host>:<port>/
<dbName>;principal=<HiveServer2_kerberos_principal>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

Kerberos Authentication with Sasl QOP

Use the following JDBC connection URL to authenticate the connecting user with Kerberos and Sasl QOP.

```
jdbc:hive2://<host>:<port>/
<dbName>;principal=<HiveServer2_kerberos_principal>;saslQop=<qop_value>;<other
<hiveConfs>#<hiveVars>
```

Non-Kerberos Authentication

Use the following JDBC connection to authenticate the connecting user without Kerberos:

```
jdbc:hive2://<host>:<port>/
<dbName>;user=<username>;password=<password>;<otherSessionConfs>?
<hiveConfs>#<hiveVars>
```

Transport Layer Security

HiveServer2 supports SSL and Sasl QOP for transport layer security. The format of the JDBC connection URL for SSL differs from the format used by Sasl QOP.

```
jdbc:hive2://<host>:<port>/
<dbName>;principal=<HiveServer2_kerberos_principal>;saslQop=auth-
conf;<otherSessionConfs>?<hiveConfs>#<hiveVars>
```

Table 2.8.

SSL Variable	Description
ssl	Specifies whether to use SSL.
sslTrustStore	TrustStore path to the SSL TrustStore.
trustStorePassword	TrustStore password to the SSL TrustStore.

Hive ODBC Driver

Hortonworks also provides an ODBC driver as an add-on to the HDP distribution:

- Download the Hortonworks Hive ODBC driver from [here](#).

- The instructions on installing and using this driver are available [here](#).

Some HiveServer2 clients, such as Apache Knox, may need to run on a host outside the Hadoop cluster. Such clients require all of the following .jar files to successfully use the Hive JDBC driver in both HTTP and HTTPS modes:

- commons-codec-1.4.jar
- commons-logging-1.1.3.jar
- hive-jdbc-0.13.0.jar
- hive-service-0.13.0.jar
- httpclient-4.2.5.jar
- httpcore-4.2.5.jar
- libthrift-0.9.0.jar

2.7. Storage-based Authorization in Hive



Important

Notes about storage-based authorization:

- This authorization mode does not recognize the GRANT and REVOKE SQL statements.
- This authorization mode does not currently consider the ACL permissions in HDFS. Rather, it verifies access using the traditional POSIX permissions model.

2.8. Troubleshooting Hive

MySQL is the default database used by the Hive metastore. Depending on several factors, such as the version and configuration of MySQL, Hive developers may encounter an error message similar to the following:

```
An exception was thrown while adding/validating classes) : Specified key was too long; max key length is 767 bytes
```

Administrators can resolve this issue by altering the Hive metastore database to use the Latin1 character set, as shown in the following example:

```
mysql> ALTER DATABASE <metastore_database_name> character set latin1;
```

2.9. Hive JIRAs

Issue tracking for Hive bugs and improvements can be found here: [Hive JIRAs](#).

3. SQL Compliance

This chapter discusses the ongoing implementation of standard SQL syntax in Hive. Although SQL in Hive does not yet entirely support the SQL-2011 standard, version 0.13 provides significant improvements to the parity between SQL as used in Hive and SQL as used in traditional relational databases.

- [Authorization with GRANT and REVOKE](#)
- [Transactions](#)
- [Subqueries in WHERE clauses](#)
- [Common Table Expressions](#)
- [Quoted Identifiers](#)
- [CHAR Data Type Support](#)

3.1. New Feature: Authorization with Grant And Revoke

Hive 0.13 provides secure authorization using the `GRANT` and `REVOKE` SQL statements. Use the following procedure to manually enable standard SQL authorization:



Note

This procedure is unnecessary if your Hive administrator installed Hive using Ambari.

1. Set the following configuration parameters in `hive-site.xml`:

Table 3.1. Configuration Parameters for Standard SQL Authorization

Configuration Parameter	Required Value
<code>hive.server2.enable.doAs</code>	<code>false</code>
<code>hive.conf.defaults.in.admin.role</code>	Comma-separated list of users granted the administrator role.

2. Start HiveServer2 with the following command-line options:

Table 3.2. HiveServer2 Command-Line Options

Command-Line Option	Required Value
<code>-hiveconf hive.security.authorization.manager</code>	<code>org.apache.hadoop.hive.q1.security.authorization.MetaStoreAuthzAPIAuthorizerEmbedOnly</code>

Config Line	Required Value
<code>hiveconf hive.security.authorization.enabled</code>	<code>true</code>
<code>hiveconf hive.security.authenticator.manager</code>	<code>org.apache.hadoop.hive ql.security.SessionStateUserAuthenticator</code>
<code>hiveconf hive.security.authorization.manager</code>	<code>' ' (a space inside single quotation marks)</code>



Note

Hive continues to provide storage-based authorization. See [Hive Authorization Without GRANT/REVOKE](#) for more information.



Note

Administrators must also specify a storage-based authorization manager for Hadoop clusters that also use storage-based authorization. The `hive.security.authorization.manager` configuration property allows multiple authorization managers in comma-delimited format, so the correct value in this case is `hive.security.authorization.manager=org.apache.hadoop.hive.ql.security`

3.2. New Feature: Transactions

Support for transactions in Hive 0.13 enables SQL atomicity of operations at the row level rather than at the level of a table or partition. This means that a Hive client may read from the same partition to which another Hive client is adding rows. In addition, transactions provide a mechanism for streaming clients to rapidly update Hive tables and partitions. However, Hive transactions are different than RDBMS transactions: each transaction has an identifier, and multiple transactions are grouped into a single transaction batch. A streaming client requests a set of transaction IDs after connecting to Hive and subsequently uses these transaction IDs one at a time during the initialization of new transaction batches. Clients write one or more records for each transaction and either commit or abort a transaction before moving to the next transaction.



Note

This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at <http://support.hortonworks.com>.



Note

Hive transactions do not work for this release when using Oracle as the Metastore database.

Understanding Compactions

Hive stores data in base files that cannot be updated by HDFS. Instead, Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. Occasionally, Hive *compacts*, or merges, the base and delta files. Hive performs all compactions in the background without affecting concurrent reads and writes of Hive clients. There are two types of compactions:

Table 3.3. Hive Compaction Types

Compaction Type	Description
Minor	Rewrites a set of delta files to a single delta file for a bucket.
Major	Rewrites one or more delta files and the base file as a new base file for a bucket.

By default, Hive automatically compacts delta and base files at regular intervals. However, Hadoop administrators can configure automatic compactions, as well as perform manual compactions of base and delta files using the following configuration parameters in `hive-site.xml`.

Table 3.4. Hive Transaction Configuration Parameters

Configuration Parameter	Description
<code>hive.txnmanager</code>	Specifies the class name of the transaction manager used by Hive. Configure this property with <code>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</code> to enable transactions. The default value is <code>org.apache.hadoop.hive.ql.lockmgr.DummyTxnManager</code> , which disables transactions.
<code>hive.jdbc.driver</code>	Specifies the class name of the JDBC driver for the Hive metastore database. Include the username and password of a Hive administrator. The default MySQL database

Configuration Parameter	Description
	uses the following JDBC connector. Change the username and password to match your environment: jdbc:mysql://hdp.example.com/hivedb?user=hivedba&password=hivedbp
hive.heap.size	Specifies the time period, in seconds, after which Hive fails a transaction if a Hive client has not sent a heartbeat. The default value is 300 seconds.
hive.open.batch	Specifies the maximum number of transactions that can be retrieved in one call to open_txn(). The default value is 1000 transactions.
hive.initiator.on	Specifies whether to run the initiator and cleaner threads on this metastore instance. The default value is false. Must be set to true for exactly one instance of the Hive metastore service.
hive.worker.threads	Specifies the number of worker threads to run on this metastore instance. The default value is 0, which must be set to greater than 0 to enable compactions. Worker threads initialize MapReduce jobs to do

Configuration Parameter	Description
	compactions. Increasing the number of worker threads decreases the time required to compact tables after they cross a threshold that triggers compactions. However, increasing the number of worker threads also increases the background load on a Hadoop cluster.
hive.hor.worker.timeout	Specifies the time period, in seconds, after which a compaction job is failed and requeued. The default value is 86400 seconds, or 24 hours.
hive.hor.check.interval	Specifies the time period, in seconds, between checks to see if any partitions require compacting. The default value is 300 seconds. Decreasing this value reduces the time required to start a compaction for a table or partition. However, it also increases the background load on the NameNode since each check requires several calls to the NameNode.

Configuration Parameter	Description
hive.compactor.delta.num.threshold	Specifies the number of delta directories in a partition that triggers an automatic minor compaction. The default value is 10.
hive.compactor.delta.pct.threshold	Specifies the percentage size of delta files relative to the corresponding base files that triggers an automatic major compaction. The default value is .1, which is 10 percent.
hive.compactor.abortedtxn.threshold	Specifies the number of aborted transactions on a single partition that trigger an automatic major compaction.

Enabling the Hive Transaction Manager

Configure the following Hive configuration properties from the table above to enable transactions:

- `hive.txn.manager`
- `hive.txn.driver`
- `hive.compactor.initiator.on`
- `hive.compactor.worker.threads`



Note

To disable automatic compactions for individual tables, set the `NO_AUTO_COMPACTION` table property for those tables. This overrides the configuration settings in `hive-site.xml`. However, this property does not prevent manual compactions.

Check the hive log file at `/tmp/hive/hive.log` for errors if you experience problems enabling hive transactions.

Performing Manual Compactions

Hive administrators use the `ALTER TABLE DDL` command to queue a request to compact base and delta files for a table or partition:

```
ALTER TABLE tablename [PARTITION (partition_key='partition_value' [...])]
  COMPACT 'compaction_type'
```



Note

`ALTER TABLE` will compact tables even if the `NO_AUTO_COMPACTION` table property is set.

Use the `SHOW COMPACTIONS` DDL command to monitor the progress of the compactions:

```
SHOW COMPACTIONS
```

This command provides the following output for each compaction:

- Database name
- Table name
- Partition name
- Major or minor compaction
- Compaction state:
 - Initiated - waiting in queue
 - Working - currently compacting
 - Ready for cleaning - compaction completed and old files scheduled for removal
- Thread ID
- Start time of compaction

Hive administrators can also view a list of currently open and aborted transactions with the `SHOW TRANSACTIONS` DDL command:

```
SHOW TRANSACTIONS
```

This command provides the following output for each transaction:

- Transaction ID
- Transaction state
- Hive user who initiated the transaction
- Host machine where transaction was initiated

New Lock Manager

Hive 0.13 utilizes a new lock manager, `DbLockManager`, to store all transaction and related lock information in the Hive metastore. Heartbeats are sent regularly from lock holders and transaction initiators to the Hive metastore to prevent stale locks and transactions. The lock or transaction is aborted if the Hive metastore does not receive a heartbeat within the amount of time specified by the `hive.txn.timeout` configuration property. Hive administrators use the `SHOW LOCKS` DDL command to view information about locks associated with transactions.

This command provides the following output for each lock:

- Database name
- Table name
- Partition, if the table is partitioned
- Lock state:
 - Acquired - transaction initiator hold the lock
 - Waiting - transaction initiator is waiting for the lock
 - Aborted - the lock has timed out but has not yet been cleaned
- Lock type:
 - Exclusive - the lock may not be shared
 - Shared_read - the lock may be shared with any number of other shared_read locks
 - Shared_write - the lock may be shared by any number of other shared_read locks but not with other shared_write locks
- Transaction ID associated with the lock, if one exists
- Last time lock holder sent a heartbeat
- Time the lock was acquired, if it has been acquired
- Hive user who requested the lock
- Host machine on which the Hive user is running a Hive client



Note

The output of the command reverts to behavior prior to Hive 0.13 if administrators use Zookeeper or in-memory lock managers.

Transaction Limitations

HDP 2.1 has the following limitations for transactions with Hive:

- The user initiating the Hive session must have write permission for the destination partition or table.
- Zookeeper and in-memory locks are not compatible with transactions.

- Only ORC files are supported.
- Destination tables must be bucketed and not sorted.
- Snapshot-level isolation, similar to READ COMMITTED. A query is provided with a consistent snapshot of the data during execution.

3.3. New Feature: Subqueries in WHERE Clauses

Previous versions of Hive allowed *subqueries* only in *FROM* clauses of SQL statements. A subquery is a SQL expression that returns a set of rows. The subquery is evaluated and its query result set used to evaluate the *parent query*, the outer query that contains the subquery. Version 0.13 of Hive expands the use of subqueries to include *WHERE* clauses, as shown in the following example.

```
SELECT state, net_payments FROM transfer_payments WHERE transfer_payments.year
IN (SELECT year FROM us_census);
```

No configuration is required to enable execution of subqueries in Hive; the feature is available by default. However, several restrictions exist for the use of subqueries in *WHERE* clauses. The next section, [Understanding Subqueries](#), describes the concepts necessary to understand these restrictions, and the following section, [Restrictions on Subqueries in WHERE Clauses](#) explains the restrictions.

3.3.1. Understanding Subqueries in SQL

SQL adheres to syntax rules like any programming language. The syntax governing the use of subqueries in *WHERE* clauses in SQL is simple and depends on the following concepts:

- Query Predicates and Predicate Operators
- Aggregated and Correlated Queries
- Conjuncts and Disjuncts

Query Predicates and Predicate Operators

A *predicate* in SQL is a condition that evaluates to a Boolean value. For example, the predicate in the preceding example returns true for a row of the *transfer_payments* table if at least one row exists in the *us_census* table with the same year as the *transfer_payments* row. The predicate starts with the first *WHERE* keyword.

```
... WHERE transfer_payments.year IN (SELECT year FROM us_census);
```

A SQL predicate in a subquery must also contain a predicate *operator*. Predicate operators specify the relationship tested in a predicate query. For example, the predicate operator in the example is the *EXISTS* keyword.

Aggregated and Correlated Queries

Aggregated queries combine one or more aggregate functions, such as *AVG*, *SUM*, and *MAX*, with the *GROUP BY* statement to group query results by one or more table columns. In the following example, the *AVG* aggregate function returns the average salary of all employees in the engineering department grouped by year.

```
SELECT year, AVG(salary) FROM Employees WHERE department = 'engineering' GROUP BY year;
```



Note

The `GROUP BY` statement may be either explicit or implicit.

Correlated queries contain a query predicate with the Equals To (=) operator. One side of the operator must reference at least one column from the parent query and the other side must reference at least one column from the subquery. The following query is a revised and correlated version of the query at the beginning of this section. It is correlated query because one side of Equals To predicate operator in the subquery references the `state` column in the `transfer_payments` table in the parent query and the other side of the operator references the `state` column in the `us_census` table.

```
SELECT state, net_payments FROM transfer_payments WHERE EXISTS (SELECT year FROM us_census WHERE transfer_payments.state = us_census.state);
```

In contrast, an *uncorrelated* query does not reference any columns in the parent query.

Conjuncts and Disjuncts

A *conjunct* is equivalent to the AND condition, while a *disjunct* is the equivalent of the OR condition. The following subquery contains a conjunct:

```
... WHERE transfer_payments.year = "2010" AND us_census.state = "california"
```

The following subquery contains a disjunct:

```
... WHERE transfer_payments.year = "2010" OR us_census.state = "california"
```

3.3.2. Restrictions on Subqueries in WHERE Clauses

The following restrictions to the use of subqueries in `WHERE` SQL clauses require an understanding of the concepts discussed in the preceding section, [Understanding Subqueries](#).

- Subqueries must appear on the right hand side of an expression.
- Nested subqueries are not supported.
- Only one subquery expression is allowed for a single query.
- Subquery predicates must appear as top level conjuncts.
- Subqueries support four logical operators in query predicates: `IN`, `NOT IN`, `EXISTS`, and `NOT EXISTS`.
- The left hand side of a subquery must qualify all references to table columns.
- References to columns in the parent query are allowed only in the `WHERE` clause of the subquery.
- Subquery predicates that reference a column in a parent query must use the Equals To (=) predicate operator.

- Subquery predicates may not refer only to columns in the parent query.
- Correlated subqueries with an implied `GROUP BY` statement may return only one row.
- All unqualified references to columns in a subquery must resolve to tables in the subquery.
- Correlated subqueries cannot contain windowing clauses.



Note

The `IN` and `NOT IN` logical operators may select only one column in a `WHERE` clause subquery.



Note

The `EXISTS` and `NOT EXISTS` operators must have at least one correlated predicate.

3.4. New Feature: Common Table Expressions

A *Common Table Expression*, or CTE, in SQL is a set of query results obtained from a simple query specified within a `WITH` clause and which immediately precedes a `SELECT` or `INSERT` keyword. A CTE exists only within the scope of a single SQL statement. One or more CTEs can be used with the following SQL statements:

- `SELECT`
- `INSERT`
- `CREATE TABLE AS SELECT`
- `CREATE VIEW AS SELECT`

The following example demonstrates the use of `q1` as a CTE in a `SELECT` statement:

```
WITH q1 AS (SELECT key from src where key = '5')
SELECT * from q1;
```

The following example demonstrates the use of `q1` as a CTE in an `INSERT` statement:

```
CREATE TABLE s1 LIKE src;
WITH q1 AS (SELECT key, value FROM src WHERE key = '5')
FROM q1 INSERT OVERWRITE TABLE s1 SELECT *;
```

The following example demonstrates the use of `q1` as a CTE in a `CREATE TABLE AS SELECT` clause:

```
CREATE TABLE s2 AS WITH q1 AS (SELECT key FROM src WHERE key = '4')
SELECT * FROM q1;
```

The following example demonstrates the use of `q1` as a CTE in a `CREATE TABLE AS VIEW` clause:

```
CREATE VIEW v1 AS WITH q1 AS (SELECT key FROM src WHERE key='5')
```

```
SELECT * from q1;
```

CTEs are available by default in Hive 0.13; Hive administrators do not need to do anything to enable them.

Limitations

Hive 0.13 imposes the following restrictions on the use of Common Table Expressions:

- Recursive queries are not supported
- The `WITH` clause is not supported within subquery blocks

3.5. New Feature: Quoted Identifiers in Column Names

Hive 0.13 allows the use of quoted *identifiers* in the names of table columns. An identifier in SQL is a sequence of alphanumeric and underscore characters surrounded by backtick characters. In the following example, ``x+y`` and ``a?b`` are valid column names for a new table.

```
CREATE TABLE test (`x+y` String, `a?b` String);
```

Quoted identifiers can be used anywhere a column name is expected, including table partitions and buckets:

```
CREATE TABLE partition_date-1 (key string, value string) partitioned by (`dt  
+x` date, region int);
```

```
CREATE TABLE bucket_test(`key?1` string, value string) clustered by (`key?1`)  
into 5 buckets;
```



Note

Identifiers are case-insensitive.



Note

Use a backtick character to escape a backtick character (`` ``).

Enabling Quoted Identifiers

Set the `hive.support.quoted.identifiers` configuration parameter to `column` in `hive-site.xml` to enable quoted identifiers in SQL column names. For Hive 0.13, the valid values are `none` and `column`.

```
hive.support.quoted.identifiers = column
```

3.6. New Feature: CHAR Data Type Support

Hive 0.13 supports the `CHAR` data type, which greatly simplifies the process of migrating data from other databases.

**Note**

Hive 0.13 ignores trailing whitespace characters for the `CHAR` data type.

The following table describes how several databases treat trailing whitespaces for the `CHAR`, `VARCHAR`, and `STRING` data types:

Table 3.5. Trailing Whitespace Characters on Various Databases

Data Type	Hive	Oracle	SQL Server	MySQL	Teradata
CHAR	Ignore	Ignore	Ignore	Ignore	Ignore
VARCHAR	Compare	Compare	Configurable	Ignore	Ignore
STRING	Compare	N/A	N/A	N/A	N/A

4. Using HDP for Metadata Services (HCatalog)

Hortonworks Data Platform deploys Apache HCatalog to manage the metadata services for your Hadoop cluster.

Apache HCatalog is a table and storage management service for data created using Apache Hadoop. This includes:

- Providing a shared schema and data type mechanism.
- Providing a table abstraction so that users need not be concerned with where or how their data is stored.
- Providing interoperability across data processing tools such as Pig, MapReduce, and Hive.

Start the HCatalog CLI with the command '`<hadoop-install-dir>\hcatalog-0.5.0\bin\hcat.cmd`'.



Note

HCatalog 0.5.0 was the final version released from the Apache Incubator. In March 2013, HCatalog graduated from the Apache Incubator and became part of the [Apache Hive project](#). New releases of Hive include HCatalog, starting with Hive 0.11.0.

HCatalog includes two documentation sets:

1. General information about HCatalog

This documentation covers installation and user features. The next section, [Using HCatalog](#), provides links to individual documents in the HCatalog documentation set.

2. WebHCat information

WebHCat is a web API for HCatalog and related Hadoop components. The section [Using WebHCat](#) provides links to user and reference documents, and includes a technical update about standard WebHCat parameters.

For more details on the Apache Hive project, including HCatalog and WebHCat, see [Using Apache Hive](#) and the following resources:

- [Hive project home page](#)
- [Hive wiki home page](#)
- [Hive mailing lists](#)

4.1. Using HCatalog

For details about HCatalog, see the following resources in the HCatalog documentation set:

- [HCatalog Overview](#)
- [Installation From Tarball](#)
- [HCatalog Configuration Properties](#)
- [Load and Store Interfaces](#)
- [Input and Output Interfaces](#)
- [Reader and Writer Interfaces](#)
- [Command Line Interface](#)
- [Storage Formats](#)
- [Dynamic Partitioning](#)
- [Notification](#)
- [Storage Based Authorization](#)

4.2. Using WebHCat

WebHCat provides a REST API for HCatalog and related Hadoop components.



Note

WebHCat was originally named *Templeton*, and both terms may still be used interchangeably. For backward compatibility the Templeton name still appears in URLs, log file names, etc.

For details about WebHCat (Templeton), see the following resources:

- [Overview](#)
- [Installation](#)
- [Configuration](#)
- **Reference**
 - [Resource List](#)
 - [GET :version](#)
 - [GET status](#)
 - [GET version](#)
 - [DDL Resources: Summary and Commands](#)
 - [POST mapreduce/streaming](#)
 - [POST mapreduce/jar](#)

- [POST pig](#)
- [POST hive](#)
- [GET queue/:jobid](#)
- [DELETE queue/:jobid](#)

4.2.1. Technical Update: WebHCat Standard Parameters

The "Security" section of the [WebHCat Overview](#) should be updated with information in the Note below:

4.2.1.1. Security

The current version supports two types of security:

- Default security (without additional authentication)
- Authentication via [Kerberos](#)

4.2.1.1.1. Standard Parameters

Every REST resource can accept the following parameters to aid in authentication:

- user.name: The user name as a string. Only valid when using default security.
- SPNEGO credentials: When running with Kerberos authentication.



Note

The user.name parameter is part of POST parameters for POST calls, and part of the URL for other calls.

For example, to specify user.name in a GET :table command:

```
% curl -s 'http://localhost:50111/templeton/v1/ddl/database/default/
table/my_table?user.name=ctdean'
```

And to specify user.name in a POST :table command:

```
% curl -s -d user.name=ctdean \
      -d rename=test_table_2 \
      'http://localhost:50111/templeton/v1/ddl/database/default/
table/test_table'
```

4.2.1.1.2. Security Error Response

If the user.name parameter is not supplied when required, the following error will be returned:

```
{  
  "error": "No user found.  Missing user.name parameter."  
}
```

5. Using Apache HBase

Hortonworks Data Platform deploys Apache HBase for your Hadoop cluster. HBase is a key-value data store designed for petabyte scale. HBase supports low latency reads, writes, and updates.

5.1. New Feature: Cell-level ACLs

HBase 0.98 introduces cell-level access control lists for HBase tables.



Note

This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at <https://support.hortonworks.com>.

5.2. New Feature: Column Family Encryption



Note

This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Hortonworks Support Portal at <https://support.hortonworks.com>.

6. Using HDP for Workflow and Scheduling (Oozie)

Hortonworks Data Platform deploys Apache Oozie for your Hadoop cluster.

Oozie is a server-based workflow engine specialized in running workflow jobs with actions that execute Hadoop jobs, such as MapReduce, Pig, Hive, Sqoop, HDFS operations, and sub-workflows. Oozie supports coordinator jobs, which are sequences of workflow jobs that are created at a given frequency and start when all of the required input data is available. A command-line client and a browser interface allow you to manage and administer Oozie jobs locally or remotely.

Access the Oozie web UI at the following URL:

`http://{your.ambari.server.hostname}:11000/oozie`

after installing a HDP 2.x cluster using Ambari 1.5.x.

For additional [Oozie documentation](#), use the following resources:

- [Quick Start Guide](#)
- Developer Documentation
 - [Oozie Workflow Overview](#)
 - [Running the Examples](#)
 - [Workflow Functional Specification](#)
 - [Coordinator Functional Specification](#)
 - [Bundle Functional Specification](#)
 - [EL Expression Language Quick Reference](#)
 - [Command Line Tool](#)
 - [Workflow Rerun](#)
 - [Email Action](#)
 - [Writing a Custom Action Executor](#)
 - [Oozie Client Javadocs](#)
 - [Oozie Core Javadocs](#)
 - [Oozie Web Services API](#)
- Administrator Documentation
 - [Oozie Installation and Configuration](#)

- [Oozie Monitoring](#)
- [Command Line Tool](#)

7. Using Apache Sqoop

Hortonworks Data Platform deploys Apache Sqoop for your Hadoop cluster. Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

This document includes the following sections:

- [Apache Sqoop Connectors](#)
- [Sqoop Import Table Commands](#)
- [Netezza Connector](#)
- [Sqoop-HCatalog Integration](#)

For additional information see the [Sqoop documentation](#), including these sections in the [User Guide](#):

- [Basic Usage](#)
- [Sqoop Tools](#)
- [Troubleshooting](#)

7.1. Apache Sqoop Connectors

Sqoop uses a connector-based architecture which supports plugins that provide connectivity to new external systems. Using specialized connectors, Sqoop can connect with external systems that have optimized import and export facilities, or do not support native JDBC. Connectors are plugin components based on Sqoop's extension framework and can be added to any existing Sqoop installation.

Hortonworks provides the following connectors for Sqoop:

- **MySQL connector:** This connector is included in the HDP 2 distribution; the instructions for invoking this connector are available [here](#).
- **Netezza connector:** This connector is included in the HDP 2 distribution and installs with Sqoop; see [below](#) for more information.
- **Oracle JDBC connector:** The instructions on using this connector are available [here](#).
- **PostgreSQL connector:** This connector is included in the HDP 2 distribution and installs with Sqoop; the instructions for invoking this connector are [here](#).
- **Hortonworks Connector for Teradata:** This connector, based on the Teradata Connector for Hadoop, can be downloaded from [here](#).

A Sqoop connector for SQL Server is available from Microsoft:

- **SQL Server R2 connector:** This connector and its documentation can be downloaded from [here](#).

7.2. Sqoop Import Table Commands

When connecting to an Oracle database, the Sqoop `import` command requires case-sensitive table names and usernames (typically uppercase). Otherwise the import fails with error message "Attempted to generate class with no columns!"

Prior to the resolution of [SQOOP-741](#), `import-all-tables` would fail for an Oracle database. See the JIRA for more information.

The `import-all-tables` command has additional restrictions. See [Chapter 8](#) in the [Sqoop User Guide](#).

7.3. Netezza Connector

Netezza connector for Sqoop is an implementation of the Sqoop connector interfaces for accessing a Netezza data warehouse appliance, so that data can be exported and imported to a Hadoop environment from Netezza data warehousing environments.

The HDP 2 Sqoop distribution includes Netezza connector software. To deploy it, the only requirement is that you acquire the JDBC jar file (named `nzjdbc.jar`) from IBM and copy it to the `/usr/local/nz/lib` directory.

7.3.1. Extra Arguments

This table describes extra arguments supported by the Netezza connector:

Table 7.1. Supported Netezza Extra Arguments

Argument	Description
<code>--partitioned-access</code>	Whether each mapper acts on a subset of data slices of a table or all. Default is "false" for standard mode and "true" for direct mode.
<code>--max-errors</code>	Applicable only in direct mode. This option specifies the error threshold per mapper while transferring data. If the number of errors encountered exceeds this threshold, the job fails. Default value is 1.
<code>--log-dir</code>	Applicable only in direct mode. Specifies the directory where Netezza external table operation logs are stored. Default value is <code>/tmp</code> .

7.3.2. Direct Mode

Netezza connector supports an optimized data transfer facility using the Netezza external tables feature. Each map task of Netezza connector's import job works on a subset of the Netezza partitions and transparently creates and uses an external table to transport data.

Similarly, export jobs use the external table to push data fast onto the NZ system. Direct mode does not support staging tables, upsert options, etc.

Direct mode is specified by the `--direct` Sqoop option.

Here is an example of a complete command line for import using the Netezza external table feature.

```
$ sqoop import \  
  --direct \  
  --connect jdbc:netezza://nzhost:5480/sqoop \  
  --table nztable \  
  --username nzuser \  
  --password nzpass \  
  --target-dir hdfsdir
```

Here is an example of a complete command line for export with tab (`\t`) as the field terminator character.

```
$ sqoop export \  
  --direct \  
  --connect jdbc:netezza://nzhost:5480/sqoop \  
  --table nztable \  
  --username nzuser \  
  --password nzpass \  
  --export-dir hdfsdir \  
  --input-fields-terminated-by "\t"
```

7.3.3. Null String Handling

In direct mode the Netezza connector supports the null-string features of Sqoop. Null string values are converted to appropriate external table options during export and import operations.

Table 7.2. Supported Export Control Arguments

Argument	Description
<code>--input-null-string <null-string></code>	The string to be interpreted as null for string columns.
<code>--input-null-non-string <null-string></code>	The string to be interpreted as null for non-string columns.

In direct mode, both the arguments must either be left to the default values or explicitly set to the same value. Furthermore, the null string value is restricted to 0-4 UTF-8 characters.

On export, for non-string columns, if the chosen null value is a valid representation in the column domain, then the column might not be loaded as null. For example, if the null string value is specified as "1", then on export, any occurrence of "1" in the input file will be loaded as value 1 instead of NULL for int columns.

For performance and consistency, specify the null value as an empty string.

Table 7.3. Supported Import Control Arguments

Argument	Description
<code>--null-string <null-string></code>	The string to be interpreted as null for string columns.

Argument	Description
<code>--null-non-string <null-string></code>	The string to be interpreted as null for non-string columns.

In direct mode, both the arguments must either be left to the default values or explicitly set to the same value. Furthermore, the null string value is restricted to 0-4 UTF-8 characters.

On import, for non-string columns in the current implementation, the chosen null value representation is ignored for non-character columns. For example, if the null string value is specified as "\N", then on import, any occurrence of NULL for non-char columns in the table will be imported as an empty string instead of \N, the chosen null string representation.

For performance and consistency, specify the null value as an empty string.

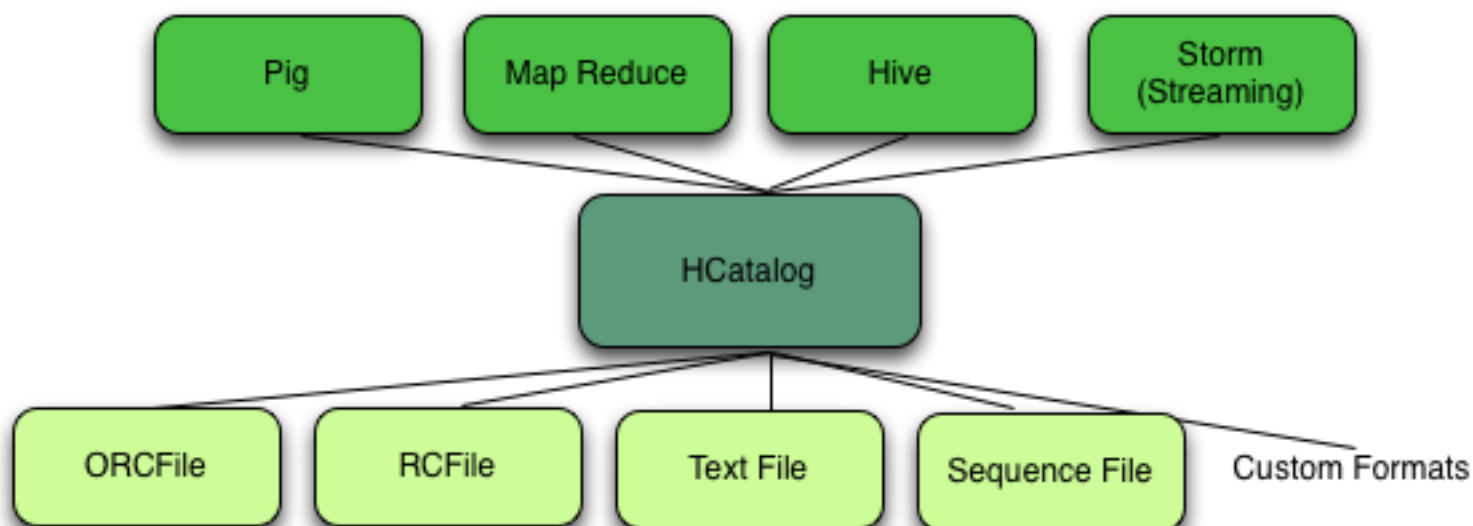
7.4. Sqoop-HCatalog Integration

This section describes the interaction between HCatalog with Sqoop.

7.4.1. HCatalog Background

HCatalog is a table and storage management service for Hadoop that enables users with different data processing tools – Pig, MapReduce, and Hive – to more easily read and write data on the grid. HCatalog's table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored: RCFile format, text files, or SequenceFiles.

HCatalog supports reading and writing files in any format for which a Hive SerDe (serializer-deserializer) has been written. By default, HCatalog supports RCFile, CSV, JSON, and SequenceFile formats. To use a custom format, you must provide the InputFormat and OutputFormat as well as the SerDe.



The ability of HCatalog to abstract various storage formats is used in providing RCFile (and future file types) support to Sqoop.

7.4.2. Exposing HCatalog Tables to Sqoop

HCatalog interaction with Sqoop is patterned on an existing feature set that supports Avro and Hive tables. This section introduces five command line options, and some command line options defined for Hive are reused.

7.4.2.1. Relevant Command Line Options

<code>--hcatalog-database</code>	Specifies the database name for the HCatalog table. If not specified, the default database name 'default' is used. Providing the <code>--hcatalog-database</code> option without <code>--hcatalog-table</code> is an error. This is not a required option.
<code>--hcatalog-table</code>	<p>The argument value for this option is the HCatalog tablename.</p> <p>The presence of the <code>--hcatalog-table</code> option signifies that the import or export job is done using HCatalog tables, and it is a required option for HCatalog jobs.</p>
<code>--hcatalog-home</code>	The home directory for the HCatalog installation. The directory is expected to have a lib subdirectory and a share/hcatalog subdirectory with necessary HCatalog libraries. If not specified, the system environment variable HCAT_HOME will be checked and failing that, a system property hcatalog.home will be checked. If none of these are set, the default value will be used and currently the default is set to /usr/lib/hcatalog. This is not a required option.
<code>--create-hcatalog-table</code>	This option specifies whether an HCatalog table should be created automatically when importing data. By default, HCatalog tables are assumed to exist. The table name will be the same as the database table name translated to lower case. Further described in Automatic Table Creation below.
<code>--hcatalog-storage-stanza</code>	This option specifies the storage stanza to be appended to the table. Further described in Automatic Table Creation below.

7.4.2.2. Supported Sqoop Hive Options

The following Sqoop options are also used along with the `--hcatalog-table` option to provide additional input to the HCatalog jobs. Some of the existing Hive import job options are reused with HCatalog jobs instead of creating HCatalog-specific options for the same purpose.

<code>--map-column-hive</code>	This option maps a database column to HCatalog with a specific HCatalog type.
--------------------------------	---

<code>--hive-home</code>	The Hive home location.
<code>--hive-partition-key</code>	Used for static partitioning filter. The partitioning key should be of type <code>STRING</code> . There can be only one static partitioning key.
<code>--hive-partition-value</code>	The value associated with the partition.

7.4.2.3. Direct Mode Support

HCatalog integration in Sqoop has been enhanced to support direct mode connectors. Direct mode connectors are high performance connectors specific to a database. The Netezza direct mode connector is enhanced to use this feature for HCatalog jobs.



Important

Only the Netezza direct mode connector is currently enabled to work with HCatalog.

7.4.2.4. Unsupported Sqoop Hive Import Options

Sqoop Hive options that are not supported with HCatalog jobs:

- `--hive-import`
- `--hive-overwrite`

In addition, the following Sqoop export and import options are not supported with HCatalog jobs:

- `--direct`
- `--export-dir`
- `--target-dir`
- `--warehouse-dir`
- `--append`
- `--as-sequencefile`
- `--as-avrofile`

7.4.2.5. Ignored Sqoop Options

All input delimiter options are ignored.

Output delimiters are generally ignored unless either `--hive-drop-import-delims` or `--hive-delims-replacement` is used. When the `--hive-drop-import-delims` or `--hive-delims-replacement` option is specified, all database columns of type `CHAR` are post-processed to either remove or replace the delimiters, respectively. (See [Delimited Text Formats and Field and Line Delimiter Characters](#) below.) This is only needed if the HCatalog table uses text format.

7.4.3. Controlling Transaction Isolation

Sqoop uses `read-committed` transaction isolation in its mappers to import data. However, this may not be ideal for all ETL workflows, and you might want to reduce the isolation guarantees. Use the `--relaxed-isolation` option to instruct Sqoop to use `read-uncommitted` isolation level.

The `read-uncommitted` transaction isolation level is not supported on all databases, such as Oracle. Specifying the `--relaxed-isolation` may also not be supported on all databases.



Note

There is no guarantee that two identical and subsequent uncommitted reads will return the same data.

7.4.4. Automatic Table Creation

One of the key features of Sqoop is to manage and create the table metadata when importing into Hadoop. HCatalog import jobs also provide for this feature with the option `--create-hcatalog-table`. Furthermore, one of the important benefits of the HCatalog integration is to provide storage agnosticism to Sqoop data movement jobs. To provide for that feature, HCatalog import jobs provide an option that lets a user specify the storage format for the created table.

The option `--create-hcatalog-table` is used as an indicator that a table has to be created as part of the HCatalog import job.

The option `--hcatalog-storage-stanza` can be used to specify the storage format of the newly created table. The default value for this option is "stored as rcfile". The value specified for this option is assumed to be a valid Hive storage format expression. It will be appended to the `CREATE TABLE` command generated by the HCatalog import job as part of automatic table creation. Any error in the storage stanza will cause the table creation to fail and the import job will be aborted.

Any additional resources needed to support the storage format referenced in the option `--hcatalog-storage-stanza` should be provided to the job either by placing them in `$HIVE_HOME/lib` or by providing them in `HADOOP_CLASSPATH` and `LIBJAR` files.

If the option `--hive-partition-key` is specified, then the value of this option is used as the partitioning key for the newly created table. Only one partitioning key can be specified with this option.

Object names are mapped to the lowercase equivalents as specified below when mapped to an HCatalog table. This includes the table name (which is the same as the external store table name converted to lower case) and field names.

7.4.5. Delimited Text Formats and Field and Line Delimiter Characters

HCatalog supports delimited text format as one of the table storage formats. But when delimited text is used and the imported data has fields that contain those delimiters, then

the data may be parsed into a different number of fields and records by Hive, thereby losing data fidelity.

For this case, one of these existing Sqoop import options can be used:

- `--hive-delims-replacement`
- `--hive-drop-import-delims`

If either of these options is provided on input, then any column of type `STRING` will be formatted with the Hive delimiter processing and then written to the HCatalog table.

7.4.6. HCatalog Table Requirements

The HCatalog table should be created before using it as part of a Sqoop job if the default table creation options (with optional storage stanza) are not sufficient. All storage formats supported by HCatalog can be used with the creation of the HCatalog tables. This makes this feature readily adopt new storage formats that come into the Hive project, such as ORC files.

7.4.7. Support for Partitioning

The Sqoop HCatalog feature supports the following table types:

- Unpartitioned tables
- Partitioned tables with a static partitioning key specified
- Partitioned tables with dynamic partition keys from the database result set
- Partitioned tables with a combination of a static key and additional dynamic partitioning keys

7.4.8. Schema Mapping

Sqoop currently does not support column name mapping. However, the user is allowed to override the type mapping. Type mapping loosely follows the Hive type mapping already present in Sqoop except that SQL types `"FLOAT"` and `"REAL"` are mapped to HCatalog type `"float"`. In the Sqoop type mapping for Hive, these two are mapped to `"double"`. Type mapping is primarily used for checking the column definition correctness only and can be overridden with the `--map-column-hive` option.

All types except binary are assignable to a `String` type.

Any field of number type (`int`, `shortint`, `tinyint`, `bigint` and `bigdecimal`, `float` and `double`) is assignable to another field of any number type during exports and imports. Depending on the precision and scale of the target type of assignment, truncations can occur.

Furthermore, `date/time/timestamps` are mapped to `string` (the full `date/time/timestamp` representation) or `bigint` (the number of milliseconds since epoch) during imports and exports.

BLOBs and CLOBs are only supported for imports. The BLOB/CLOB objects when imported are stored in a Sqoop-specific format and knowledge of this format is needed for processing these objects in a Pig/Hive job or another Map Reduce job.

Database column names are mapped to their lowercase equivalents when mapped to the HCatalog fields. Currently, case-sensitive database object names are not supported.

Projection of a set of columns from a table to an HCatalog table or loading to a column projection is allowed (subject to table constraints). The dynamic partitioning columns, if any, must be part of the projection when importing data into HCatalog tables.

Dynamic partitioning fields should be mapped to database columns that are defined with the NOT NULL attribute (although this is not validated). A null value during import for a dynamic partitioning column will abort the Sqoop job.

7.4.9. Support for HCatalog Data Types

All the primitive HCatalog types are supported. Currently all the complex HCatalog types are unsupported.

BLOB/CLOB database types are only supported for imports.

7.4.10. Providing Hive and HCatalog Libraries for the Sqoop Job

With the support for HCatalog added to Sqoop, any HCatalog job depends on a set of jar files being available both on the Sqoop client host and where the Map/Reduce tasks run. To run HCatalog jobs, the environment variable HADOOP_CLASSPATH must be set up as shown below before launching the Sqoop HCatalog jobs.

```
HADOOP_CLASSPATH=$(hcat -classpath)
export HADOOP_CLASSPATH
```

The necessary HCatalog dependencies will be copied to the distributed cache automatically by the Sqoop job.

7.4.11. Examples

- Create an HCatalog table, such as:

```
hcat -e "create table txn(txn_date string, cust_id string, amount float,
store_id int)
partitioned by (cust_id string) stored as rcfile;"
```

- Then Sqoop import and export of the "txn" HCatalog table can be invoked as follows:

Import

```
`${SQOOP_HOME}/bin/sqoop import --connect <jdbc-url> -table <table-
name> --hcatalog-table txn
```

Export

```
$SQOOP_HOME/bin/sqoop export --connect <jdbc-url> -table <table-name> --hcatalog-table txn
```