APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Succinctly®

ton Stoneman

Syncfusion | Technology Resource Portal

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Hire Smart

By

Elton Stoneman

Foreword by Daniel Jebaraj

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Syncfusion®

Deliver innovation with ease®

Copyright © 2016 by Syncfusion, Inc.

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

50

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

USA

All rights reserved.

INITIAL HERE

SIGN HERE

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a

APPROVED COMPLETED CONFIDENTIAL

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders sh claim, damages, or any other liability arising from, out of, or in ormation in this book.

Please do not use this book if the liste ble.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: James McCaffrey
Copy Editor: John Elderkin
Acquisitions Coordinator: Morgan Weston, social media marketing manager, Syncfusion, Inc.
Proofreader: Darren West, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books **7**

About the Author **9**

Chapter 1 Introducing Hive **10**

What is Hive? **10**

External tables 14

Views 15

Indexes 17

Summary **18**

Chapter 2 Running Hive **20**

Hive runtime options **20**

Installing Hive **20**

Running Hive in a Docker container **21**

Getting started with Hive **22**

Ho **24**

Summary **25**

Chapter 3 Internal Hive Tables **26**

Why use Internal tables? **26**

Defining internal tables **26**

File formats **28**

Simple data types **29**

Number types 29

Character types 31

Date and time types 33

Other types 35

Summary **36**

Chapter 4 External Tables Over HDFS **37**

Why use Hive with HDFS? **37**

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Defining external tables over HDFS files **37**

File formats **39**

Mapping bad data 41

Complex data types **42**

Mapping JSON files **44**

Summary **46**

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Mapping columns and column families 49

Converting data types **52**

Bad and missing data **53**

Connecting Hive to HBase **55**

INITIAL HERE

SIGN HERE

Configuring Hive to c 56

Hive, HBase and Docker Compose 56

Summary **58**

Chapter 6 ETL with Hive **59**

Extract, transform, load **59**

Loadi **59**

APPROVED

COMPLETED

CONFIDENTIAL

Inserting from query results **63**

Multiple inserts 67

Create table as select 67

Temporary tables **68**

Summary **72**

Chapter 7 DDL and DML in Hive **73**

HiveQL and ANSI-SQL **73**

Data definition **73**

Databases and schemas 73

Creating database objects 74

Modifying database objects 75

Removing database objects 78

Data manipulation **81**

ACID storage and Hive transactions 81

Summary **82**

Chapter 8 Partitioning Data **83**

Sharding data **83**

Partitioned tables **83**

INSERT, UPDATE, DELETE 89

Querying partitioned tables 92

Bucketed tables **93**

Creating bucketed tables 94

Populating bucketed tables 96

Querying bucketed tables 97

Summary **99**

Chapter 9 Querying with HiveQL **100**

The Hive Query Language **100**

Aggregation and windowing **100**

Built-in functions **107**

Date and timestamp functions 108

String functions 108

Mathematical functions 109

Collection functions 110

Other functions 111

User defined functions 112

Summary **113**

Next steps **113**

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The Story behind the *Succinctly* Series

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

an

Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

INITIAL HERE

SIGN HERE

Whenever platforms or tools are shipping out [illegible] seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While [illegible] omi[illegible] et [illegible] are being [illegible] s th[illegible] e as [illegible] ibit us is the inability to find concise technology overview books.

APPROVED

COMPLETED

CONFIDENTIAL

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep [illegible] ries of concise technical books that would be targeted at developers work [illegible] atform.

We firmly believe, given the backgrou [illegible] velopers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

SIGN HERE

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you

INITIAL HERE

APPROVED

COMPLETED

CONFIDENTIAL

Please follow us on Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

ch oft
connecting systems since 2000, and in recent years he's been designing and building Big Data solutions using a variety of Hadoop technologies and a mixture of on-premise and cloud deliveries.

Elton's latest Pluralsight courses have covered Big Data in detail on Microsoft's Azure cloud, which provides managed clusters for Hadoop and key parts of the Hadoop ecosystem, including Spark, HBase, and H

INITIAL HERE

SIGN HERE

Hive Succinctly is Elton's second eBook for Syncfusion, and it is accompanied by source code on GitHub and a Hive container image on the Docker Hub:

- https://github.com/sixeyed/hive-succinctly
- https://hub.docker.com/r/sixeyed/hive-succinctly

HBase Succinctly, Elton's first eBook for Syncfusion, remains available, and you'll also find him online blogging at https://blog.sixeyed.com and tweeting at @EltonStoneman.

APPROVED

COMPLETED

CONFIDENTIAL

Chapter 1 Introducing Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Hive is a data warehouse for Big Data. It allows you to take unstructured, variable data in Hadoop, apply a fixed external schema, and query the data with an SQL-like language. Hive abstracts the complexity of writing and running map/reduce jobs in Hadoop, presenting a familiar and accessible interface for Big Data.

INITIAL HERE

SIGN HERE

Hadoop is the most p ng and pr uantities of data. It runs on a cluster of machines—at the top end of the scale are Hadoop deployments running across thousands of servers, storing petabytes of data. With Hadoop, you query data using jobs that can be broken up into tasks and distributed around the cluster. These map/reduce tasks are powerful, but they are complex, even for simple queries.

Hive is an open source project from Apache that originated at Facebook. It was built to address the problem of making petabytes of data available to data scie chnical back g e and he Face at S on skill, so they designed Hive to provide an SQL facade over Hadoop.

APPROVED

COMPLETED

CONFIDENTIAL

Hive is essentially an adaptor between HiveQL, with the Hive Query Language based on SQL and a Hadoop data source. You can submit a query such as `SELECT * FROM people` to Hive, and it will generate a batch job to process the query. Depending on the source, the job Hive generates could be a map/reduce run Hadoop or a Java query over HBase tables.

Hive allows you to join across multipl you can write data as well as read it. This means you can run complex q esults in a simplified format for visualization. Hive can be accessed u aking it easy to integrate into your existing technology landscape, and Hive works well with other Big Data technologies such as HBase and Spark.

In this book, we'll learn how Hive works, how to map Hadoop and HBase data in Hive, and how to write complex queries in HiveQL. We'll also look at running custom code inside Hive queries using a variety of languages.

Use cases for Hive

Hive is an SQL facade over Big Data, and it fits with a range of use cases, from mapping specific parts of data that need ad-hoc query capabilities to mapping the entire data space for analysis. Figure 1 shows how data might be stored in an IoT solution in which data from devices is recorded in HBase and server-side metrics and logs are stored in Hadoop.

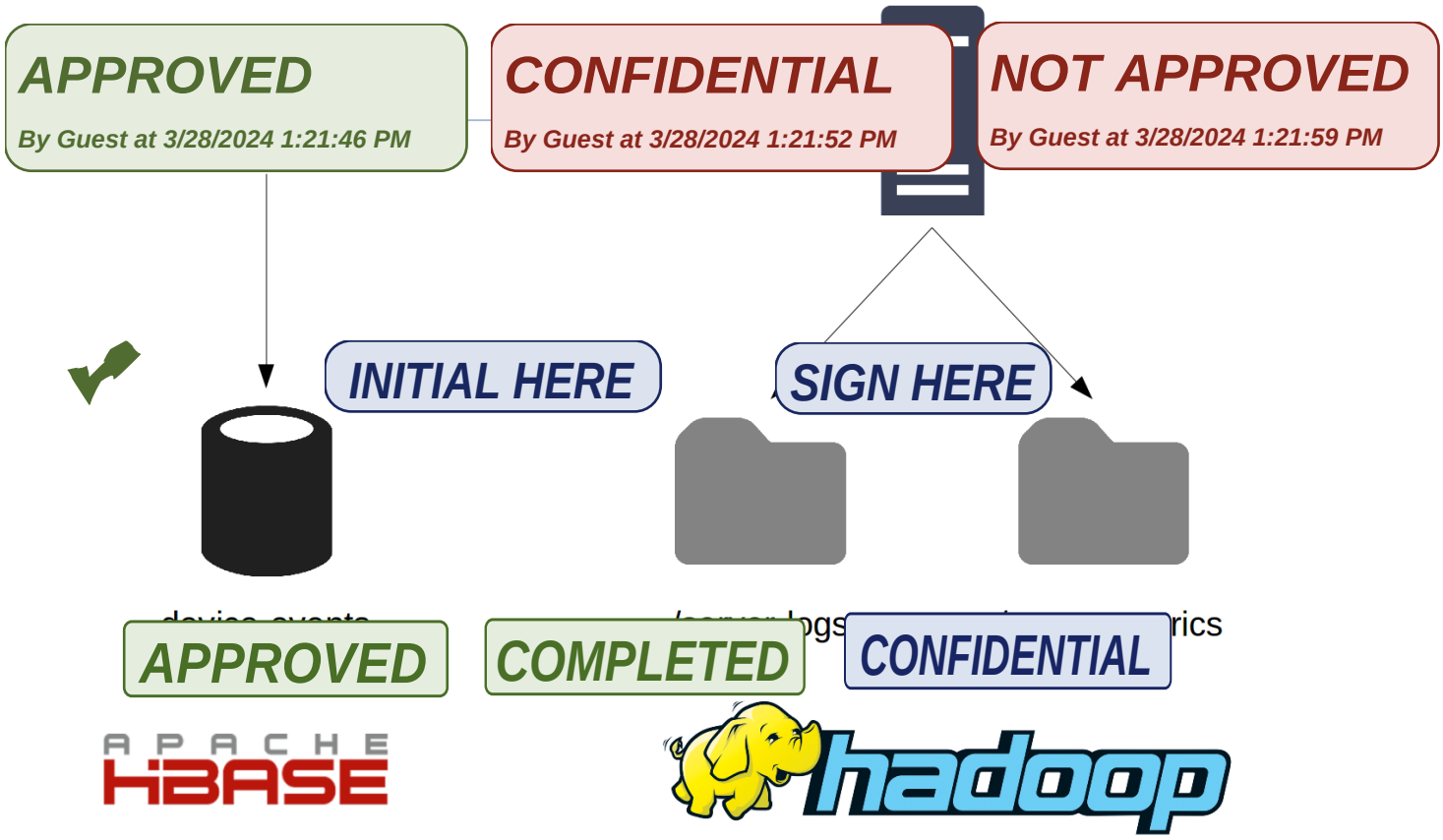

Figure 1: Big Dat Storage Sources

As Figure 2 demonstrates, this space ee tables in Hive: **`device_metrics`**, **`server_metrics`**, and **`server_logs`** e source folder and table names have hyphens, that isn't supported in **`vice-events`** becomes **`device_events`**.

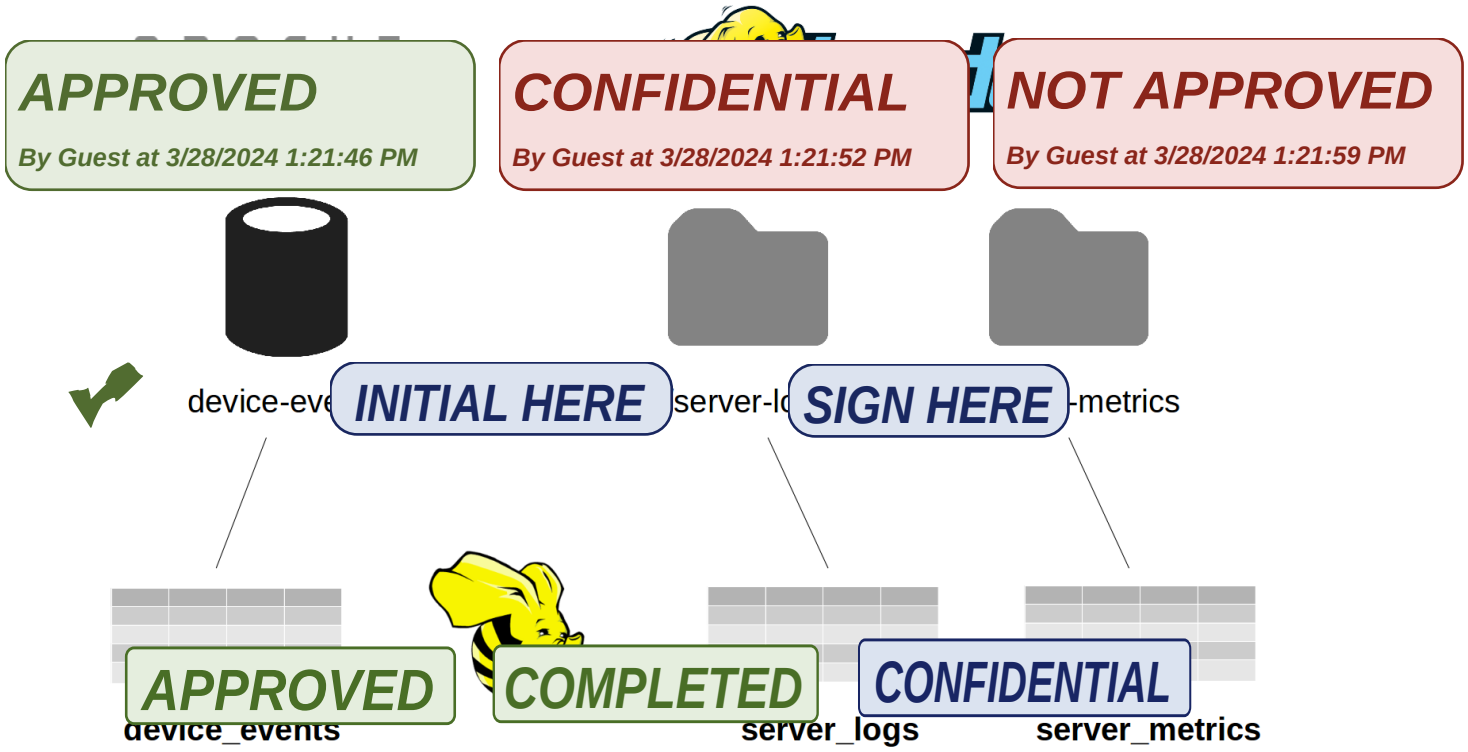

Figure 2: Mapping Multiple Sources with Hive

You can find the most active devices for a period by running a **`group by`** query over the **`device_metrics`** table. In that instan HBase driver, which provides real-time data access, and you can expec

If you want to correlate server errors an JOIN across the **`server_metrics`** and **`server_logs`** t be stored in a tab-separated variable (TSV) format, and metrics mi ve will abstract those formats, enabling you to query them in the same way. These files are stored in the Hadoop Distributed File System (HDFS), which means Hive will run them as a map/reduce job.

Hive doesn't only abstract the format of the data, it also abstracts the storage engine, which means you can query across multiple data stores. If you have a list of all known device IDs in a CSV file, you can upload that to HDFS and write an **`outer join`** query over the **`device_metrics`** HBase table and the **`device_ids`** CSV file in order to find devices that have not sent any metrics for a period.

We'll look more closely at running those types of queries in Chapter 9 Querying with HiveQL.

Hive typically runs on an existing Hadoop, HBase, or Spark cluster, which means you don't need additional infrastructure to support it. Instead, it provides another way to run jobs on your existing machines.

Hive's metastore, which is the database Hive uses to store table definitions separately from the data sources it maps, constitutes its only significant overhead. However, the metastore can be

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Hive data model

I've used a lot of terminology from SQL, such as tables and queries, joins, and grouping. HiveQL is mostly SQL-92 compliant, so that most of the Hive concepts are SQL concepts based on modeling data wit

INITIAL HERE

SIGN HERE

However, Hive doesn't support all the constructs available in SQL databases. There are no primary keys or foreign keys, which means you can't explicitly map data relations in Hive. Hive does support tables, indexes, and views in which tables are an abstraction over the source data, indexes are a performance boost for queries, and views are an abstraction over tables.

The main difference between tables in SQL and Hive comes with how the data is stored and acces th nd s xample, a tab s st use SQL, while Hive can use multiple data sources.

APPROVED

COMPLETED

CONFIDENTIAL

Hive can manage storage using internal tables, but it can also use tables to map external data sources. The definition you create for external tables tells Hive where to find the data and how to read it, but the data itself is stored and managed by another system.

Internal tables

Internal tables are defined in and phy e. When an internal table is queried, Hive executes the query by r anaged storage. We'll see how and where Hive stores the data for interna nternal Hive Tables.

In order to define an internal table in Hive, we use the standard **`create table`** syntax from SQL, specifying the column details—name, data type, and any other relevant properties. **Error! Reference source not found.** shows a valid statement that will create a table called **`server_log_summaries`**.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 1: Creating an Internal Table

```
      host STRING,
      logLevel STRING,
      count INT
)
```

This statement create[illegible] internal Hive table we can use to record summaries for our server logs. This way the raw [illegible] HDFS files [illegible] kept in Hive. Because Code Listing 1's statement is standard SQL, we can run it on either Hive or MySQL.

INITIAL HERE

SIGN HERE

External tables

External tables are defined in Hive, but they are physically managed outside of Hive. Think of external tables as a logical view of another data source. When we define an external table, Hive recor[illegible] not [illegible]urce[illegible]

APPROVED

COMPLETED

CONFIDENTIAL

When we query an external table, Hive translates the query into the relevant API calls for the data source (map/reduce jobs or HBase calls) and schedules the query as a Hadoop job. When the job finishes, the output is presented in the format specified in the Hive table definition.

External tables are defined with the **`create external table`** statement, which has a similar syntax to internal tables. However, Hive must also know where the data lives, how the data is stored, and how rows and columns a[illegible]ng 2 shows a statement that will create an external table over HDFS fi[illegible]

Code List[illegible]al Table

```
CREATE EXTERNAL TABLE server_l[illegible]
(
      serverId STRING,
      loggedAt BIGINT,
      logLevel STRING,
      message STRING
)
STORED AS TEXTFILE
LOCATION '/server-logs';
```

This statement would fail in a standard SQL database because of the additional properties we give Hive to set up the mapping:

- EXTERNAL TABLE—specifies that the data is stored outside of Hive.
- STORED AS TEXTFILE—indicates the format of the external data.
- LOCATION—shows folder location in HDFS where the data is actually stored.

Hive will assume by default that text files are delimited format, using ASCII character \001 (ctrl-A) as the field delimiter and the new line character as the row delimiter. And, as we'll see in

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

line it will map the fields in order. In the first field it will expect a string as the **`serverId`** value, and in the next field it will expect a long for the timestamp.

Mapping in Hive is robust, which means any blank fields in the source will be surfaced as NULL in the row. Any additional data after the mapped fields will be ignored. Rows with invalid data—incomplete number of fields, invalid field formats—will be returned when you query the table, but only fields that can be mapped.

INITIAL HERE

SIGN HERE

Views

Views in Hive work in exactly the same way as views in an SQL database—they provide a projection over tables in order to give a subset of commonly used fields or to expose raw data using a friendlier mapping.

APPROVED

COMPLETED

CONFIDENTIAL

In Hive, views are used to abstract the underlying storage of the data. For example, a Hive table can map an HDFS folder of CSV files, and it can offer a view providing access to the table. If all clients use the view to access data, we can change the underlying storage engine and the data structure without affecting the clients (provided we can alter the view and map it from the new structure).

Because views are an abstraction over tables, and because table definitions are where Hive's custom properties are used, the **`create view`** syntax in Hive is the same as in SQL databases. We specify a name for the view, and a select statement in order to provide the data, which can contain functions for transforming the data and can join across tables.

Unlike with SQL databases, views in Hive are not materialized. The underlying data is always retained in the original data source—it is not imported into Hive, and views remain static in Hive. If the underlying table structures change after a view is created, the view is not automatically refreshed.

Code Listing 3 creates a view over the **`server_logs`** table, where the UNIX timestamp (a long value representing the number of seconds since 1 January 1970) is exposed as a date that can be read using the built-in HiveQL function FROM_UNIXTIME. The log level is mapped with a CASE statement.

Code Listing 3: Creating a View

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
 CASE logLevel
  WHEN 'F' THEN 'FATAL'
  WHEN 'E' THEN 'ERROR'
  WHEN 'W' THEN 'WARN'
  WHEN 'I' THEN 'INFO'
 END,
 message
FROM server_logs
```

INITIAL HERE

SIGN HERE

Hive offers various client options for working with the database, including a REST API for submitting queries and an ODBC driver for connecting SQL IDEs or spreadsheets. The easiest option, which we'll use in this book, is the command line—called Beeline.

Code APPROVED nd COMPLETED g fe CONFIDENTIAL **ogs** table **orm** v ap e the data friendlier.

Code Listing 4: Reading from Tables and Views using Beeline

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-----------------------+-----------------------+-----------------------+--
| SCSVR1                | 1453562878            | W                     |
edbeuydbyuwfu        |
+-----------------------+-----------------------+-----------------------+--
1 row selected (0.063 seconds)

> select * from s                                  limit 1
+-------------------------------+---------------------------+-----------
| server_logs_formatted.serverid  | server_logs_formatted._c1  |
server_logs_formatted._c2  | server_logs_formatted.message  |
+-------------------------------+---------------------------+-----------
| SCSVR1                          | 2016-01-23                | WARN
| edbeuydbyuwfu                  |
+-------------------------------+---------------------------+-----------
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Indexes

Conceptually, Hive indexes are the same as SQL indexes. They provide a fast lookup for data in an existing table, which can significantly improve query performance, and they can be created over internal or external tables, giving us a simple way to index key columns in HDFS or HBase data.

An index in Hive is created as a sepa populated from a map/reduce job that Hive runs when we rebuild the in atic background index rebuilding, which means indexes must be rebuilt as changed.

Surfacing indexes as ordinary tables allows us to query them directly, or we can query the base table and let the Hive compiler find the index and optimize the query.

Code Listing 5 shows an index being created and then populated over the **`serverId`** column in the **`system_logs`** table (with some of the Beeline output shown).

Code Listing 5: Creating and Populating an Index

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
> alter index ix_server_logs_serverid on server_logs rebuild;

…

INFO  : The url t                    ://local
INFO  : Job runni                    Hadoop)
INFO  : 2016-01-25 07:30:48,507 Stage-1 map = 100%,  reduce = 100%
INFO  : Ended Job = job_local1186116405_0001
INFO  : Loading data to table
default.default__server_logs_ix_server_logs_serverid__ from
file:/user/hive/warehouse/default__server_logs_ix_server_logs_serverid__/.hive-
stag                    07-                    879
INFO                    Fau                    erv                    tats:
[numFiles=1, numRows=3, totalSize=142, rawDataSize=139]
No rows affected (1.936 seconds)
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Although the CREATE INDEX statement is broadly the same as SQL, specifying the table and column name(s) to index, it contains two additional clauses:

- 'COMPACT'—Hive supports a e which means we can use COMPACT indexes, suitable f th many values, or BITMAP indexes, which are more effici smaller set of repeated values.
- DEFERRED REBUILD—withc e populated when the CREATE INDEX statement runs. Deferring rebuild means we can populate the index later using the ALTER INDEX … REBUILD statement.

As in SQL databases, indexes can provide a big performance boost, but they do create overhead with the storage used for the index table along with the time and compute required to rebuild the index.

Summary

This chapter's overview of Hive addressed how the key concepts are borrowed from standard SQL, and it showed how Hive provides an abstraction over Hadoop data by mapping different sources of data as tables that can be further abstracted as views.

We've seen some simple HiveQL statements for defining tables, views, and indexes, and we have noted that the query language is based on SQL. HiveQL departs from standard SQL only

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

command line.

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Chapter 2. Running Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Because Hive sits naturally alongside other parts of Hadoop, it typically runs alongside an existing cluster. For nonproduction environments, Hadoop can run in local or pseudo-distributed mode, and Hive can submit jobs to Hadoop, which means it will use whatever runtime is configured.

INITIAL HERE SIGN HERE

Hive typically execute to Hado iginal map/reduce engine or, more commonly now, with Yet Another Resource Negotiator (YARN), the job management framework from Hadoop 2. Hive can run smaller queries locally using its own Java virtual machine (JVM) rather than submitting a job to the cluster. That's useful when developing a query because we can quickly run it over a subset of data and then submit the full job to Hadoop.

APPROVED COMPLETED CONFIDENTIAL

Setting up a production Hadoop cluster isn't a trivial matter, bu ily confi h th ude HDInsight cluste e h means it can run as part of a Hadoop, HBase, or Storm cluster with no extra configuration. When using Amazon's Elastic MapReduce, you will need to specify Hive as an option when creating a cluster.

Hive is a Java system, and it's not complex to install, but Hadoop must already be set up on your machine. The easiest way to run Hive for development and testing is with Docker, and in this chapter we'll look at using an image I've published on the Docker Hub that will help you get started with Hive.

Installing Hive

Hive is a relatively easy part of the Hadoop stack to install. It's a Java component with only a few options, and it is installed onto the existing Hadoop nodes.

For a single node dev or test environment, you should install HDFS first, before Hive. In both cases, you simply download the latest tarball and extract it. Hive's runtime behavior can be changed from a variety of settings specified in the **`hive-site.xml`** config file, but these changes are mostly optional.

Because Hive stores its own data in HDFS, you will need to set up the folders it expects to use and the necessary permissions using **`hdfs dfs`**, as shown in Code Listing 6.

Code Listing 6: Creating Hive Folders in HDFS

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
hdfs dfs -chmod g+w /tmp

hdfs dfs -chmod g+w /user/hive/warehouse
```

Note: Full insta ed in the ere: https://cwiki.apache.org/confluence/display/Hive/Administration. Or you can see the steps captured in the Dockerfile for the `hive-succinctly` image on GitHub: https://github.com/sixeyed/hive-succinctly/tree/master/docker.

INITIAL HERE

SIGN HERE

Running Hive in a Docker container

APPROVED COMPLETED CONFIDENTIAL

Dock r ex arn an spin up and kill instances with very little overhead, and you don't need to worry about any software or service conflicts with your development machine.

Docker is a cross-platform tool, which means you can run it on Windows, OS/X, or Linux, and installation is relatively simple. You can follow the instructions at http://docker.com. In addition to the runtime, Docker has a public registry of images, the Docker Hub, where you can publish and share your own images or pull image ared.

The image **`hive-succinctly`** on the put together expressly for use with this book. It comes with Hive already d, and the image is also preloaded with sample data you can use for tryi image, install Docker and execute the command in Code Listing 7.

Code Listing 7: Running Hive in Docker

```
docker run -d --name hive -h hive \

 –p 8080:8080 –p 8088:8088 –p 8042:8042 –p 19888:19888 \

 sixeyed/hive-succinctly
```

Some of the settings in the **`docker run`** command are optional, but if you want to code along with the sample in this book, you'll need to run the full command. If you're not familiar with Docker, here is a brief listing of the command's functions:

- Pulls the image called **`hive-succinctly`** from the **`sixeyed`** repository in the public Docker Hub.
- Runs the image in a local container with all the key ports exposed for the Hive Web UI.

- Names the image **hive**, allowing you to control it with other Docker commands without knowing the container ID that Docker will assign.

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

tn us
try wh
command runs, but with future runs the container will start in a few seconds and you'll have Hive running in a container with all the server ports exposed.

Note: The** `hive-succinctly` **image uses Hive 1.2.1 and Hadoop 2.7.2. It will remain at those v you can r s from this book using the exact s Hadoop ted mode—although it starts quickly it may take a couple of minutes for the servers to come online and make Hive available.

INITIAL HERE

SIGN HERE

Getting started with Hive

There nter ina repla l is doe ections, which means it must be run from the Hive master node. That limitation, along with issues concerning long-running queries, means the original CLI has been deprecated in favor of Beeline. Although there are some benefits to using the Hive CLI, we'll focus on Beeline in this book.

APPROVED

COMPLETED

CONFIDENTIAL

Beeline is an extension of the open source SQLLine JDBC command-line client, which means you can run it remotely and connect t d connect to any other JDBC-compliant server.

If you're running the **hive-succinct** e command from Code Listing 8 will connect you to the Hive container an t.

Code Listing 8: Starting Beeline in Docker

```
docker exec -it hive beeline
```

With Beeline, standard HiveQL queries are sent to the server, but for internal commands (such as connecting to the server) you will use a different syntax that is prefixed with the exclamation mark. Code Listing 9 shows how to connect to the Hive server running on the local machine at port 10000 as the user **root**.

Code Listing 9: Connecting to Hive from Beeline

```
0
```

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

In Code Listing 10, I select the first row from the **`server_logs`** table, which is already created and populated in the Docker image.

Code Listing 10: Running a Query in Beeline

```
> select * from s
+-----------------------+-----------------------+-----------------------+--
| server_logs.serverid  | server_logs.loggedat  | server_logs.loglevel  |
server_logs.message                             |
+-----------------------+-----------------------+-----------------------+--
| SCSVR1                | 1439546226            | W                     |
9c1224a9-294b-40a3-afbb-d7ef99c9b1f49c1224a9-294b-40a3-afbb-d7ef99c9b1f4  |
+---------------------------------------------------------------------------+--
1 ro                                  con
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The datasets in the image are small, just tens of megabytes, but if they are of Big Data magnitude, you might wait minutes or hours to get the results of your query. But, however large your data, and however complex your query, you should eventually get a result because Hive executes jobs using the core Had

When you have long-running jobs :or them with the standard UI interfaces from Hadoop—the YARN monitor port 8080, which is exposed in the Docker container, so that you car host machine. Figure 3 shows the UI for a running job at http://loca e you can drill down to the individual map/reduce tasks.

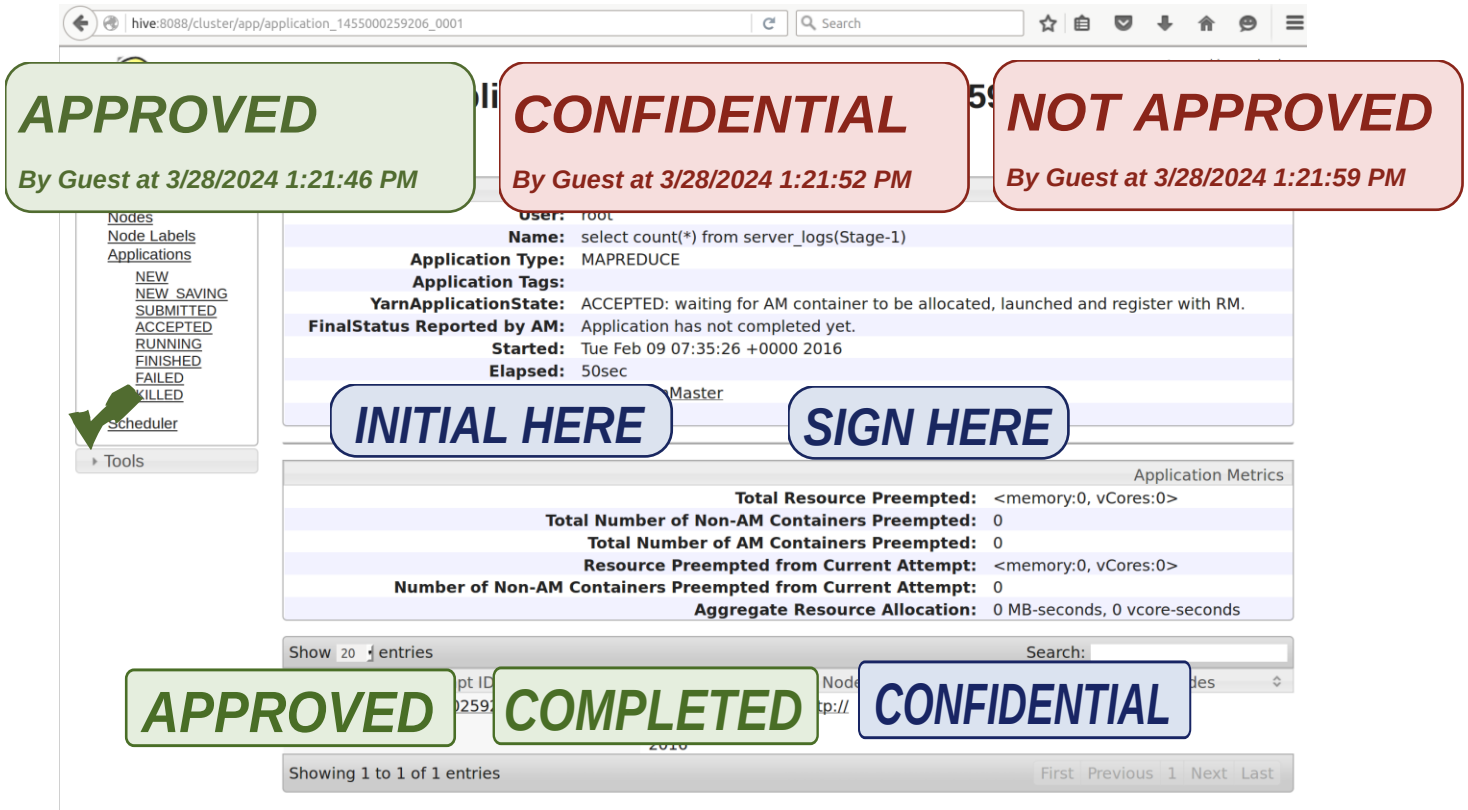

Figure 3: Monitoring Hive Jobs in the YARN UI

How the Hive runtime w

The key element of Hive is the compi rage-agnostic HiveQL query and translates it into a job to be executed or Hive tables mapped over files in HDFS, the compiler will generate a Ja for tables mapped over HBase, it will generate queries using the HBase Java API.

Hive sends the compiled job to the execution engine, which typically means creating multiple jobs in YARN—a master job for coordination that spawns multiple maps and reduces jobs. Figure 4 shows the steps from HiveQL query to YARN jobs.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

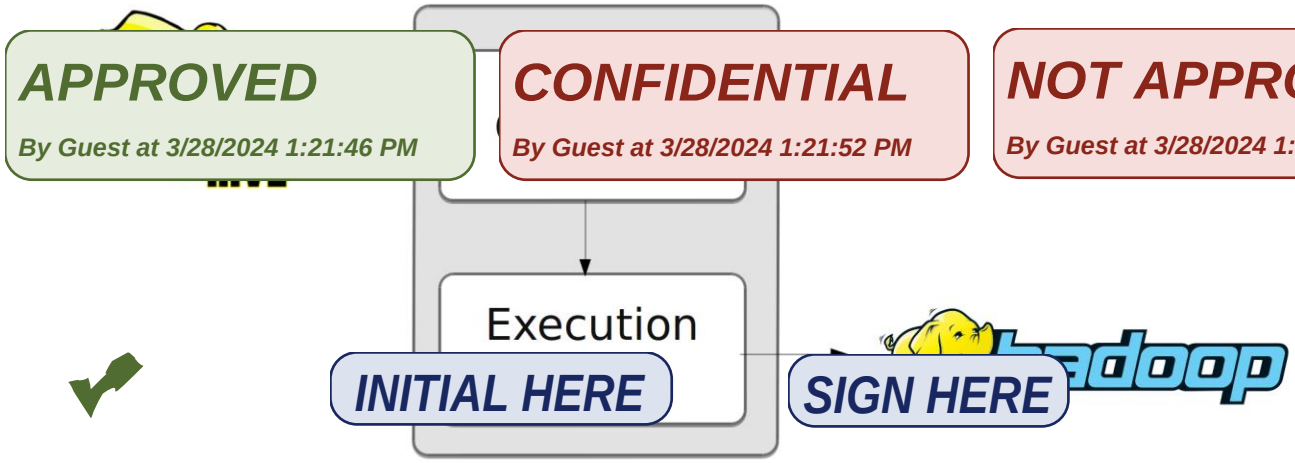

Figure 4: The Hive-Hadoop Architecture

Because the Hive compiler has a pluggable transform architecture, the new query functionality was provided by the HBase Storage Handler when HBase sup As Hive expa ec d ne provide the q

APPROVED

COMPLETED

CONFIDENTIAL

Summary

Hive is essentially a façade with a set of built-in adapters. When a HiveQL query runs, the compiler translates it into a map/redu nt storage handler, and the execution engine sends the job to Ha

Typically, Hive jobs will be run on a cl YARN, but for smaller queries and in nonproduction environments, Hive using its own JVM process.

Chapter 3 Internal Hive Tables

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Internal tables, also known as native or managed tables, are controlled by Hive—in fact, Hive owns the storage of these tables. They're still continued in HDFS, which means you get all the benefits of reliable, widely available data, and, if you choose a common format, you can still query Hive table files using other Hadoop tools.

INITIAL HERE

SIGN HERE

You receive the majo ctionality tables. Currently, updating or deleting data is only possible with managed tables (we'll cover this more in Chapter 7 DDL and DML in Hive), and there are many edge cases with HiveQL that work only with internal tables.

Using internal tables lets you focus on modeling the data in the way you want to use it while Hive worries about how the data is physically maintained. And for large datasets, you can configure sharding, so that tables are physically split across different files in order to improve perfo

APPROVED

COMPLETED

CONFIDENTIAL

Hive also allows for and manages temporary tables, and those are useful for storing intermediate result sets that Hive destroys when the session ends. The full set of Hive's Extract, Transform, and Load (ETL) tools are available for internal tables, which means they are a good choice for storing new data to be accessed primarily through Hive.

Hive stores internal tables as files in HDFS, which will allow you to access them using other Hadoop tools. The more optimized st upported by the entire Hadoop ecosystem. You can use internal Hive using a range of tools, but you will need to choose an interoperable form

Defining internal tables

The **`create table`** statement will create an internal table unless you specify the **`external`** modifier (which we will cover in Chapter 4 External Tables Over HDFS and Chapter 5 External Tables Over HBase). The simplest statement, shown in Code Listing 11, will create a table with a single column, using all default values.

Code Listing 11: Creating an Internal Hive Table

```
create table dual(r string);
```

The default root location in HDFS for Hive tables is **`/user/hive/warehouse`**, and in Code Listing 12 we can see that when the **`create table`** statement runs, Hive creates a directory called **`dual`**, but the directory will be empty.

Code Listing 12: HDFS Folder Created by Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
drwxrwxr-x   - root root       4096 2016-01-25 18:02
/user/hive/warehouse/dual

root@hive:/hive-setup# hdfs dfs -ls /user/hive/warehouse/dual
root@hive:/hive-setup#
```

INITIAL HERE

SIGN HERE

When we insert data ll create pulate it. Code Listing 13 shows an **insert** statement in the new table, with all the output from Beeline, which allows us to see what Hive is doing.

Code Listing 13: Inserting a Row into a Hive Table

APPROVED

COMPLETED

CONFIDENTIAL

```
> insert into dual(r) values('1');
INFO  ta  e t  rator
INFO  1
INFO  : Submitting tokens for job: job_local1569518498_0001
INFO  : The url to track the job: http://localhost:8080/
INFO  : Job running in-process (local Hadoop)
INFO  : 2016-01-25 18:07:39,487 Stage-1 map = 100%,  reduce = 0%
INFO  : Ended Job = job_local1569518498_0001
INFO  : Stage-4 is selected by  .
INFO  : Stage-3 is filtered ou  olver.
INFO  : Stage-5 is filtered ou  olver.
INFO  : Moving data to: file:/  e/dual/.hive-
staging_hive_2016-01-25_18-07-  4589876-2/-ext-10000 from
file:/user/hive/warehouse/dual/.hive-staging_hive_2016-01-25_18-07-
37_949_178012634824589876-2/-ext-10002
INFO  : Loading data to table default.dual from
file:/user/hive/warehouse/dual/.hive-staging_hive_2016-01-25_18-07-
37_949_178012634824589876-2/-ext-10000
INFO  : Table default.dual stats: [numFiles=1, numRows=1, totalSize=2,
rawDataSize=1]
No rows affected (1.724 seconds)
```

There is a lot of detail in the INFO level output, and some of it offers useful information:

- What is the URL for tracking the job in Hive's Web UI (good for long-running queries).
- How the job is running (in-process rather than through YARN).
- How the job is structured (into map and reduce stages).

- What Hive is doing with the data (first loading it to a staging file).

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Hive deals with appending data to HDFS via the staging file. Hive writes all the new data to it, and when the write is committed Hive moves the file to the correct location in HDFS. Inserting rows into internal tables is an ACID operation, and when it finishes we can view the file and its contents in HDFS using the list and cat commands in Code Listing 14.

File Contents

INITIAL HERE

SIGN HERE

```
root@hive:/hive-setup# hdfs dfs -ls /user/hive/warehouse/dual/

Found 1 items

-rwxrwxr-x   1 root root          2 2016-01-26 07:00
/user/hive/warehouse/dual/000000_0

root
1
```

APPROVED

COMPLETED

CONFIDENTIAL

The **`hdfs dfs -ls`** command tells us there is no one file in the directory for the **`dual`** table, called **`000000_0`**, and that file's contents make up one row with a single character, the '1' we inserted into the table. We can read the contents of the file because the default format is text, but Hive also supports other, more efficient file formats.

File formats

The **`create table`** command suppo se that specifies the physical file format for the data files. The clause is optional, and if you omit it as we did in Code Listing 14, Hive assumes the default **`stored as textfile`**.

Hive has native support for other file types that are supported by other tools in the Hadoop ecosystem. Here are three of the most popular:

- AVRO—schema-based binary format, interoperable across many platforms.
- ORC—Optimized Row Columnar format, built for Hive as an efficient format.
- PARQUET—a compressed columnar format, widely used in Hadoop.

If you are creating new data with Hive, the ORC format is typically the optimal choice for storage size and access performance, but it is not widely supported by other Hadoop tools. If you want to use other tools to access the same data (such as Pig and Spark), Parquet and Avro are more commonly supported.

Columnar file formats have a more intelligent structure than flat text files, and they typically store data in blocks along with a lightweight index that Hive uses to locate the exact block it needs to

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

in CPU time needed to compress and decompress data is usually negligible compared to the time saved in transferring smaller files across the network or blocks from disk into memory.

Note: You can change the file format of an existing table `using alter table… set fileformat` ***but Hive does not automatically convert all the existing data. If you have data in a ta file form ore data, you will have multiple files in different formats—Hive read from the table. If you want to change the format, you should use one of the ETL options described in Chapter 6 ETL with Hive.***

INITIAL HERE

SIGN HERE

Simple data types

APPROVED

COMPLETED

CONFIDENTIAL

One eat str semistructured Hadoop data. When you define a table in Hive, each column uses a specific data type. You need not worry about how the data is mapped with internal tables because Hive owns the storage and takes care of serializing and deserializing the data on disk.

Hive provides all the basic data types used in typical databases, and it also includes some higher-value data types that allow you to more accurately model your data. For complex data types, Hive uses columns that can co different types of collections. We'll look at those in Chapter 4 External T n the context of mapping existing data.

Number types

With built-in functions for mathematical operations such as log, square root, and modulus, Hive offers richer support for numerical data than many relational databases. Hive can also implicitly convert between multiple integer and floating-point types (so long as you are converting from a smaller capacity to a larger one).

In ascending order of capacity, here are the four integer types:

- TINYINT—from -128 to +127, postfix with 'Y' in literals.
- SMALLINT—from -32768 to +32767, postfix with 'S' in literals.
- INT—from -2147483648 to +2147483647, no postfix needed.
- BIGINT—from -9223372036854775808 to +9223372036854775807, postfix 'L'.

INT is the default integer type, but the +/-2Bn range will be limiting if you are storing sequences, counts, or UNIX timestamps (although Hive has a specific data type for that).

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 15 shows the result of breaching column capacity—in this case turning a positive TINYINT into a negative value and a negative SMALLINT into a positive value.

Code Listing 15: Wrapping Numbers from Positive to Negative

INITIAL HERE

SIGN HERE

```
> select (127Y +
+-------+--------+--+
|  _c0  |  _c1   |
+-------+--------+--+
| -128  | 32767  |
+-------+--------+--+
```

APPROVED COMPLETED CONFIDENTIAL

- FLOAT—single precision, 4-byte capacity.
- DOUBLE—double precision, 8-byte capacity.
- DECIMAL—variable precision, 38-digit capacity.

DECIMAL types default to a precision of zero, but they are typically defined with specific values—e.g., a five-digit number with two decimal places would be defined as DECIMAL (5,2) and could hold a max

With the DECIMAL type using zero scale, you can store larger integers than is possible with BIGINT. The postfix for integer numbers represented as a decimal is BD (the DECIMAL type is based on Java's BigDecimal type), and as Code Listing 16 shows, this allows you to work beyond the BIGINT limits.

Code Listing 16: Representing Large Integers with BigDecimal

```
> select (9223372036854775807L * 10), (9223372036854775807BD * 10);

+------+----------------------+--+
| _c0  |          _c1         |
+------+----------------------+--+
| -10  | 92233720368547758070 |
+------+----------------------+--+
```

Hive supports both scientific notation and standard notation for floating-point numbers, and it allows a mixture of them in the same rows and tables.

Hive will approximate to zero or infinity when you reach the minimum and maximum limits of the decimal types it can store. But those limits are at powers of approximately +/-308, which means

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
> select 1E308, 1E330, -1E-308, -1E-330;

+----------+-----------+------------+-------+--+
|   _c0    |    _c1    |    _c2     |  _c3  |
+----------+-----------+------------+-------+--+
|  1.0E308 | Infi                   -0.0  |
+----------+-----------+------------+-------+--+
```

INITIAL HERE

SIGN HERE

Character types

The primary character type in Hive is STRING, which does not impose a maximum length and practically supports strings of any size. User-Defined Functions [illegible] typically use s[illegible]ces

APPROVED

COMPLETED

CONFIDENTIAL

- STRING—unlimited size, literals can be delimited with single or double quotes.
- VARCHAR—specified maximum size, whitespace in input is preserved.
- CHAR—fixed length, smaller input is padded with whitespace.

You can compare strings in columns [illegible] types, but you must know whether or not whitespace will affect the comp[illegible] ARCHAR types preserve whitespace, which means values with [illegible] but that end with different numbers of spaces are not equal. CHAR fields [illegible] the set length with spaces, so that only the text is compared.

Code Listing 18 creates a table with three character columns and inserts rows that include the same text in each field and with differing amounts of trailing whitespace.

Code Listing 18: Creating Tables with Character Fields

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
> insert into strings(a_string, a_varchar, a_char) values('a', 'a', 'a');

No rows affected (1.812 seconds)

> insert into strings(a_string, a_varchar, a_char) values('b', 'b', 'b');

No rows affected (1.381 seconds)
```

INITIAL HERE

SIGN HERE

Because the table is in the default file format, it is stored as text, and when we read the files we can see where the whitespace is being persisted, as in Code Listing 19 (where '#' represents the default separator \0001).

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 19: Reading Tables with Character Fields

```
root@hive:/hive-setup# hdfs dfs -cat /user/hive/warehouse/strings/*

a#a#a

b         #b      #b
```

If we query that table to find rows with [illegible] will receive only the first row because the fields in the second row [illegible] trailing whitespace. But if we use the **trim()** function to clear the white[illegible]on, both rows are returned—as in Code Listing 20.

Code Listing 20: Comparing Values in Character Fields

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
+-------------------+--------------------+-----------------+--+
| a                 | a                  | a               |
+-------------------+--------------------+-----------------+--+
1 row selected (0.088 seconds)

> select * from s                    tring) =                    and
trim(a_string) = a_char;

+-------------------+--------------------+-----------------+--+
| strings.a_string  | strings.a_varchar  | strings.a_char  |
+-------------------+--------------------+-----------------+--+
| a                 | a                  | a               |
| b
+---
2 rows selected (0.095 seconds)
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Date and time types

Hive explicitly supports date and time e of storing high-precision timestamps or dates without a time c

- TIMESTAMP—UNIX-style tim e elapsed since epoch. Precision can vary from seconds to nan
- DATE—a date with no time component.

Both types support literal expressions as strings in the formats **`'yyyy-MM-dd'`** (for dates) and **`'yyyy-MM-dd HH:mm:ss.fff'`** (for timestamps, with up to nine decimal places supporting nanosecond precision).

If you have values recorded as integer UNIX timestamps, you can insert them into TIMESTAMP columns using the **`from_unixtime()`** function. Note that only second precision is supported here, and you cannot use functions in an **`insert … values`** statement, which means the syntax differs for string and integer timestamp insertion, as shown in Code Listing 21.

Code Listing 21: Converting Dates and Timestamps

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
> insert into datetimes(a_timestamp, a_date) values('2016-01-27
07:19:01.001', '2016-01-27');

...

No rows affected

> from dual insert into table datetimes select from_unixtime(1453562878),
'2016-01-23';

...

No rows affected (1.296 seconds)
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Here [illegible] n, and we insert two rows with different levels of precision in the timestamp. The **`from dual`** is a trick that lets us use a select statement with functions as the source clause for an insert (we'll cover that more in Chapter 6 ETL with Hive).

Hive supports conversion between timestamps and dates, and the built-in date functions apply to both types of column. With timestamps, the time portion will be lost if you convert to date type, and with dates any time-based f[illegible] LL. Code Listing 22 shows those conversions and some sample functio[illegible]

Code Listing 22: Working with Date Types

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+--------------------------+-------+------------+-------+--+
|       a_timestamp        |  _c1  |   a_date   |  _c3  |
+--------------------------+-------+------------+-------+--+
| 2016-01-27 07:19:01.001  | 2016  | 2016-01-27 | 2016  |
| 2016-01-23 15:27:58.0    | 2016  | 2016-01-23 | 2016  |
+--------------------------+-------+------------+-------+--+
2 rows selected (
```

INITIAL HERE

SIGN HERE

```
> select a_timestamp, hour(a_timestamp), a_date, hour(a_date) from
datetimes;

+--------------------------+-------+------------+-------+--+
|       a_timestamp        |  _c1  |   a_date   |  _c3  |
+--------------------------+-------+------------+-------+--+
| 20
| 2016-01-23 15:27:58.0    | 15    | 2016-01-23 | NULL  |
+--------------------------+-------+------------+-------+--+
2 rows selected (0.073 seconds)
```

APPROVED

COMPLETED

CONFIDENTIAL

```
> select cast(a_timestamp as date), cast(a_date as timestamp) from
datetimes;

+--------------+--------------
| a_timestamp  |         a_dat
+--------------+--------------
| 2016-01-27   | 2016-01-27 00:00:00.0  |
| 2016-01-23   | 2016-01-23 00:00:00.0  |
+--------------+-----------------------+--+
2 rows selected (0.067 seconds)
```

Other types

There are two other simple types in Hive:

- BOOLEAN—for true/false values.
- BINARY—for arbitrary byte arrays, which Hive does not interpret.

Boolean values are represented with the literals **`true`** and **`false`**; you can cast other types as boolean, but you might not get the expected result. Unlike with other languages, in Hive **`false`**

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

the blob data type in SQL databases. Binary column values in Hive are stored in-line in the data file with the rest of the row, rather than as a pointer to a blob. Because Hive doesn't interact with binary data, binary values are not widely used.

Summary

INITIAL HERE

SIGN HERE

In this chapter we've looked at using internal tables with Hive. The underlying data for internal tables is stored as files in HDFS, which means Hive gets all the reliability and scalability of Hadoop for free. By using internal tables, Hive controls reading and writing at the file level, so that the full feature set of Hive is available.

We also looked at the different file formats Hive provides, with text files as the default and more efficient binary formats available. When you choose a file format, you need to access the raw data. If not, Hive's ORC format is a good choice; otherwise Parquet and Avro are well supported in Hadoop. Flat files can be supported by many options.

APPROVED

COMPLETED

CONFIDENTIAL

Lastly, we looked at all the simple data types available in Hive, noting that these are equally suitable to internal and external tables (provided they can be correctly mapped from the source data). In the next chapters we'll look at using external tables instead, and we'll see how to use Hive with existing HDFS files and with

Chapter 4 External Tables Over HDFS

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Hive allows us to write queries as though we're accessing a consistent, structured data store by applying a fixed schema over a variety of formats in HDFS. HiveQL presents a familiar, simple entry point that lets users run complex queries without having to understand Java or the map/reduce API.

INITIAL HERE

SIGN HERE

With Hive, we can ap at will sim rs work with higher-level constructs such as tables and views, again without needing to understand the properties in the underlying data files.

Hive has native support for all the major file formats in Big Data problems—CSV, TSV, and JSON (together with more exotic formats such as ORC and Parquet). As with other tools in the Hadoop ecosystem, Hive also uses native support for compression, so that if raw data is compressed with GZip, BZip2, or Snappy, Hive can access it

APPROVED

COMPLETED

CONFIDENTIAL

And b d v entially in standard SQL, Hive metadata acts as living documentation over Hadoop files, with the mappings clearly defining the expected content of the data.

Defining external tables over HDFS files

When you use HDFS as the backing ou actually map the table to a folder in HDFS. So if you are appending eve using a time-base structure, as in Figure 5, you define the Hive table at ructure.

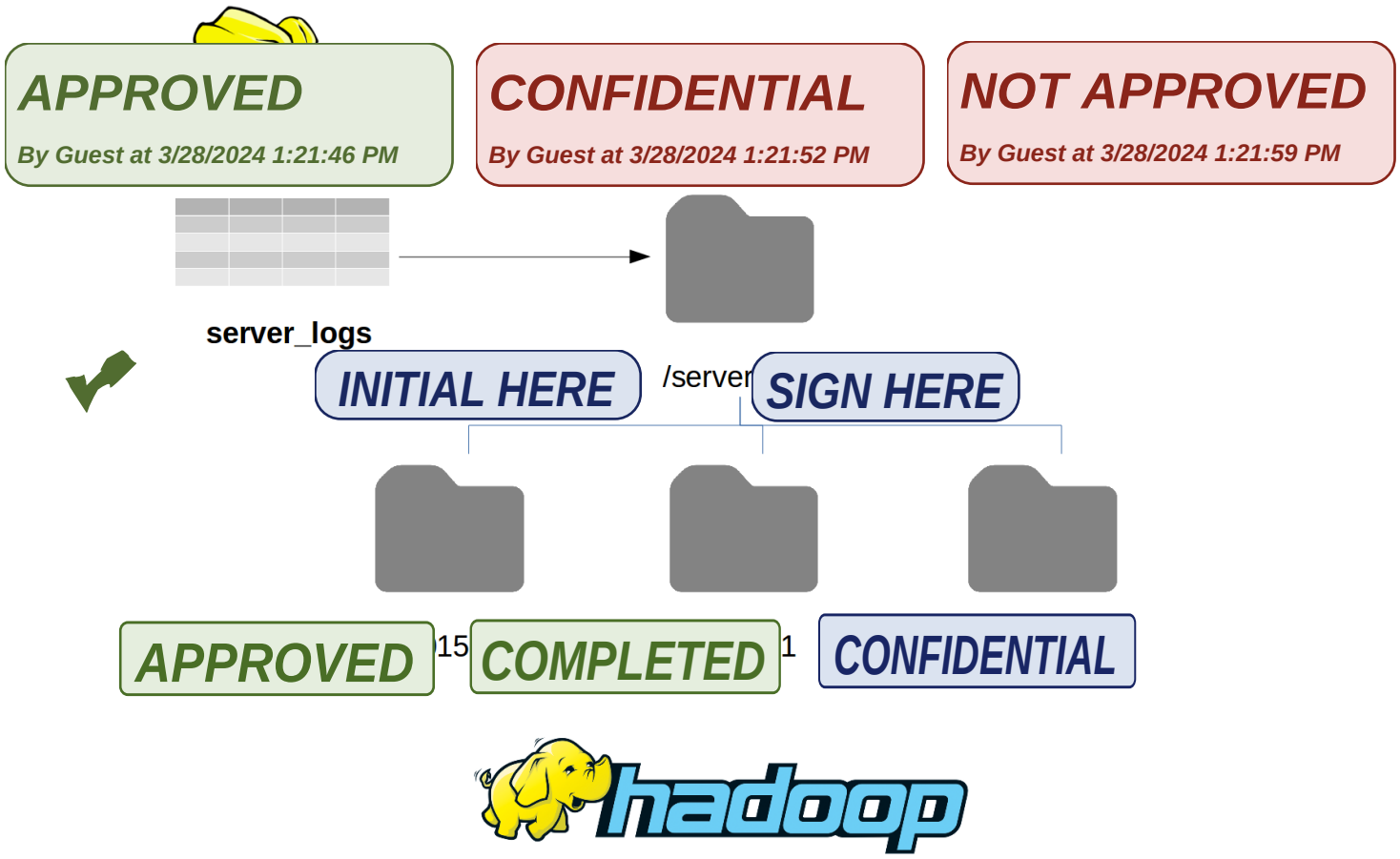

Figure 5 *in Hive*

In this case, the **`server_logs`** table c e **`/server/logs`** folder as the root location, with all the files under all the available when we query the table. The Hive table definition specifies onl n, and Hive will not list out the files and construct map jobs to read them until you submit a query.

The **`server-logs`** table is already configured in the **`hive-succinctly`** Docker image. The files are in the **`data`** directory in the HDFS file system root, and they are stored in CSV format. Code Listing 23 shows some sample rows from one file using the **`hdfs dfs -cat`** command.

Code Listing 23: Sample Lines from HDFS Files

```
b58e-971656c20462
SCSVR1,1448727463,D,44610125-4bdb-4046-b363-aa7a0cd28bde44610125-4bdb-4046-
b363-aa7a0cd28bde
```

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

The server log files have one line for each log entry, and entries have the same fields in the following order:

INITIAL HERE

SIGN HERE

- Timestamp—UNIX timestamp of the log entry.
- Hostname—name of the server writing the log.
- Level—log level, using the standard Apache log4* levels (e.g., D=debug, W=warn, E=error).
- APPROVED e. COMPLETED CONFIDENTIAL

To map that data in Hive, we need to use the **`create external table`** statement, which specifies the field mappings, data format, and location of the files. Code Listing 24 shows one valid statement for creating the table.

Code Listing 24: Mapping HDFS Files as an External Table

```
create external table server_l
(serverid string, loggedat big                    ng, message string)
 row format delimited
 fields terminated by ','
 stored as textfile
 location '/server-logs';
```

Columns are defined using positional mapping, so that the first column in the table will be mapped to the first field in each row, and the last column will be mapped to the last field. We're using some of the simple data types we've already seen—BIGINT and STRING—and we'll work with more complex data types later in this chapter.

File formats

The same file formats available for internal tables can also be used for external tables. The file structure for columnar formats such as ORC and Avro is well known—you shouldn't need to customize the table in Hive unless you are specifying **`stored as`** ORC or Avro.

However, text files might have any structure, and the Hive default of delimiting rows with the newline character and fields with \001 is not always suitable. The **create external table** te g

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

ha is files in which the rows are delimited by the new-line character, fields are delimited with vertical tab (ASCII character 011), and special characters are escaped with the tilde.

Code Listing 25: Specifying Delimiters for External Tables

```
create external table server_metrics
(serverId string, , cpuPc
 memoryPc decimal(3,1), storagepc decimal(3,1),
row format delimited
fields terminated by '\011'
escaped by '~'
lines terminated by '\n'
stored as textfile
location '';
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Hive will take this unusual format and map it into usable rows and columns. Code Listing 26 shows the first line of the raw data in HDFS, followed by the same data mapped as a row in Hive.

Code Listing 26: Mapping Unusual File Formats

```
root@hive:/hive-setup# hdfs dfs                    ics/server_metrics.txt |
head -n 1

SCSVR1~     LON   2016-01-28 18                    .1  12.2

> select * from server_metrics limit 1;
+-------------------------+---------------------------+-------------------
| server_metrics.serverid  | server_metrics.recordedat  |
server_metrics.cpupc  | server_metrics.memorypc  | server_metrics.storagepc
|
+-------------------------+---------------------------+------------------
| SCSVR1    LON               | 2016-01-28 18:05:01.0      | 32.6
| 64.1                     | 12.2                      |
+-------------------------+---------------------------+-------------------
```

Here are the clauses used to define the structure of an external HDFS source:

- ROW FORMAT—either 'DELIMITED' for flat files or 'SERDE' for complex formats with a custom serializer/deserializer (as with JSON, which we'll see later in this chapter).

- LINES TERMINATED BY—the delimiter between lines of data, mapped to rows in the Hive table. Currently only the new-line ('\n') character is allowed.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

- ESCAPED BY—the character used to escape the field delimiter, e.g., if your file is a CSV you can escape commas inside fields with a backslash so that '\,' means a comma in a string field, not the start of a new field.

All clauses except ROW FORMAT take a single character, and you can use backslash notation with ASCII values for e.g., **'\01** The same clauses are available for inter s, which c nt Hive to own the data, but you will need a custom format that you can use with other tools.

INITIAL HERE

SIGN HERE

Mapping bad data

When you run a **`create external table`** statement, no data validation occurs when the statement runs. Hive will create the HDFS folder if it doesn't e exist Hive her ent of fields. The mappings are performed at runtime when you query the table.

APPROVED

COMPLETED

CONFIDENTIAL

Within each row, Hive attempts to map HDFS data at the field level. If a row is missing a field, the column mapped from that field will be returned as null for that row. If a row has a field containing data that Hive can't convert to the specified column type, that column will be returned as null for the row.

In the **`hive-succinctly`** Docker ima **ogs** is already created, and there are several files in the location folder. have valid data, matching the external table definition, so that Hive all rows for those files.

One file in the location is not in the co sage field is missing. Also note that the timestamp and server name fields are in the wrong order. Hive will still attempt to read data from that file, but, as Code Listing 27 shows, some of the mappings will fail.

Code Listing 27: Mapping Bad Data

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
server_logs.message  |
+----------------------+----------------------+----------------------+---
| 1453562878000        | NULL                 | W                    |
NULL                 |
| 1453562879000        | NULL                 | F                    |
NULL
```

INITIAL HERE

SIGN HERE

Because Hive can convert a numeric value into a string, the **`serverId`** column is returned, but the **`loggedAt`** timestamp is NULL—Hive can't convert the string data into long values. And because Hive continues processing the row even when it finds an error, the **`logLevel`** fields are mapped, but the **`message`** column (which is missing in the CSV file) is NULL.

If the source is incorrectly formated, for example if a location is mapped as ORC in Hive but the data APPROVED COMPLETED CONFIDENTIAL Code Listin

Code Listing 28: Mapping the Wrong File Type

```
> select * from server_logs_orc;

Error: java.io.IOException: java.io.IOException: Malformed ORC file
file:/server-logs/server_logs_ostscript. (state=,code=0)
```

Tip: This can happen if you ormat of a table that exists in Hive—the expected format gets store, but the existing files aren't converted. You can recover the k to the original format.

Complex data types

In addition to primitive data types (such as INT, STRING, and DATE, as seen in Chapter 3 Internal Hive Tables), Hive supports three complex data types that can be used to represent collections of data:

- ARRAY—an ordered collection in which all elements have the same data type.
- MAP—an unordered collection of key-value pairs. Keys must all have the same data type—a primitive type—and values must have the same data type = which can be any type.
- STRUCT—a collection of elements with a fixed structure applied.

When collection types are mapped from HDFS files, the mappings in the **`create external table`** statement must specify the delimiters for the elements of the collection. Code Listing 29

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
SCSVR1,192.168.2.1:192.168.2.2,8:32:1500,country=gbr:dc=london
SCSVR2,192.168.20.1:192.168.20.2,4:16:500,country=gbr:dc=london
SCSVR3,192.168.100.3:192.168.100.4,16:32:500,country=roi:dc=dublin
```

In this file, fields are separated after the first two commas. Field 2 contains the IP addresses separated by colons. Field 3 contains hardware details, again separated by colons. Field 4 contains the server's location as key-value pairs.

INITIAL HERE

SIGN HERE

We can map these to the relevant collection types in Hive using an array for the IP addresses, a struct for the hardware, and a map for the location. Code Listing 30 shows how to specify those mappings when we create the table.

Code Listing 30: Mapping Collection Columns

APPROVED

COMPLETED

CONFIDENTIAL

```
create external table servers
 (name string, ipAddresses array<string>,
  hardware struct<cores:int, ram:int, disk:int>,
  site map<string, string>)
row format delimited
fields terminated by ','
collection items terminated by
map keys terminated by '='
lines terminated by '\n'
stored as textfile
location '/servers';
```

We specify three clauses for Hive in order to identify the delimiters in collections:

- FIELDS TERMINATED BY—the field separator, comma in this case.
- COLLECTION ITEMS TERMINATED BY—separator for collection elements, colon in this case.
- MAP KEYS TERMINATED BY—the separator for key-value pairs, equal sign in this case.

Code Listing 31 shows how Hive represents collection columns when we fetch rows.

Code Listing 31: Reading Rows with Collection Columns

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
|              servers.site                |
+---------------+---------------------------------+------------------------
| SCSVR1        | ["192.168.2.1","192.168.2.2"]    |
{"cores":8,"ram":32,"disk":1500}  | {"country":"gbr","dc":"london"}  |
| SCSVR2        | ["192.168.20.1","192.168.20.2"]  |
{"cores":4,"ram":1                ountry"                  on"}  |
+---------------+-----------------------------------------------------------
```

INITIAL HERE

SIGN HERE

Note:** The `terminated by` **clauses are specified once and apply to the entire table, which means your delimiter fields must be consistent in the source file. If your source has multiple complex types, they all must use the same delimiters—you can't have arrays delimited by semicolons and structs delimited by underscores in the same ta

APPROVED

COMPLETED

CONFIDENTIAL

Mapping JSON files

Hive uses a pluggable serializer/d (called "SerDe") to read and write text files. For all the native data types hould use will be implicitly specified with the **`stored as`** clause. With custo provide your own SerDe and configure it in the **`create table`** statement.

Several open source SerDe components can provide JSON file support in Hive, and one of the best, which allows reading and writing JSON, comes from Roberto Congiu on GitHub. You can download the latest Java Archive (JAR) file from Roberto's website.

You will need to register the JAR file with Hive in order to use the custom SerDe, and you must map from the JSON format, typically representing JSON objects as nested structs. Code Listing 32 shows the steps.

Code Listing 32: Creating a Table Using JSON SerDe

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
INFO  : Added resources: [/tmp/json-serde-1.3.7-jar-with-dependencies.jar]

> create external table devices

> (device struct<deviceClass:string, codeName:string,

> firmwareVersio                u:struct                :int>>)

> row format serde 'org.openx.data.jsonserde.JsonSerDe'

> location '/devices';
```

INITIAL HERE

SIGN HERE

The **row format** clause specifies the class name of the SerDe

APPROVED

COMPLETED

CONFIDENTIAL

(**org.** **.Js** , th ies the root f

You can map the source JSON data in different ways with the complex data types available in Hive, which allows you to choose the most appropriate format for accessing the data. A sample JSON object for my **devices** table is shown in Code Listing 33.

Code Listing 33: Sample JSON Source Data

```
{
      "device": {
            "deviceClass": "t
            "codeName": "jeri
            "firmwareVersions": ["1.0.0", "1.0.1"],
            "cpu": {
                  "speed": 900,
                  "cores": 2
            }
      }
}
```

Properties from the root-level JSON object can be deserialized directly into primitive columns, and simple collections deserialized into arrays. You can also choose to deserialize objects into maps, so that each property is presented as a key-value pair or as a struct in which each value is a named part of a known structure.

How you map the columns depends on how the data will be used. In this example, I use nested structs for the **device** and **cpu** objects, and I use an array for the **firmwareVersions** property. We can fetch entire JSON objects from Hive, or we can query on properties—as in Code Listing 34.

Code Listing 34: Querying the JSON

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-----------------------------------------------------------------------
|
{"deviceclass":"tablet","codename":"jericho","firmwareversions":["1.0.0","1
.0.1"],"cpu":{"speed":900,"cores":2}}  |
|
{"deviceclass":"phone","codename":"discus","firmwareversions":["1.4.1","1.5
.2"],"cpu":{"spe                                                  |
+-----------------------------------------------------------------------

> select * from devices where device.cpu.speed > 1000;

+-----------------------------------------------------------------------
|
{"deviceclass":"phone","codename":"discus","firmwareversions":["1.4.1","1.5
.2"],"cpu":{"speed":1200,"cores":4}}  |
+-----------------------------------------------------------------------
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Note: For files with a custom SerDe, the `stored as` and `terminated by` clauses are not needed, because the SerDe will expect a known format. In this case, the JSON SerDe expects text files with one JSON object per line, which is a common Hadoop format.

Summary

Presenting an SQL-like interface over unstructured data in HDFS is one of the key features of Hive. In this chapter we've seen how we can define an external table in Hive, where the underlying data exists in a folder in HDFS. That folder can contain terabytes of data split across thousands of files, and the batch nature of Hive will allow you to query them all.

Hive supports a variety of file formats, including standard text files and more efficient columnar file types such as Parquet and ORC. When we define an external table, we specify how the rows and columns are mapped, and at runtime Hive uses the relevant deserializer to read the data. Custom serialization is supported with a plug-in SerDe framework.

Tables defined in Hive have a fixed schema with known columns of fixed data types. The usual primitive data types we might find in a relational database are supported, but Hive also provides collection data types. Columns in Hive can contain arrays, structs, or maps, which allows us to surface complex data in Hive and query it using typical SQL syntax.

The separation between table structure and the underlying storage handler that reads and writes data means that Hive queries look the same whether they run over internal Hive tables or ch ex

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Chapter 5 External Tables Over HBase

APPROVED
By Guest at 3/28/2024 1:21:46 PM
CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM
NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

HBase

HBase is a Big Data storage technology that provides real-time access to huge quantities of data. We won't go into much HBase detail here, but Syncfusion has it covered with another free eBook in their Succinctly series—**HBase Succinctly** (also written by me).

The architecture of HBase allows for very fast read cells by row key and columns, but its semistructured nature means that cells can be difficult to work with. By mapping HBase tables as Hive tables, you get all the benefits of a fixed structure, which you can query with HiveQL, along with all the speed benefits of HBase.

INITIAL HERE
SIGN HERE

In HBase, data is stored as rows inside tables. All rows have a row key as unique identifier, and all tables have one or more column families specified. Column families are dynamic structures that can contain different columns for different rows in the same table.

Hive tables can be exposed to HBase tables, with each column within a column family mapped to a column represented as key-value pairs. As with HDFS, Hive doesn't import any data from HBase, so that when you query an HBase table with Hive, it will be executed as a map/reduce job, and it will execute tasks using the HBase Java API.

APPROVED
COMPLETED
CONFIDENTIAL

Combining HBase and Hive offers major advantages over using HBase alone. HBase doesn't provide indexes—you will need to query tables by their row key, which can be a slow process. Using Hive, however, you can create an index on a column in an HBase table, which means you can efficiently query HBase on field values other than the row key.

Defining external tables over HBase tables

In Hive, tables using HBase as storage are defined as external tables, and they use the same command syntax as HDFS-stored tables. There's no need to specify a data format or SerDe with HBase, because Hive uses the HBase API to access data and the internal data format need not be known.

HBase tables must be declared using a specific storage engine in the **`stored by`** clause that includes properties to identify the HBase table name. Code Listing 35 shows a **`create table`** statement for accessing data in the HBase table called **`device-events`**.

Code Listing 35: Mapping an HBase Table in Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,cf1:data')

TBLPROPERTIES ('hbase.table.name' = 'device-events');
```

We are only mapping the row key from the HBase table with this simple statement, along with one column from one column is se tables by the storage handler, and ata colun amily (this is specific to my table).

INITIAL HERE

SIGN HERE

For external HBase tables, here are the clauses you need to specify:

- STORED BY—this will be the fixed value **`org.apache.hadoop.hive.hbase.HBaseStorageHandler`** for all HBase tables. The of will HBase ve cha

APPROVED

COMPLETED

CONFIDENTIAL

- WITH SERDEPROPERTIES—the source columns from HBase are specified in the **`hbase.columns.mapping`** property. They are positional, so that the first column in the table definition is mapped to the first column in the property list.

- TBLPROPERTIES—as a minimum, provide the source table name in the **`hbase.table.name`** property. schema name.

HBase stores all data as byte arrays, ponsibility—in this case, the Hive storage handler's responsibility—to d e relevant format. When you declare data types for columns in Hiv encoded byte array in HBase can be decoded to the type you specify. If e data, it will return NULLs.

Mapping columns and column families

In order to minimize storage and maximize access performance, tables in HBase typically include just one or two column families with very short names. With Hive we can map individual columns within column families as primitive data types or map the entire column family as a MAP.

I use two column families in my **`device-events`** table in HBase—**`e`** for the event data and **`m`** for the metadata properties. If I want to expose that table through Hive, with specific event columns and all the metadata columns, I can use the **`with serdeproperties`** clause, as shown in Code Listing 36.

Code Listing 36: Mapping Specific HBase Columns

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,e:n,e:t,e:p,m:')

TBLPROPERTIES ('hbase.table.name' = 'device-events');
```

INITIAL HERE

SIGN HERE

Column mappings are separated s columns are named using the **{co name}** sy es are named using the **{column family}:** syntax.

Table 1 shows the HBase source for each of the Hive columns.

Table 1: Table Structure in HBase

APPROVED

COMPLETED

CONFIDENTIAL

Hive Column	HBase Column Fa	HBase Column	Notes
rowkey			Built-in :key property
eventName	e	n	
timestamp	e	t	
payload	e	p	
metadata	m		Whole family

With this mapping, we can read HBas e structured format, and we can use higher-level HiveQL functionality ation from the data.

Code Listing 37 shows how the raw d le using the HBase Shell to read all the cells with row key **rk1** in the table **device-events**.

Code Listing 37: Reading Data in HBase

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
 e:n                                  timestamp=1453966490613, value=power.on
 e:p                                  timestamp=1453966518206,
value={"some":"json"}
 e:t                                  timestamp=1453966495748,
value=1453562878
 m:d                                  timestamp=[illegible]lue=device-id
 m:u                                  timestamp=[illegible]alue=elton
 m:v                                  timestamp=1453966547593, value=1.0.0
```

INITIAL HERE

SIGN HERE

Note that the timestamps shown are internal fields in which HBase records the last modification time of the cell value. The cell values are all stored as strings, which simplifies interop between HBase and other tools.

C[illegible]t same[illegible]e.

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 38: Reading HBase Data in Hive

```
> select * from device_events where rowkey='rk1';

+-----------------------+-------------------------+-----------------------
| device_events.rowkey  | dev[illegible]ne  |
device_events.receivedat  | d[illegible]ad  |
device_events.metadata
+-----------------------+----[illegible]----+-----------------------
| rk1                   | pow[illegible]    | 1453562878
| {"some":"json"}        | {"d":"device-id","u":"elton","v":"1.0.0"}  |
+-----------------------+-------------------------+-----------------------
```

Code Listing 39 depicts how we can use basic HiveQL to make more sense of the data—in this case showing the UNIX timestamp in the **`receivedAt`** field as a date and extracting the user name from the metadata map.

Code Listing 39: Formatting HBase Data in Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
| eventname  |          _c1          |     payload      |  _c3   |
+------------+-----------------------+------------------+--------+--+
| power.on   | 2016-01-23 15:27:58   | {"some":"json"}  | elton  |
+------------+-----------------------+------------------+--------+--+
```

INITIAL HERE

SIGN HERE

Converting data types

Because HBase stores all dat can arise if you use HBase in a multiplatf e s acrifice some storage performance in order to support interoperable data and store all HBase data as standard UTF-8 encoded strings.

COMPLETED

CONFIDENTIAL

APPROVED

We'll in Chapter 9 Querying with HiveQL, but here it is useful to know that the **`cast`** function converts between primitive data types and that we can include data conversion in a view in order to give more logical access to HBase data.

Direct access to rows always comes via the row key in HBase, and the keys are often cryptic combinations of multiple values. We can use string functions from HiveQL to split the row key into component parts, surfacing them as typed columns in a view and making the data much more readable.

Code Listing 40 shows a sample cell e table where all values are being stored as encoded strings.

Code Listing 40: Querying One Cell in HBase

```
hbase(main):002:0> get 'device-events', 'uuid|20160128', 'e:n'

COLUMN                              CELL

 e:n                                timestamp=1454002528064, value=power.off
```

The row key is constructed from a device ID (which would be a real UUID in the actual database) and a date period separated by the pipe character. A common issue in HBase is that the row key design must support the primary data access vector (in this case the device ID), and if you want to query by a secondary vector (the date period), you must enact a full table scan.

Code Listing 41 shows how to create a view over the Hive table that will, by splitting the row key into two parts, expose the HBase data in a more useful way.

Code Listing 41: Splitting HBase Row Keys with Hive Views

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
receivedat FROM device_events;
```

Tip: In this example, the view maintains the full row key as a separate column. That's a good pr ults cont you can use to query HBase dir the sour

INITIAL HERE

SIGN HERE

Now we can search for rows in a given period with a clear and logical HiveQL statement, as in Code Listing 42.

Code Listing 42: Querying HBase by Partial Row Key

APPROVED

COMPLETED

CONFIDENTIAL

```
1:                1:                 de                  here
sub               20
+-----------------------------+-------------------------------+----------
| device_events_period.rowkey  | device_events_period.deviceid  |
device_events_period.period  | device_events_period.eventname  |
device_events_period.receivedat  |
+-----------------------------+-------------------------------+----------
| uuid|20160128                                                | 20160128
| power.off                                                        |
+-----------------------------+-------------------------------+----------
```

Here we use the **`substr`** function to compare the first six characters of the string field with the literal '201601,' so we return rows in which the date comes in January 2016. That simple query will read a subset of the HBase data by matching a portion in the middle of the row key—something you cannot do with HBase alone.

Hive's inability to create indexes over views, which would let us create a secondary index for HBase tables by using the parts of the row key, is a feature currently missing from the component. We also can't include functions in the **`create index`** statement; if we want a secondary index over the parts of the row key, we'll need an ETL process (which we'll see in Chapter 6 ETL with Hive).

Bad and missing data

Because column families are dynamic in HBase, the mappings we define in Hive might not apply to every row. In fact, rows might have no cells for specific columns we expect to find, or the data in mapped cells might not be in the expected format.

Hive takes the same optimistic approach for HBase that it takes with other sources. When rows do not contain the expected data, it will map any columns that are correct, and return NULL for any columns that it cannot map. Even if the majority of columns can't be mapped, Hive will return a row with a majority of NULLs rather than no row at all.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Row	Row Key	Event Name (e:n)	Valid in HBase	Valid in Hive
✔	uu			Yes
2	uuid2	power.off	Yes	No

INITIAL HERE

SIGN HERE

The first row includes the row key in the valid format but no data in the **`eventName`** (**`e:n`**) column. The second row has data in the **`eventName`** column but an unexpected row key format. Both are valid in HBase, which doesn't require rows to have columns or to adhere to an explicit row k

APPROVED

COMPLETED

CONFIDENTIAL

Hive does what it can with that data. If we explicitly map a column that doesn't exist, Hive won't include any rows in the response that do not have that column. However, if we exclude that column from the query, the rows with missing values do get returned, as we see in Code Listing 43, in which the **`rowkey`** for uuid3 is seen in the first set of results, but not the second.

Code Listing 43: Querying Unexpected HBase Data in Hive

```
                                                  fr
                                                  -
                                                  pa
                                                  -
| uuid2            | {"other":"json"}   |
| uuid3|20160128   | {"more":"json"}    |
| uuid|20160128    | {"other":"json"}   |
+-----------------+--------------------+--+
3 rows selected (0.22 seconds)
select rowkey,                                    e_events
+----------------+------------+--+
|     rowkey     | eventname  |
+----------------+------------+--+
| uuid2          | power.off  |
| uuid|20160128  | power.off  |
+----------------+------------+--+
```

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Swal Hiv data in the source, but it also means that long-running queries won't be broken by bad data.

Connecting Hive to HBase

Hive can run on an HBase cluster or cluster configured to talk to HBase. Most people choose an option depen s—if you share hardware, you'll be competing for resources.

If you'll be running Hive loads at the s ase usage, separate clusters is the way to go. But if your Hive queries are overnight jobs and HBase is used during the day, sharing a cluster works fine.

Hive comes packaged with the HBase storage handler, and, provided all the HBase libraries are available, you need only to configure Hive with the address of the HBase Zookeeper quorum.

When you query HBase data from Hive, the Hive compiler defers to the HBase storage handler in order to generate the data access jobs. In this case, Hive will generate Java code to query HBase using the native Java API, and it will use the configured Zookeeper quorum to find the address of the HMaster and HRegion nodes.

HBase is designed for real-time access, and queries are typically executed very quickly. For optimum performance, your tables should be structured so that the storage handler can query HBase by scanning a subset of rows rather than the entire table (that's something we don't have space for here, but if you're interested, the area to research is "filter push-down").

Configuring Hive to connect to HBase

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 44 shows a snippet from **`hive-site.xml`** that contains the HBase ZooKeeper quorum addresses—typically multiple servers (here named **`zk1`** to **`zk3`**), and they should all be accessible to Hive via DNS or using fully qualified domain names.

INITIAL HERE

SIGN HERE

```
<property>
      <name>hbase.zookeeper.quorum</name>
      <value>zk1,zk2,zk3</value>
</property>
```

Hive ships with the HBase storage handler and the Zookeeper library, but you will need to add the c

APPROVED

COMPLETED

CONFIDENTIAL

HBas

Hive uses the HBase libraries at two points in the runtime—on the server when a query is compiled, in order to build the map/reduce job, and also on each data node in the cluster when the job runs. Hive supports shipping dependencies to data nodes by making the libraries available in HDFS and listing all the dependencies in config. Code Listing 45 shows a sample of the HBase libraries configured in the **`hive-succinctly`** Docker image.

```
<property>
      <name>hive.aux.jars.pat
      <value>hdfs://localhost                    base-server-
1.1.2.jar,hdfs://localhost:9000/hive/auxlib/protobuf-java-2.5.0.jar,
      ...</value>
</property>
```

Tip: You can get the correct list of HBase dependencies by logging onto the HBase Master node and running the command line `hbase mapredcp | tr ':' '\n'`. You can download the HBase tarball, extract the files in the list, put them into HDFS, and add them to the `hive-site.xml` config.

Hive, HBase and Docker Compose

In order to try out connecting Hive to a remote HBase server, we can use two Docker containers and configure them using Docker Compose, which is an extension to Docker used for specifying groups of containers that run together as a logical unit.

The easiest way to accomplish that is to use my **`hbase-succinctly`** and **`hive-succinctly`** images on the Docker Hub, which you can connect together using the Docker Compose YAML file on my GitHub repository with the code for this course.

Docker Compose uses a very simple syntax in the YAML file. The HBase container exposes all the ports Hive needs and is given the hostname **`hbase`**. The Hive container is linked to the

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

In the **`hive-succinctly`** container image, the configuration contains the HBase Zookeeper quorum address (set t brary folde ill already have all th additional librarie which m to configure anything yourself.

INITIAL HERE

SIGN HERE

You will need to install Docker Compose as well as Docker, then download **`docker-compose.yml`** and navigate to the directory where you saved it. Code Listing 46 shows how to control the lifecycle of the containers through Docker Compose.

Code Listing 46: Starting and Stopping Containers

APPROVED

COMPLETED

CONFIDENTIAL

```
docker-compose up -d

docker-compose stop

docker-compose start

docker-compose kill

docker-compose rm
```

Here are the definitions of Code Listir

- **`Up`**—get the latest images, then create, configure and start containers.
- **`Stop`**—stop the containers running, but leave them in-place with their state saved.
- **`Start`**—start (or restart) the saved containers.
- **`Kill`**—stop the containers and destroy their state.
- **`Rm`**—remove the containers.

Both containers are already set up with the sample data, tables, and views from this chapter, so that you can connect to the Hive container, run Beeline, and begin querying data that lives in the HBase container, which will be running in the background.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
docker-compose start.
```

When you run HBase queries in Hive, the compiler builds a job that uses the HBase API, then Hive fetches the data from the HBase server(s) and maps it to the table format. HBase owns reading (and writing) the data, and Hive owns query preparation and data translation.

INITIAL HERE

SIGN HERE

Summary

Hive works well with HBase, taking advantage of real-time data access for fast querying and adding structured layers on top of the semistructured data in HBase.

The nature of HBase storage means that column and family names are typically very short (one chara to the often crypt da ive structured, typed columns, adding a layer of meaning and making queries easier to comprehend.

APPROVED

COMPLETED

CONFIDENTIAL

As with HDFS, Hive presents a consistent SQL interface for HBase, so that users need not have programming knowledge in order to query (HBase provides multiple server interfaces, but they all require programming).

We've now seen how to define Hive t rage, external HDFS files, and HBase tables. External tables are a p hen you want to use existing data in a more accessible way, but the full ality is only available for internal tables.

When you need the more advanced functionality that comes with internal tables, you can still use HDFS and HBase files as the source and load them into Hive tables using ETL functions, which we'll cover in the next chapter.

Chapter 6 ETL with Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

When you have existing data you want to load into Hive without creating a link to the source data by using an external table, Hive offers many options. In fact, Hive has several commands to support ETL, all of which result in populating an internal Hive table.

You can populate Hive from HBase into parts for indexing, or you can INITIAL HERE or the load SIGN HERE of the load, you can transform the data into more usable representations, or you can strip out only the parts you need.

Hive offers multiple options for ETL, but all of the processing is done through map/reduce jobs, which means loading data of any size is possible. Hive is particularly well suited for the alternative data input approach ETL because it supports very efficient loading of data in the native format, which can then be transformed using Hive and

APPROVED COMPLETED CONFIDENTIAL

In this major data loading into Hive in the format of your choosing.

Loading from files

The simplest ETL tool is the **`load`** command. It loads a file or set of files into an existing internal Hive table. Running **`load`** when the data in the source files matches the target table schema, because it does not include any transformations in this option.

You should also know that **`the load`** statement will not do any verification, which means you can load files with the wrong format into a table and the command will run, but you won't be able to read the data back again.

With **`load`** you can specify the source file(s) using either a local filepath or an HDFS path. Code Listing 47 shows an example that loads a syslog file from the local **`/tmp`** directory into the **`syslogs_flat`** table and then fetches the first row.

Code Listing 47: Loading Data into Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
INFO  : Loading data to table default.syslogs_flat from
file:/tmp/syslogs/syslog
INFO  : Table default.syslogs_flat stats: [numFiles=1, totalSize=462587]
No rows affected (0.153 seconds)
```

INITIAL HERE

SIGN HERE

```
> select * from syslogs_flat limit 1;
+-----------------------------------------------------------------------+-
|                           syslogs_flat.entry                          |
+-----------------------------------------------------------------------+-
| Jan 28 20:35:00 sc-ub-xps thermald[785]: Dropped below poll threshold  |
+-----------------------------------------------------------------------+-
-+
```

APPROVED

COMPLETED

CONFIDENTIAL

The **`load`** command has two qualifiers for changing its behavior:

- LOCAL—specifies that the source filepath is on the local machine. If omitted, the filepath is assumed to be HDFS.
- OVERWRITE—deletes the ex ble before loading the new files. If omitted, the new data is appe

Code Listing 48 shows an alternative and—appending the existing data in the **`syslogs_flat`** table from a file efore and after the load in order to show the number of rows in the table.

Code Listing 48: Appending Data to Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
s:
| _c0  |
+-------+--+
| 3942  |
+-------+--+
1 row selected (16.474 seconds)
> load data inpa 9000/tmp able
syslogs_flat;

INFO  : Loading data to table default.syslogs_flat from
hdfs://localhost:9000/tmp/syslog.1
INFO  : Table default.syslogs_flat stats: [numFiles=2, totalSize=1753418]
No rows affected (0.195 seconds)
> s sys
+--
|  _c0   |
+--------+--+
| 15642  |
+--------+--+
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Note: the source file path is RI here in order to show that a remote HDFS cluster can be us specify relative or absolute paths for files in the home HDFS clus

The **`load`** command can only be used with internal tables because its single function is to copy files from the specified source to the underlying folder in HDFS for the Hive table. Code Listing 49 shows the contents of the table folder after the two preceding **`load`** operations.

Code Listing 49: Listing Files Loaded into Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
-rwxrwxr-x   1 root supergroup     462587 2016-02-02 07:28
/user/hive/warehouse/syslogs_flat/syslog

-rwxrwxr-x   1 root supergroup    1290831 2016-02-02 07:32
/user/hive/warehouse/syslogs_flat/syslog.1
```

INITIAL HERE

SIGN HERE

With the **`overwrite`** flag, Hive deletes the existing files before copying the new source files. Code Listing 50 shows the result of overwriting the table with a new file and also shows the HDFS file listing after the load.

Code Listing 50: Load with Overwrite

APPROVED

COMPLETED

CONFIDENTIAL

```
> l
table syslogs_flat;

INFO  : Loading data to table default.syslogs_flat from
hdfs://localhost:9000/tmp/syslog.2.gz
INFO  : Table default.syslogs_flat stats: [numFiles=1, numRows=0,
totalSize=253984, rawDataSize=0]
No rows affected (0.154 secon

> select count(*) from syslog

+--------+--+
|  _c0   |
+--------+--+
| 15695  |

…

root@hive:/hive-setup# hdfs dfs -ls /user/hive/warehouse/syslogs_flat

Found 1 items
-rwxrwxr-x   1 root supergroup     253984 2016-02-02 07:44
/user/hive/warehouse/syslogs_flat/syslog.2.gz
```

`The load` statement is a powerful and quick way of loading data into Hive because it simply does an HDFS put to copy the files from the source into the Hive table structure. However, its use is limited to cases in which the source files are in the correct format for the table and no transformation can occur as part of the load.

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

functions effectively to back up a table on one Hive instance and recreate it on another.

We can see the limitations of having untransformed data in the **`syslogs_flat`** table. For example, syslog files from a Linux machine can be easily loaded, but the format is not easily mapped in Hive. Each entry uses space separated fields for date, machine name, and event type, along with a co ata can't rt of a load, so I have a Hive table with a single string column in which each row contains all the log entry data in one string value.

INITIAL HERE

SIGN HERE

The rows in the **`syslogs_flat`** table aren't well suited for querying if we use **`load`**, but now that the data is in Hive we can use other options to transform the data into a more usable format and load it into other tables.

This l, tr in nto Hive in its st, ans ed as map/reduce jobs by Hive.

APPROVED

COMPLETED

CONFIDENTIAL

Inserting from query results

Hive supports loading a table with the ther objects, and you can include transformation functions in the query. internal or external tables this way, with the query parser doing some bas ensure that the columns from the source query will fit into the target tab

Columns are positionally matched be t, which means you need to craft your **`select`** statement so that the columns in the result set are in the same order as the columns defined in the target table.

If there are too many or too few columns in the query, Hive won't try to match them to the target table and you'll get an error, as in Code Listing 51.

Code Listing 51: Invalid Insert Statement

```
> insert into table server_logs select 's1', 123L, 'E' as loglevel from
dual;

Error: Error while compiling statement: FAILED: SemanticException [Error
10044]: Line 1:18 Cannot insert into target table because column
number/types are different 'server_logs': Table insclause-0 has 4 columns,
but query has 3 columns. (state=42000,code=10044)
```

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

databases), and then you can use** `dual` **in the** `from` **clause. You will find that** `insert into [table] select 'a', 'b', 3` **will fail, but** `insert into [table] select 'a', 'b', 3 from dual` **will succeed.

Hive doesn't verify the data types of the column, which means you can load data into the wrong columns without any errors. The type mismatch will manifest itself when you attempt to query the data and Hive ca the resul

INITIAL HERE

SIGN HERE

The query clause can contain any valid HiveQL, including joins and unions between internal and external tables, function calls, and aggregation, which allows for a lot of extract and transform logic. In order to make a set of syslog files more usable, I've defined a new table with separate columns for the data items, as in Code Listing 52.

Code Listing 52: A Structured Table for Syslogs

APPROVED

COMPLETED

CONFIDENTIAL

```
> d

+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| loggedat  | timestamp  |          |
| host      | string     |          |
| process   | string     |          |
| pid       | int        |          |
| message   | string     |          |
+-----------+------------+---
```

We'll see more HiveQL functions in Chapter 9 Querying with HiveQL, and there is a useful string function called **`sentences`** that takes an input string and returns it tokenized as an array of sentences, each containing an array of words. I can use that to pull out specific words from the log entry, as in Code Listing 53, in which I also cast strings to other types.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
cast(sentences(entry)[0][7] as int) as pid from syslogs_flat limit 5;

+------------+-----------------------------------+-------+--+
|    host    |              process              |  pid  |
+------------+-----------------------------------+-------+--+
| sc-ub-xps  | a
| sc-ub-xps  | anacron                           |       |
| sc-ub-xps  | org.gnome.zeitgeist.SimpleIndexer | 1395  |
| sc-ub-xps  | systemd-timesyncd                 | 625   |
| sc-ub-xps  | systemd                           | 1293  |
+------------+-----------------------------------+-------+--+
```

INITIAL HERE

SIGN HERE

The t APPROVED is COMPLETED bec CONFIDENTIAL ord the year in the log, but if we assume the current year we can use a combination of string and date functions to prepend the year to the date and time in the entry, then convert it all to a timestamp, as in Code Listing 53.

Code Listing 54: Transforming Strings to Timestamps

```
> select unix_timestamp(conca                    _date) as string), ' ',
substr(entry, 0, 15)), 'yyyy                     from syslogs_flat limit 1;

+-------------+--+
|     _c0     |
+-------------+--+
| 1453755712  |
+-------------+--+
```

The final column is the log message, which is a straightforward substring after the closing square bracket from the process ID, as in Code Listing 55.

Code Listing 55: Extracting Substrings

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
> select trim(substr(entry, instr(entry, ']')+2)) from syslogs_flat limit
--
+-----------------------------+--+
| Job 'cron.daily' terminated  |
+-----------------------------+--+
```

INITIAL HERE

SIGN HERE

Putting it all together, **syslogs** **syslogs_flat** table with a single **insert … select**. The query aspect is clunky because of the nature of the input data, but once it runs we can make much more useful selections over the formatted **syslogs** table, as in Code Listing 56.

Code Listing 56: Transforming Raw Data

APPROVED

COMPLETED

CONFIDENTIAL

```
> insert into syslogs select unix_timestamp(concat(cast(year(current_date)
as  en  MMM
sen  se  [0][7]
as int), trim(substr(entry, instr(entry, ']')+2)) from syslogs_flat;

...
No rows affected (16.757 seconds)

> select process, count(proce  syslogs where host = 'sc-
ub-xps'  group by process ord  limit 5;

…

+-----------------+----------
|     process     | entries  |
+-----------------+----------+--+
| kernel          | 7862     |
| thermald        | 2906     |
| systemd         | 1450     |
| NetworkManager  | 1245     |
| avahi-daemon    | 310      |
+-----------------+----------+--+
```

Inserting query results to tables also supports the **overwrite** clause, which effectively truncates the target table before inserting the results of the query.

Multiple inserts

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Multiple inserts are useful when you need to populate different Hive projections from a single source, because multiple inserts run very efficiently. Hive will scan the source data once, then run each query over the scanned data, inserting it into the target.

In Code Listing 57 we use the unformatted **`syslogs`** table as the source and load the formatted table and a summary

INITIAL HERE

SIGN HERE

Code Listing 57: Multiple Inserts

APPROVED

COMPLETED

CONFIDENTIAL

```
from syslogs_flat sf

insert overwrite table syslogs select
unix_timestamp(concat(cast(year(current_date) as string), ' ',
sub                                                    s')
sen                                                    [0]
cast(sentences(sf.entry)[0][7] as int), trim(substr(sf.entry,
instr(sf.entry, ']')+2))

insert overwrite table syslog_summaries select unix_timestamp(),
sentences(sf.entry)[0][5] as host, count(sentences(sf.entry)[0][5]) as
entries group by sentences(sf.entry)[0][5]

…

> select * from syslog_summar

+-----------------------------                          ----------+-----------------
| syslog_summaries.processedat  | syslog_summaries.host  |
syslog_summaries.entries  |
+-------------------------------+------------------------+-----------------
| 2016-02-02 20:00:48.062       | sc-ub-xps              | 15695
|
+-------------------------------+------------------------+-----------------
```

Create table as select

A final variation on inserting data from query results is **`create table as select`** (CTAS), which lets us define a table and populate it with a single statement. The table can only be internal, but we can specify the normal **`stored by`** clauses for internal tables.

In a CTAS table, the structure is inferred from the query, which means we don't need to explicitly specify the columns. Queries should cast values into the data types we want for the

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 58 shows a CTAS statement for loading the individual words from syslog entries into a new table. The select statement parses the log message as sentences and extracts the first sentence as an array of strings, which is how Hive defines the column.

Code Listing 58: Create Table as Select

INITIAL HERE

SIGN HERE

```
> create table syslog_sentences stored as orc as select
sentences(trim(substr(entry, instr(entry, ']')+2)))[0] words from
syslogs_flat;

...

No rows affected (12.758 seconds)

> d[illegible]ce[illegible]

+-----------+----------------+----------+--+
| col_name  |   data_type    | comment  |
+-----------+----------------+----------+--+
| words     | array<string>  |          |
+-----------+----------------
1 row selected (0.064 seconds

> select * from syslog_senten[illegible]

+-----------------------------------+--+
|       syslog_sentences.words      |
+-----------------------------------+--+
| ["Job","cron.daily","terminated"] |
| ["Normal","exit","1","job","run"] |
+-----------------------------------+--+
```

APPROVED

COMPLETED

CONFIDENTIAL

Temporary tables

Hive supports temporary tables, which are very useful for interim data transformations in ETL/ELT workloads that cannot achieve the full transform in a single step. Temporary tables don't support indexes, and they exist only for the life of the Hive session—they are automatically deleted when the session ends.

Temporary tables are internal tables stored for the user in the working directory within HDFS. They can be specified with a supported file format, so that you benefit from efficient storage

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 59: Using Temporary Tables

```
> create temporary table etl_progress(status string, stage string,
processedat timestamp, rowcount bigint) stored as orc;

No rows affected

> insert into etl_progress(status, stage, processedat, rowcount)
values('Done', 'Transform.1', '2016-02-02 07:03:01', 328648);

No rows affected (14.853 seconds)

> select * from etl_progress;

+----------------------+----------------------+--------------------------+-
| etl_progress.status  | etl_progress.stage   | etl_progress.processedat  |
etl_progress.rowcount  |
+----------------------+----------------------+--------------------------+-
| Done                 | Transform.1          | 2016-02-02 07:03:01.0     |
328648                 |
+----------------------+----------------------+--------------------------+-
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

In this case the **insert** statement us e Hive doesn't support using functions in the values clause for inse able. If you try using a built-in function, such as **unix_timestamp**, i nt time, you'll receive an error.

However, you can use functions in a select clause, which means you can use the same trick of selecting literals from **dual**, as shown in Code Listing 60.

Code Listing 60: Inserting from Functions

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
10293]: Unable to create temp file for insert values Expression of type
TOK_FUNCTION not supported in insert/values (state=42000,code=10293)

> insert into etl_progress select 'Done', 'Transform.2', unix_timestamp(),
12435358 from dual;
...

No rows affected (12.437 seconds)
```

INITIAL HERE

SIGN HERE

Whether the session is interactive with Beeline or a job submitted through an external interface, Hive deletes the table at the end of the session. The temporary table is visible only in the sessi

APPROVED

COMPLETED

CONFIDENTIAL

When the session that created the temporary table ends, the data and metadata for the table are removed, and when the next session starts the table will be gone, as we see in Code Listing 61.

Code Listing 61: Temporary Tables Being Removed

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
| etl_progress.status  | etl_progress.stage   | etl_progress.processedat  |
etl_progress.rowcount  |
+----------------------+----------------------+---------------------------+-
| Done                 | Transform.1          | 2016-02-02 07:03:01.0     |
328648                 |
| Done                 |                      |                 01:23.674   |
12435358               |
+----------------------+----------------------+---------------------------+-
2 rows selected (0.091 seconds)

> !close

Closing: 0: jdbc:hive2://127.0.0.1:10000

bee                    iv                    -n

Connecting to jdbc:hive2://127.0.0.1:10000

…

> select * from etl_progress;

Error: Error while compiling                                  SemanticException [Error
10001]: Line 1:14 Table not f                                  (state=42S02,code=10001)
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

In this example, when the Beeline user disconnects with the **!close** command, the session ends, and the Hive server deletes the temporary table. When the user connects again, a new session begins, and the temporary table will be gone.

Summary

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

For new data loads, an ELT process makes better use of Hive by initially using **`load`** to put the raw data into Hive tables and transforming it using HiveQL. The **`insert … select`** and **`create table … as select`** statements allow you to craft a complex query with functions to transform your data and have it populated in Hive through scalable map/reduce jobs.

INITIAL HERE

SIGN HERE

Once you have the data in Hive, you can create views to make the information approachable. When your next set of data is extracted, simply repeat the inserts and the new data will be appended to the existing tables.

In the next chapter we'll look more closely at defining objects and modifying data using HiveQL, the Hive Query Language.

APPROVED

COMPLETED

CONFIDENTIAL

Chapter 7 DDL and DML in Hive

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

We've looked at some HiveQL queries in previous chapters, and we can see that they are predominantly SQL, including some Hive-specific statements and clauses. HiveQL isn't fully ANSI-SQL compatible (although achieving SQL-92 compatibility is an aim for future releases), but the differences are found around the edges—anyone with SQL experience can easily pick up HiveQL.

INITIAL HERE

SIGN HERE

As with SQL, HiveQL statements either define the structure of the database with Data Definition Language (DDL), change the content of the data queries with Data Modification Language (DML), or read data.

Hive provides only table, view, and index objects, which means there are a limited number of DDL statements, and because its primary function is as a data warehouse, the standard SQL DML statements aren't supported in all cases.

APPROVED

COMPLETED

CONFIDENTIAL

In this [illegible] key [illegible] ing data structures and writing data. We'll also cover all the major statements, but the Language Manual on the Hive Wiki has excellent documentation for all statements, including the versions of Hive that support them.

Data definition

DDL statements are used to define or [illegible] es and database objects. The functionality of HiveQL has evolved w [illegible] means not all statements, and not all clauses, are available in all versior [illegible] cover the most commonly used DDL statements in the current versior [illegible] 1.2.1.

Databases and schemas

All objects in Hive live inside a schema, but you need not specify a particular schema, in fact the default is often used. If you want to segregate your data, you can create different schemas and refer to objects by prefixing the schema name.

The terms **`database`** and **`schema`** are interchangeable in Hive, so the following statements can be used with **`schema`** or **`database`** and work in the same way:

- CREATE SCHEMA [name]—create a new schema.
- USE [name]—switch to the named schema.

Schemas can be created (or altered) using the **`with dbproperties`** clause in order to store a collection of key-value pairs as metadata about the schema. That can be useful for storing a

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
> create database iot with dbproperties('maintainer'='elton@sixeyed.com',
'release'='2016R1');

No rows affected (0.249 seconds)

> describe schem

+----------+----------+-----------------------------------------------
| db_name  | comment  |                    location                    |
owner_name  | owner_type  |                parameters                 |
+----------+----------+-----------------------------------------------
| iot      |          | hdfs://localhost:9000/user/hive/warehouse/iot.db  |
root                                  six                    16R1}
|
+----------+----------+-----------------------------------------------
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Creating database objects

The only database objects in Hive are exes, which means the only **`create`** statements are the ones we already v 1 Introducing Hive:

- CREATE TABLE—create an i
- CREATE EXTERNAL TABLE— ch data is stored outside of Hive.
- CREATE VIEW—create a view over one or more tables.
- CREATE INDEX—create an index over an existing table (internal or external).

Because all of those statements support the **`if not exists`** clause, Hive will only create them if they don't already exist in the database.

Tables are created specifying the column names and data types, the storage engine, and the data location. Internal tables can also be partitioned to improve scalability, which we'll cover more in Chapter 8 Partitioning Data.

Views can be created for any HiveQL query that returns a value, which means you can create views to provide simple access to complex combinations of data using **`join`** or **`union`** constructs as required.

Indexes are created for one or more columns over a single table.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

the data exists, but any column mappings are not verified.

The **`create`** statements are broadly standard SQL, but Hive provides a time-saving option to create tables with the same structure as an existing table, **`create table like`**—as shown in Code Listing 63.

INITIAL HERE

g Tables l

SIGN HERE

```
> describe dual;

+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| c         |            |          |
+--
```

APPROVED

COMPLETED

CONFIDENTIAL

```
> create table dual2 like dual;

> describe dual2;

+-----------+------------+---------+--+
| col_name  | data_type  | co
+-----------+------------+---
| c         | string     |
+-----------+------------+---
```

`The create table like` statement is frequently useful for working with data in temporary tables or moving between internal and external tables in which the structure is the same. It does not copy any data, nor does it link the tables, which means changing the original table structure won't affect the new one.

Modifying database objects

Existing objects can be changed with **`alter`** statements, but typically these affect only the structure of the object in Hive's metadata store and will not change any of the existing data.

For that reason, you must be careful when altering table definitions, as you can easily modify your table and make reading from it impossible. With **`alter table`** you can rename the table, change the file format, and add, remove, or change columns.

To change an existing column, the syntax is **`alter table … change`**, which allows you to change the column name, data type, and order in the table. Code Listing 64 shows an existing

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
> describe syslogs;

+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| loggedat  | timestamp  |          |
| host      | string     |          |
| process   | string     |          |
| pid       | int        |          |
| message   | string     |          |
+-----------+------------+----------+--+

> alter table syslogs change host loggedat string;

> describe syslogs;

+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| host      | string     |          |
| loggedat  | string     |          |
| process   | string     |          |
| pid       | int        |          |
| message   | string     |          |
+-----------+------------+----------+--+
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The change is made successfully, as we can see from the second **`describe`** statement, but Hive hasn't changed the data inside the table. The data files maintain the original structure, so that the first column is a timestamp (the original **`loggedAt`** column), and the second column is a string (the original **`host`** column).

If we try to query this table, Hive reads the timestamp value in the first column, which contains the **`loggedAt`** timestamp, but tries to read it as a string. The formats are not compatible, so we get an error as in Code Listing 65.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 65: Tables with Structural Mismatch

```
> select * from syslogs limit 5;

Error: java.io.IOException:
org.apache.hadoop.hive.ql.metadata.HiveException:
java.lang.ClassC
org.apache.hadoop.hive.serde2.io.TimestampWritable cannot be cast to
org.apache.hadoop.io.Text (state=,code=0)
```

INITIAL HERE

SIGN HERE

We can still access the other columns in the table, but now the data is in the wrong place—the metadata says the **`loggedAt`** column is in the second position, but in the data files that position contains the **`host`** field, as we see in Code Listing 66.

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 66: Wrong

```
0: jdcbc:hive2://127.0.0.1:10000> select loggedat, process, pid, message
from syslogs limit 1;

+------------+----------+------+-----------------------------+--+
|  loggedat  | process  | pid                                age            |
+------------+----------+----                                 ---------------+--+
| sc-ub-xps  | anacron  | 804                                 ' terminated  |
+------------+----------+----                                 ---------------+--+
```

Because **`alter table`** works at the metadata level, it's easy to repair the damage by altering the table back to its original definition. If you do need to change the existing structure of a table (other than adding or renaming columns), a better approach is to define a new table and load it from the existing one, using whichever transforms you need.

With **`alter view`** you can change the select statement that projects the view. You can change the column layout, data types, and order by changing the query, and, provided the HiveQL query is valid, the view will be valid. Views are not materialized in Hive, which means there is no data file sitting behind the view that can get out of sync with the definition.

If the view does not already exist, **`alter view`** will raise an error. Code Listing 67 alters the existing view over the HBase **`device-events`** table, thereby removing the original **`rowkey`** column and adding a clause, so that only rows with a value in the **`period`** column will be returned.

Code Listing 67: Altering Views

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

The **`alter index`** statement only allows you to rebuild an existing index—you can't change the column or table the index uses. Hive doesn't automatically rebuild indexes, which means you will need to manually rebuild using **`alter index`** whenever you modify data in the underlying table.

INITIAL HERE

SIGN HERE

Code Listing 68 rebu terialized able) over the external **`hbase_table`**. Note that the table for the index must be explicitly specified.

Code Listing 68: Rebuilding Indexes

```
> alter index ix_hbase_table_cf1_data on hbase_table rebuild;

...
No rows affected (51.917 seconds)
```

APPROVED

COMPLETED

CONFIDENTIAL

Removing database objects

You can remove objects from Hive wi **op view**, and **`drop index`** statements while optionally using the

When you drop an index, Hive remov he internal table used to store it. Removing indexes has no functional explicitly referenced the internal index table in any queries. Otherwise, plicitly use the index will run more slowly, but they will still produce the same results.

When you drop a view, it gets removed from the database, and any queries using it will fail with a 'table not found' error from Hive, as in Code Listing 69.

Code Listing 69: Dropping Views

```
> drop view device_events_period;

> select * from device_events_period;

Error: Error while compiling statement: FAILED: SemanticException [Error
10001]: Line 1:14 Table not found 'device_events_period'
(state=42S02,code=10001)
```

Because there are no materialized views in Hive, dropping a view will not remove any data.

When you drop a table, the effect on the data varies depending upon the type of table being used. With external tables, Hive doesn't remove the source data, so that if you drop a table

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

scanning the underlying **device-events** table in HBase.

Code Listing 70: Dropping External Tables

```
> drop table device events;
> select * from

Error: Error while compiling statement: FAILED: SemanticException [Error
10001]: Line 1:14 Table not found 'device_events' (state=42S02,code=10001)

…

hbase(main):004:0> scan 'device_events'

ROW
 uuid2                                  column=e:n, timestamp=1454520396475,
value=power.off
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

With internal tables, Hive manages the storage so that when you drop them, all the data will be deleted. With some platform setups, the underlying files may be moved to a recoverable trash folder—but don't depend on it.

To ensure permanent deletion, specif in Code Listing 71, which shows the HDFS file listing before and after table is dropped.

Code Listing 71: Dropping Internal Tables

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
-rwxrwxr-x   1 root supergroup        423 2016-02-02 21:21
/user/hive/warehouse/syslog_summaries/000000_0

..

> drop table syslog_summaries;

…

root@hive:/hive-setup# hdfs dfs -ls /user/hive/warehouse/syslog_summaries

ls: `/user/hive/warehouse/syslog_summaries': No such file or directory
```

INITIAL HERE

SIGN HERE

Hive APPROVED deletes COMPLETED tables CONFIDENTIAL that have been created on the Hive server. If you drop a table referenced by views, the views will remain but will error if you try to query them. When you drop an indexed table, the indexes and the underlying index tables will be silently deleted. If you want to remove the data from an internal Hive table but leave the table in place, the standard SQL **`truncate`** statement deletes all the rows, as in Code Listing 72.

Code Listing 72: Truncating Data

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
+--------+--+
| 15695  |
+--------+--+

> truncate table

No rows affected (0.09 seconds)

> select count(*) from syslogs;

+------+--+
| _c0  |
+--
| 0
+------+--+
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The **`truncate`** statement works by deleting the underlying files in HDFS while leaving the folder structure in place, so that the table can be populated again. Because it is only valid for internal tables, Hive will raise the error 'Cannot truncate non-managed table' if you try to **`truncate`** an external table.

Data manipulation

The full range of DML statements is a and isn't supported by all storage engines. From its origin as a data warehouse, Hive wasn't originally conceived to update or delete existing data; it only supported appending data with **`load`** and **`import`** statements.

Since Hive 0.14, **`update`** and **`delete`** statements have been provided for storage engines that support them. The **`insert`** statement has also been extended in order to allow direct insertion of values, whereas in previous versions we could only **`insert`** the results of a **`select`** query.

ACID storage and Hive transactions

The full set of DML statements is only available on tables that support the ACID properties of typical RDBMS designs. The ACID principles ensure data consistency, so that simultaneous reads and writes to the same table won't cause conflicts.

ACID principles are not inherent in HDFS, which doesn't allow data in files to be changed and doesn't lock files that are being appended. That means you can't update an existing data item in a file, and you can't stop readers from accessing new data as it's being written.

Hive works around the HDFS limitations by creating delta files—the Wiki page on Hive Transactions explains the complexity involved. Currently, only internal tables support Hive transactions, and only if they are s[illegible]iv[illegible]up[illegible] internal table.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

- ORC format storage.
- Bucketed but [illegible]n Chapter [illegible]
- Flagged with the `transactional` property.

INITIAL HERE

SIGN HERE

Summary

Because Hive has a smaller number of objects than SQL databases, so that there are very few DDL [illegible]tal[illegible]e a[illegible]-specific claus[illegible]ool[illegible]and[illegible]

APPROVED

COMPLETED

CONFIDENTIAL

Remembering the disconnect between the object structure, which is stored in the Hive metastore, and the actual structure of data in files, is the key takeaway concerning DDL. Because Hive typically doesn't enforce the structure, if you alter tables the structure and content will be out of sync and the data will be unreachable.

From its origins as an append-or-ove[illegible] Hive has grown to support the majority of SQL DML statements, alb[illegible]of table types. The support for ACID table types and transactions is useful [illegible] Hive is not intended as a transactional database. If you find yo[illegible]ited by the DML support in Hive, you may not be using Hive appropriat[illegible]

In the next chapter, we'll look at one of the major performance-boosting factors in Hive table design—partitioning data across multiple physical stores.

Chapter 8 Partitioning Data

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Splitting a logical database object across multiple physical storage locations is the key to achieving high performance and scalability. The more storage locations, the more compute nodes can concurrently access the files. Intensive jobs can be run with a high degree of parallelism, which means they will run more efficiently and finish more quickly.

Some databases call INITIAL HERE t of the ir SIGN HERE typically comes in the form of greater complexity in accessing the data. In some implementations you must specify which shard to insert or read from, and the administration—such as redistributing data if some shards become overloaded—is not trivial.

Hive supports two different types of sharding for storing internal tables—partitions and buckets. The sharding approach is specified when the table is created, and Hive uses column values to decide which rows go into which shard. This action abstracts some of the complexity from the user.

APPROVED COMPLETED CONFIDENTIAL

Sharding data is a key performance technique, and it is typically used in Hive for all but the smallest tables. We address it later in this book because we now have a good understanding of how Hive logically and physically stores and accesses data. Learning about sharding will be straightforward at this point.

Partitioned tables

Sharding a table into multiple partitior e storage into many folders in HDFS. If you create an internal Hive t the data files will be stored (by default) in HDFS at **`/user/hive/warehouse/[table_name]`**. How you populate that table dictates how many files get created, but they will all be in the same folder.

Figure 6 shows how the **`syslogs`** table can be physically stored when it has been populated.

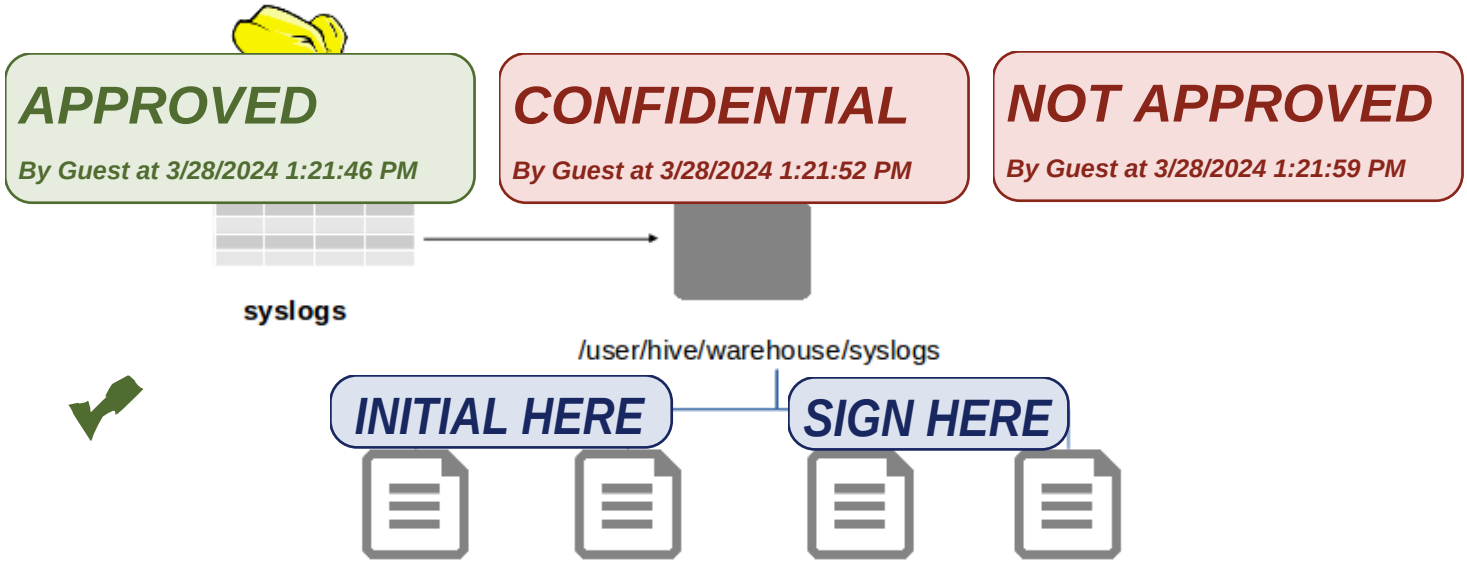

Figure 6: Folder Structure for an Unpartitioned Table

APPROVED COMPLETED CONFIDENTIAL

Wher nte ubf the partition, and the data files reside in the lowest-level subfolder. We specify how to partition the data, and the most efficient partition scheme will reflect the access patterns for the data.

If we typically load and query logs in batches for a particular date and server, we can partition the table by period and server name. As the table gets populated, Hive will create a nested folder structure using **`/user/hive/warehouse/[table_name]`** as the root and **`./[period]/[server]`** for the su .

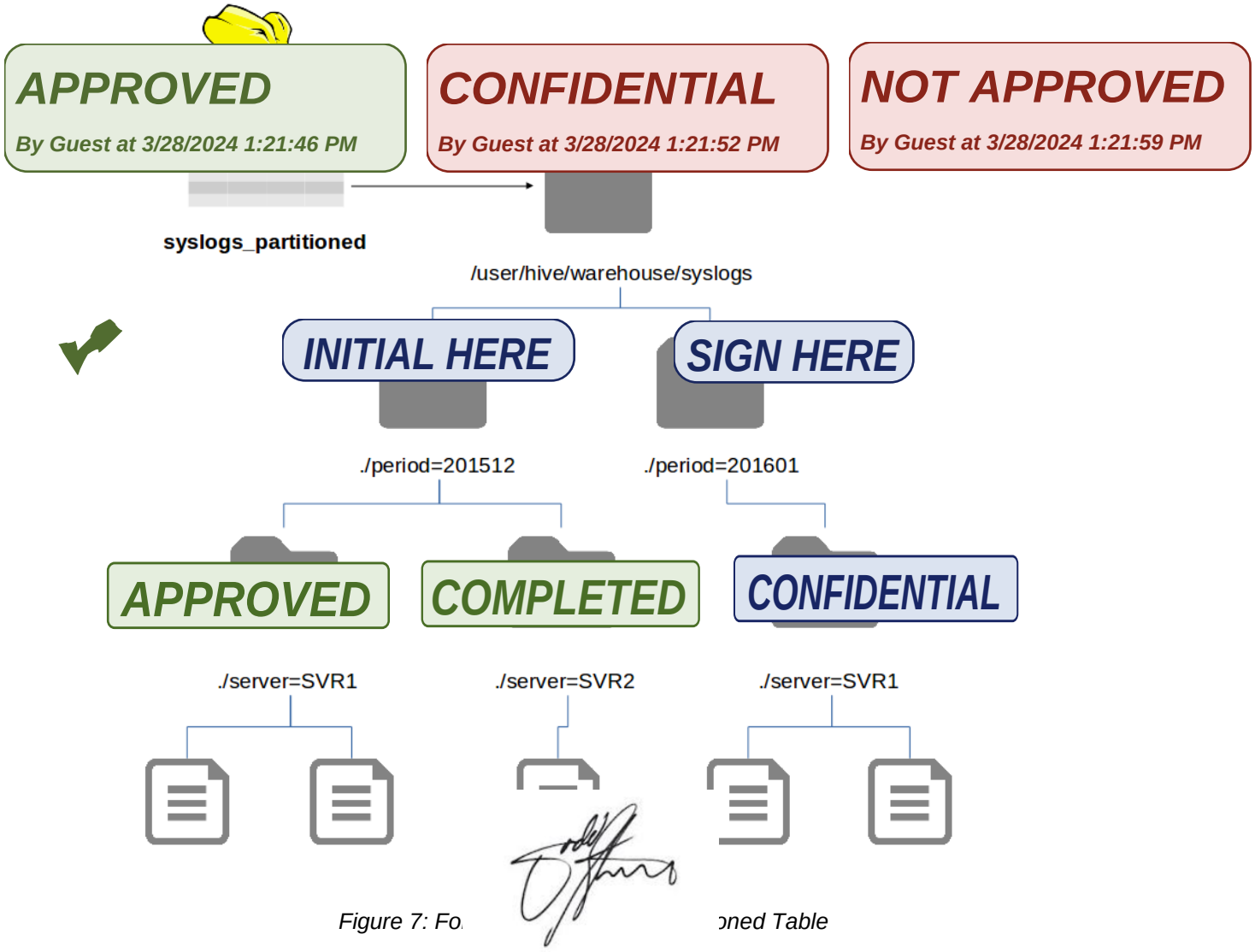

Figure 7: Fo ned Table

Creating partitioned tables

The **`create table`** statement supports a **`partitioned by`** clause in which you specify the columns used to partition the data files. You can specify multiple columns, and each will add another level of nesting to the folder structure.

For comparison, Code Listing 73 shows the structure of a table that is not partitioned, along with the structure of the folders in HDFS.

Code Listing 73: File Listing for an Unpartitioned Table

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-----------+------------+----------+--+
| period    | string     |          |
| host      | string     |          |
| loggedat  | timestamp  |          |
| process   | st
| pid       | in
| message   | string     |          |
+-----------+------------+----------+--+

…

root@28b0162f637b:/hive-setup# hdfs dfs -ls
/us
Found 2 items

-rwxrwxr-x   1 root supergroup        692 2016-02-04 17:52
/user/hive/warehouse/syslogs_no_partitions/000000_0

-rwxrwxr-x   1 root supergroup        697 2016-02-04 17:53
/user/hive/warehouse/syslogs_                  00_0_copy_1
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The two files in this listing contain diff period and host, but because the table is not partitioned, the files are in able, and any file could contain rows with any period and host name.

Code Listing 74 shows how to create a partitioned version of the **`syslogs`** table.

Code Listing 74: Creating a Partitioned Table

```
> create table syslogs_with_partitions(loggedat timestamp, process string,
pid int, message string) partitioned by (period string, host string) stored
as ORC;

No rows affected (0.222 seconds)
```

Note: The columns you use to partition a table are not part of the column list, which means that when you define your table, you specify columns for all the data

fields that are not part of the partitioning scheme, and you will specify partition columns separately.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 75: File Listing for a Partitioned Table

```
root@28b0162f637b:/hive-setup# hdfs dfs -ls
/user/hive/warehouse/syslogs_with_partitions/*/*

Found 1 items
-rwxrwxr-x   1 root supergroup        502 2016-02-04 17:58
/user/hive/warehouse/syslogs_with_partitions/period=201601/host=sc-ub-
xps/000000_0

Found 1 items

-rwxrwxr-x   1 root supergroup        502 2016-02-04 17:58
/user/hive/warehouse/syslogs_with_partitions/period=201601/host=sc-win-
xps/000000_0

Found 1 items

-rwxrwxr-x   1 root supergroup        502 2016-02-04 17:56
/user/hive/warehouse/syslogs_with_partitions/period=201602/host=sc-ub-
xps/000000_0
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Here we have two folders under the r specifying the partition column name and value (**period=201601** an eath those folders we have another folder level that specifies the next par **-ub-xps** and **host=sc-win-xps**).

In the partitioned table, the data files live under the period/host folder, and each file contains only rows for a specific combination of period and host.

Populating partitioned tables

Because rows in a partitioned table need to be located in a specific location, all the data population statements must tell Hive the destination target. This is done with the **partition** clause, which specifies the column name and value for all the rows being loaded.

The data being loaded cannot contain values for the partition columns, which means Hive treats them as a different type of column.

Code Listing 76 shows the description for the partitioned **syslogs** table in which the partition columns are shown as distinct from the data columns.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
|
+-------------------------+----------------------+----------------------
| loggedat                | timestamp            |
| process                 | string               |
| pid                     |                      |
| message                 |                      |
| period                  | string               |
| host                    | string               |
|
| # Partition Information | NULL                 | NULL
| # col_name              | data_type            | comment
| period                  | string               |
| host                    |                      |
+-------------------------+----------------------+----------------------
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

The data for the partition columns is still available to read in the normal way, but it must be written separately from other columns. Code Listing 77 shows how to insert a single row into the partitioned **syslogs** table.

Code Listing [illegible] oned Table

```
> insert into syslogs_with_pa [illegible] (period='201601', host='sc-
win-xps') values('2016-01-04 [illegible] , 123, 'msg2');

No rows affected (11.726 seconds)
```

The data to be inserted is split between clauses:

- PARTITION—contains the column names and values for the partition columns.
- VALUES—contains the values for data columns (names can be omitted, as in this example, which uses positional ordering).

If you try to populate a partitioned table without specifying the correct partition columns, Hive will raise an error.

Tip: This split between partition columns and data columns will add complexity to your data loading, but it ensures that every row goes into a file in the correct folder. If this seems odd at first, it's simply a case of remembering that columns in the

partition clause of the create statement need to go in the partition clause for inserts, and they aren't included in the normal column list.

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 78 populates **`syslogs_partitioned`** by selecting from the **`syslogs`** table.

Code Listing 78: Selecting and Inserting into a Partitioned Table

```
> insert into sy                tition(p                host='sc-ub-
xps') select log                message                re
year(date(loggedat)) = 2016 and month(date(loggedat)) = 01 and host = 'sc-
ub-xps';

…

INFO  : Partition default.syslogs_partitioned{period=201601, host=sc-ub-
xps} stats: [numFiles=2, numRows=3942, totalSize=
raw

No rows affected (13.106 seconds)
```

INITIAL HERE SIGN HERE APPROVED COMPLETED CONFIDENTIAL

Similarly, if the target table is partitioned, the **`load`** statement requires the partition clause. Apart from the partition columns, load works in the same way as we noted in Chapter 6 ETL with Hive—essentially it copies the source file to HDFS, using the partition specification to decide the target folder.

INSERT, UPDATE, DELETE

The key DML statements are only supported for ACID tables, but where they are supported the syntax remains the same as standard SQL. As of release 1.2.1, Hive supports some DML statements for tables that it does not recognize as ACID, as shown in **Error! Reference source not found.**.

Table 3: DML Statement Support

Table Storage	INSERT	UPDATE	DELETE
Internal - ACID	Yes	Yes	Yes
Internal – not ACID	Yes	No	No
External - HDFS	Yes	No	No
External - HBase	Yes	No	No

Code Listing 79 creates a table that supports Hive transactions and specifies the ORC format, bucketed partitioning, and a custom table property in order to identify it as transactional.

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 79: Creating a Table for Transactions

```
(host string, loggedat timestamp, process string, pid int,
 message string, hotspot boolean)
clustered by(host) into 4 buckets
stored as ORC
tblproperties ("transactional" = "true");
```

INITIAL HERE

SIGN HERE

In order to work with [illegible] need to set [illegible]ion values. We can do this in **`hive-site.xml`** by making extensive use of transactional tables, or we can do this per session if we have only transactional tables.

The **hive-succinctly** Docker image is already configured in **`hive-site.xml`** in order to support the transaction manager, and it uses the following settings:

APPROVED

COMPLETED

CONFIDENTIAL

- [illegible]ncy
- "hive.enforce.bucketing" = "true".
- "hive.exec.dynamic.partition.mode" = "nonstrict".
- "hive.txn.manager" = "org.apache.hadoop.hive.ql.lockmgr.DbTxnManager".
- "hive.compactor.initiator.on" =
- "hive.compactor.worker.threa[illegible]

With the transaction settings configur[illegible] statements with the ACID table, they will run under the transaction ma[illegible]ble to insert, update, and delete data. Code Listing 80: Inserting to an ACID TableCode Listing 80 shows the insertion of all the syslog data from the existing non-ACID table to the new ACID table.

Code Listing 80: Inserting to an ACID Table

```
> insert into syslogs_acid select host, loggedat, process, pid, message,
false from syslogs;

...

INFO  : Table default.syslogs_acid stats: [numFiles=4, numRows=15695,
totalSize=82283, rawDataSize=0]
```

If we want to modify that data, we can use **`update`** and **`delete`** on the table. As with SQL, Hive accepts a **`where`** clause in order to specify the data to act on. In Hive, the updates will be made in a map/reduce job, so that we can use complex queries and act on large result sets.

In Code Listing 81 we populate the hotspot column to identify processes that do a lot of logging, using a count query.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 81: Identifying Hotspots

```
from syslogs_acid group by process having count(process) > 1000);

> select * from syslogs_acid where hotspot = true limit 1;

+--------------------+-------------------------+----------------------+--
| syslogs_acid.h                loggedat                  .process  |
syslogs_acid.pid                  essage                  otspot  |
+--------------------+-------------------------+----------------------+--
| sc-ub-xps          | 1970-01-17 19:52:02.838  | systemd              | 1
| Started CUPS Scheduler.  | true                  |
+--------------------+-------------------------+----------------------+--
```

INITIAL HERE

SIGN HERE

Now ies ot h g 82.

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 82: Deleting from an ACID Table

```
> delete from syslogs_acid where hotspot = false;

> select count(distinct(process)), count(*) from syslogs_acid;

...

+-----+--------+--+
| c0  |   c1   |
+-----+--------+--+
| 4   | 13463  |
+-----+--------+--+
```

This offers a straightforward way to identify that a minority of processes generates the majority of logs and leaves us with a table that contains the raw data for the main subset of processes.

Querying partitioned tables

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 83 shows a basic select for all the columns in the **`syslogs_partitioned`** table, which will return all the partition columns and all the data columns without distinguishing between them.

Code Listing 83: Selecting from a Partitioned Table

INITIAL HERE

SIGN HERE

```
> select * from                                    limit 2;
+-------------------------------+-------------------------------+-----------
| syslogs_partitioned.loggedat  | syslogs_partitioned.process   |
syslogs_partitioned.pid  |  syslogs_partitioned.message  |
syslogs_partitioned.period  | syslogs_partitioned.host  |
+-------------------------------+-------------------------------+-----------
| 2016-01-17 19:53:33.3         | thermald                      | 785
| D                                   res                                -xps
|
| 2016-01-17 19:53:33.3         | thermald                      | 785
| thd_trip_cdev_state_reset     | 201601                       | sc-ub-xps
|
+-------------------------------+-------------------------------+-----------
```

APPROVED

COMPLETED

CONFIDENTIAL

You can also use a combination of pa   s in the selection criteria, as in Code Listing 84.

Code Listing 8   ition Columns

```
> select host, pid, message f                                    oned where period =
'201601' and host like 'sc-ub%' and process = 'anacron' limit 2;
+------------+------+-----------------------------+--+
|    host    | pid  |           message           |
+------------+------+-----------------------------+--+
| sc-ub-xps  | 781  | Job `cron.daily' terminated |
| sc-ub-xps  | 781  | Job `cron.weekly' started   |
+------------+------+-----------------------------+--+
```

This query makes efficient use of the partitions in order to help distribute the workload. The **`where`** clause specifies a value for the period that will limit the source files to the folder **`period=201601`** under the table root. There's a wildcard selection by **host**, so that all the files in all the **`host=sc-ub*`** folders will be included in the search.

Hive can split this job into multiple map steps, one for each file. Hadoop will run as many of those map steps in parallel as it can, given its cluster capacity. When the map steps are finished, one or more reducer steps collate the interim results, then the query completes.

Using partitioned tables correctly will give you a good performance boost whenever you read or write from the table, but you should think carefully about your partition scheme. If you use a

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

overhead of running very large numbers of small map jobs outweighs the benefit of parallel processing. If you need to support multiple levels of sharding, you can use a combination of partitions and buckets.

Bucketed tab

INITIAL HERE

SIGN HERE

Bucketing is an alternative data-sharding scheme provided by Hive. Tables can be partitioned or bucketed, or they can be partitioned and bucketed. This works differently from partitioning, and the storage structure is well suited to sampling data, which means you can work with small subsets of a large dataset.

With partitions, the partition columns define the folder scheme, and populating data in Hive will

APPROVED

COMPLETED

CONFIDENTIAL

buckets.

Buckets shard data at the file level rather than the folder level, so that if you create a table **`syslogs_bucketed`** with five buckets, the folder structure will look like Figure 8.

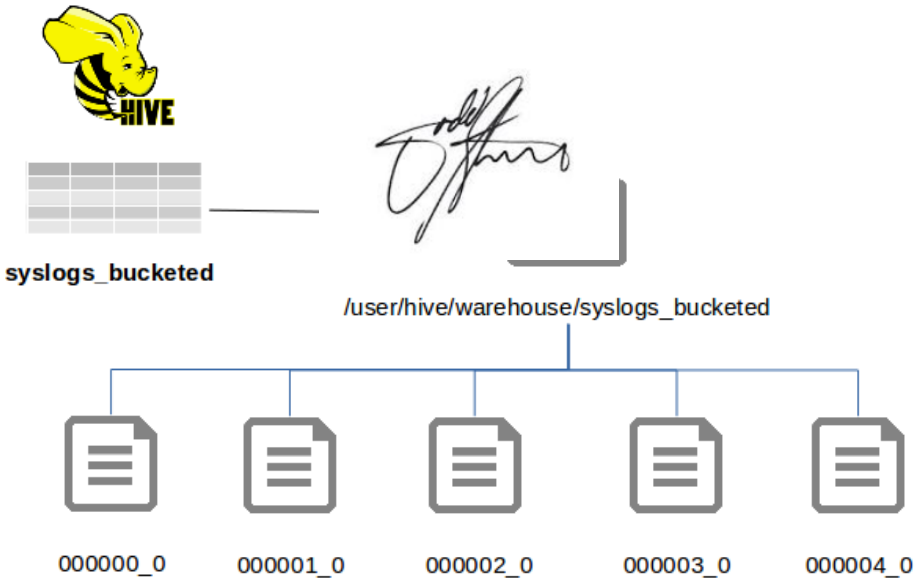

Figure 8: Folder Structure for a Bucketed Table

Here we still have the benefit of sharding data, but we don't have the issue of heavily nested folders, and we have better control over how many files the data is split across. Bucketed tables are also easier to work with because the bucket columns are normal data columns, which means we don't have to specify a partition when we load data.

Creating bucketed tables

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Code Listing 85 creates a bucketed version of the **syslogs** table using the period and host columns for the buckets. In this example the file is in ORC format, but that is not a requirement.

Code Listing 85: Creating a Bucketed Table

```
create table s d string ggedat
timestamp, proces message s by(period,
host) into 12 buckets stored as orc;

No rows affected (0.226 seconds)
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

bu wh cate d loc sam lways be in the same location, but rows with different combinations of period and host can be in different buckets.

Hive doesn't create any folders or files until we start to populate the table, but after inserting a single row, the file structure for all the buckets will be created, as we see in Code Listing 86.

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Code Listing 86: Folder Listing for a Bucketed Table

```
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000001_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000002_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000003_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000004_0
-rwxrwxr-x   1 root supergroup        646 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000005_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000006_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000007_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000008_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000009_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000010_0
-rwxrwxr-x   1 root supergroup         49 2016-02-05 18:07
/user/hive/warehouse/syslogs_bucketed/000011_0
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

> **Tip: You can change the number of buckets for an existing table using the `alter table` statement with the `clustered by` clause, but Hive will not reorganize the data in the existing files to match the new bucket specification. If you want to modify the bucket count, it is preferable to create a new table and populate it from the existing one.**

A subclause of **`clustered by`** directs Hive to create a sorted bucketed table. With sorted tables, the same physical bucket structure is used, but data within the files is sorted by a specified column value. In Code Listing 87 a variant of the **`syslogs`** table is created that is bucketed by period and host and is sorted by the process name.

Code Listing 87: Creating a Sorted Bucketed Table

```
[illegible] buck[illegible]
[illegible] ng[illegible]
[illegible] t, [illegible] ou[illegible]
  stored as orc;
```

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

Sorting the data in buckets gives a further optimization at read time. At write time, we can take advantage of Hive's enforced bucketing to ensure data enters the correct buckets.

Populating buc[illegible]

INITIAL HERE

SIGN HERE

When bucketed tables get populated, the storage structure is transparent, as far as the insert is concerned. Standard columns are used to determine the storage bucket, which means the standard insert statements work without any additional clauses.

The **`load`** statement cannot be used with bucketed tables because **`load`** simply copies from the sourc[illegible] ifyi[illegible] der[illegible] eted table[illegible] sou[illegible] e d[illegible] ts, which will lo[illegible] **load** in any case.

APPROVED

COMPLETED

CONFIDENTIAL

In order to populate bucketed tables, we need to use an **`insert`**. Code Listing 88 shows a simple insert into a bucketed table with specific values. Some key lines from Hive's output log are also shown.

Code Listing [illegible] eted Table

```
> insert into syslogs_buckete[illegible] 2', unix_timestamp(),
'kernel', 1, 'message' from d[illegible]
...
INFO  : Hadoop job informatio[illegible] er of mappers: 1; number of
reducers: 12
...
INFO  : Table default.syslogs_bucketed stats: [numFiles=12, numRows=1,
totalSize=1185, rawDataSize=396]
No rows affected (29.69 seconds)
```

The log entries from Beeline tell us that Hive used a single map task and 12 reduce tasks—one for each bucket in the table, which ensures data will end up in the right location. (Hive uses a hash of the bucketed column values to decide on the target bucket.) The table stats in the last line tell us there are 12 files but only one row in the table.

Data in a bucketed table can get out of sync, with rows in the wrong buckets, if the number of reducers for an insert job does not match the number of buckets. Hive will address that if the setting **`hive.enforce.bucketing`** is true in the Hive session (or in **`hive-site.xml`**, as is the case in the **`hive-succinctly`** Docker image).

With bucketing enforced by Hive, we don't have to specify which bucket to populate, and so we can insert data into multiple buckets from a single statement. In Code Listing 89 we take the

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
> insert into table syslogs_bucketed select date_format(loggedat,
'yyyyMM'), host, loggedat, process, pid, message from syslogs;
…
INFO  : Table default.syslogs_bucketed stats: [numFiles=12, numRows=3942,
totalSize=58903,
```

INITIAL HERE

SIGN HERE

Querying bucketed tables

Bucketed tables are queried like any other tables, but query execution is optimized if the bucket columns are included in the **`where`** clause. In that case, Hive will limit the map input files to those it knows will contain the data, so that the initial search space is restricted.

APPROVED

COMPLETED

CONFIDENTIAL

There ce (or rom tables which are not bucketed—Code Listing 90 shows a query using all the columns in the table.

Code Listing 90: Querying Bucketed Tables

```
> select period, process, pid from syslogs_bucketed where host = 'sc-ub-
xps' limit 2;
+---------+----------------+-
| period  |    process     |
+---------+----------------+-
| 201601  | gnome-session  |
| 201601  | gnome-session  |
+---------+----------------+-------+--+
```

Columns can be used in any part of the **`select`** statement irrespective of whether or not they are plain data columns or they form part of the bucketing (or sorting) specification.

Bucketed tables are particularly useful if you want to query a subset of the data. We've seen the **`limit`** clause in previous HiveQL queries, but that only constrains the amount of data that gets returned—typically the query will run over the entire table and return only a small portion.

With bucketed tables, we can specify a query over a sample of the data using the **`tablesample`** clause. Because Hive provides a variety of ways to sample the data, we can pick from one or more buckets or from a percentage of the data. In Code Listing 91 we fetch data from the fifth bucket in the bucketed **`syslogs`** table.

Code Listing 91: Sampling from Table Buckets

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-----------+--+
| 42553050  |
+-----------+--+
> select count(*) from syslogs_bucketed tablesample(bucket 5 out of 12);
INFO  : MapReduce Total cumulative CPU time: 2 seconds 930 msec
+--------+--+
|  _c0   |
+--------+--+
| 40695  |
+--------+--+
```

INITIAL HERE

SIGN HERE

The entire table count returned 42 million rows in 21 seconds, yet a single bucket count took only

APPROVED

COMPLETED

CONFIDENTIAL

In order to sample a subset of data from multiple buckets, you can specify a percentage of the data size or a desired size for the sample. Hive reads from HDFS at the file block level, which means you might get a larger sample than you specified—if Hive fetches a block that takes it over the requested size, it will still use the entire block. Code Listing 92 fetches at least 3% of the data.

Code Listi of Data

```
> select count(*) from syslog ple(3 percent);
INFO  : MapReduce Total cumul seconds 740 msec
+-----------+--+
|    _c0    |
+-----------+--+
| 10525740  |
+-----------+--+
```

The efficient sampling returned with bucketed tables can be a huge timesaver when crafting a complex query. You can iterate over the query using a subset of data that will return quickly, and when you're happy with the query you can submit it to the cluster to run over the entire data set.

Summary

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Hive provides two approaches to sharding that can be used independently or in combination. Which approach you choose depends on how your data is logically grouped and how you're likely to access it. Typically, the clauses you most frequently query over but which have a relatively small number of distinct values are good candidates for sharding—those might be time periods or the identifications.

INITIAL HERE

SIGN HERE

Partitioned tables shard data physically by using a nested folder structure with one level of nesting for each partitioned column. The number of partitions is not fixed, and as you insert data into new partition column values, Hive will create partitions for you. Partition columns are a separate part of the normal table structure, which means your data loads are made more complex as they need to be partition aware.

The alternative to partitioned tables is bucketed tables that sp rather than er xed d, and standard data columns are used to decide the target bucket for new rows.

APPROVED

COMPLETED

CONFIDENTIAL

Bucketed tables are easier to work with because there is no distinction between the data columns and the bucket columns, and as Hive allocates data to buckets using a hash there will be an even spread with no hotspots if your column values are evenly spread. With bucketed columns you get the added bonus of efficient sampling in which Hive can pull a subset of data from one or more buckets.

Combining both approaches in a part can provide the optimal solution, but you must select your sharding col

In the final chapter, we'll look more cl and covering the higher value functionality HiveQL provides for Big Data analysis.

Chapter 9 Querying with HiveQL

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

Hive's biggest drivers are the broad functionality of HiveQL and its easy adoption for anyone with SQL experience. The complexity of identifying, loading, and transforming data can be isolated in development or ops teams, which will leave analysts free to query huge amounts of data using a familiar syntax.

INITIAL HERE

SIGN HERE

HiveQL keeps expan Hive, and en been integrated into Apache Spark, so that in-memory Big Data workloads can be based on HiveQL, too.

The language statements we've seen so far have been fundamentally similar to their SQL counterparts, and the same is true for the more advanced HiveQL features. We'll cover those in this chapter, along with some of the functions Hive provides and the mechanism for incorporating custom functionality in Hive.

APPROVED

COMPLETED

CONFIDENTIAL

Joining data sources

HiveQL supports inner, outer, cross, and semi joins. However, joins are not as richly supported as in SQL databases, because Hive only supports joins in which the comparison is for equal values—you can't join tables based on columns that have different values (`x <> y`) in Hive.

The basic joins use standard SQL sy oins the `servers` and `server_logs` tables and returns the first log entry.

Code Listing 93: Inner Joins

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+---------+--------------+-------------+-------------+--+
| SCSVR1  | 192.168.2.1  | 1439546226  | W           |
+---------+--------------+-------------+-------------+--+
> select s.name, s.ipaddresses[0], l.loggedat, l.loglevel from server_logs
l, servers s where l.serverid = s.name limit 1;
+---------+--------------+-------------+-------------+--+
| s.name  |     _c1      | l.loggedat  | l.loglevel  |
+---------+--------------+-------------+-------------+--+
| SCSVR1  | 192.168.2.1  | 1439546226  | W           |
+---------+--------------+-------------+-------------+--+
```

INITIAL HERE

SIGN HERE

Both queries return the same results, as the explicit syntax (**`join ... on`**) is interchangeable with the in[illegible]ples[illegible]spe[illegible]clause).

APPROVED

COMPLETED

CONFIDENTIAL

Similarly, **`outer join`**s are specified in the same way as SQL databases and have the same effect on the output—Code Listing 94 returns all the servers that have never recorded any logs.

Code Listing 94: Left Outer Joins

```
> select s.name, s.site["dc"] from servers s left outer join server_logs l
on s.name = l.serverid where
+---------+---------+--+
| s.name  |   _c1   |
+---------+---------+--+
| SCSVR2  | london  |
| SCSVR3  | dublin  |
+---------+---------+--+
```

The **`cross join`** statement returns the Cartesian product of the tables, as in SQL, and only **`left semi join`** is unusual. Most SQL databases support this join, but they don't use an explicit clause. This join is equivalent to fetching all the rows in one table from which a matching column value in another table exists.

In SQL databases, that is usually done in a **`where exists`** clause, but HiveQL has an explicit join type for it. Code Listing 95 shows how that looks as it returns only those servers which have recorded logs.

Code Listing 95: Left Semi Joins

```
APPROVED
By Guest at 3/28/2024 1:21:46 PM
CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM
NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM
+---------+---------+--+
| SCSVR1  | london  |
+---------+---------+--+
```

Provided your join is valid, you can join any database objects no matter what storage engine the use. The examp external t internal table **server_logs**. Code L table (**all_**), rnal HDFS table in JSON format (**devices**), and a view over an external HBase table (**device_events_period**).

INITIAL HERE

SIGN HERE

Code Listing 96: Joining Hive and HBase Tables

```
> select de.period, de.eventname, d.device.deviceclass from
device_events_period de join all_devices ad on ad.deviceid = de.deviceid
joi vic vic od
lik
+------------+---------------+--------------+--+
| de.period  | de.eventname  | deviceclass  |
+------------+---------------+--------------+--+
| 20160128   | power.off     | tablet       |
+------------+---------------+--------------+--+
```

APPROVED

COMPLETED

CONFIDENTIAL

As with any SQL database, joining la nance implication. Hive optimizes the joins wherever it can. With releas he Hive query engine is becoming increasingly sophisticated at join opti HiveQL supports query hints for explicit optimization, often they are n

For example, Hive can join onto small tables much more efficiently if the entire table is loaded into memory in a map task. This can be explicitly requested in a query with the **mapjoin** hint, but Hive can do this automatically if the setting **hive.auto.convert.join** is true.

Joining results with the **union** clause will be the last join we'll cover. This lets us combine two result sets with the same column structure. The current version of Hive (1.2.1) supports **union all**, which includes duplicate rows, and **union distinct**, which omits duplicates.

Code Listing 97 shows the result of two **union** clauses and also demonstrates an outer query (the **count**) with a subquery (the **union**)—showing that subqueries must be named in HiveQL.

Code Listing 97: Union All and Union Distinct

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-----+--+
| 4   |
+-----+--+
> select count(1) from (select * from devices a union distinct select *
from devices b) s
+-----+--+
| c0  |
+-----+--+
| 2   |
+-----+--+
```

INITIAL HERE

SIGN HERE

Agg APPROVED nd COMPLETED

CONFIDENTIAL

Hive supports basic aggregation to group results by one or more columns, as well as more advanced windowing functions.

Basic aggregation in Hive is done with the **`group by`** clause, which defines one or more columns to aggregate by and supports standard SQL aggregation functions such as **`sum`**, **`avg`**, **`count`**, **`min`**, and **`max`**. You can use se rent columns in the same query.

In Code Listing 98 we group syslog e at generated them, selecting only processes with more than 1,000 entri cess name, number of entries, and size of the largest message logged b

Code Listing 98: Group by Aggregation

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
+-----------------+-------------+-----------------+--+
|     process     | entrycount  | largestmessage  |
+-----------------+-------------+-----------------+--+
| NetworkManager  | 1863        | 201             |
| kernel          | 7444        | 101             |
| thermald        | [illegible] | [illegible]     |
| systemd         | 1232        | 93              |
+-----------------+-------------+-----------------+--+
```

INITIAL HERE

SIGN HERE

> **Tip: You can't include functions like `count` and `max` in the `order by` clause of a query, but if you alias the results of those functions in the `select` clause, you can o[illegible] this [illegible] by [illegible] e tried o[illegible]ess [illegible] r fi[illegible]**

APPROVED

COMPLETED

CONFIDENTIAL

As in SQL, you cannot include columns in a **`group by`** query unless they are in the group clause or are an aggregate function over the grouped set. The SQL:2003 specification defines window functions that let you aggregate over multiple partitions of the data set in a single query.

> **Note: Hive supports SQL:20[illegible] [illegible] eware of the terminology collision—partitions in window [illegible] ation to Hive's table partitioning; they are separate features with [illegible]**

With analytical functions, you can get [illegible] gregates in the same query. Code Listing 99 shows a query over **`syslo`**[illegible] ocesses and process IDs have log entries containing the word 'CRON' (with abbreviated results).

Code Listing 99: Partitioning Queries

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
+--------+----------+-------------------------------------------------------
|   c0   | process  |                                                  message
| c3  |
+--------+----------+-------------------------------------------------------
--------------------------------------+-------+
| 19:53  | cron                                  nning @
| 5   |
...
| 19:53  | anacron  | Job `cron.daily' started
| 1   |
| 19:53  | anacron  | Job `cron.daily' terminated
| 2   |
| 1                                    b                                   ted
| 2   |
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Here we get complex results from a straightforward query. The query selects the message text and information about the log entry, partitioning by the message text and counting at that level. The results show the same aggregation as grouping by text and counting, but they include row level details—we can see the 'anacron' processed the same message twice, and we can see the different times it was logged.

Analytic partitions can occur over mul an interesting way to present the data. Adding an **`order by`** in the **ove** query lets us view the results in time order with a running total of how man s have been logged up to that point, as shown in Code Listing 100.

Code Listing 100: Partitioned and Ordered Queries

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

```
+-----------+----------+-------------------------------------+-----+--+
|    c0     | process  |                message              | c3  |
+-----------+----------+-------------------------------------+-----+--+
| 19:52:21  | cron     | (CRON) INFO (Running @reboot jobs)  | 1   |
| 19:52:43  | cron     | (CRON) INFO (Running @reboot jobs)  | 2   |
| 19:53:21  | cron     | (CRON) INFO (Running @reboot jobs)  | 3   |
| 19:53:32  | cron     | (CRON) INFO (Running @reboot jobs)  | 4   |
| 19:54:09  | cron     | (CRON) INFO (Running @reboot jobs)  | 5   |
+-----------+----------+-------------------------------------+-----+--+
```

INITIAL HERE

SIGN HERE

Partitioning by message and ordering by the timestamp gives us an incremental view of when the message in question was logged. We can take that one stage further using windowing funct

APPROVED

COMPLETED

CONFIDENTIAL

Windowing functions let you produce a result set in which values for a row are compared to values in the rows that precede or follow the current row. You can set the range explicitly for a window, or you can use functions that implicitly define a window (usually defaulting to single row on either side of the current row).

You can combine scalar values, row-level aggregates, and windowing functions in the same query. Code Listing 101 repeats the p each row it specifies the distance in time from this row to the one before it

Code Listing 101: Windowing with Lag and Lead

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
like '%CRON%REBOOT%';
...
+-----------+----------+-----------------------------------+-----+-------
|    c0     | process  |              message              | c3  |
c4          |          c5                |
+-----------+----------+-----------------------------------+-----+-------
| 19:52:21  | cron     | (CRON) INFO (Running @reboot jobs)  | 1   | NULL
| 0 00:00:21.455000000  |
| 19:52:43  | cron     | (CRON) INFO (Running @reboot jobs)  | 2   | -0
00:00:21.455000000  | 0 00:00:37.773000000  |
| 19:53:21  | cron     | (CRON) INFO (Running @reboot jobs)  | 3   | -0
00:00:37.773000000  | 0 00:00:10.958000000  |
| 19:54:09  | cron     | (CRON) INFO (Running @reboot jobs)  | 5   | -0
00:00:37.695000000  | NULL                  |
+-----------+----------+-----------------------------------+-----+-------
```

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL

Here the **lag** function gets the **loggedAt** value for the previous row (which is NULL for the first row) and **lead** gets the **loggedAt** val e current row's **loggedAt** value is subtracted, which gives us the time d .

Windowing functions are a relatively s rful results—like finding the change of values over time periods, identifyin g percentiles and ranks.

Built-in functions

We've seen many examples of HiveQL's built-in functions already, and they are usually well-named and syntactically clear, so that they stand without much introduction. Hive has a suite of built-in functions which ship with the runtime, which means they are the same no matter which platform you work with.

The Language Manual for Built-in Functions is a comprehensive reference that lists all available functions by the data type they apply to and includes the version from which they were introduced.

Hive includes more than 150 built-in functions; here I'll cover some of the most useful.

Date and timestamp functions

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

In Code Listing 102 we fetch the current UNIX timestamp (in seconds, using the system clock) and convert between string and long-integer date values.

Code Listing 102: Date Conversion Functions

```
> select unix_timestamp() from_clock, unix_timestamp('2016-02-07 21:00:00')
from_string, from_unix_timestamp(1454878800L) from_long;
+-------------+-------------+---------------------+--+
| from_clock  | from_string |      from_long      |
+-------------+-------------+---------------------+--+
| 1454879183  | 1454878800  | 2016-02-07 21:03:16 |
+-------------+-------------+---------------------+--+
```

INITIAL HERE

SIGN HERE

Once we have added the numbers, and get the

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 103: Date Manipulation Functions

```
> select weekofyear(to_date(current_timestamp)) week, date_add('2016-02-07
21:00:00', 10) addition, datediff('2016-02-07', '2016-01-31') difference;

+-------+-------------+------
| week  |  addition   | diffe
+-------+-------------+------
| 5     | 2016-02-17  | 7
+-------+-------------+------
```

String functions

Hive includes all the usual functions for finding text within string (**`instr`**), splitting strings (**`substr`**), and joining them (**`concat`**). It also includes some useful overloaded functions that allow for common tasks with a single statement, as in Code Listing 104, which manipulates a URL.

Code Listing 104: String Manipulation Functions

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
| blog | protocol |
+-------------------+-----------+--+
| blog.sixeyed.com | https |
+-------------------+-----------+--+
```

INITIAL HERE

SIGN HERE

String functions for ri ams and c ide textual analysis for word frequency in sentences) and standard language processing functions also exist. Code Listing 105 shows the distance between two words (using the common Levenshtein measure of similarity) and the phonetic representation of a word.

Code Listing 105: Language Processing Functions

APPROVED

COMPLETED

CONFIDENTIAL

```
> s ve ('h
+------+-------+--+
| _c0 | _c1 |
+------+-------+--+
| 3 | H100 |
+------+-------+--+
```

Mathematical functions

As with strings, Hive supports all the ctions (**round**, **floor, abs**), along with more unusual ones—such as trig in, **cos**, **tan**). Having a wide range of built-in math functions makes complex analysis possible with simple queries.

Code Listing 106 shows some useful mathematical functions.

Code Listing 106: Mathematical Functions

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+----------+--------------------+------------+--+
| 2        | 9.539392014169456  | 720        |
+----------+--------------------+------------+--+
```

Collection func

INITIAL HERE

SIGN HERE

With its support for collection data types, Hive provides functions for working with arrays and maps. Only a small number of functions are provided, but they cover all the functionality you're likely to need.

Code Listing 107 shows how to extract a value from an array and sort the array, then combine them to find the largest value (the

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 107: Collection Functions

```
> select array(26, 54, 43)[0], sort_array(array(26, 54, 43)),
sort_array(array(26, 54, 43))[size(array(26, 54, 43))-1];

+------+-------------+------+--+
| _c0  |     _c1     | _c2  |
+------+-------------+------+
| 26   | [26,43,54]  | 54   |
+------+-------------+------+
```

Similar functions can be used with MAP column types so that you can also extract the set of keys or the set of values as arrays. A key function suitable with either type is **`explode`**, which generates a table from a collection, as shown in Code Listing 108.

Code Listing 108: Exploding Collection Data

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
+-------------+--+
| Hive        |
| Succinctly  |
+-------------+--+
```

INITIAL HERE

SIGN HERE

With **explode**, you ca om a sing se that to join against other tables.

Other functions

Standard SQL functions are available in Hive, such as converting between types (**cast**), checl not aris eturn one v o s n C

APPROVED

COMPLETED

CONFIDENTIAL

Code Listing 109: Choosing between Values

```
> select nvl(NULL, 'default') `nvl`, coalesce('first', 'second', NULL)
`coalesce`, case true when cast(1 as boolean) then 'expected' else
'unexpected' end `case`;

+----------+-----------+-----
|   nvl    | coalesce  |   ca
+----------+-----------+-----
| default  | first     | expe
+----------+-----------+-----------+--+
```

More unusual functions are also occasionally useful, as shown in Code Listing 110.

Code Listing 110: Miscellaneous Functions

```
> select pi() `pi`, current_user() `whoami`, hash('@eltonstoneman') `hash`;

+--------------------+---------+-------------+--+
|         pi         | whoami  |    hash     |
+--------------------+---------+-------------+--+
| 3.141592653589793  | root    | 1326977505  |
+--------------------+---------+-------------+--+
```

Hive continues to add functions to the built-in set with AES encryption, SHA, and MD5 hashing, and CRC32 calculations available in Hive 2.0.0.

User defined functions

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

That means you can write functions in your preferred language and ensure the quality of custom components with unit testing and versioning. Provided you can wrap functions in a command line that reads from standard input and writes to standard output, you can also use existing libraries of code.

INITIAL HERE

SIGN HERE

A UDF consists of tw ary avail e and registering the function so you can use it in Hive queries.

Java UDFs are the simplest operation. You write a class that extends **`org.apache.hadoop.hive.ql.exec.UDF`**, then build a JAR and copy it to Hive's auxiliary folder (specified with the **`HIVE_AUX_JARS_PATH`** setting). Next, you register the class **`with create temporary function [alias] as [UDF_class_name]`**. With that, you can call the function using the UDF alias.

APPROVED

COMPLETED

CONFIDENTIAL

Strea littl e P adds Value Added Tax to an integer amount, as an example in Code Listing 111.

Code Listing 111: A Simple Python Script

```
#!/usr/bin/python
import sys
for line in sys.stdin:
  line = line.strip()
  print int(line) * 1.2
```

Next we need to copy the .py file to H command in Code Listing 112.

Code Listing 112: Adding the Python Script to Hive

```
> add file /tmp/add_vat_udf.py;

INFO  : Added resources: [/tmp/add_vat_udf.py]
```

And now we can access the UDF with the **`transform … using`** clause, which will invoke the command once for each row in the ResultSet of the Hive query. Code Listing 113 shows the UDF being invoked.

Code Listing 113: Invoking the Python Script in HiveQL

APPROVED
By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL
By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED
By Guest at 3/28/2024 1:21:59 PM

```
| vat_added  |
+------------+--+
| 12.0       |
| 144.0      |
| 180.0      |
```

INITIAL HERE

SIGN HERE

Note: You can't include columns from the result set and transformed column—the final result has to come entirely from the transform. So if I wanted to include the original net value in my query, I couldn't add it to the select statement before the transform, I'd need to write out the input in my Python script and tab-separate it from the output.

APPROVED

COMPLETED

CONFIDENTIAL

Summary

The familiarity of SQL, combined with complex analytical functions and the ability to make functions of your own code, make HiveQL a powerful and extensible query language.

For the most part, you can run the exact same queries over any data source—whether that's an internal Hive table, a HDFS folder stru ands of TSV files, or an HBase table with billions of rows.

The Hive compiler generates the mos duce jobs that it can in order to represent the HiveQL query in a form l on Hadoop. Presenting a simple interface that can trigger hundreds of compute hours behind the scenes with no further user interaction makes Hive an attractive component in the Big Data stack.

Next steps

We've covered a lot of ground in this short book, but there's plenty more to learn. In order to address Hive succinctly, I've focused on the functional parts of the language and the runtime, which means I haven't even touched on performance, query optimization, or use of Hive with other clients. Those are the obvious next steps if you want to learn more about Hive.

The **`hive-succinctly`** Docker image is a good place to get started. It's set up with Hadoop running in pseudo-distributed mode and configured for YARN, which means it's suitable for testing long-running queries. You can drop your own data onto the container using Docker's copy command, then load it into Hive using the tools we saw in Chapter 6 ETL with Hive.

After that, if you want to try out Hive in production to see how it performs with your data at a much bigger scale, you can easily fire up a Hadoop cluster running Hive in the cloud. The

APPROVED

By Guest at 3/28/2024 1:21:46 PM

CONFIDENTIAL

By Guest at 3/28/2024 1:21:52 PM

NOT APPROVED

By Guest at 3/28/2024 1:21:59 PM

INITIAL HERE

SIGN HERE

APPROVED

COMPLETED

CONFIDENTIAL