Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ **Introduction to Impala and Hive** ▪ Querying With Impala and Hive ▪ Impala and Hive Data Management ▪ Data Storage and Performance	**Introduction to Impala and Hive**
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ Conclusion	Course Conclusion

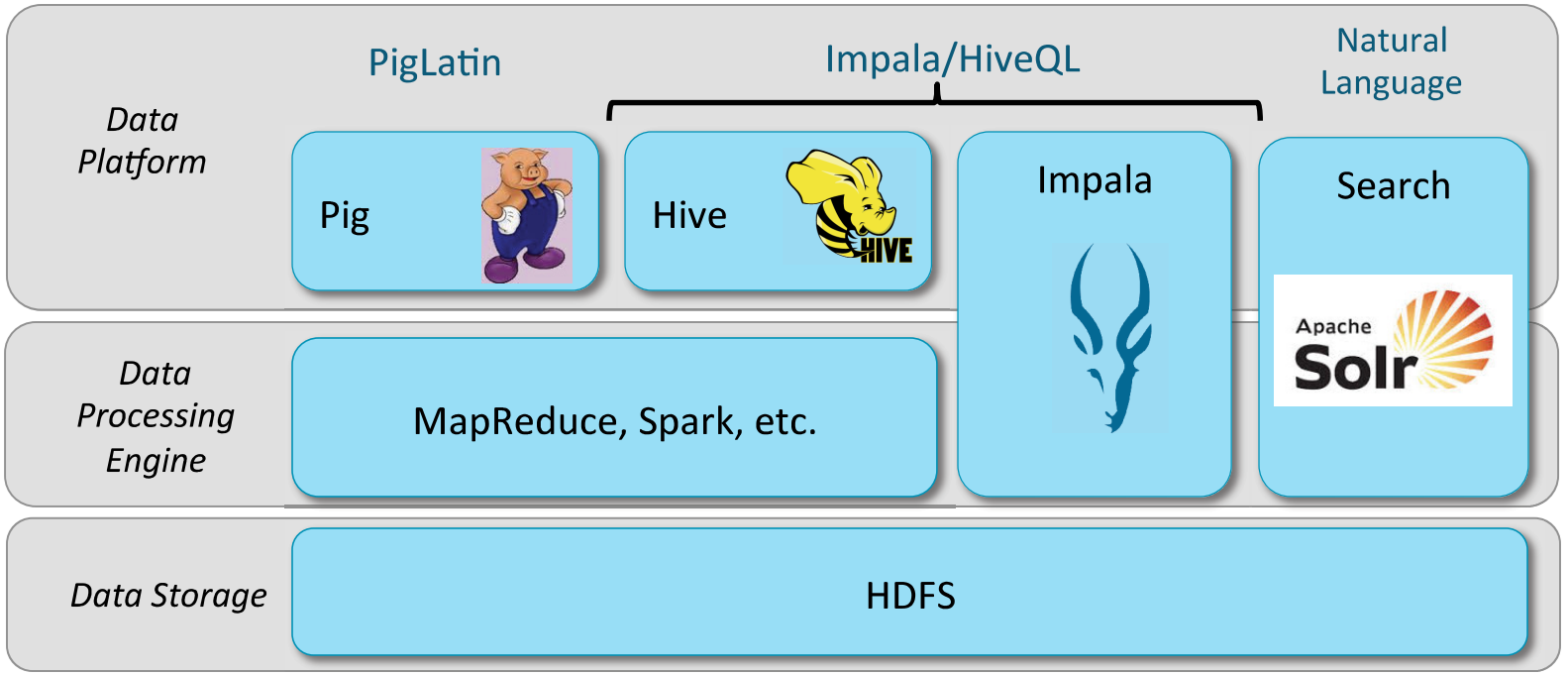
Introduction to Impala and Hive

In this chapter, you will learn

- **What Hive is**
- **What Impala is**
- **How Hive and Impala differ from a relational database**
- **Ways in which organizations use Hive and Impala**

Review: Hadoop Data Processing and Analysis

- **Hadoop includes many tools for data processing and analysis**



Chapter Topics

Introduction to Impala and Hive

- **What is Hive?**
- What is Impala?
- Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Conclusion

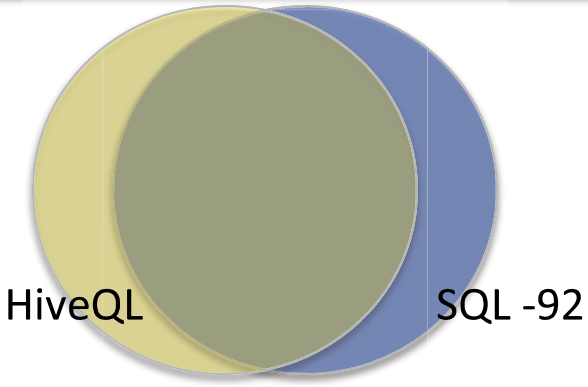
What is Apache Hive?

- **Apache Hive is a high-level abstraction on top of MapReduce**
  - Uses a SQL-like language called HiveQL
  - Generates jobs that run on the Hadoop cluster
  - Originally developed by Facebook for data warehousing
    - Now an open-source Apache project

HiveQL

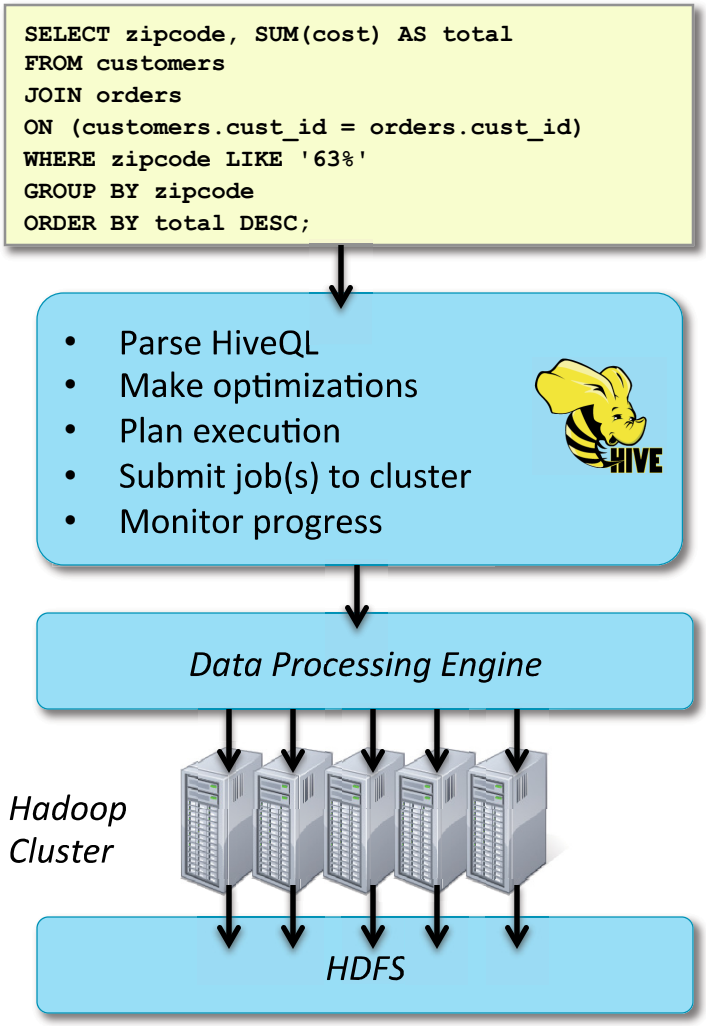
- **HiveQL implements a subset of SQL-92**
  - Plus a few extensions found in MySQL and Oracle SQL dialects

```
SELECT zipcode, SUM(cost) AS total
FROM customers
JOIN orders
ON (customers.cust_id = orders.cust_id)
WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```



Hive High-Level Overview

- **Hive**
  - Turns HiveQL queries into data processing jobs
  - Submits those jobs to the data processing engine (MapReduce) to execute on the cluster



Chapter Topics

Introduction to Impala and Hive

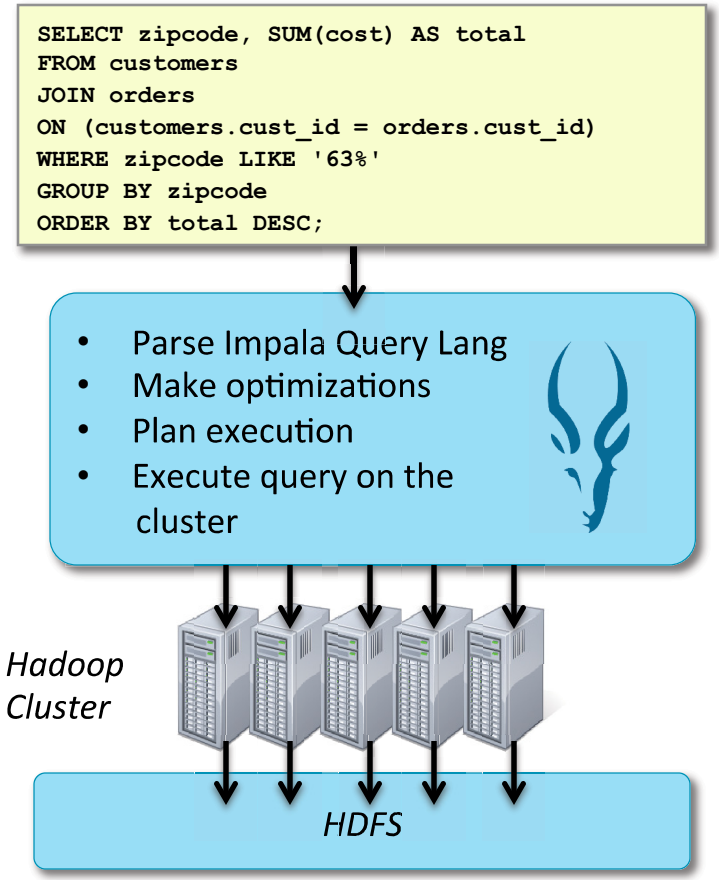
- What is Hive?
- **What is Impala?**
- Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Conclusion

What is Impala?

- **Impala is a high-performance SQL engine for vast amounts of data**
  - Massively-parallel processing (MPP)
  - Inspired by Google's Dremel project
  - Query latency measured in milliseconds
- **Impala runs on Hadoop clusters**
  - Can query data stored in HDFS or HBase tables
  - Reads and writes data in common Hadoop file formats
- **Developed by Cloudera**
  - 100% open source, released under the Apache software license

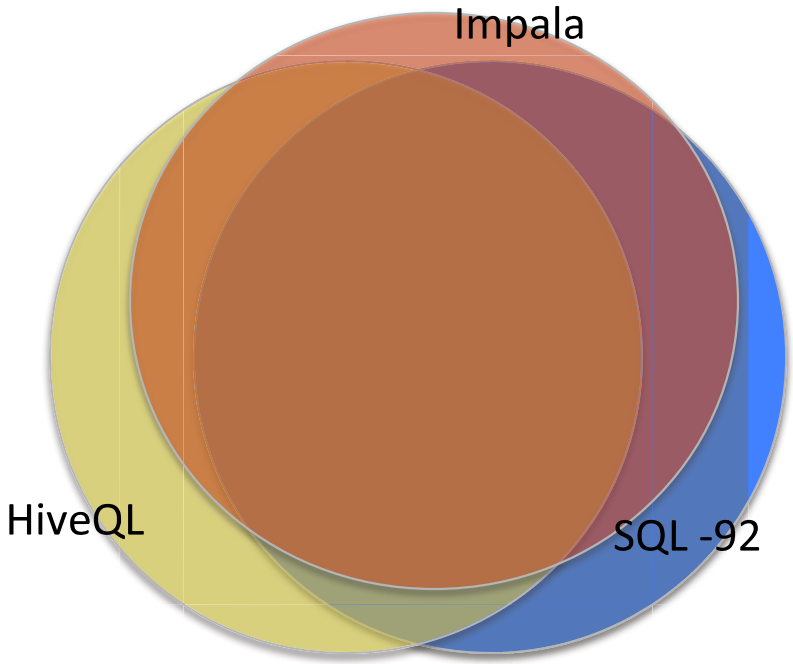
Impala High-Level Overview

- **Impala**
  - Executes Impala SQL queries directly on the Hadoop Cluster
  - Does not rely on a general-purpose data processing engine such as MapReduce
  - Highly-optimized for queries
  - Almost always at least five times faster than either Hive or Pig
    - Often 20 times faster or more



Impala Query Language

- **Impala supports a subset of HiveQL**
  - Plus a few additional commands



Why Use Hive and Impala?

- **More productive than writing MapReduce directly**
  - Five lines of HiveQL/Impala SQL might be equivalent to 200 lines or more of Java
- **Brings large-scale data analysis to a broader audience**
  - No software development experience required
  - Leverage existing knowledge of SQL
- **Offers interoperability with other systems**
  - Extensible through Java and external scripts
  - Many business intelligence (BI) tools support Hive and/or Impala

Comparing Hive and Impala

- **Hive and Impala are different tools, but are closely related**
  - Both use very similar variants of SQL
  - Both share the same data warehouse and metadata storage
  - They are often used together
- **The next few chapters focus on the areas that are the same between Hive and Impala**
  - HiveQL and Impala SQL are nearly identical
  - Minor differences will be noted
- **Later chapters will focus on the areas in which they differ**

Where to Get Hive and Impala

- **CDH is the easiest way to install Hadoop, Hive and Impala**
  - A Hadoop distribution which includes HDFS, MapReduce, Spark, Pig, Hive, Impala, Sqoop, HBase, and other Hadoop ecosystem components
  - Available as RPMs, Ubuntu/Debian/SuSE packages, or a tarball
  - Simple installation
  - 100% free and open source
- **Installation is outside the scope of this course**
  - Cloudera offers a training course for System Administrators, *Cloudera Administrator Training for Apache Hadoop*

Chapter Topics

Introduction to Impala and Hive

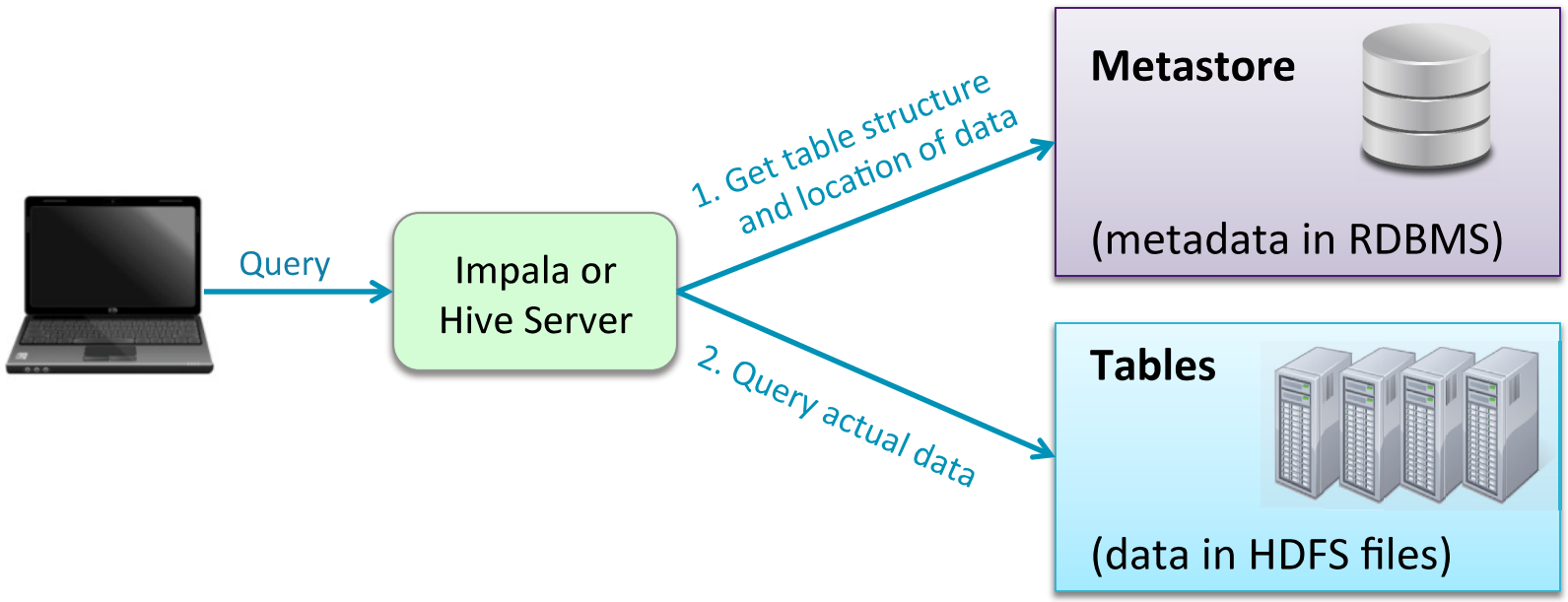
- What is Hive?
- What is Impala?
- **Schema and Data Storage**
- Comparing Hive and Impala to Traditional Databases
- Hive and Impala Use Cases
- Conclusion

How Hive and Impala Load and Store Data (1)

- **Queries operate on tables, just like in an RDBMS**
  - A table is simply an HDFS directory containing one or more files
  - Default path: `/user/hive/warehouse/<table_name>`
  - Supports many formats for data storage and retrieval
- **What is the structure and location of tables?**
  - These are specified when tables are created
  - This metadata is stored in the *Metastore*
    - Contained in an RDBMS such as MySQL
- **Hive and Impala work with the same data**
  - Tables in HDFS, metadata in the Metastore

How Hive and Impala Load and Store Data (2)

- **Hive and Impala use the Metastore to determine data format and location**
  - The query itself operates on data stored on a filesystem (typically HDFS)



Chapter Topics

Introduction to Impala and Hive

- What is Hive?
- What is Impala?
- Schema and Data Storage
- **Comparing Hive and Impala to Traditional Databases**
- Hive and Impala Use Cases
- Conclusion

Your Cluster is Not a Database Server

- **Client-server database management systems have many strengths**
  - Very fast response time
  - Support for transactions
  - Allow modification of existing records
  - Can serve thousands of simultaneous clients
- **Your Hadoop cluster is not an RDBMS**
  - Hive generates processing engine jobs (MapReduce) from HiveQL queries
    - Limitations of HDFS and MapReduce still apply
  - Impala is faster but not intended for the throughput speed required for an OLTP database
  - No transaction support

Comparing Hive and Impala To A Relational Database

	Relational Database	Hive	Impala
Query language	SQL (full)	SQL (subset)	SQL (subset)
Update individual records	Yes	No	No
Delete individual records	Yes	No	No
Transactions	Yes	No	No
Index support	Extensive	Limited	No
Latency	Very low	High	Low
Data size	Terabytes	Petabytes	Petabytes

Chapter Topics

Introduction to Impala and Hive

Use Case: Log File Analytics

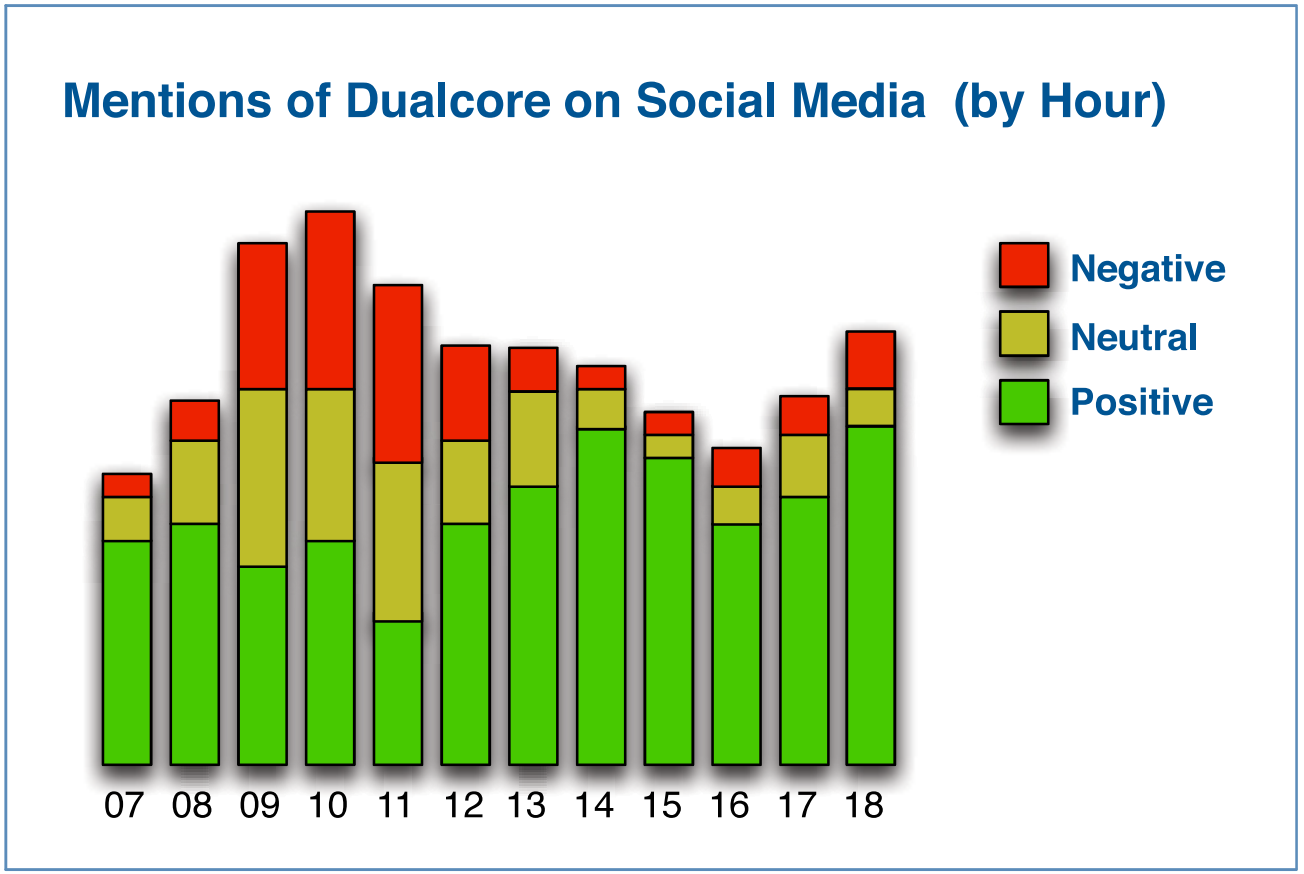
- **Server log files are an important source of data**
- **Hive and Impala allow you to treat a directory of log files like a table**
  - Allows SQL-like queries against raw data

Dualcore Inc. Public Web Site (June 1 - 8)

Product	Unique Visitors	Page Views	Average Time on Page	Bounce Rate	Conversion Rate
Tablet	5,278	5,894	17 seconds	23%	65%
Notebook	4,139	4,375	23 seconds	47%	31%
Stereo	2,873	2,981	42 seconds	61%	12%
Monitor	1,749	1,862	26 seconds	74%	19%
Router	987	1,139	37 seconds	56%	17%
Server	314	504	53 seconds	48%	28%
Printer	86	97	34 seconds	27%	64%

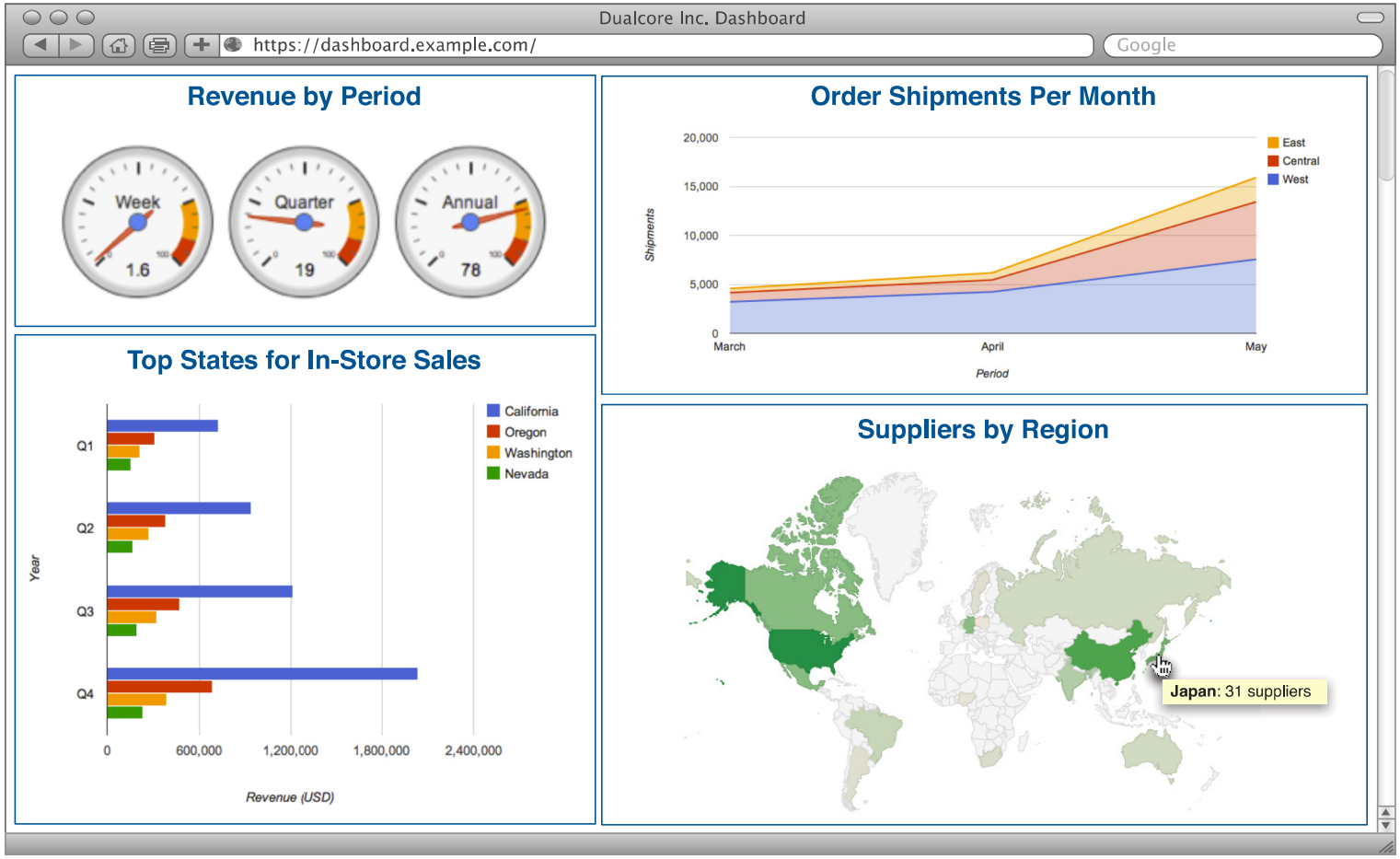
Use Case: Sentiment Analysis

- **Many organizations use Hive or Impala to analyze social media coverage**



Use Case: Business Intelligence

- **Many leading business intelligence tools support Hive and Impala**



Chapter Topics

Introduction to Impala and Hive

- What is Hive?
- What is Impala?
- Schema and Data Storage
- Comparing Hive and Impala to Traditional Databases
- Hive and Impala Use Cases
- **Conclusion**

Essential Points

- **Impala and Hive are tools for performing SQL queries on data in HDFS**
- **HiveQL and Impala SQL are very similar to SQL-92**
  - Easy to learn for those with relational database experience
  - However, does *not* replace your RDBMS
- **Hive generates jobs that run on the Hadoop cluster data processing engine**
  - Runs MapReduce jobs on Hadoop based on HiveQL statements
- **Impala execute queries directly on the Hadoop cluster**
  - Uses a very fast specialized SQL engine, not MapReduce
- **Tables are really directories of files in HDFS**
  - Information about those tables is kept in the Metastore (in an RDBMS)

Bibliography

The following offer more information on topics discussed in this chapter

- ***Cloudera Impala* (free O'Reilly book)**
  - `http://tiny.cloudera.com/impalabook`
- ***Programming Hive* (O'Reilly book)**
  - `http://tiny.cloudera.com/programminghive`
- **Data Analysis with Hadoop and Hive at Orbitz**
  - `http://tiny.cloudera.com/dac09b`
- **Sentiment Analysis Using Apache Hive**
  - `http://tiny.cloudera.com/dac09c`
- ***Wired* Article on Impala**
  - `http://tiny.cloudera.com/wiredimpala`

cloudera®
Ask Bigger Questions

Querying with Impala and Hive

Chapter 9

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ **Querying With Impala and Hive** ▪ Impala and Hive Data Management ▪ Data Storage and Performance	**Introduction to Impala and Hive**
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ Conclusion	Course Conclusion

Querying with Hive and Impala

In this chapter, you will learn

- **How to explore databases and tables with HiveQL and Impala SQL**
- **How HiveQL, Impala SQL, and SQL syntax compare**
- **Which data types HiveQL and Impala SQL support**

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- **Databases and Tables**
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Tables

- **Data for Hive and Impala tables is stored on the filesystem (typically HDFS)**
  - Each table maps to a single directory
- **A table's directory may contain multiple files**
  - Typically delimited text files, but many formats are supported
  - Subdirectories are not allowed
    - Unless the table is partitioned (see later)
- **The Metastore gives context to this data**
  - Helps map raw data in HDFS to named columns of specific types

Databases

- **Each table belongs to a specific database**
- **Early versions of Hive supported only a single database**
  - It placed all tables in the same database (named `default`)
  - This is still the default behavior
- **Hive supports multiple databases as of release 0.7.0**
  - Helpful for organization and authorization

Exploring Databases and Tables (1)

- **Which databases are available?**

```
SHOW DATABASES;
+-----------------+
| database_name   |
+-----------------+
| sales           |
| default         |
+-----------------+
```

Exploring Databases and Tables (2)

- **Impala and Hive both connect to the `default` database by default**
- **Switch between databases with the `USE` command**

```
$ impala-shell
> SELECT * FROM customers;
> USE sales;
> SELECT * FROM customers;
```

Queries table in the `default` database

Queries table in the `sales` database

Exploring Databases and Tables (2)

- **Use SHOW TABLES to list the tables in a database**

Shows tables in the current database

```
USE accounting;
SHOW TABLES;
+-----------+
| tab_name  |
+-----------+
| invoices  |
| taxes     |
+-----------+
```

```
SHOW TABLES IN sales;
+-----------+
| tab_name  |
+-----------+
| customers |
| prospects |
+-----------+
```

Shows tables in another database

Exploring Databases and Tables (3)

- **The `DESCRIBE` command displays basic structure for a table**

```
> DESCRIBE orders;
+------------+-----------+---------+
| name       | type      | comment |
+------------+-----------+---------+
| order_id   | int       |         |
| cust_id    | int       |         |
| order_date | timestamp |         |
+------------+-----------+---------+
```

- **Use `DESCRIBE FORMATTED` command for detailed info**
  - Input and output formats, file locations, etc.

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- **Basic Hive and Impala Query Language Syntax**
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

An Introduction To HiveQL and Impala Query Language

- **HiveQL is Hive's query language**
  - Based on a subset of SQL-92, plus Hive-specific extensions
- **Impala Query Language is very similar to HiveQL**
  - Lacks some features (will be noted during class)
- **Some limitations compared to 'standard' SQL**
  - Some features are not supported
  - Others are only partially implemented

Syntax Basics

- **Keywords are not case-sensitive**
  - Though they are often capitalized by convention
- **Statements are terminated by a semicolon**
  - A statement may span multiple lines
- **Comments begin with `--` *(double hyphen)***
  - Only supported in scripts
  - There are no multi-line comments

myscript.sql

```
SELECT cust_id, fname, lname
   FROM customers
   WHERE zipcode='60601';    -- downtown Chicago
```

Selecting Data from Tables

- **The `SELECT` statement retrieves data from tables**
  - Can specify an ordered list of individual columns

```
SELECT cust_id, fname, lname FROM customers;
```

  - An asterisk matches all columns in the table

```
SELECT * FROM customers;
```

Limiting and Sorting Query Results

- **The `LIMIT` clause sets the maximum number of rows returned**

```
SELECT fname, lname FROM customers LIMIT 10;
```

- **Caution: no guarantee regarding *which* 10 results are returned**
  - Use `ORDER BY` for top-N queries
  - The field(s) you `ORDER BY` must be `SELECT`ed

```
SELECT cust_id, fname, lname FROM customers
   ORDER BY cust_id DESC LIMIT 10;
```

Using a WHERE Clause to Restrict Results

- **WHERE clauses restrict rows to those matching specified criteria**
  - String comparisons are case-sensitive

```
SELECT * FROM orders WHERE order_id=1287;
```

```
SELECT * FROM customers WHERE state
   IN ('CA', 'OR', 'WA', 'NV', 'AZ');
```

- **You can combine expressions using AND or OR**

```
SELECT * FROM customers
   WHERE fname LIKE 'Ann%'
      AND (city='Seattle' OR city='Portland');
```

Table Aliases

- **Table aliases can help simplify complex queries**

```
SELECT o.order_date, c.fname, c.lname
   FROM customers c JOIN orders o
   ON (c.cust_id = o.cust_id)
   WHERE c.zipcode='94306';
```

  - Note: Using `AS` to specify table aliases is also supported starting with CDH 5.1

Combining Query Results with UNION ALL

- **Unifies output from SELECTs into a single result set**
  - The name, order, and types of columns in each query must match
  - Only supports UNION ALL

```
SELECT cust_id, fname, lname
    FROM customers
    WHERE state='NY'
  UNION ALL
  SELECT cust_id, fname, lname
    FROM customers
    WHERE zipcode LIKE '073%'
```

- **UNION ALL can also be used with subqueries**

Subqueries

- **Subqueries are supported only in the `FROM` clause of the `SELECT` statement**

```
SELECT prod_id, brand, name
  FROM (SELECT *
          FROM products
          WHERE (price - cost) / price > 0.65
          ORDER BY price DESC
          LIMIT 10) high_profits
  WHERE price > 1000
  ORDER BY brand, name;
```

- **Supports arbitrary levels of subqueries**
  - Each subquery must be named (like `high_profits` above)
  - Correlated subqueries supported in Impala 2.0 and later

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- **Data Types**
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Data Types

- **Each column is associated with a data type**
- **Hive and Impala support more than a dozen types**
  - Most are similar to ones found in relational databases
  - Hive also supports three complex types (covered later)
- **Use the `DESCRIBE` command to see a table's column types**

```
DESCRIBE products;
+-------------+--------+---------+
| name        | type   | comment |
+-------------+--------+---------+
| prod_id     | int    |         |
| brand       | string |         |
| name        | string |         |
| price       | int    |         |
| cost        | int    |         |
| shipping_wt | int    |         |
+-------------+--------+---------+
```

Integer Types

- **Integer types are appropriate for whole numbers**
  - Both positive and negative values allowed

Name	Description	Example Value
`TINYINT`	Range: -128 to 127	`17`
`SMALLINT`	Range: -32,768 to 32,767	`5842`
`INT`	Range: -2,147,483,648 to 2,147,483,647	`84127213`
`BIGINT`	Range: ~ -9.2 quintillion to ~ 9.2 quintillion	`632197432180964`

Decimal Types

- **Decimal types are appropriate for floating point numbers**
  - Both positive and negative values allowed
  - Use DECIMAL when exact values are required!

Name	Description	Example Value
`FLOAT`	Decimals	`3.14159`
`DOUBLE`	Very precise decimals	`3.141592653589793238 46`
`DECIMAL(p,s)` *	Exact precision	`3.14` (p=3, s=2)

* Available in version CDH 5.1 and later

Character Types

- **Character Types**

Name	Description	Example Value
`STRING`	Character sequence	**`Impala rules!`**
`CHAR(`*`n`*`)` *	Fixed length character sequence	**`Impala rules!`**■■■ (n=15)
`VARCHAR(`*`n`*`)` *	Variable length character sequence (max length *n*)	**`Impala rul`** (n=10)

* Available in CDH 5.2 and later

Other Simple Types

- **Other types**

Name	Description	Example Value
`BOOLEAN`	True or False	**`TRUE`**
`TIMESTAMP`	Instant in time	`2013-06-14 16:51:05`
`BINARY`* (Hive Only)	Raw bytes	N/A

* Available in Hive 0.8 and later

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- **Differences Between Impala and Hive Query Syntax**
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Differences Between Impala and Hive Query Syntax

- **Hive and Impala Query languages are very similar but not identical**
- **Some Hive features are not available in Impala**
  - These will be covered in a later chapter
- **There are also a few minor syntax differences**

Impala Metadata Cache Invalidation

- **Changes made to tables outside Impala require cache invalidation**
  - For example, creating tables with Hive or importing data with Sqoop

```
INVALIDATE METADATA;
```

```
REFRESH tablename;
```

- **Details on Impala metadata caching and invalidation are covered in a later chapter**

Selecting Data in Impala

- **Impala does not require a `FROM` clause**

```
SELECT SQRT(64) AS square_root;
```

Data Type Conversion

- **Hive auto-converts a `STRING` column used in numeric context**


```
SELECT zipcode FROM customers LIMIT 1;
60601
SELECT zipcode + 1.5 FROM customers LIMIT 1;
60602.5
```

- **Impala requires an explicit `CAST` operation for this**

```
SELECT zipcode + 1.5 FROM customers LIMIT 1;
ERROR: AnalysisException: Arithmetic operation ...
```

```
SELECT CAST(zipcode AS float) + 1.5
  FROM customers LIMIT 1;
60602.5
```

Limiting and Sorting Query Results

- **In versions of Impala before CDH 5.2**
  - When using `ORDER BY`, the `LIMIT` clause was mandatory in Impala

```
SELECT cust_id, fname, lname FROM customers
  ORDER BY cust_id DESC LIMIT 10;
```

  - Only one `DISTINCT` clause was allowed per query in Impala
    - Typical workaround is to use subselect and `UNION`
  - Both of these restrictions are not present in CDH 5.2 and later

Handling of NULL Values

- **Impala and Hive handle out-of-range values differently**
  - Hive returns `NULL`
  - Impala returns the maximum value for that type

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- **Using Hue to Execute Queries**
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

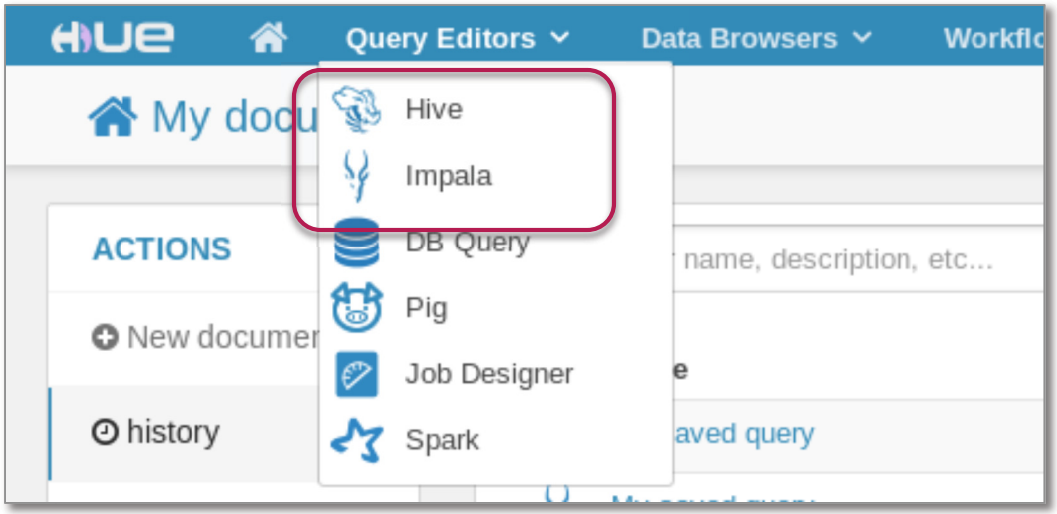
Interacting with Hive and Impala

- **Hive and Impala offer many interfaces for running queries**
  - Hue Web UI
    - Hive Query Editor (also know as Beeswax)
    - Impala Query Editor
    - Metastore Manager
  - Command-line shell
    - Hive: Beeline
    - Impala: Impala shell
  - ODBC / JDBC

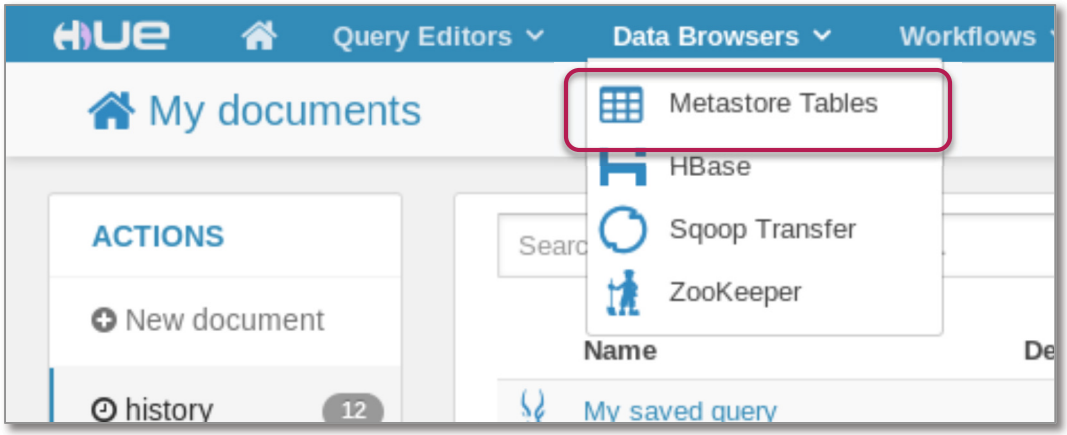
Using Hue with Hive and Impala

You can use Hue to…

Query data with Hive or Impala

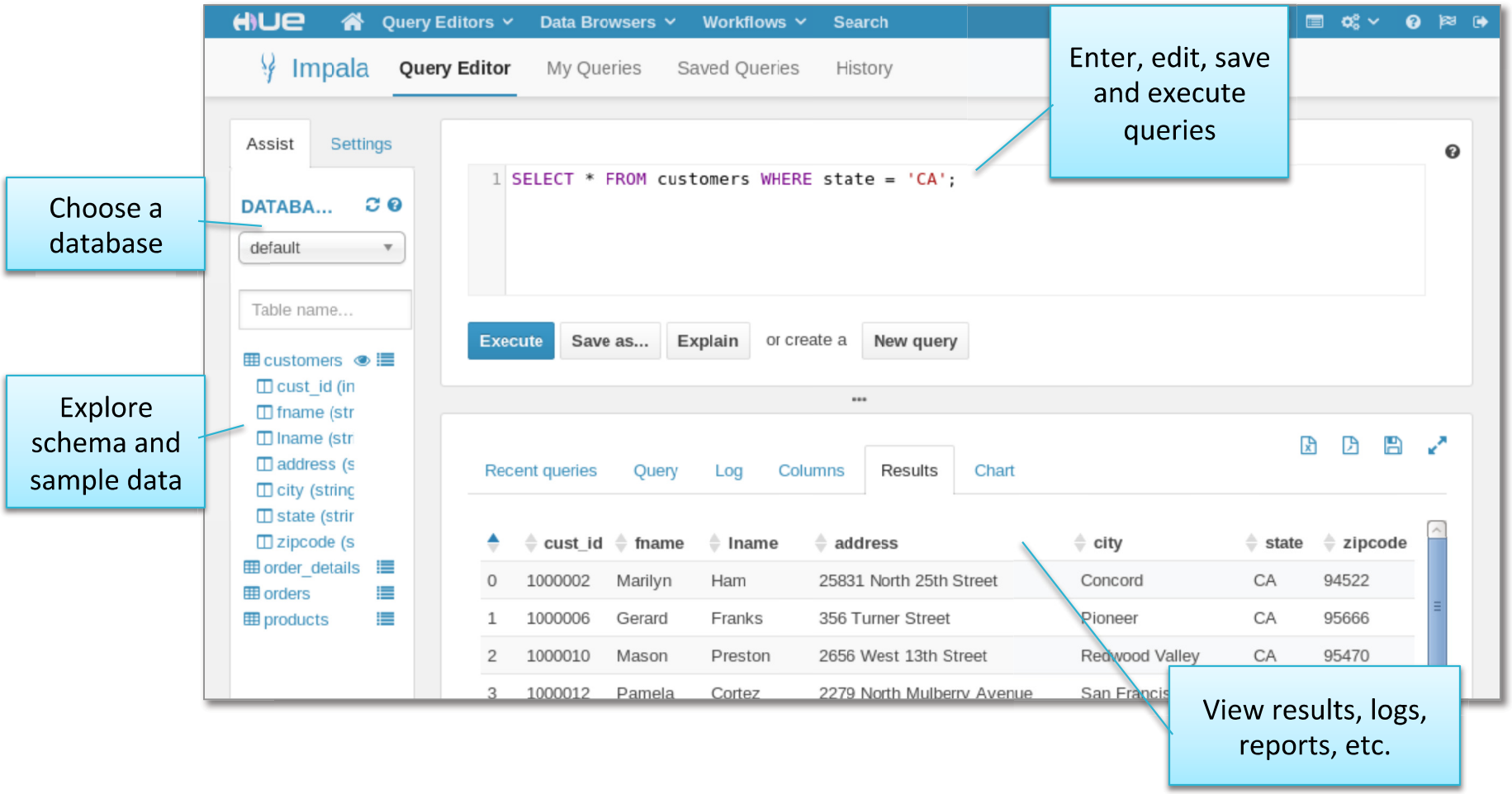


View and manage the Metastore



The Hue Query Editor

- **The Impala and Hive Query editors are nearly identical**



Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- **Using the Impala Shell**
- Using Beeline (Hive's Shell)
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Starting the Impala Shell

- **You can execute statements in the Impala shell**
  - This interactive tool is similar to the shell in MySQL
- **Execute the `impala-shell` command to start the shell**
  - Some log messages truncated to better fit the slide

```
$ impala-shell
Connected to localhost.localdomain:21000
Server version: impalad version 1.4.0-cdh5 (…)
Welcome to the Impala shell.
[localhost.localdomain:21000] >
```

- **Use `-i hostname:port` option to connect to another server**

```
$ impala-shell -i myserver.example.com:21000
[myserver.example.com:21000] >
```

Using the Impala Shell

- **Enter semicolon-terminated statements at the prompt**
  - Hit [Enter] to execute a query or command
  - Use the `quit` command to exit the shell
- **Use `impala-shell --help` for a full list of options**

Executing Queries in the Impala Shell

```
> SELECT lname,fname FROM customers WHERE state = 'CA'
limit 50;

Query: select lname,fname FROM customers WHERE state =
'CA' limit 50
+------------+------------+
| lname      | fname      |
+------------+------------+
| Ham        | Marilyn    |
| Franks     | Gerard     |
| Preston    | Mason      |
| Cortez     | Pamela     |
…
| Falgoust   | Jennifer   |
+------------+------------+
Returned 50 row(s) in 0.17s

>
```

Note: shell prompt abbreviated as **>**

Interacting with the Operating System

- **Use `shell` to execute system commands from within Impala shell**

```
> shell date;
Mon May 20 16:44:35 PDT 2013
```

- **No direct support for HDFS commands**
  - But could run `hdfs dfs` using `shell`

```
> shell hdfs dfs -mkdir /reports/sales/2013;
```

Running Impala Queries from the Command Line

- **You can execute a file containing queries using the `-f` option**

```
$ impala-shell -f myquery.hql
```

- **Run queries directly from the command line with the `-q` option**

```
$ impala-shell -q 'SELECT * FROM users'
```

- **Use `-o` to capture output to file**
  - Optionally specify delimiter

```
$ impala-shell -f myquery.hql \
      -o results.txt \
      --output_file_field_delim='\t'
```

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- **Using Beeline (Hive's Shell)**
- Conclusion
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Starting Beeline (Hive's Shell)

- **You can execute HiveQL statements in the Beeline shell**
  - Interactive shell based on the SQLLine utility
  - Similar to the Impala shell
- **Start Beeline by specifying the URL for a Hive2 server**
  - Plus username and password if required

```
$ beeline -u jdbc:hive2://host:10000 \
-n username -p password

0: jdbc:hive2://localhost:10000>
```

Executing Queries in Beeline

- **SQL commands are terminated with semi-colon (;)**
- **Similar to Impala shell**
  - Results formatting is slightly different

```
1: url> SELECT lname,fname FROM customers
. . . > WHERE state = 'CA' LIMIT 50;

+-------------+-------------+
|    lname    |    fname    |
+-------------+-------------+
| Ham         | Marilyn     |
| Franks      | Gerard      |
| Preston     | Mason       |
...
| Falgoust    | Jennifer    |
+-------------+-------------+
50 rows selected (15.829 seconds)

1: url>
```

Using Beeline

- **Execute Beeline commands with '!'**
  - No terminator character
- **Some commands**
  - `!connect` *`url`* – connect to a different Hive2 server
  - `!exit` – exit the shell
  - `!help` – show the full list of commands
  - `!verbose` – show added details of queries

```
0: jdbc:hive2://localhost:10000> !exit
```

Executing Hive Queries from the Command Line

- **You can also execute a file containing HiveQL code using the `-f` option**

```
$ beeline -u ... -f myquery.hql
```

- **Or use HiveQL directly from the command line using the `-e` option**

```
$ beeline -u ... -e 'SELECT * FROM users'
```

- **Use the `--silent` option to suppress informational messages**
  - Can also be used with `-e` or `-f` options

```
$ beeline -u ... --silent
```

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- **Conclusion**
- Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue

Essential Points

- **HiveQL and Impala SQL syntax are familiar to those who know SQL**
  - A subset of SQL-92, plus extensions
- **Every table belongs to exactly one database**
  - The `SHOW DATABASES` command lists databases
  - The `USE` command switches the active database
  - The `SHOW TABLES` command lists all tables in a database
- **Every column in a table has an associated data type**
  - Most simple column types are similar to SQL
- **Hive and Impala both have command line shells and Query Editors in Hue**

Bibliography

The following offer more information on topics discussed in this chapter

- **HiveQL Language Manual on the Hive Wiki**
  - `http://tiny.cloudera.com/hqlmanual`
- **Impala Documentation at Cloudera Web site**
  - `http://tiny.cloudera.com/cdh5impala`
- **Beeline CLI reference**
  - `http://sqlline.sourceforge.net`

Chapter Topics

Querying with Impala and Hive | **Introduction to Impala and Hive**

- Databases and Tables
- Basic Hive and Impala Query Language Syntax
- Data Types
- Differences Between Impala and Hive Query Syntax
- Using Hue to Execute Queries
- Using the Impala Shell
- Using Beeline (Hive's Shell)
- Conclusion
- **Hands-On Exercise: Running Queries from the Shell, Scripts, and Hue**

Hands-on Exercise: Running Queries from the Shell, Scripts, and Hue

- **In this Hands-On Exercise, you will run queries from the Impala and Hive shells, scripts, and Hue**
- **Please refer to the Hands-On Exercise Manual for instructions**

cloudera®
Ask Bigger Questions

Data Management

Chapter 10

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ Querying With Impala and Hive ▪ **Impala and Hive Data Management** ▪ Data Storage and Performance	**Introduction to Impala and Hive**
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ Conclusion	Course Conclusion

Data Management

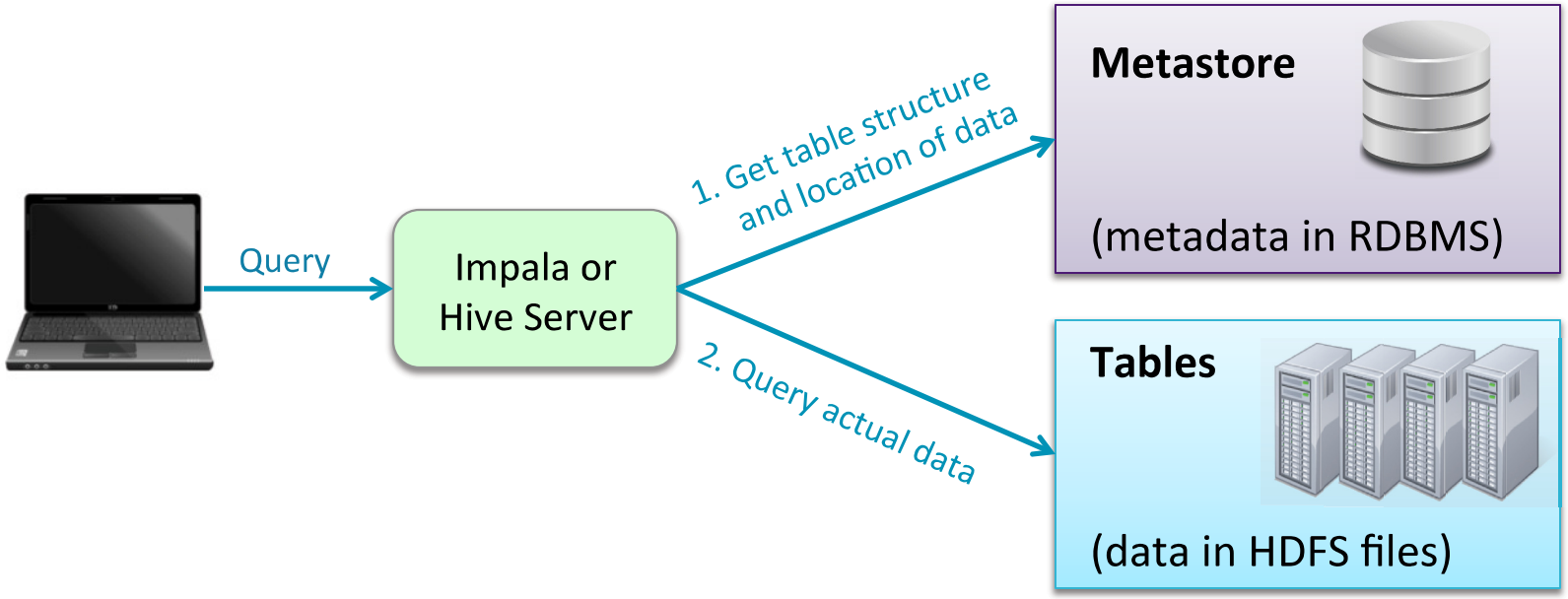
In this chapter you will learn

- **How Hive and Impala encode and store data**
- **How to create databases, tables, and views**
- **How to load data into tables**
- **How to alter and remove tables**
- **How to save query results into tables and files**

Chapter Topics

Recap: How Hive and Impala Load and Store Data

- **Hive and Impala use the metastore to determine data format and location**
  - The query itself operates on data stored in a filesystem (typically HDFS)

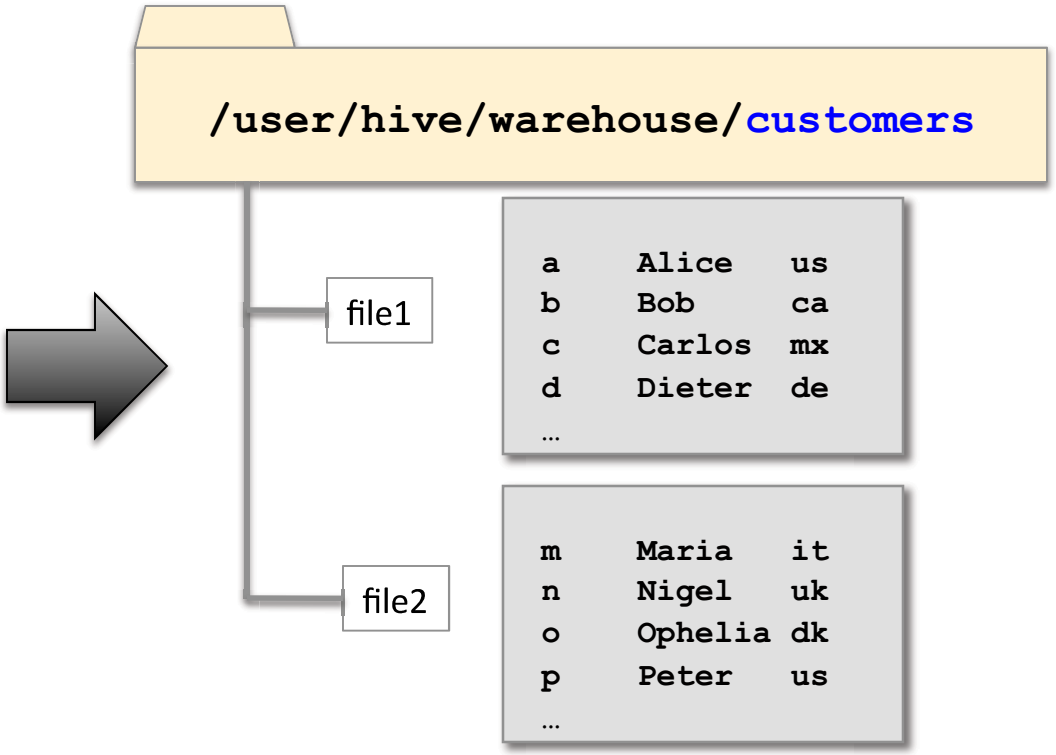


The Data Warehouse Directory

- **By default, data is stored in the HDFS directory `/user/hive/warehouse`**
- **Each table is a subdirectory containing any number of files**

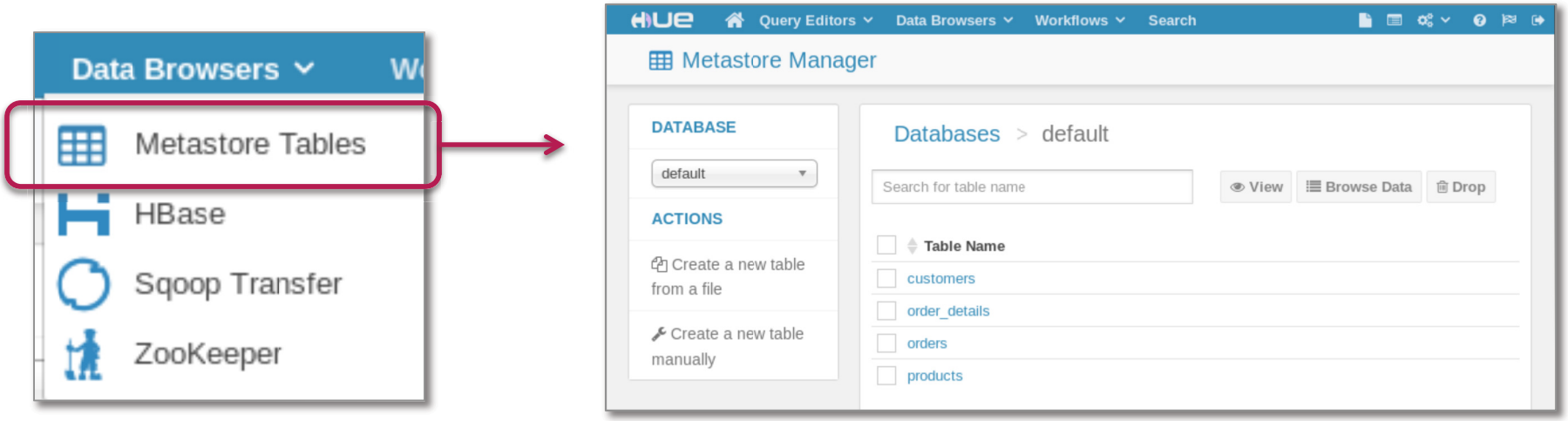
customers table

cust_id	name	country
001	Alice	us
002	Bob	ca
003	Carlos	mx
…	…	…
392	Maria	it
393	Nigel	uk
394	Ophelia	dk
…	…	…



Using the Hue Metastore Manager

- **The Hue Metastore Manager**
  - An alternative to using SQL commands to manage metadata
  - Allows you to create, load, preview, and delete databases and tables



Chapter Topics

Impala and Hive Data Management | **Introduction to Impala and Hive**

- Data Storage
- **Creating Databases and Tables**
- Loading Data
- Altering Databases and Tables
- Simplifying Queries with Views
- Storing Query Results
- Conclusion
- Hands-On Exercise: Data Management

Creating a Database

- **Hive and Impala databases are simply namespaces**
  - Helps to organize your tables
- **To create a new database**

```
CREATE DATABASE dualcore;
```

1. Adds the database definition to the metastore
2. Creates a storage directory in HDFS

   e.g. `/user/hive/warehouse/dualcore.db`

- **To conditionally create a new database**
  - Avoids error in case database already exists (useful for scripting)

```
CREATE DATABASE IF NOT EXISTS dualcore;
```

Creating a Table (1)

- **Basic syntax for creating a table:**

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|…}
```

- **Creates a subdirectory in the database's warehouse directory in HDFS**
  - Default database:
    `/user/hive/warehouse/tablename`
  - Named database:
    `/user/hive/warehouse/dbname.db/tablename`

Creating a Table (2)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELD
  STORE
```

Specify a name for the table, and list the column names and datatypes (see later)

Creating a Table (3)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORE
```

This line states that fields in each file in the table's directory are delimited by some character. Hive's default delimiter is Control-A, but you may specify an alternate delimiter...

Creating a Table (4)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|…}
```

…for example, tab-delimited data would require that you specify `FIELDS TERMINATED BY '\t'`

Creating a Table (5)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|…}
```

Finally, you may declare the file format. `STORED AS TEXTFILE` is the default and does not need to be specified

Example Table Definition

- **The following example creates a new table named `jobs`**
  - Data stored as text with four comma-separated fields per line

```
CREATE TABLE jobs (
    id INT,
    title STRING,
    salary INT,
    posted TIMESTAMP
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
```

  - Example of corresponding record for the table above

```
1,Data Analyst,100000,2013-06-21 15:52:03
```

Creating Tables Based On Existing Schema

- **Use `LIKE` to create a new table based on an existing table definition**

```
CREATE TABLE jobs_archived LIKE jobs;
```

- **Column definitions and names are derived from the existing table**
  - New table will contain no data

Controlling Table Data Location

- **By default, table data is stored in the warehouse directory**
- **This is not always ideal**
  - Data might be shared by several users
- **Use `LOCATION` to specify the directory where table data resides**

```
CREATE TABLE jobs (
    id INT,
    title STRING,
    salary INT,
    posted TIMESTAMP
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/dualcore/jobs';
```

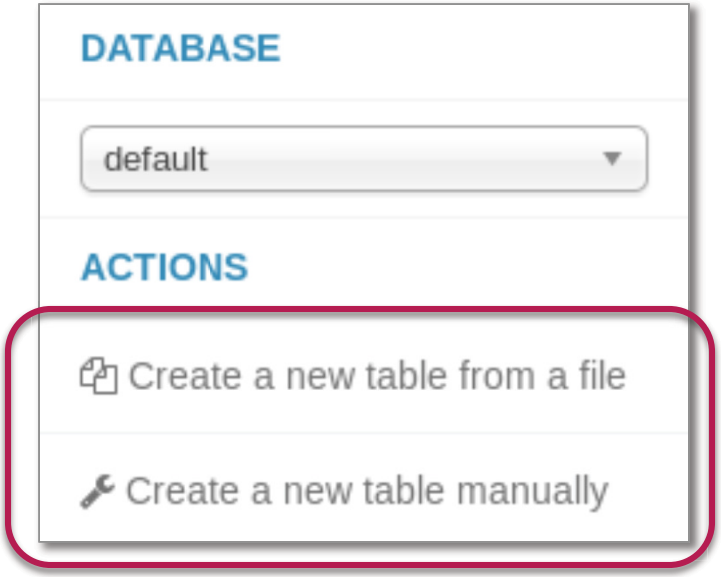
Externally Managed Tables

- **CAUTION: Dropping a table removes its data in HDFS**
- **Using `EXTERNAL` when creating the table avoids this behavior**
  - Dropping an *external* table removes only its *metadata*

```
CREATE EXTERNAL TABLE adclicks
  ( campaign_id STRING,
    click_time TIMESTAMP,
    keyword STRING,
    site STRING,
    placement STRING,
    was_clicked BOOLEAN,
    cost SMALLINT)
  LOCATION '/dualcore/ad_data';
```

Creating Tables Using the Metastore Manager

- **The Metastore Manager provides a table creation wizard**
- **Supports most but not all table options**



Chapter Topics

Data Validation

- **Impala and Hive are 'schema on read'**
  - Unlike an RDBMS, they do not validate data on insert
    - Files are simply moved into place
  - Loading data into tables is therefore very fast
  - Errors in file format will be discovered when queries are performed
- **Missing or invalid data will be represented as `NULL`**

Loading Data From HDFS Files

- **To load data, simply add files to the table's directory in HDFS**
  - Can be done directly using the `hdfs dfs` commands
  - This example loads data from HDFS into the `sales` table

```
$ hdfs dfs -mv sales.txt /user/hive/warehouse/sales/
```

- **Alternatively, use the `LOAD DATA INPATH` command**
  - Done from within Hive or Impala
  - This *moves* data within HDFS, just like the command above
  - Source can be either a file or directory

```
LOAD DATA INPATH 'sales.txt'
  INTO TABLE sales;
```

Overwriting Data From Files

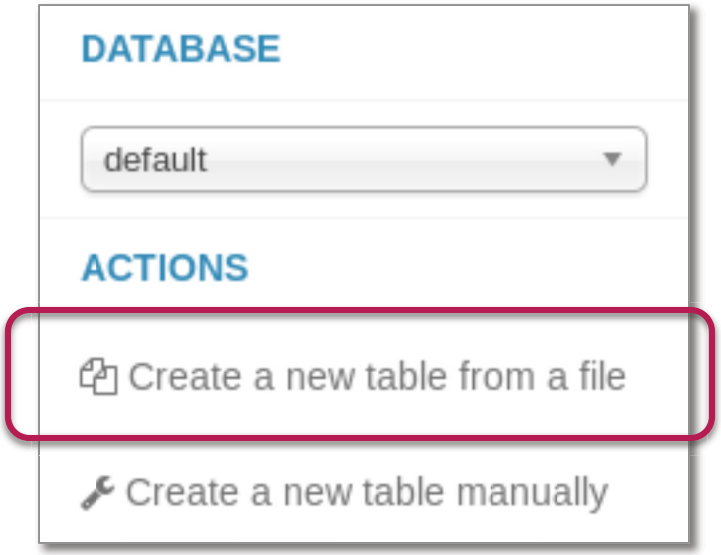
- **Add the `OVERWRITE` keyword to delete all records before import**
  - Removes all files within the table's directory
  - Then moves the new files into that directory

```
LOAD DATA INPATH 'sales.txt'
  OVERWRITE INTO TABLE sales;
```

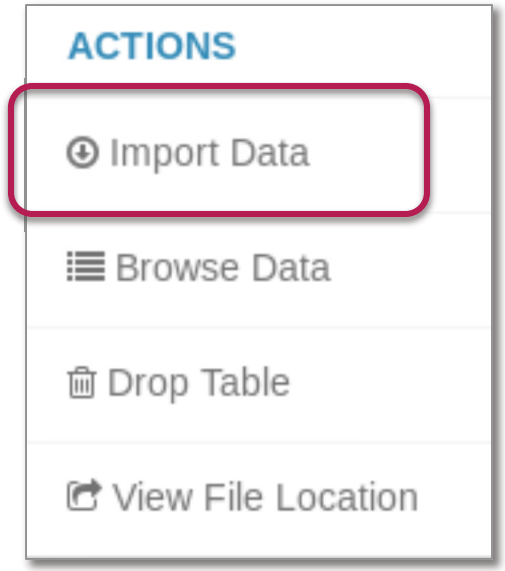
Loading Data Using the Metastore Manager

- **The Metastore Manager provides two ways to load data into a table**

Table creation wizard



Import data wizard



Loading Data From a Relational Database

- **Sqoop has built-in support for importing data into Hive and Impala**
- **Add the `--hive-import` option to your Sqoop command**
  - Creates the table in the Hive metastore
  - Imports data from the RDBMS to the table's directory in HDFS

```
$ sqoop import \
  --connect jdbc:mysql://localhost/dualcore \
  --username training \
  --password training \
  --fields-terminated-by '\t' \
  --table employees \
  --hive-import
```

Chapter Topics

Removing a Database

- **Removing a database is similar to creating it**
  - Just replace **CREATE** with **DROP**

```
DROP DATABASE dualcore;
```

```
DROP DATABASE IF EXISTS dualcore;
```

- **These commands will fail if the database contains tables**
  - In Hive: Add the **CASCADE** keyword to force removal
  - Caution: this command removes data in HDFS!


```
DROP DATABASE dualcore CASCADE;
```

Removing a Table

- **Table removal syntax is similar to database removal**

```
DROP TABLE customers;
```

```
DROP TABLE IF EXISTS customers;
```

- **Regular (managed) tables**
  - Metadata is removed
  - Data in HDFS is removed
  - *Caution: No rollback or undo feature!*
- **External tables**
  - Metadata is removed
  - Data in HDFS is *not* removed

Renaming Tables and Columns

- **Use `ALTER TABLE` to modify a table's metadata**
  - This command does not change the data in HDFS
- **Rename an existing table**

```
ALTER TABLE customers RENAME TO clients;
```

- **Rename a column by specifying its old and new names**
  - Type must be specified even if it is unchanged

```
ALTER TABLE clients
  CHANGE fname first_name STRING;
```

Old Name New Name Type

Modifying and Adding Columns

- **You can also modify a column's type**
  - The old and new column names will be the same
  - You must ensure data in HDFS conforms to the new type

```
ALTER TABLE jobs CHANGE salary salary BIGINT;
```

Old Name New Name Type

- **Add new columns to a table**
  - Appended to the end of any existing columns
  - Existing data will have **NULL** values for new columns

```
ALTER TABLE jobs
  ADD COLUMNS (city STRING, bonus INT);
```

Reordering and Removing Columns

- **Use `AFTER` or `FIRST` to reorder columns**
  - Ensure that your data in HDFS matches the new order

```
ALTER TABLE jobs
  CHANGE bonus bonus INT AFTER salary;
```

```
ALTER TABLE jobs
  CHANGE bonus bonus INT FIRST;
```

- **Use `REPLACE COLUMNS` to change the entire table's column definitions**
  - Any column not listed will be dropped from the metadata

```
ALTER TABLE jobs REPLACE COLUMNS
  (id INT,
   title STRING,
   salary INT);
```

Chapter Topics

Impala and Hive Data Management | **Introduction to Impala and Hive**

- Data Storage
- Creating Databases and Tables
- Loading Data
- Altering Databases and Tables
- **Simplifying Queries with Views**
- Storing Query Results
- Conclusion
- Hands-On Exercise: Data Management

Simplifying Complex Queries

- **Complex queries can become cumbersome**
  - Imagine typing this several times for different orders

```
SELECT o.order_id, order_date, p.prod_id, brand, name
  FROM orders o
  JOIN order_details d
  ON (o.order_id = d.order_id)
  JOIN products p
  ON (d.prod_id = p.prod_id)
  WHERE o.order_id=6584288;
```

Creating Views

- **Views are conceptually like a table, but backed by a query**
  - You cannot directly add data to a view

```
CREATE VIEW order_info AS
  SELECT o.order_id, order_date, p.prod_id, brand, name
  FROM orders o
  JOIN order_details d
  ON (o.order_id = d.order_id)
  JOIN products p
  ON (d.prod_id = p.prod_id);
```

- **Our query is now greatly simplified**

```
SELECT * FROM order_info WHERE order_id=6584288;
```

Inspecting and Removing Views

- Use `DESCRIBE FORMATTED` to see the underlying query (Hive only)


```
DESCRIBE FORMATTED order_info;
```

- Use `DROP VIEW` to remove a view

```
DROP VIEW order_info;
```

Chapter Topics

Saving Query Output to a Table

- **`SELECT` statements display their results on screen**
- **To save results to a table, use `INSERT OVERWRITE TABLE`**
  - Destination table must already exist
  - Existing contents will be deleted

```
INSERT OVERWRITE TABLE ny_customers
  SELECT * FROM customers
  WHERE state = 'NY';
```

- **`INSERT INTO TABLE` adds records without first deleting existing data**

```
INSERT INTO TABLE ny_customers
  SELECT * FROM customers
  WHERE state = 'NJ' OR state = 'CT';
```

Creating Tables Based On Existing Data

- **Create a table based on a `SELECT` statement**
  - Often know as 'Create Table As Select' (CTAS)

```
CREATE TABLE ny_customers AS
  SELECT cust_id, fname, lname
  FROM customers
  WHERE state = 'NY';
```

- **Column definitions are derived from the existing table**
- **Column names are inherited from the existing names**
  - Use aliases in the `SELECT` statement to specify new names

Writing Output To HDFS in Hive

- **Hive also lets you save output to a file in HDFS**

```
INSERT OVERWRITE DIRECTORY '/dualcore/ny/'
  SELECT * FROM customers
  WHERE state = 'NY';
```

- **Produces text files delimited by Ctrl-A characters**
  - Workaround: Create an *external* table with delimiters of your choices, save the results to the table, then delete the table

Saving to Multiple Directories in Hive (1)

- **We just saw that you can save output to an HDFS file**

```
INSERT OVERWRITE DIRECTORY 'ny_customers'
  SELECT cust_id, fname, lname
    FROM customers WHERE state = 'NY';
```

- **This query could also be written as follows**

```
FROM customers c
  INSERT OVERWRITE DIRECTORY 'ny_customers'
    SELECT cust_id, fname, lname WHERE state='NY';
```

Saving to Multiple Directories in Hive (2)

- **We sometimes need to extract data to multiple files**
  - Hive `SELECT` queries can take a long time to complete
- **Hive allows us to do this with a single query**
  - Much more efficient than using multiple queries
  - Result is two directories in HDFS

```
FROM customers c
  INSERT OVERWRITE DIRECTORY 'ny_names'
    SELECT fname, lname
      WHERE state = 'NY'
  INSERT OVERWRITE DIRECTORY 'ny_count'
    SELECT count(DISTINCT cust_id)
      WHERE state = 'NY';
```

Saving to Multiple Directories in Hive (3)

- **The following query produces the same result**

```
FROM (SELECT * FROM customers WHERE state='NY') nycust
  INSERT OVERWRITE DIRECTORY 'ny_names'
    SELECT fname, lname
  INSERT OVERWRITE DIRECTORY 'ny_count'
    SELECT count(DISTINCT cust_id);
```

Chapter Topics

Impala and Hive Data Management | **Introduction to Impala and Hive**

- Data Storage
- Creating Databases and Tables
- Loading Data
- Altering Databases and Tables
- Simplifying Queries with Views
- Storing Query Results
- **Conclusion**
- Hands-On Exercise: Data Management

Essential Points

- **Each table maps to a directory in HDFS**
  - Table data is stored as one or more files
  - Default format: plain text with delimited fields
- **The Hue Metastore Manager can create and load simple tables**
- **Dropping managed tables deletes data in HDFS**
  - External tables require manual data deletion
- **`ALTER TABLE` is used to add, modify, and remove columns**
- **Views can help to simplify complex and repetitive queries**

Chapter Topics

Impala and Hive Data Management | **Introduction to Impala and Hive**

- Data Storage
- Creating Databases and Tables
- Loading Data
- Altering Databases and Tables
- Simplifying Queries with Views
- Storing Query Results
- Conclusion
- **Hands-On Exercise: Data Management**

Hands-On Exercise: Data Management

- **In this Hands-On Exercise, you will create and load tables using the Hue Metastore Manager, Impala SQL, and Sqoop**
- **Please refer to the Hands-On Exercise Manual for instructions**

cloudera®
Ask Bigger Questions

Data Storage and Performance

Chapter 11

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ Querying With Impala and Hive ▪ Impala and Hive Data Management ▪ **Data Storage and Performance**	**Introduction to Impala and Hive**
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ Conclusion	Course Conclusion

Data Storage and Performance

In this chapter you will learn

- **How to improve query performance with partitioning**
- **How to choose the best file format for your use case**
- **How to use HCatalog to manage metadata**
- **Available options for access control**

Chapter Topics

Table Partitioning

- **By default, all data files are stored in a single directory**
  - *All* files in the directory are read during a query
- **Partitioning subdivides the data**
  - Data is physically divided during loading, based on values from one or more columns
- **Speeds up queries that involve partition columns**
  - Only the files containing the selected data need to be read
  - Queries that filter on partitioned fields limit the amount of data read
- **Does not prevent you from running queries that span partitions**

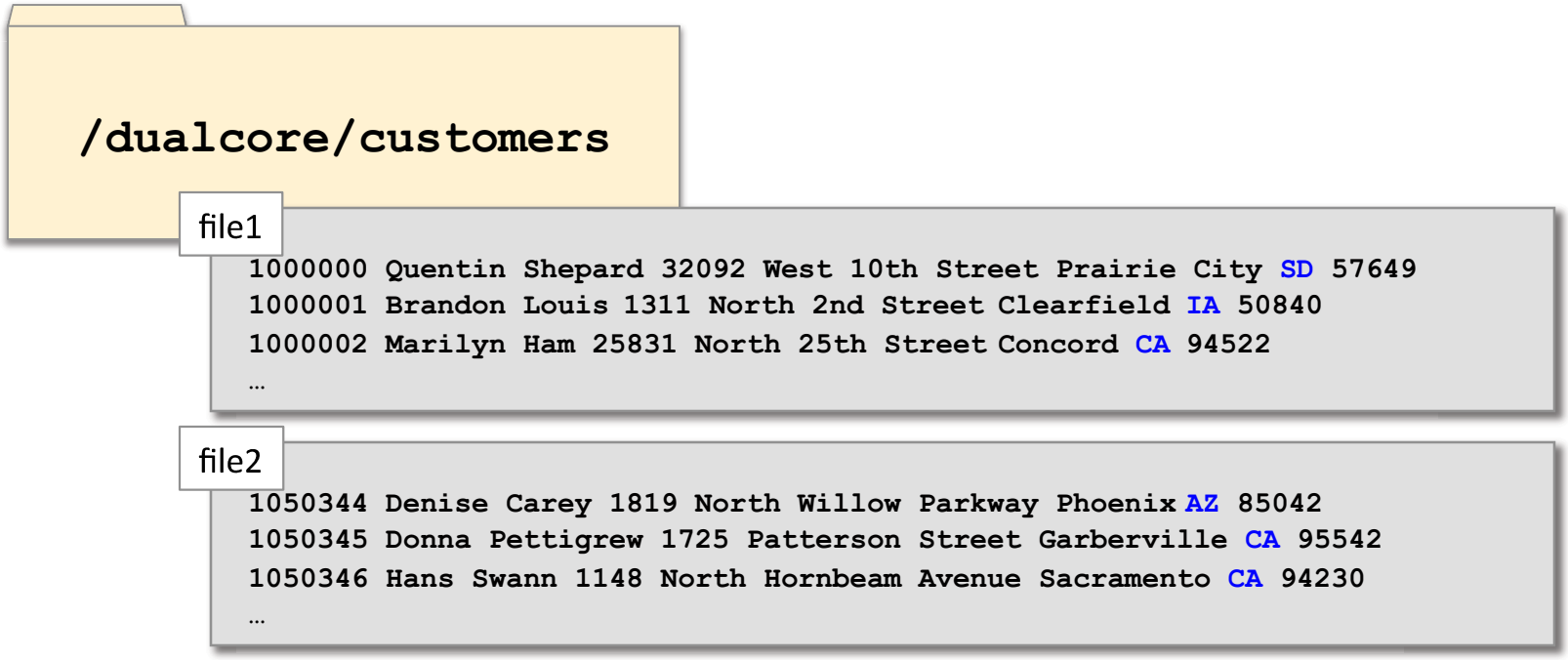
Example: Partitioning Customers By State (1)

- **Example: `customers` is a non-partitioned table**

```
CREATE EXTERNAL TABLE customers (
    cust_id int,
    fname string,
    lname string,
    address string,
    city string,
    state string,
    zipcode string)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LOCATION '/dualcore/customers';
```

Example: Partitioning Customers By State (2)

- **Data files are stored in a single directory**
- **All files are scanned for every query**



Example: Partitioning Customers By State (3)

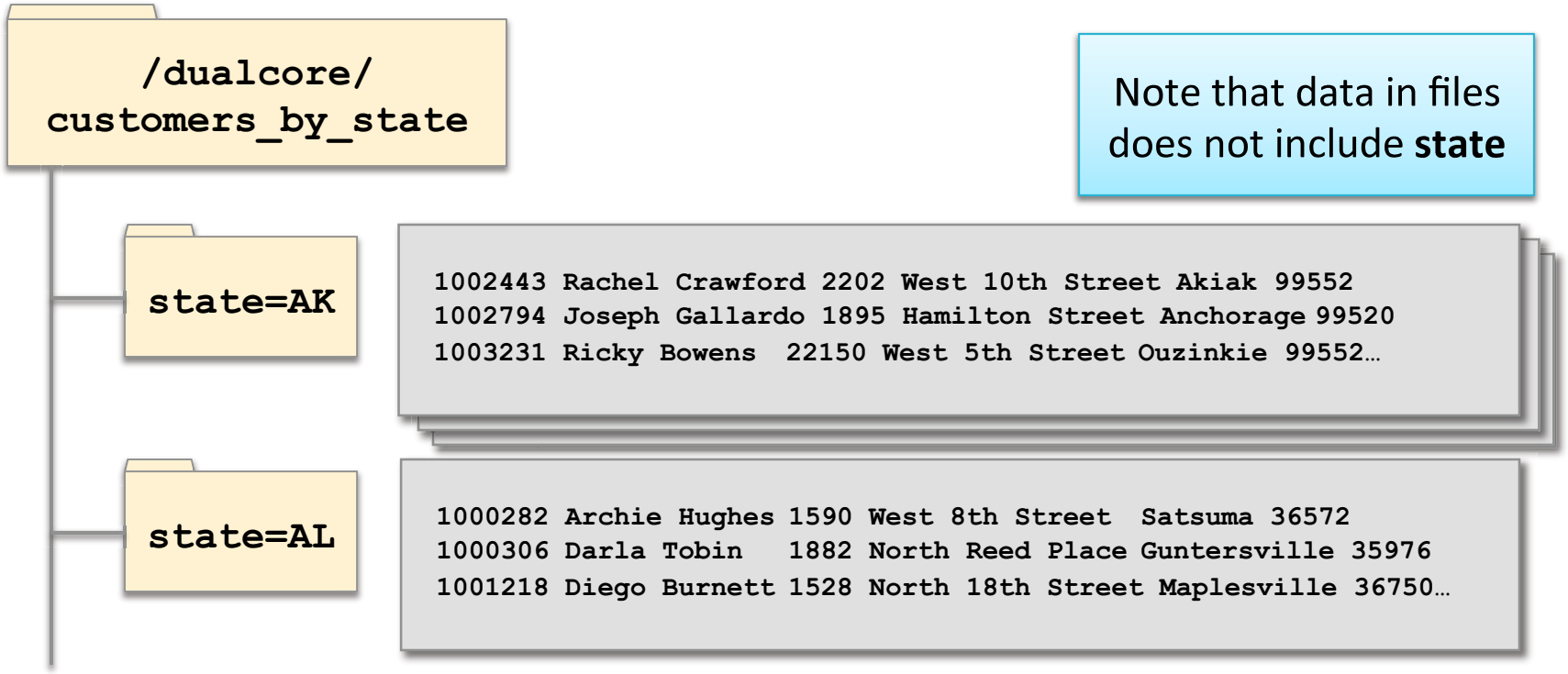
- **What if most of Dualcore's analysis on the customer table was done by state? For example:**

```
SELECT o.order_date, c.fname, c.lname
  FROM customers c JOIN orders o
  ON (c.cust_id = o.cust_id)
  WHERE c.state='NY';
```

- **By default, all queries have to scan *all* files in the directory**
- **Use partitioning to store data in separate files by state**
  - State-based queries scan only the relevant files

Partitioning File Structure

- **Partitioned tables store data in subdirectories**
  - Queries that filter on partitioned fields limit amount of data read



Creating a Partitioned Table

- **Create a partitioned table using PARTITIONED BY**

```
CREATE EXTERNAL TABLE customers_by_state (
    cust_id int,
    fname string,
    lname string,
    address string,
    city string,
    zipcode string)
  PARTITIONED BY (state STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LOCATION '/dualcore/customers_by_state';
```

Partition Columns

- **The partition column is displayed if you `DESCRIBE` the table**

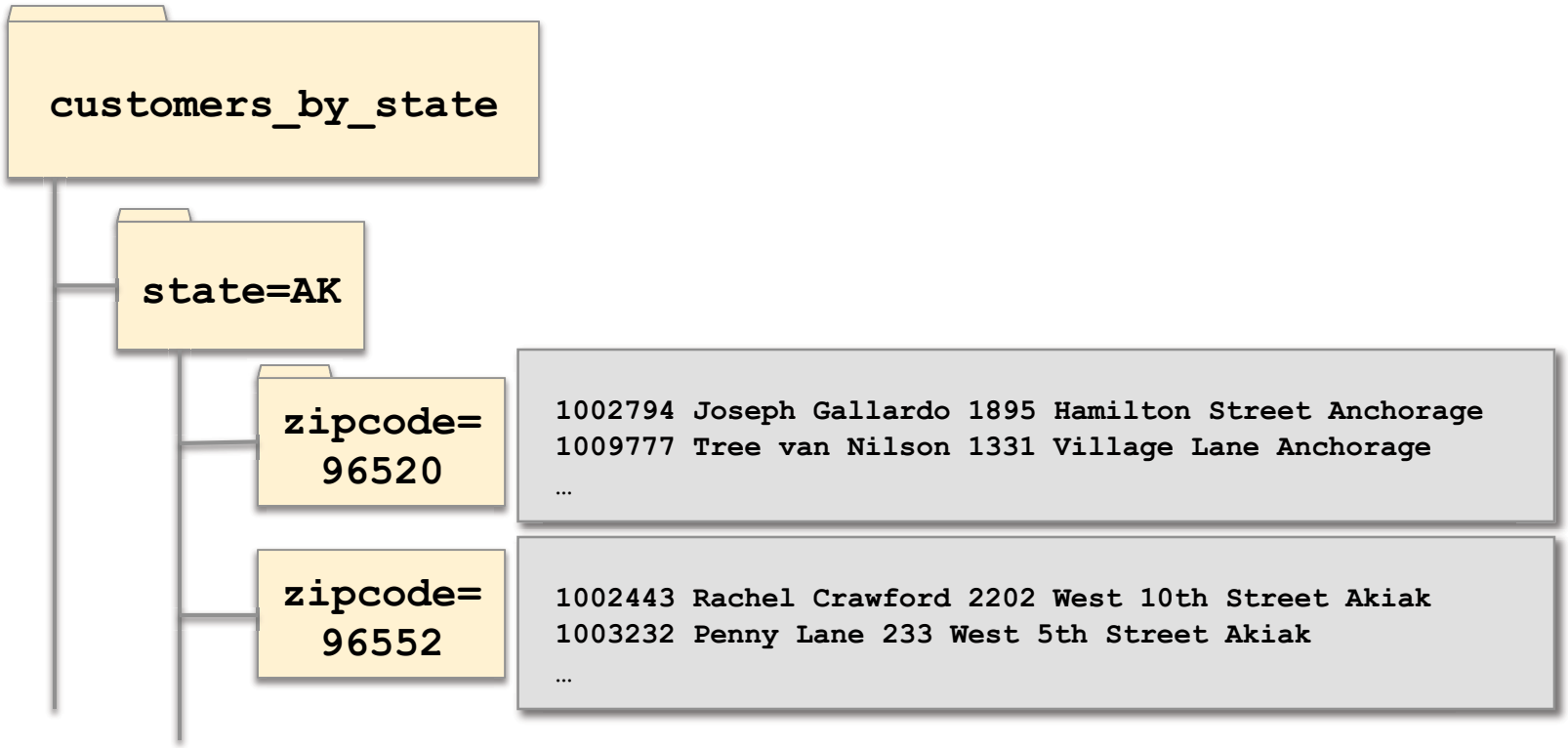
```
DESCRIBE customers_by_state;
+---------+--------+---------+
| name    | type   | comment |
+---------+--------+---------+
| cust_id | int    |         |
| fname   | string |         |
| lname   | string |         |
| address | string |         |
| city    | string |         |
| zipcode | string |         |
| state   | string |         |
+---------+--------+---------+
```

A partition column is a "virtual column"; data is not stored in the file

Nested Partitions

- **You can also created nested partitions**

```
… PARTITIONED BY (state STRING, zipcode STRING)
```



Loading Data Into a Partitioned Table

- **Dynamic partitioning**
  - Impala/Hive add new partitions automatically as needed at load time
  - Data is stored into the correct partition (subdirectory) based on column value
- **Static partitioning**
  - You define new partitions using `ADD PARTITION`
  - When loading data, you specify which partition to store data in

Dynamic Partitioning

- **We can create new partitions dynamically from existing data**

```
INSERT OVERWRITE TABLE customers_by_state
  PARTITION(state)
  SELECT cust_id, fname, lname, address,
    city, zipcode, state FROM customers;
```

- **Partitions are automatically created based on the value of the *last* column**
  - If the partition does not already exist, it will be created
  - If the partition does exist, it will be overwritten

Static Partitioning Example: Partition Calls by Day (1)

- **Dualcore's customer service phone system generates logs detailing calls received**
  - Analysts use this data to summarize previous days' calls
  - e.g.

```
SELECT event_type, COUNT(event_type)
  FROM call_log
  WHERE call_date = '2014-10-01'
  GROUP BY event_type;
```

Static Partitioning Example: Partition Calls by Day (2)

- **Logs are generated daily, e.g.**

call-20141001.log

```
19:45:19,312-555-7834,CALL_RECEIVED
19:45:23,312-555-7834,OPTION_SELECTED,Shipping
19:46:23,312-555-7834,ON_HOLD
19:47:51,312-555-7834,AGENT_ANSWER,Agent ID N7501
19:48:37,312-555-7834,COMPLAINT,Item not received
19:48:41,312-555-7834,CALL_END,Duration: 3:22
…
```

call-20141002.log

```
03:45:01,505-555-2345,CALL_RECEIVED
03:45:09,505-555-2345,OPTION_SELECTED,Billing
03:56:21,505-555-2345,AGENT_ANSWER,Agent ID A1503
03:57:01,505-555-2345,QUESTION
…
```

Static Partitioning Example: Partition Calls by Day (3)

- **In the previous example, existing data was partitioned dynamically based on a column value**
- **This time we use static partitioning**
  - Because the data files do not include the partitioning data

Static Partitioning Example: Partition Calls by Day (4)

- **The partitioned table is defined the same way**

```
CREATE TABLE call_logs (
    call_time STRING,
    phone STRING,
    event_type STRING,
    details STRING)
  PARTITIONED BY (call_date STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
```

Loading Data Into Static Partitions (1)

- **With static partitioning, you create new partitions as needed**
- **e.g. For each new day of call log data, add a partition:**

```
ALTER TABLE call_logs
  ADD PARTITION (call_date='2014-10-01');
```

- **This command**
  1. Adds the partition to the table's meta-data
  2. Creates subdirectory
     `/user/hive/warehouse/call_logs/call_date=2014-10-02`

Loading Data Into Static Partitions (2)

- **Then load the day's data into the correct partition**

```
LOAD DATA INPATH 'call-20141001.log'
  INTO TABLE call_logs
  PARTITION(call_date='2014-10-01');
```

  - This command moves the HDFS file `call-20141002.log` to the partition subdirectory

- **To overwrite all data in a partition**

```
LOAD DATA INPATH 'call-20141001.log'
  INTO TABLE call_logs OVERWRITE
  PARTITION(call_date='2014-10-01');
```

Hive Only: Shortcut for Loading Data Into Partitions

- **Hive will create a new partition if the one specified doesn't exist**

```
LOAD DATA INPATH 'call-20141002.log'
  INTO TABLE call_logs
  PARTITION(call_date='2014-10-01');
```

- **This command**
  1. Adds the partition to the table's metadata if it doesn't exist
  2. Creates subdirectory `/user/hive/warehouse/call_logs/call_date=2014-10-02` if it doesn't exist
  3. Moves the HDFS file `call-20141002.log` to the partition subdirectory

Viewing, Adding, and Removing Partitions

- **To view the current partitions in a table**

```
SHOW PARTITIONS call_logs;
```

- **Use `ALTER TABLE` to add or drop partitions**

```
ALTER TABLE call_logs
   ADD PARTITION (call_date='2013-06-05')
   LOCATION '/dualcore/call_logs/call_date=2013-06-05';
```

```
ALTER TABLE call_logs
   DROP PARTITION (call_date='2013-06-06');
```

When To Use Partitioning

- **Use partitioning for tables when**
  - Reading the entire data set takes too long
  - Queries almost always filter on the partition columns
  - There are a reasonable number of different values for partition columns
  - The data generation or ETL process segments data by file or directory names
    - Partition column values are not in the data itself

When *Not* To Use Partitioning

- **Avoid partitioning data into numerous small data files**
  - Partition on columns with too many unique values
- **Caution: This can happen easily when using dynamic partitioning!**
  - For example, partitioning customers by first name could produce thousands of partitions

Partitioning in Hive (1)

- **In older versions of Hive, dynamic partitioning is not enabled by default**
  - Enable it by setting these two properties

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

- **Note: Hive variables set in Beeline are for the current session only.**
  - Your system administrator can configure settings permanently

Partitioning in Hive (2)

- **Caution: if the partition column has many unique values, many partitions will be created**
- **Three Hive configuration properties exist to limit this**
  - `hive.exec.max.dynamic.partitions.pernode`
    - Maximum number of dynamic partitions that can be created by any given node involved in a query
    - Default 100
  - `hive.exec.max.dynamic.partitions`
    - Total number of dynamic partitions that can be created by one HiveQL statement
    - Default 1000
  - `hive.exec.max.created.files`
    - Maximum total files (on all nodes) created by a query
    - Default 100000

Chapter Topics

Choosing a File Format

- **Hive and Impala support many different formats for data storage**

```
CREATE TABLE tablename (colname DATATYPE, ...)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY char
   STORED AS format
```

- **Format options**
  - **TEXTFILE**
  - **SEQUENCEFILE**
  - **PARQUET**
  - **AVRO**
  - **RCFILE**
  - **ORCFILE** (Hive only)

Considerations for Choosing a File Format

- **Hadoop and its ecosystem support many file formats**
  - May ingest data in one format, but convert to another as needed
- **Selecting the format for your data set involves several considerations**
  - Ingest pattern
  - Tool compatibility
  - Expected lifetime
  - Storage and performance requirements
- **Which format is best? It depends on *your* data and use cases**
  - The following slides offer some general guidance on common formats

Text File Format (1)

- **Default data format is plain text**
  - One record per line (record separator is `\n`)
  - Columns are delimited by `^A` (Control-A) by default
- **You can override these delimiters**
  - Specify alternate values during table creation

Text File Format (2)

- **Text files are the most basic file type in Hadoop**
  - Can be read or written from virtually any programming language
  - Comma- and tab-delimited files are compatible with many applications
- **Text files are human readable, since everything is a string**
  - Useful when debugging
- **At scale, this format is inefficient**
  - Representing numeric values as strings wastes storage space
  - Difficult to represent binary data such as images
    - Often resort to techniques such as Base64 encoding
  - Conversion to/from native types adds performance penalty
- **Verdict: Good interoperability, but poor performance**

SequenceFile Format

- **SequenceFiles store key-value pairs in a binary container format**
  - Less verbose and more efficient than text files
  - Capable of storing binary data such as images
  - Format is Java-specific and tightly coupled to Hadoop
- **Verdict: Good performance, but poor interoperability**

Avro File Format

- **Apache Avro is an efficient data serialization framework**
- **Avro also defines a data file format for storing Avro records**
  - Similar to SequenceFile format
- **Efficient storage due to optimized binary encoding**
- **Widely supported throughout the Hadoop ecosystem**
  - Can also be used outside of Hadoop
- **Ideal for long-term storage of important data**
  - Can read and write from many languages
  - Embeds schema in the file, so will always be readable
  - Schema evolution can accommodate changes
- **Verdict: Excellent interoperability and performance**
  - Best choice for general-purpose storage in Hadoop

Using Avro File Format

- **Avro embeds the schema definition in the file itself**
  - Requires the schema be specified

```
CREATE TABLE order_details_avro ()
  STORED AS AVRO
  TBLPROPERTIES ('avro.schema.literal'=
    '{"name": "order",
      "type": "record",
      "fields": [
          {"name":"order_id", "type":"int"},
          {"name":"cust_id", "type":"int"},
          {"name":"order_date", "type":"string"}
        ]}');
```

Columnar Formats

- **Hadoop also supports a few *columnar* formats**
  - These organize data storage by column, rather than by row
  - Very efficient when selecting only a subset of a table's columns

id	name	city	occupation	income	phone
1	Alice	Palo Alto	Accountant	85000	650-555-9748
2	Bob	Sunnyvale	Accountant	81500	650-555-8865
3	Bob	Palo Alto	Dentist	196000	650-555-7185
4	Bob	Palo Alto	Manager	87000	650-555-2518
5	Carol	Palo Alto	Manager	79000	650-555-3951
6	David	Sunnyvale	Mechanic	62000	650-555-4754

Organization of data in traditional row-based formats

id	name	city	occupation	income	phone
1	Alice	Palo Alto	Accountant	85000	650-555-9748
2	Bob	Sunnyvale	Accountant	81500	650-555-8865
3	Bob	Palo Alto	Dentist	196000	650-555-7185
4	Bob	Palo Alto	Manager	87000	650-555-2518
5	Carol	Palo Alto	Manager	79000	650-555-3951
6	David	Sunnyvale	Mechanic	62000	650-555-4754

Organization of data in columnar formats

Columnar File Formats: RCFile and ORCFile

- **RCFile**
  - A *column-oriented* format originally created for Hive tables
  - All data stored as strings (inefficient)
  - **Verdict**: poor performance, and limited interoperability
- **ORCFile**
  - An improved version of RCFile currently under development
  - Currently supported only in Hive, not Impala
  - More efficient than RCFile, but not well supported outside of Hive
  - **Verdict**: improved performance, but limited interoperability

Columnar File Formats: Parquet

- **Parquet is an open-source columnar format originally developed by engineers at Cloudera and Twitter**
  - Now an Apache Incubator project
- **Supported in MapReduce, Hive, Pig, Impala, Spark, Crunch, and others**
- **Contains embedded metadata**
- **Uses advanced optimizations described in Google's Dremel paper**
  - Reduces storage space
  - Increases performance
- **Most efficient when adding many records at once**
  - Some optimizations rely on identifying repeated patterns
- **Verdict: Excellent interoperability and performance**
  - Best choice for column-based access patterns

Using Parquet File Format (1)

- **Create a new table stored in Parquet format**

```
CREATE TABLE order_details_parquet (
    order_id INT,
    prod_id INT)
  STORED AS PARQUET;
```

* `STORED AS PARQUET` supported in Impala, and in Hive 0.13 and later

Using Parquet File Format (2)

- In Impala, use `LIKE PARQUET` to use column metadata from an existing Parquet data file
- Example: Create a new table to access existing Parquet output from another tool

```
CREATE EXTERNAL TABLE ad_data
  LIKE PARQUET '/dualcore/ad_data/datafile1.parquet'
  STORED AS PARQUET
  LOCATION '/dualcore/ad_data/';
```

Chapter Topics

Data and Metadata

- ***Data* refers to the information you store and process**
  - Billing records, sensor readings, and server logs are examples of data
- ***Metadata* describes important aspects of that data**
  - Field name and order are examples of metadata

Metadata

Data

id	name	title	bonus
108424	Alice	Salesperson	2500
101837	Bob	Manager	3000
107812	Chuck	President	9000
105476	Dan	Accountant	3000
104028	Eve	Manager	3500
105293	Frank	Salesperson	2000

The Metastore and HCatalog

- **HCatalog is a Hive sub-project that provides access to the Metastore**
  - Accessible via command line and REST API
  - Allows you to define tables using Hive syntax
  - Access those tables from Hive, Impala, MapReduce, Pig, and other tools
  - Included with CDH 4.2 and later

Creating Tables in HCatalog

- **HCatalog uses Hive's DDL (data definition language) syntax**
  - You can specify a single command using the `-e` option

```
$ hcat -e "CREATE TABLE vendors \
      (id INT, company STRING, email STRING) \
      ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
      LOCATION '/dualcore/vendors'"
```

- **Tip: save longer commands to a text file and use the `-f` option**
  - If the file has more than one command, separate each with a semicolon

```
$ hcat -f createtable.txt
```

Displaying Metadata in HCatalog

- **The `SHOW TABLES` command also shows tables created directly in Hive**

```
$ hcat -e 'SHOW TABLES'
employees
vendors
```

- **The `DESCRIBE` command lists the fields in a specified table**
  - Use `DESCRIBE FORMATTED` instead to see detailed information

```
$ hcat -e 'DESCRIBE vendors'
id          int
company     string
email       string
```

Removing a Table in HCatalog

- **The `DROP TABLE` command has the same behavior as it does in Hive and Impala**
  - Caution: this will remove the data as well as the metadata (unless table is `EXTERNAL`)!

```
$ hcat -e 'DROP TABLE vendors'
```

Using HCatalog to Integrate Pig with Hive and Impala

- **HCatalog simplifies sharing data between Pig, Hive, and Impala**
  - Pig can read and write to Hive/Impala tables using `HCatLoader` and `HCatStorer` functions

```
grunt> allproducts = LOAD 'products'
         USING org.apache.hcatalog.pig.HCatLoader();
grunt> filtered = FILTER allproducts
         BY brand=='Dualcore';
grunt> STORE filtered INTO 'products_dualcore'
         USING org.apache.hcatalog.pig.HCatStorer();
```

Chapter Topics

Data Storage and Performance | **Introduction to Impala and Hive**

- Partitioning Tables
- Choosing a File Format
- Managing Metadata with HCatalog
- **Controlling Access to Data**
- Conclusion
- Hands-On Exercise: Data Storage and Performance

Controlling Access to Data

- **Types of Security**
  - Authentication – who is the user?
    - LDAP or Kerberos
  - Authorization – what is the user allowed to do or see?
    - Apache Sentry
  - Auditing – who did what?
    - Query logging can be enable in both Impala and Hive
  - Encryption – only valid users may access data
    - Network encryption via SSL (Secure Socket Layer)
    - HDFS supports “at rest” encryption

Authorization with Apache Sentry

- **Sentry is a plug-in for enforcing authorization**
- **Sentry can be enabled in HiveServer2 and in Impala**
- **Sentry provides fine-grained, role-based authorization for Hive and Impala data and metadata**
- **Sentry was developed at Cloudera**
  - Available starting with CDH 4.3
  - Project is in incubation status at Apache
  - `http://incubator.apache.org/projects/sentry.html`

Sentry Access Control Model

- **What does Sentry control access to?**
  - Server
  - Database
  - Table
  - View
- **Who can access Sentry-controlled objects?**
  - Users in a Sentry role
    - Sentry roles = one or more groups
- **How can role members access Sentry-controlled objects?**
  - Read operations (`SELECT` privilege)
  - Write operations (`INSERT` privilege)
  - Metadata definition operations (`ALL` privilege)

Chapter Topics

Essential Points

- **Partitioning splits table storage by column values for improved query performance**
- **A variety of file formats are supported by Hive and Impala**
- **HCatalog is a service for managing Metastore directory, for use by Hive, Impala and others**
- **Hive and Impala offer several ways to control access to data**

Bibliography

The following offer more information on topics discussed in this chapter

- **Blog about Impala partitioning at Allstate Insurance**
  - `http://tiny.cloudera.com/allstateimpala`
- **Introducing Parquet: Efficient Columnar Storage for Apache Hadoop**
  - `http://tiny.cloudera.com/dac16a`
- **Using HCatalog**
  - `http://tiny.cloudera.com/hcatalog`

Chapter Topics

Data Storage and Performance | **Introduction to Impala and Hive**

- Partitioning Tables
- Choosing a File Format
- Managing Metadata with HCatalog
- Controlling Access to Data
- Conclusion
- **Hands-On Exercise: Data Storage and Performance**

Hands-On Exercise: Data Storage and Performance

- **In this Hands-On Exercise, you will create a table for ad click data which is partitioned by the network on which the ad was displayed**
- **Please refer to the Hands-On Exercise Manual for instructions**

cloudera®
Ask Bigger Questions

Relational Data Analysis With Impala and Hive

Chapter 12