Hive Data Management

In this chapter you will learn

- **How Hive encodes and stores data**
- **How to create Hive 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**
- **How to control access to data in Hive**

Chapter Topics

Hive Data Management

- **Hive Data Formats**
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Hive Text File Format

- **Each Hive table maps to a directory of data, typically in HDFS**
  - Data stored in one or more files
- **Default data format is plain text**
  - One record per line (record separator is `\n`)
  - Columns are delimited by `^A` (field separator is Control-A)
  - Complex type elements are separated by `^B` (Control-B)
  - Map keys/values are separated by `^C` (Control-C)
- **Hive allows you to override these delimiters**
  - Specify alternate values during table creation

Hive Binary File Formats

- **Hive also supports a few binary formats**
  - Pro: may offer better performance than text files
  - Con: may limit ability to read data from other programs
- **SequenceFile is a Hadoop-specific binary format**
  - Avoids the need to convert all data to/from strings
  - Supports block- and record-level data compression
- **RCFile is a binary format created especially for Hive**
  - RC stands for **R**ecord **C**olumnar
  - Column-oriented format efficient for some queries
  - Otherwise, very similar to sequence files

Chapter Topics

Hive Data Management

- Hive Data Formats
- **Creating Databases and Hive-Managed Tables**
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Creating Databases in Hive

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

```
hive> CREATE DATABASE dualcore;
```

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

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

Creating a Table In Hive (1)

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

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

- **Creates a subdirectory under `/user/hive/warehouse` in HDFS**
  - This is Hive's *warehouse directory*

Creating a Table In Hive (2)

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

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

Creating a Table In Hive (3)

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

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 In Hive (4)

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

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

Creating a Table In Hive (5)

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

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
```

Complex Column Types

- **Recap: Hive has support for complex data types**

Column Type	Description
`ARRAY`	Ordered list of values, all of the same type
`MAP`	Key-value pairs, each of the same type
`STRUCT`	Named fields, of possibly mixed types

Creating Tables with Complex Column Types

- **Complex columns are typed**
  - Arrays have a single type
  - Maps have a key type and a value type
  - Structs have a type for each attribute
- **These types are specified with angle brackets**

```
CREATE TABLE stores
   (store_id SMALLINT,
    departments ARRAY<STRING>,
    staff MAP<STRING, STRING>,
    address STRUCT<street:STRING,
                   city:STRING,
                   state:STRING,
                   zipcode:STRING>
);
```

Customizing Complex Type Storage

- **We can control which delimiters are used for complex types**

```
CREATE TABLE stores
   (store_id SMALLINT,
    departments ARRAY<STRING>,
    staff MAP<STRING, STRING>,
    address STRUCT<street:STRING,
                   city:STRING,
                   state:STRING,
                   zipcode:STRING>
   )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ','
   MAP KEYS TERMINATED BY ':';
```

Row Format Example for Complex Types

- **The following is an example of a record in that table**

```
1|Audio,Photo|A123:Abe,B456:Bob|123 Oak St.,Ely,MN,55731
```

- **Examples queries on this record**

```
hive> SELECT departments[0] FROM stores;
Audio

hive> SELECT staff['B456'] FROM stores;
Bob

hive> SELECT address.city FROM stores;
Ely
```

Chapter Topics

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- **Loading Data into Hive**
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Data Validation in Hive

- **Hadoop and its ecosystem are 'schema on read'**
  - Unlike an RDBMS, Hive does 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 in Hive will be represented as `NULL`**

Loading Data From Files (1)

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

```
$ hadoop fs -mv sales.txt /user/hive/warehouse/sales/
```

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

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

Loading Data From Files (2)

- **Add the `LOCAL` keyword to load data from the local disk**
  - Copies a local file (or directory) to the table's directory in HDFS

```
hive> LOAD DATA LOCAL INPATH '/home/bob/sales.txt'
      INTO TABLE sales;
```

  - This is equivalent to the following **`hadoop fs`** command

```
$ hadoop fs -put /home/bob/sales.txt \
 /user/hive/warehouse/sales
```

Loading Data From Files (3)

- **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

```
hive> LOAD DATA INPATH '/depts/finance/salesdata'
      OVERWRITE INTO TABLE sales;
```

Loading Data From a Relational Database

- **Sqoop has built-in support for importing data into Hive**
- **Just add the `--hive-import` option to your Sqoop command**
  - Creates the table in Hive (metastore)
  - Imports data from RDBMS to 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

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- **Altering Databases and Tables**
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Removing a Database

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

```
hive> DROP DATABASE dualcore;
```

```
hive> DROP DATABASE IF EXISTS dualcore;
```

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

```
hive> DROP DATABASE dualcore CASCADE;
```

Removing a Table

- **Syntax is similar to database removal**

```
hive> DROP TABLE customers;
```

```
hive> DROP TABLE IF EXISTS customers;
```

- **Caution: These commands can remove data in HDFS!**
  - Hive does not have a rollback or undo feature

Renaming Tables and Columns

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

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

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

```
hive> 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

```
hive> 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

```
hive> 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

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

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

- **Use `REPLACE COLUMNS` to remove columns**
  - Any column not listed will be dropped from metadata

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

Chapter Topics

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- **Self-Managed Tables**
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Controlling Table Data Location

- **Hive stores data associated with a table in its warehouse directory**
- **Storing data below Hive's warehouse is not always ideal**
  - Data might be shared by several users
- **Use `LOCATION` to specify directory where table data resides**

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

Self-Managed (External) Tables

- **Recall that 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';
```

Chapter Topics

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- **Simplifying Queries with Views**
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

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 in Hive 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**

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

Inspecting and Removing Views

- Use `DESCRIBE FORMATTED` to see underlying query

```
hive> DESCRIBE FORMATTED order_info;
```

- Use `DROP VIEW` to remove a view

```
hive> DROP VIEW order_info;
```

Chapter Topics

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- **Storing Query Results**
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- Conclusion

Saving Query Output to a Table

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

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

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

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

Creating Tables Based On Existing Data

- **Hive supports creating 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 a Filesystem

- **You can save output to a file in HDFS**

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

- **Add LOCAL to store results to local disk instead**

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/home/bob/ny/'
        SELECT * FROM customers
        WHERE state = 'NY';
```

- **Both produce text files delimited by Ctrl-A characters**

Writing Output To HDFS, Specifying Format

- **To write the files to HDFS with a user-specified format:**
  - Create an external table in the required format
  - Use `INSERT OVERWRITE TABLE`

```
hive> CREATE EXTERNAL TABLE ny_customers
        (cust_id INT,
         fname STRING,
         lname STRING)
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
        STORED AS TEXTFILE
        LOCATION '/dualcore/nydata';

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

Multi-Table Insert (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';
```

Multi-Table Insert (2)

- **We sometimes need to extract data to multiple tables**
  - 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
- **The following example demonstrates multi-table insert**
  - 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';
```

Multi-Table Insert (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

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- **Controlling Access to Data**
- Hands-On Exercise: Data Management with Hive
- Conclusion

Hive Authentication

- **HiveServer2 can be configured for Kerberos or LDAP authentication**
  - You must provide a user ID and password when connecting to HiveServer2 from Beeline

```
[training@localhost analyst]$ beeline
Beeline version 0.10.0-cdh4.2.1 by Apache Hive

beeline> !connect jdbc:hive2://localhost:10000
training mypwd org.apache.hive.jdbc.HiveDriver

Connecting to jdbc:hive2://localhost:10000
Connected to: Hive (version 0.10.0)
Driver: Hive (version 0.10.0-cdh4.2.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ

beeline>
```

Hive 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 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

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- **Hands-On Exercise: Data Management with Hive**
- Conclusion

Hands-on Exercise: Data Management with Hive

- **In this Hands-On Exercise, you will create, load, modify, and delete tables in Hive.**
- **Please refer to the Hands-On Exercise Manual for instructions**

Chapter Topics

Hive Data Management

- Hive Data Formats
- Creating Databases and Hive-Managed Tables
- Loading Data into Hive
- Altering Databases and Tables
- Self-Managed Tables
- Simplifying Queries with Views
- Storing Query Results
- Controlling Access to Data
- Hands-On Exercise: Data Management with Hive
- **Conclusion**