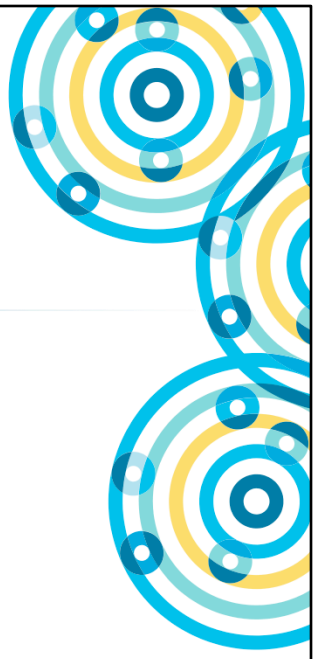
cloudera®



Data File Partitioning

Chapter 8

Course Chapters

1	Introduction	Course Introduction
2	Introduction to Hadoop and the Hadoop Ecosystem	Introduction to Hadoop
3	Hadoop Architecture and HDFS	
4	Importing Relational Data with Apache Sqoop	**Importing and Modeling Structured Data**
5	Introduction to Impala and Hive	
6	Modeling and Managing Data with Impala and Hive	
7	Data Formats	
8	**Data File Partitioning**	
9	Capturing Data with Apache Flume	Ingesting Streaming Data
10	Spark Basics	Distributed Data Processing with Spark
11	Working with RDDs in Spark	
12	Aggregating Data with Pair RDDs	
13	Writing and Deploying Spark Applications	
14	Parallel Processing in Spark	
15	Spark RDD Persistence	
16	Common Patterns in Spark Data Processing	
17	Spark SQL and DataFrames	
18	Conclusion	Course Conclusion

Data File Partitioning

In this chapter you will learn

- **How to improve query performance with data file partitioning**
- **How to create and populate partitioned tables in Impala and Hive**

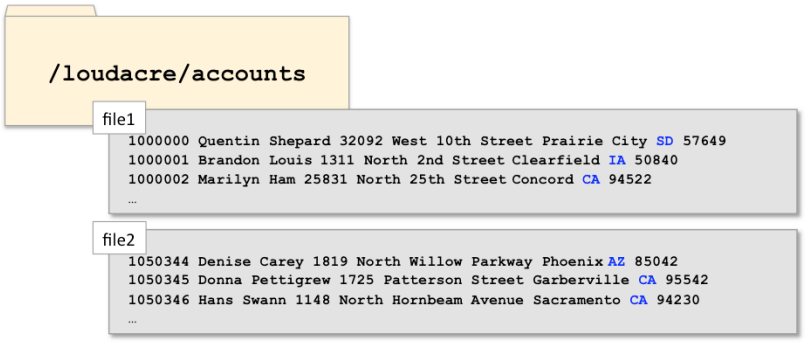
Chapter Topics

Data File Partitioning | **Importing and Modeling Structured Data**

- **Partitioning Overview**
- Partitioning in Impala and Hive
- Conclusion
- Homework: Partition Data in Impala or Hive

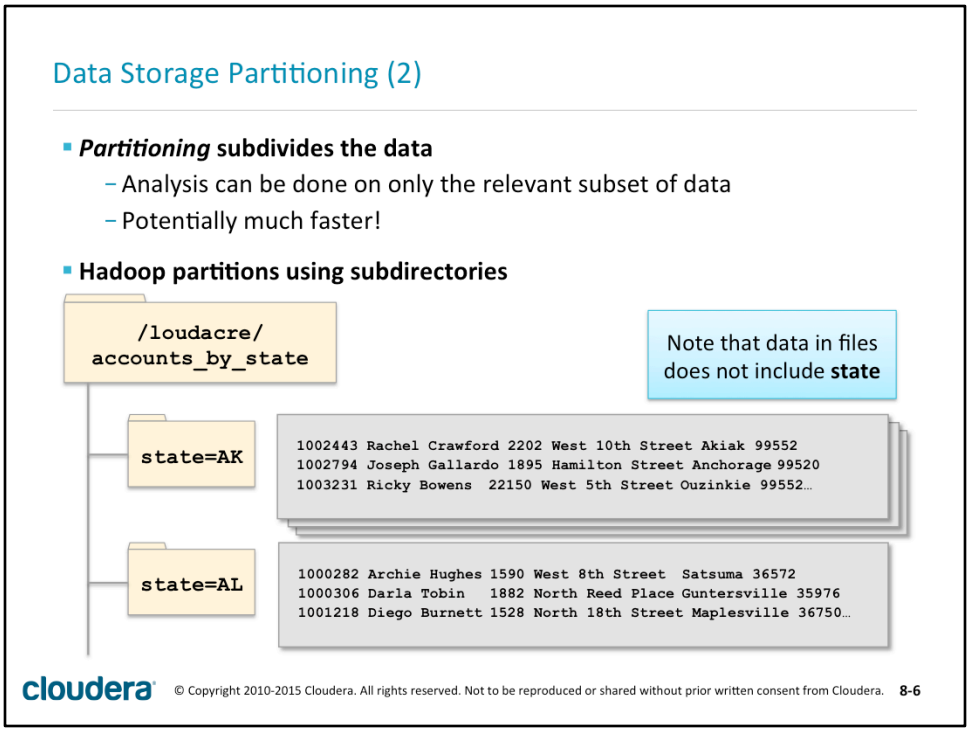
Data Storage Partitioning (1)

- **By default, all files in a data set are stored in a single HDFS directory**
  - *All* files in the directory are read during analysis or processing
  - "Full table scan"



Data storage or data file partitioning is not to be confused with other sorts of "partitioning" that are referred to in the Hadoop Ecosystem, e.g.

- partitioners which partition data during processing in MapReduce or Spark
- Kafka "partitions"
- disk partitioning in underlying file systems (Linux)
- Partitions in Spark RDDs



This is sometimes called “horizontal partitioning” and is seen as an antipattern with traditional RDBMSs, but is nonetheless a relatively common hack employed when an organization starts hitting the scalability limits for those databases. It’s not a hack in Hadoop, however, because it’s built in and implemented fairly transparently to the end user. Partitioning your data on a field means that queries that filter on that field will be much faster because it limits the amount of data that query needs to read (though if you run a query that spans partitions, it may need to read all data). You should use partitions for fields that you frequently used for filtering data in your queries. You can still run a query that spans multiple partitions (including a query that doesn’t filter on the partitioned field at all).

Regarding the note that the files themselves do not contain the STATE column data: students often ask, what if for some reason that data needs to be *in* the file (for example, if they plan to have some other tool access the data that doesn’t use the “virtual” state column, or because the source of the data puts state in there regardless.) That is a reasonable use case, and in order to do that, you can define an *additional* column (for example, “STATE2”). If it happens that the data for the column is at the end of the line in a text-based file format, you could simply not refer to it in the table definition and it will be disregarded by Hive/Impala; this would be an unusual situation though, as discussed in the previous chapter, most often we won’t be using text format.

Hadoop Partitioning

- **Partitioning is involved at two phases**
  - Storage – putting the data into correct partition (subdirectory)
  - Retrieval – getting the data out of the correct partition based on the query or analysis being done
- **Hadoop with built-in support for partitioning**
  - Hive and Impala (covered in next section)
  - Sqoop – When using the **`--hive-import`** option you can specify flags **`--hive-partition-key`** and **`--hive-partition-value`**
- **Other tools can be used to store partitioned data**
  - Spark and MapReduce
  - Flume (at ingestion)

Chapter Topics

Data File Partitioning | **Importing and Modeling Structured Data**

- Partitioning Overview
- **Partitioning in Impala and Hive**
- Conclusion
- Homework: Partition Data in Impala or Hive

Example: Impala/Hive Partitioning Accounts By State (1)

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

```
CREATE EXTERNAL TABLE accounts(
    cust_id INT,
    fname STRING,
    lname STRING,
    address STRING,
    city STRING,
    state STRING,
    zipcode STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/loudacre/accounts';
```


This is an example of a non-partitioned table, as a lead-in to partitioned tables coming next.

Here we are showing the customers table, which students have already been using in the homework assignments, so it should be familiar. The syntax was already covered in the last chapter so this is a recap.

Example: Impala/Hive Partitioning Accounts By State (2)

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

```
SELECT fname, lname
  FROM accounts
  WHERE 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

This query displays the order date and customer name of all orders from customers located in NY. What is important is that this is a typical analysis for Dualcore – they analyze data by state.

Example: Impala/Hive Partitioning Accounts By State (3)

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

```
CREATE EXTERNAL TABLE accounts_by_state(
    cust_id INT,
    fname STRING,
    lname STRING,
    address STRING,
    city STRING,
    state STRING,
    zipcode STRING)
  PARTITIONED BY (state STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/loudacre/accounts_by_state';
```


This is the same as the create table statement for customers, except that customers_by_state has no state column in the column list; instead, it has PARTITIONED BY (state). Other than that it is identical.

The name of the partitioned column should appear ***only*** in the `PARTITION` clause (it does not – and must not – also appear in the section within parentheses among the other fields.

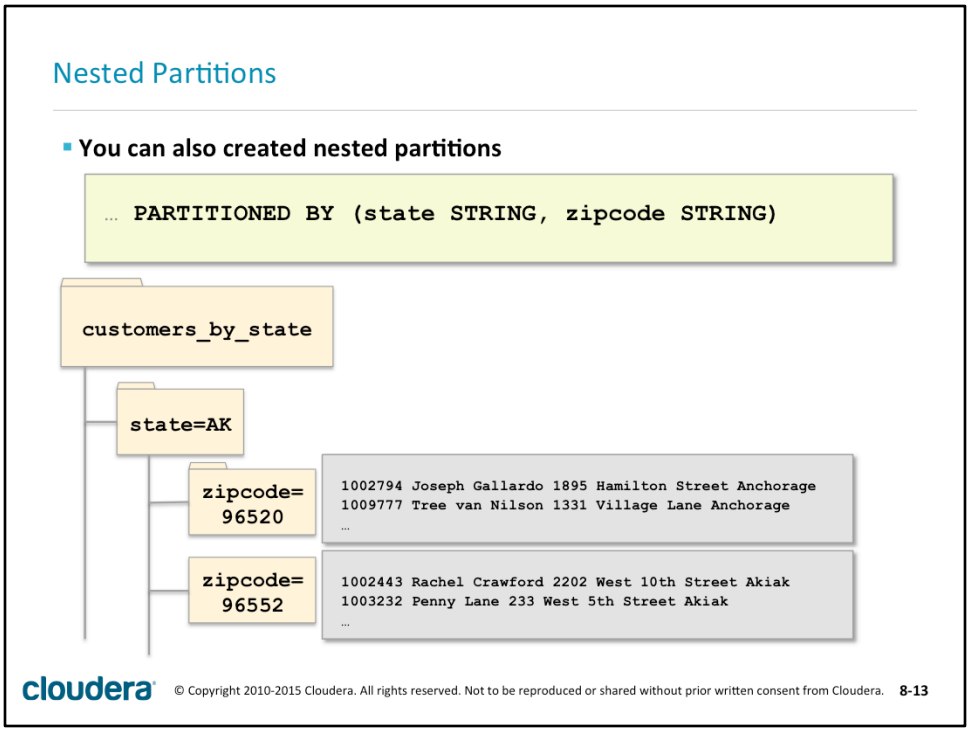
Partition Columns

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

```
DESCRIBE accounts_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

You can use the `SHOW PARTITIONS table` command to see all the partitions currently in the table.



Nested partitions create sub-subdirectories. The second partition level is stored in a subdirectory to the first.

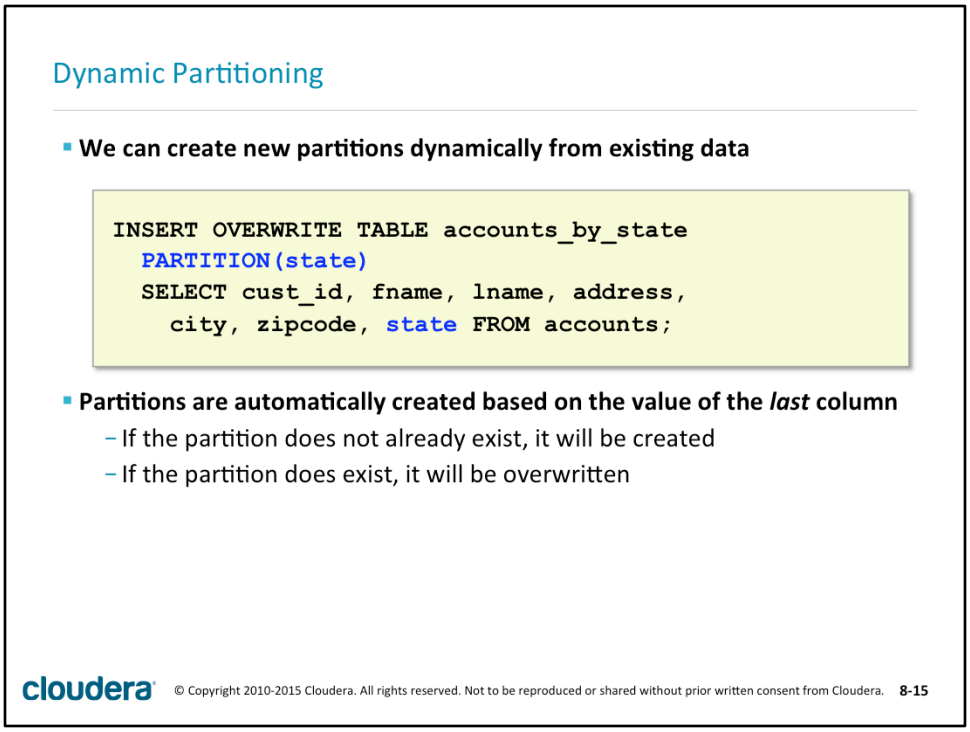
Note that Impala and Hive (MapReduce) performs better on *large* files rather than many small files. Creating too many nested partitions will create more, smaller files. Although it is possible two have more than two levels of nesting, we recommend no more than two, because it will negatively impact performance.

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

There are two ways to load data into a partitioned table. Which to use depends on the source and nature of the data.

In Dynamic partitioning, Impala/Hive create the partitions for you; in static, you create them yourself.



Dynamic partitioning is used when you have an existing table, with data that includes a column on which to partition.

The INSERT OVERWRITE TABLE …. SELECT syntax was covered in the last chapter; this example just adds the PARTITION command.

Point out that when inserting records this way, you must include the partitioned field ('state' in this example) in both the `PARTITION` portion of the `INSERT` statement and the field list in the `SELECT` statement, as shown here.

The dynamic partitioning feature was added in Hive 0.7.0 [`https://issues.apache.org/jira/browse/HIVE-936`]. The Dynamic Partition design document [`https://cwiki.apache.org/Hive/dynamicpartitions.html`] points out that in `INSERT` … `SELECT` … queries, the dynamic partition columns must be **specified last** among the columns in the `SELECT` statement and **in the same order** in which they appear in the `PARTITION()` clause. This article [`http://www.brentozar.com/archive/2013/03/introduction-to-hive-partitioning/`] is also quite helpful in explaining dynamic table inserts.

Per the Hive Wiki [`https://cwiki.apache.org/Hive/languagemanual-ddl.html`], you cannot use a CTAS to create a partitioned table, so the partitioned table must be created in an earlier step.

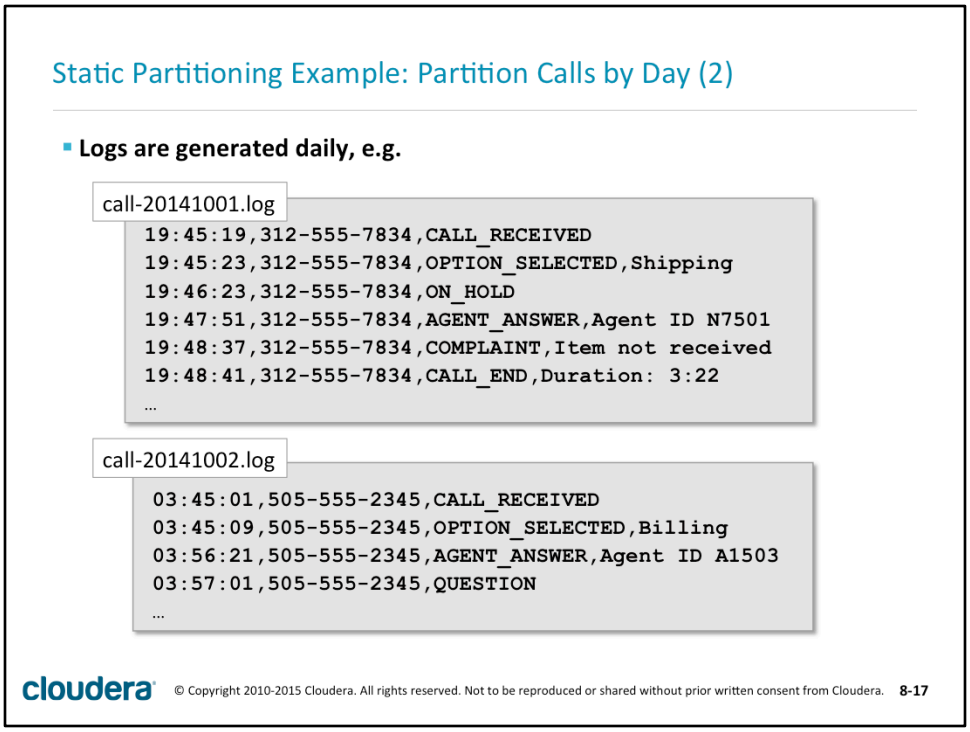
Static Partitioning Example: Partition Calls by Day (1)

- **Loudacre's customer service phone system generates logs detailing calls received**
  - Analysts use this data to summarize previous days' calls
  - For example:

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


Static partitioning can be used when new data is being added to new partitions periodically, and when the partition data (in this example the call_data) is not part of the data itself.

To explain this, we start here with an example, a typical query against call log data.



Here's some sample data that would be in that table. Each record includes the time of the call, the number of the caller, and event type (call received, call end, etc.) and optionally notes entered by the customer service rep.

Note that each day's data is in a separate file, and that the date itself is not part of the data. This is what leads us to partitioning by date, because our data is already pre-partitioned.

Students may ask, what if we had the date in the data, for example, not just the time. This would be typical output for, say, a webserver log. In that case you have three options:

1) Preprocess the data to remove the date, e.g. a pig script.
2) create a non-partitioned table which includes a "date" column, and then use dynamic partitioning to copy the data into the correct partition. Note that this option results in two copies of the data which is probably not optimal if it's a lot of data.
3) Go ahead and define a partition column (e.g. "call_date") as well as a regular column (e.g. "log_date"). The two columns would always be equal and therefore redundant which might lead to confusion, but it will work fine.

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 ',';
```


The definition of partitioned tables is the same, no matter how you plan to load the data. In fact, you can use either method or both on the same table.

note from docs: For time-based data, split out the separate parts into their own columns, because Impala cannot partition based on a TIMESTAMP column.

Partitioning on a TIMESTAMP column could create too many partitions.

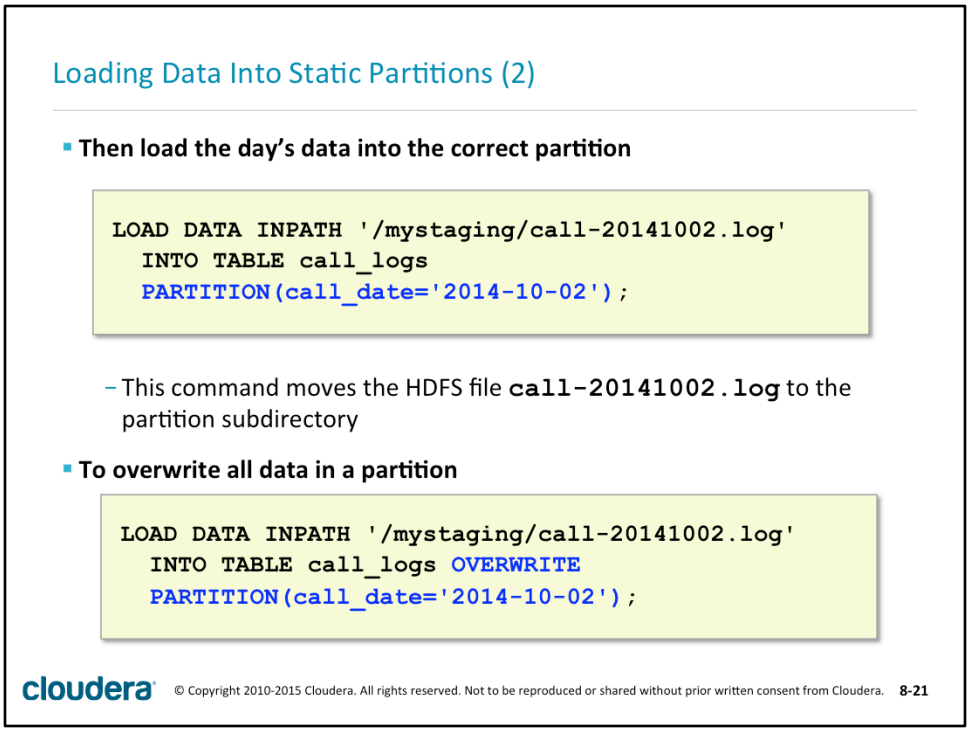
Loading Data Into Static Partitions (1)

- **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='2014-10-02');
```

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

The ADD PARTITION statement doesn't copy any data, it only adds the partition to the metadata and warehouse.



The Hive shell has been deprecated in favor of beeline.

The LOAD statement will work using a path relative to the users' home directory in the Hive shell and the beeline shell, but not using the Impala shell. An absolute path must be used in Impala.

```
impala-shell> LOAD DATA INPATH 'myfile' INTO TABLE
accounts_by_state PARTITION(state='CA');
Query: load DATA INPATH 'myfile' INTO TABLE
accounts_by_state PARTITION(state='CA')
ERROR: AnalysisException: URI path must be absolute:
myfile
```

If the above statement is run in the Hive shell (which is deprecated), the file myfile must be in the user's home directory of HDFS (/user/training on our VMs).

If the above statement is run in the beeline shell, the file myfile must be in /user/hive (or the home directory associated with the 'user' configured to run Beeline, which is a configuration option)

No massaging makes the statement work in Impala; it wants an absolute path to the file in HDFS.

Hive Only: Shortcut for Loading Data Into Partitions

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

```
LOAD DATA INPATH '/mystaging/call-20141002.log'
  INTO TABLE call_logs
  PARTITION(call_date='2014-10-02');
```

- **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

HIVE ONLY: Hive will create the partition if needed.

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 '/loudacre/call_logs/call_date=2013-06-05';
```

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


SHOW PARTITIONS in Impala is only in Impala 1.4 and later

Creating Partitions from Existing Partition Directories in HDFS

- **Partition directories in HDFS can be created and populated outside Hive or Impala**
  - For example, by a Spark or MapReduce application
- **In Hive, use the `MSCK REPAIR TABLE` command to create (or recreate) partitions for an existing table**

```
MSCK REPAIR TABLE call_logs;
```


This works in Hive but not Impala. (Syntax error message.)
Buggy in Hue: New partitions not shown in Hue with show partitions <table> but do show up using Hive CLI. Data can still be queried in Hue.
The docs say you can also use ALTER TABLE RECOVER PARTITIONS but I could not get this working.

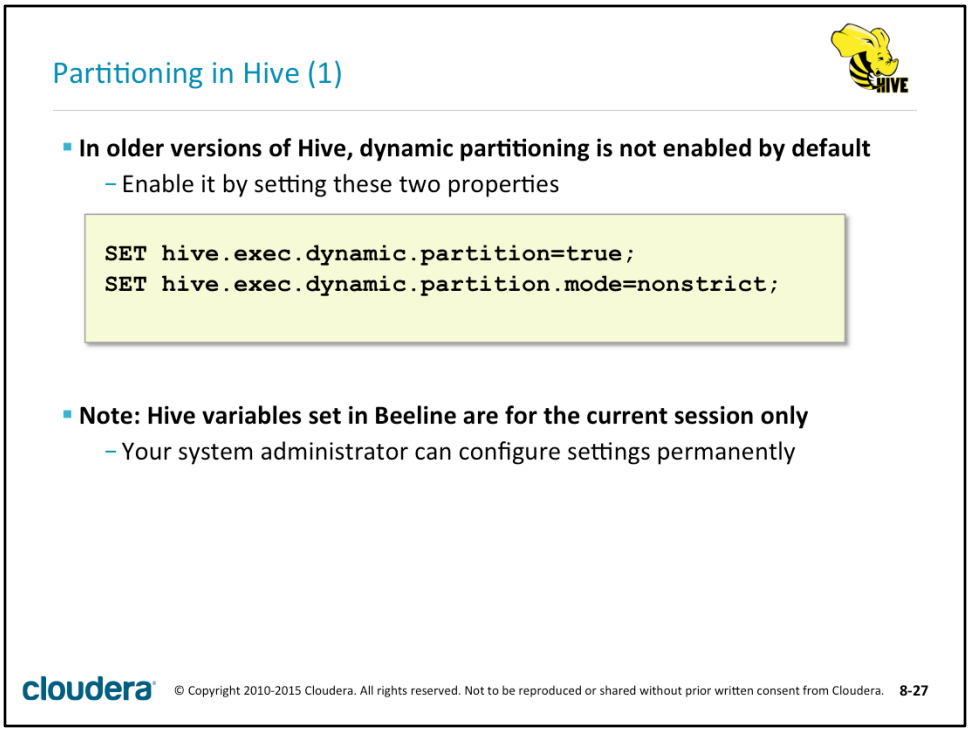
When To Use Partitioning

- **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

From the docs:
When to Use Partitioned Tables
Partitioning is typically appropriate for:
Tables that are very large, where reading the entire data set takes an impractical amount of time.
Tables that are always or almost always queried with conditions on the partitioning columns. In our example of a table partitioned by year, SELECT COUNT(*) FROM school_records WHERE year = 1985 is efficient, only examining a small fraction of the data; but SELECT COUNT(*) FROM school_records has to process a separate data file for each year, resulting in more overall work than in an unpartitioned table. You would probably not partition this way if you frequently queried the table based on last name, student ID, and so on without testing the year.
Columns that have reasonable cardinality (number of different values). If a column only has a small number of values, for example Male or Female, you do not gain much efficiency by eliminating only about 50% of the data to read for each query. If a column has only a few rows matching each value, the number of directories to process can become a limiting factor, and the data file in each directory could be too small to take advantage of the Hadoop mechanism for transmitting data in multi-megabyte blocks. For example, you might partition census data by year, store sales data by year and month, and web traffic data by year, month, and day. (Some users with high volumes of incoming data might even partition down to the individual hour and minute.)
Data that already passes through an extract, transform, and load (ETL) pipeline. The values of the partitioning columns are stripped from the original data files and represented by directory names, so loading data into a partitioned table involves some sort of transformation or preprocessing.

When *Not* To Use Partitioning

- **Avoid partitioning data into numerous small data files**
  - Don't 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



To set the variable at the command line when invoking Beeline use:
`beeline --hiveconf key=value`

You could also set them in Hive's XML configuration file (i.e. `/etc/hive/conf/`), although as with other configuration changes to the XML files, this is probably better left for advanced users or system administrators.

From the docs:

hive.exec.dynamic.partition.mode=strict << default
hive.exec.dynamic.partition.mode=nonstrict

In strict mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions, in nonstrict mode all partitions are allowed to be dynamic.

Note: `.hiverc` (discussed in prior versions of these slides, before the Hive CLI was deprecated) is not used by Beeline.

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

See PH1e p. 75 for more info on these parameters.

Chapter Topics

Data File Partitioning | **Importing and Modeling Structured Data**

- Partitioning Overview
- Partitioning in Impala and Hive
- **Conclusion**
- Homework: Partition Data in Impala or Hive

Essential Points

- **Partitioning splits table storage by column values for improved query performance**
- **Partitions are HDFS directories**
  - Names follow the format ***`column=value`***
- **Partitions can be defined and loaded dynamically or statically**
- **Only partition on columns with a reasonable number of possible values**

REVIEW QUESTIONS (optional, but will help the instructor gauge whether the class is absorbing the material, and to provide some elasticity if you're running ahead of schedule):

- **Q1**: Where is the data for partitioned tables stored? (Answer: In subdirectories under the main table warehouse directory)
- **Q2**: What are important factors to consider when considering which fields you might use for partitioning a table? (Answer: Whether that field is frequently used in `WHERE` clauses and how many unique values it might contain)

Bibliography

The following offer more information on topics discussed in this chapter

- **Impala documentation on partitioning**
  - `http://tiny.cloudera.com/impalapart`
- **Improving Query Performance Using Partitioning in Apache Hive (Cloudera Engineering Blog)**
  - `http://tiny.cloudera.com/partblog`

Chapter Topics

Data File Partitioning | **Importing and Modeling Structured Data**

- Partitioning Overview
- Partitioning in Impala and Hive
- Conclusion
- **Homework: Partition Data in Impala or Hive**

Homework: Partition Data in Impala or Hive

- **In this homework assignment you will**
  - Create a table for accounts that is partitioned by area code
- **Please refer to the Homework description**

