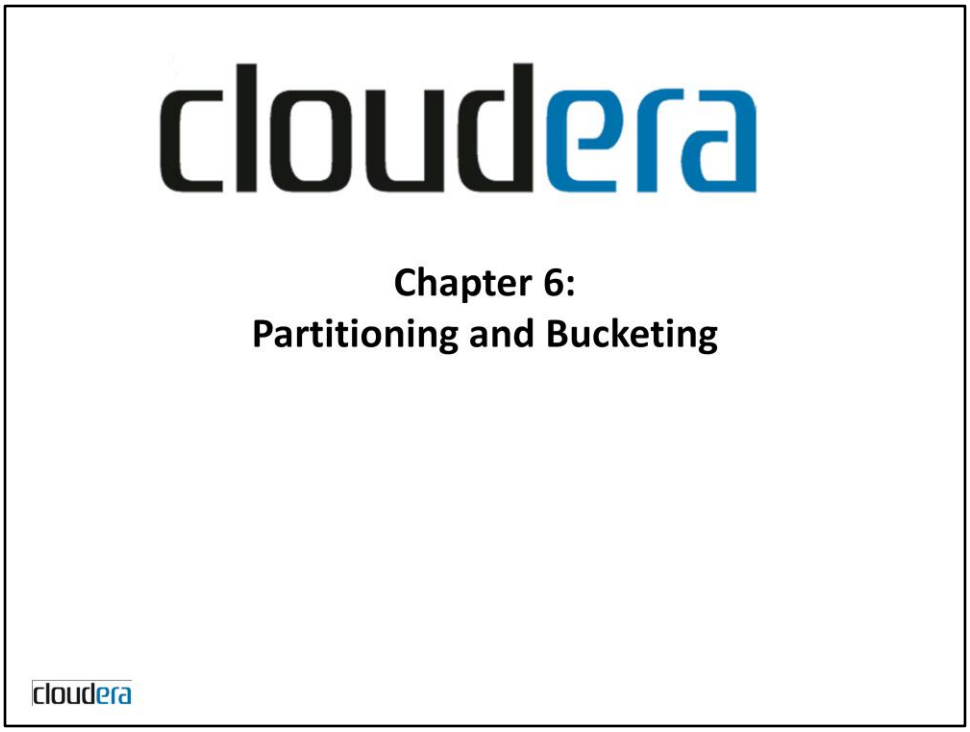


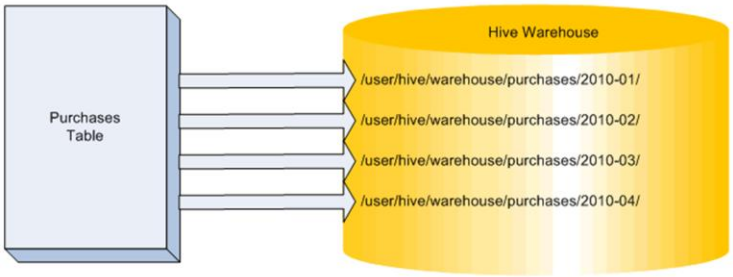
In this chapter, you will learn

- How to create partitions
- How to load data into partitions
- How to change partition definitions
- How to use buckets

cloudera

Partitioning

- Horizontal partitioning divides a table into distinct groups of rows
- Physically stored in separate directories



cloudera

Horizontal partitioning is a technique for dividing a table into distinct groups of rows. Each partition would be a separate directory in the Hive warehouse. There are several benefits of partitioning a large table, such as faster queries that need to access only certain partitions. It is also convenient to remove a subset of rows by dropping a partition. It is common to partition by date, such as a partition per month or day.

Creating partitions

- `CREATE TABLE tablename`

  `(columns..)`

  `PARTITIONED BY (column type);`

- The partition column is not part of the data, but a virtual column
- The type is required so Hive can cast the data in a query

cloudera

To create a partitioned table, add the `PARTITIONED BY` clause to `CREATE TABLE`. The column to partition on is not part of the data, but a user-defined value that is specified when loading individual partitions. Hive is strongly typed, so it is necessary to specify a type for the partition column. It is also possible to partition by multiple columns.

Loading data into partitions

- `LOAD DATA INPATH 'filepath'`
  `INTO TABLE tablename`
  `PARTITION (col=val);`
- Must specify partition when loading data into a partitioned table
- Subdirectory auto-created in Hive warehouse

cloudera

To load data into a partitioned table, use `LOAD DATA INPATH` with a `PARTITION` clause. It is required to use the `PARTITION` clause when loading data into a table that was created with the `PARTITIONED BY` clause. In the Hive warehouse in HDFS, a subdirectory is created for each unique partition value.

An example

```
LOAD DATA LOCAL INPATH
'/tmp/new_logs.txt'
INTO TABLE logs
PARTITION (d='2010-04-01');
```

cloudera

This command will load the file `/tmp/new_logs.txt` into the partitioned named '2010-04-01'.

Note: This is different than how RDBMSs partition data. Most RDBMS implementations would read the data that is being loaded and put each row into a relevant partition based on a value or values in the record. In Hive, it is the user who loads the data that is specifying which partition this file belongs to. Dynamic partitions are coming soon to Hive.

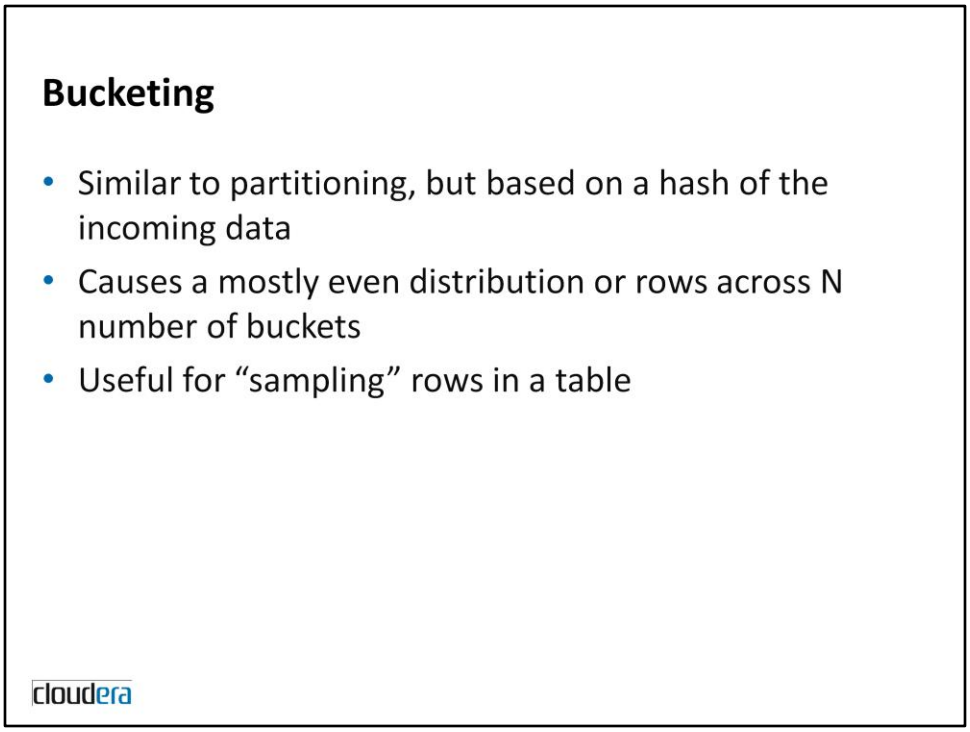
Other useful commands

- `ALTER TABLE tablename`
  `ADD PARTITION (col=val)`
- `ALTER TABLE tablename`
  `DROP PARTITION (col=val)`
- `SHOW PARTITIONS tablename;`

cloudera

Partitions may be added or removed using an `ALTER TABLE` statement. The `ADD PARTITION` clause adds a new, empty partition (and the corresponding directory). This would likely be followed by a `LOAD DATA INPATH`.

The `DROP PARTITION` clause removes the partition and any data in that partition.



Bucketing is similar to partitioning in that it distributes the rows into separate groups that are stored in their own subdirectory. Unlike partitioning, the user does not have to define which rows go into a particular bucket. Hive will use this formula for assigning a row to a bucket: `hash(column) MOD (number of buckets)`. This will usually cause an even distribution of rows across the buckets.

Create a bucketed table

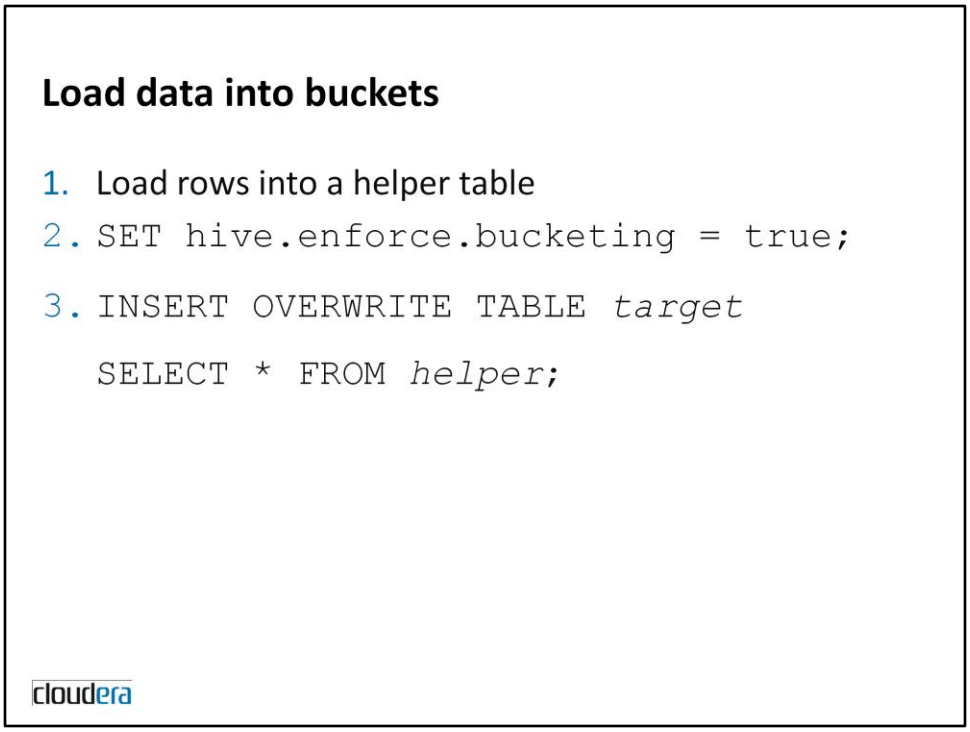
- CREATE TABLE *tablename* (*columns*)

  CLUSTERED BY (*col*) INTO *N* BUCKETS;

cloudera

Creating a bucketed table is as simple as adding:
`CLUSTERED BY (column_name) INTO N BUCKETS`

However, loading the data into the bucketed table requires a bit more work.



Here are the steps to load data into a bucketed table:

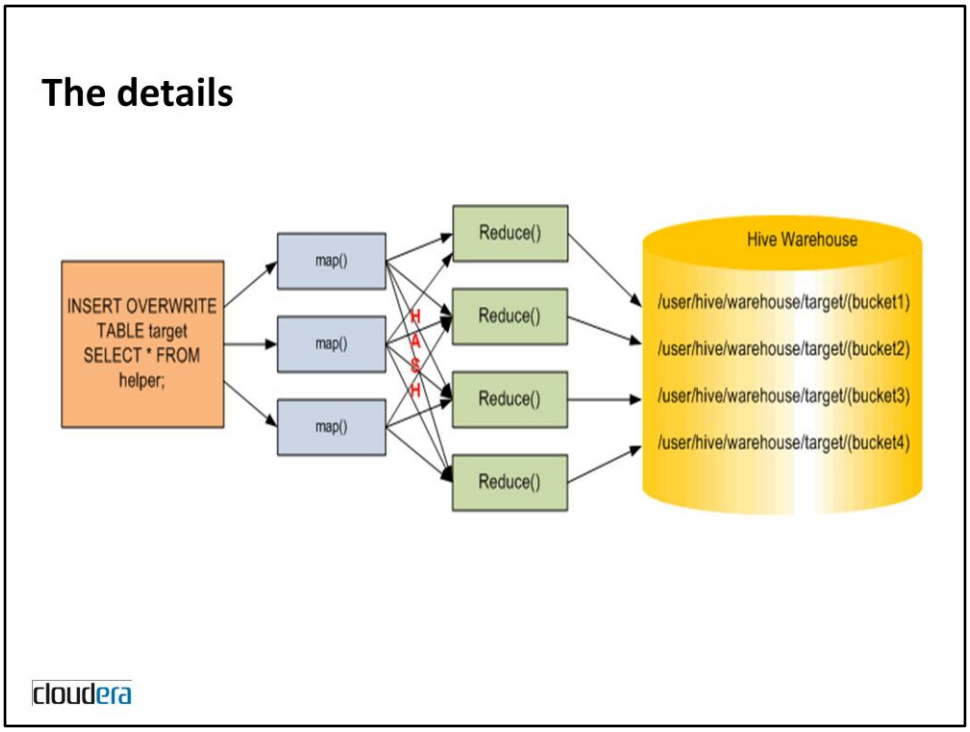
1. Create an unbucketed table that has the rows intended for the bucketed table.
2. Set the `hive.enforce.bucketing` parameter to true.
3. Use an `INSERT OVERWRITE` statement to copy the rows from the unbucketed table into the bucketed table.

In older releases that do not have the `hive.enforce.bucketing` parameter, two things must be changed for this example:

```
SET mapred.reduce.tasks = (number of buckets on the
   table);
```

Add a `CLUSTER BY` clause to the `SELECT` in step 3.

Note: `hive.enforce.bucketing` is introduced in Hive 0.6



The way Hive implements the `INSERT` into the bucketed table is by running a MapReduce job. The map tasks read the helper table, then a hash algorithm is executed to send the data to N number of reducers (where N is the number of buckets on the target table). Then each reducer writes its rows to a file representing that bucket.

Sampling

- Reading a subset of the data is very efficient if the table is bucketed
- `SELECT * FROM tablename`

  `TABLESAMPLE (BUCKET 1 OUT OF 4 ON col)`

cloudera

Often it is useful to look at a sample of the rows in a table instead of all of them. Hive's built-in bucketing feature makes this very efficient. If a table has 4 buckets, this query will read only the first bucket:

```
SELECT * FROM tablename
TABLESAMPLE (BUCKET 1 OUT OF 4 ON col)
```

Say a table has 32 buckets. If the query asks to sample bucket 1 out of 16, two buckets would actually be read (since 32/16 = 2). They would be bucket 1 and 17. If they query asked for bucket 1 out of 2, half the buckets would be used.

Note: Using the `TABLESAMPLE` feature on a non-bucketed table will cause a full table scan to fetch the sample of rows.

Conclusion

In this chapter, you have learned

- How to create partitions
- How to load data into partitions
- How to change partition definitions
- How to use buckets

cloudera