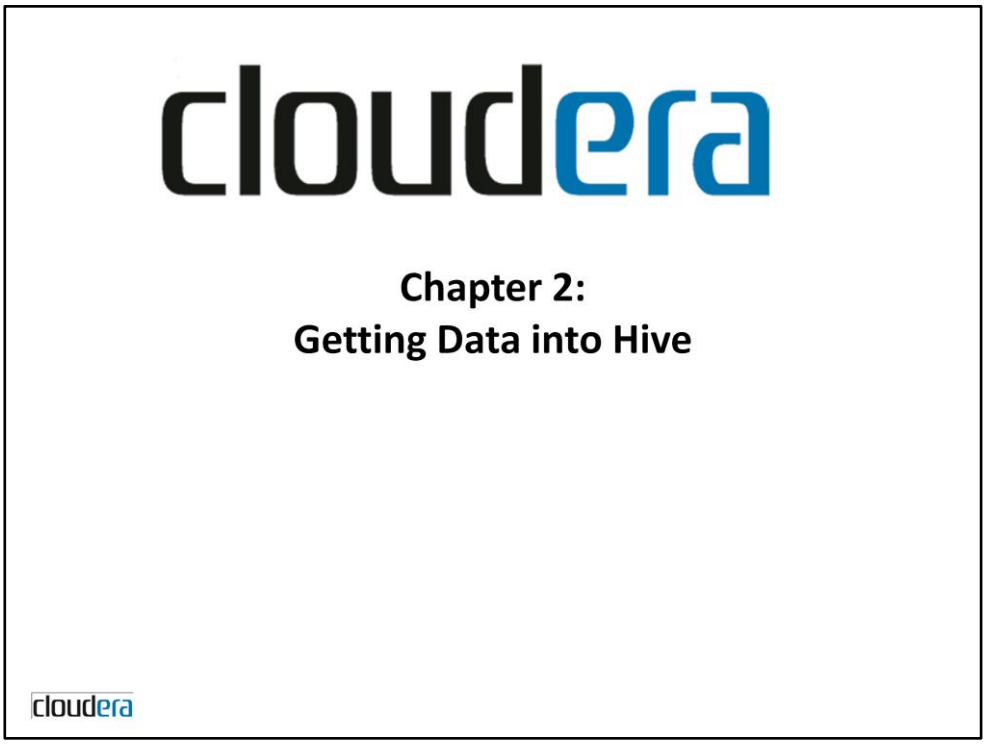
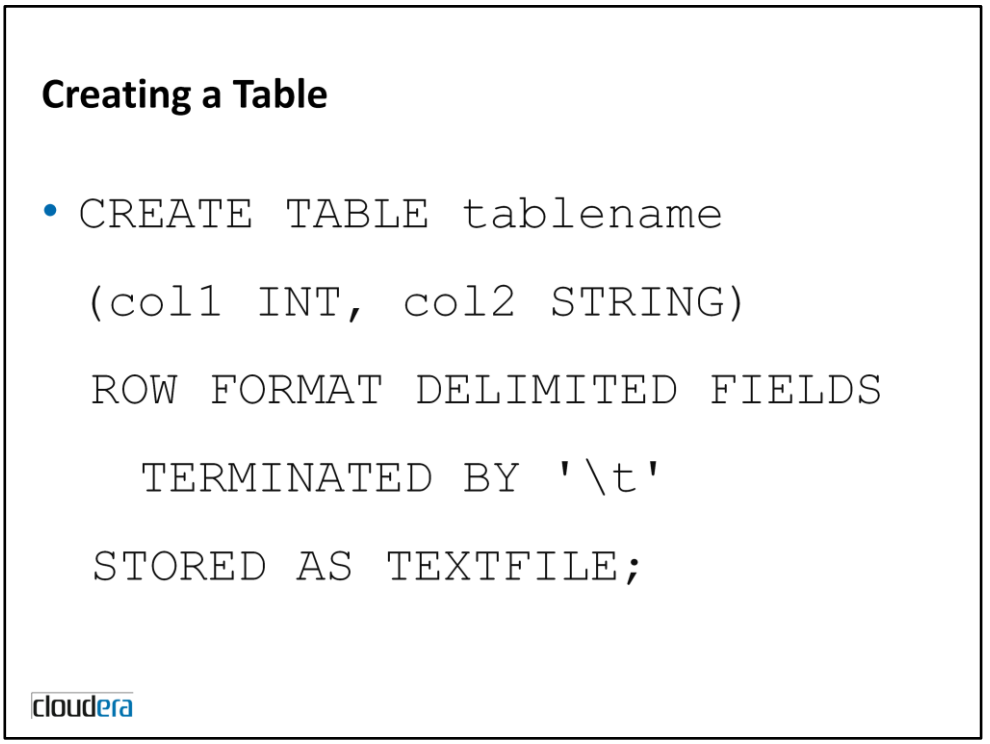


## In this chapter, you will learn

- How to create tables in Hive
- How Hive uses InputFormats and SerDes to parse data
- How to remove tables from Hive
- How to load data into Hive
- How to import data with Sqoop

cloudera




This example would create a new table called *tablename* that has two columns, an integer and a string. If a table called *tablename* already exists, this statement would give an error. Optionally an `IF NOT EXISTS` clause can be added: `CREATE TABLE IF NOT EXISTS` *tablename*.

When data is later loaded into this table, the `ROW FORMAT` will be used to parse the incoming data. In this example, the fields will be delimited by tab (\t). Hive defaults to use a field delimiter of ^A (ctrl-a) and a newline (\n) to separate lines.

Additional keywords can be used when creating a table to use Hive features such as partitioning and bucketing. These are discussed later in the chapter.

Note: identifiers should generally not be keywords or contain special characters (like spaces). To use an identifier like this, it must be enclosed in backticks (`).

## External Tables

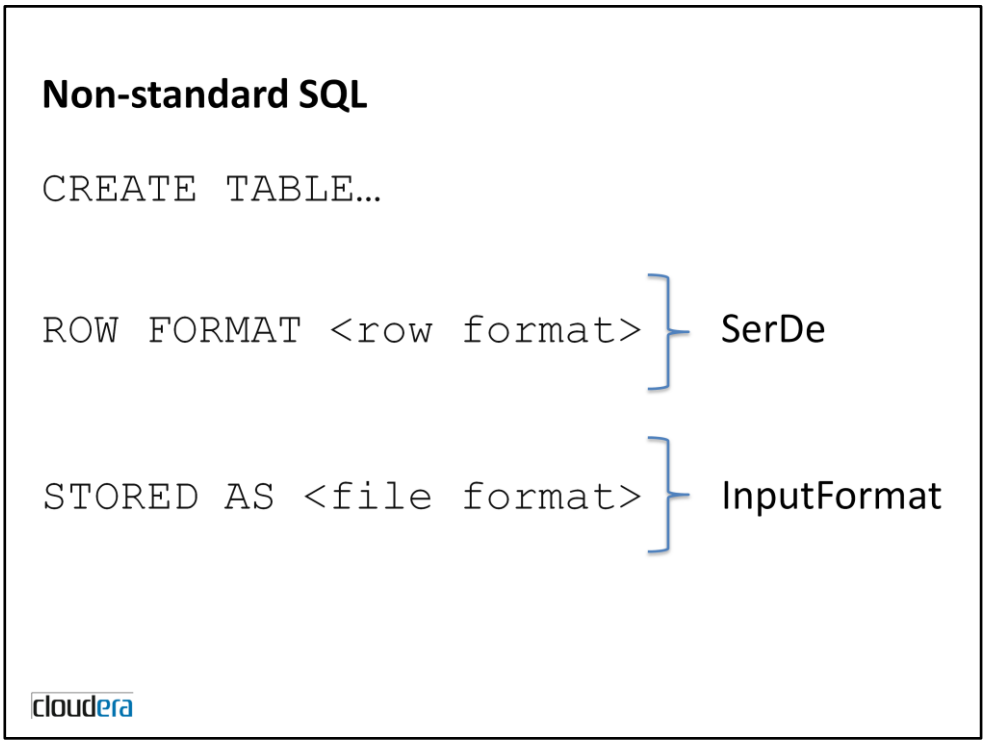
- A table can use any files in HDFS
- CREATE **EXTERNAL** TABLE tablename

```
    (columns..)
ROW FORMAT DELIMITED FIELDS
   TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/user/hadoop/data';
```

cloudera

An external table allows users to create tables in Hive whose data is in HDFS, but not in Hive's warehouse directory. This can be useful to query data with Hive, even if other users rely on the data being in a specific location. If an external table is removed from Hive (`DROP TABLE`), the file is not deleted.

Note: it is not necessary for the data to exist when creating the external table. This is useful if you want to lazily add the data after creating the table.




The `ROW FORMAT` and `STORED AS` clauses are not part of standard SQL.

The `STORED AS` clause indicates what file format the data uses. This translates to an InputFormat class which reads rows of data. Currently the possible values are TextFile (the default) or SequenceFile.

The `ROW FORMAT` clause tells Hive how to read the columns in each row. If `ROW FORMAT` is not specified, then Hive uses a built-in SerDe that assumes ctrl-A as a field terminator.

## File Formats

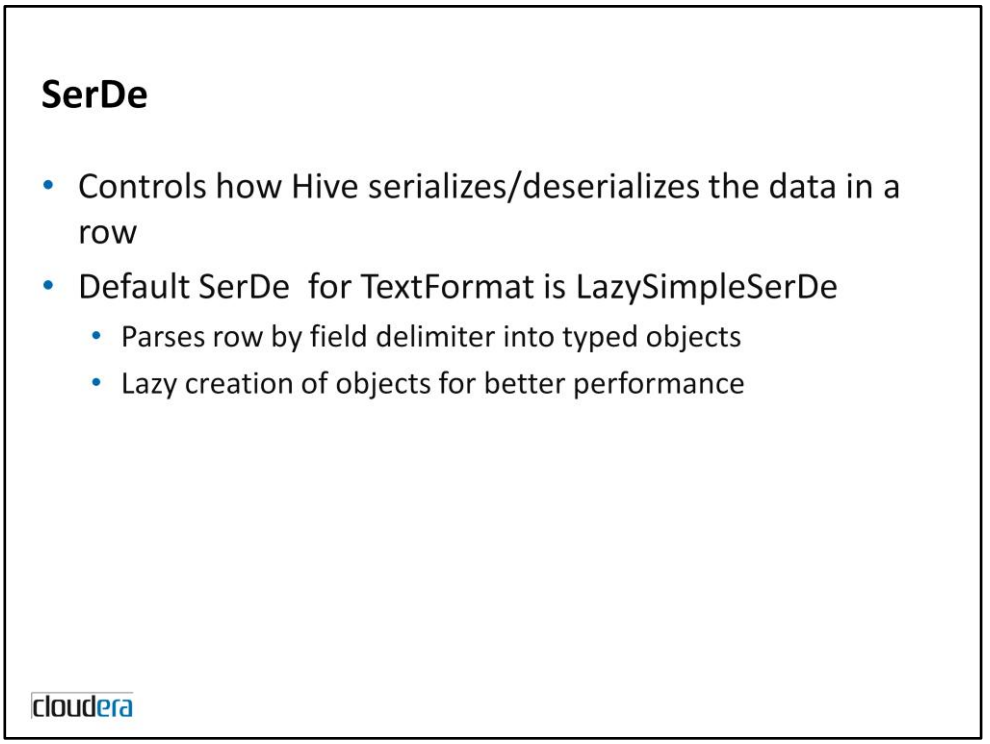
| | TEXT FILE | SEQUENCE FILE |
|---|---|---|
| DATA FORMAT | Text | Text or binary |
| COMPRESSION | File-level | Block-level |
| SPLITABLE | Maybe | Yes |

cloudera

Hive has support for various file formats. The most common ones are text files and sequence files. Text files are plain text. They can be compressed by standard compression tools prior to being loaded into Hive. Depending on the codec, this file may or may not be "splitable" which is a technique used by HDFS to make processing large files more efficient. For example, gzip is not splitable but bzip2 is.

Sequence files can contain text or binary data. The data is in (key, value) format. Sequence files can be compressed on a block-level instead of a whole file-level. This is advantageous because compressed sequence files are splitable regardless of the codec used.

The default file format is TextFile. It can be changed by setting a property: `hive.default.fileformat`.




A SerDe controls how Hive serializes/deserializes the data in a row. In other words, it knows how to parse a row into one or more column values. The default SerDe for TextFormat is called the LazySimpleSerDe. It parses a row by the field delimiter (if no field delimiter is specified, then ctrl-A is the default). Each column value is turned into a typed object (though the objects are only created if necessary).

## Example of Storing Apache Logs in Hive

```
CREATE TABLE apache_log (
  host STRING, identity STRING, user STRING,
  time STRING, request STRING, status STRING,
  size STRING, referer STRING, agent STRING)…
```

cloudera

In order to move Apache log files into Hive, first a table must be created with the proper columns. For example, here we create several STRING fields for the host, identity, user, etc.

## Example of Storing Apache Logs in Hive

- … **ROW FORMAT SERDE**

```
'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES ( "input.regex" =
"([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^
\"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^
\"]*|\"[^\"]*\") ([^ \"]*|\"[^\"]*\"))?",
"output.format.string" = "%1$s %2$s %3$s
%4$s %5$s %6$s %7$s %8$s %9$s" )
STORED AS TEXTFILE;
```

cloudera

It is necessary to change the ROW FORMAT to the SerDe class. This example uses the RegexSerDe (provided by Hive) which can parse a row using a given regular expression. Whenever this table is queried, Hive will use the regular expression to read each row of the apache log file and map it to fields.

Note: For this example to work, the jar that contains the RegexSerDe must be loaded with `ADD JAR /usr/lib/hive/lib/hive_contrib.jar`

## Delete a table

- `DROP TABLE tablename;`

| Regular table | External table |
|---|---|
| Removes table definition and all data files | Only removes table definition |

cloudera

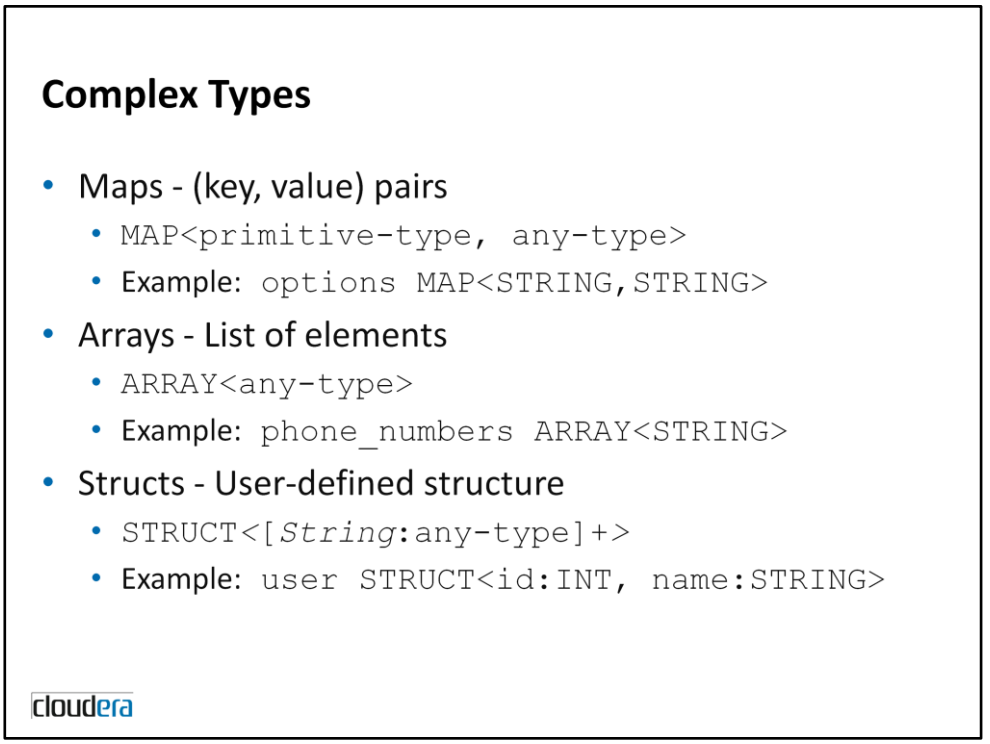
The opposite of `CREATE TABLE` is `DROP TABLE`. For regular Hive tables, this command will remove the table from the metastore as well as delete the files from the Hive warehouse in HDFS. For external tables, only the table metadata is removed.

## Primitive Data Types

| Type | Description |
|---|---|
| TINYINT | 1 byte |
| SMALLINT | 2 bytes |
| INT | 4 bytes |
| BIGINT | 8 bytes |
| FLOAT | Single precision |
| DOUBLE | Double precision |
| STRING | Sequence of characters |
| BOOLEAN | True/false |

cloudera

The `CREATE TABLE` statement can use a variety of data types for columns. The primitive types are either numbers, strings or booleans. For integers, there are 4 sizes available. A TINYINT uses only 1 byte but is limited to a small range of numbers (-256 to 255). Floating-point numbers can be single or double precision. A string is a sequence of characters in a particular character set. Boolean can only be true, false or null.




There are three complex data types in Hive. These are an extension to the SQL language.

A map is a set of (key, value) pairs. For example, a group of options could be represented as a map where the key and value are both strings. To retrieve a value from a map, reference it by key such as *options['myopt']*.

An array is a list of items of the same type. Individual items can be accessed by a zero-based index. For example, to retrieve the first phone number in an array called phone_numbers, use *phone_numbers[0]*.

A struct is a user-defined structure of any number of typed fields. To access the fields, use a dot notation such as `user.id` and `user.name`.

The complex types can also be nested.

## Creating a table with an array

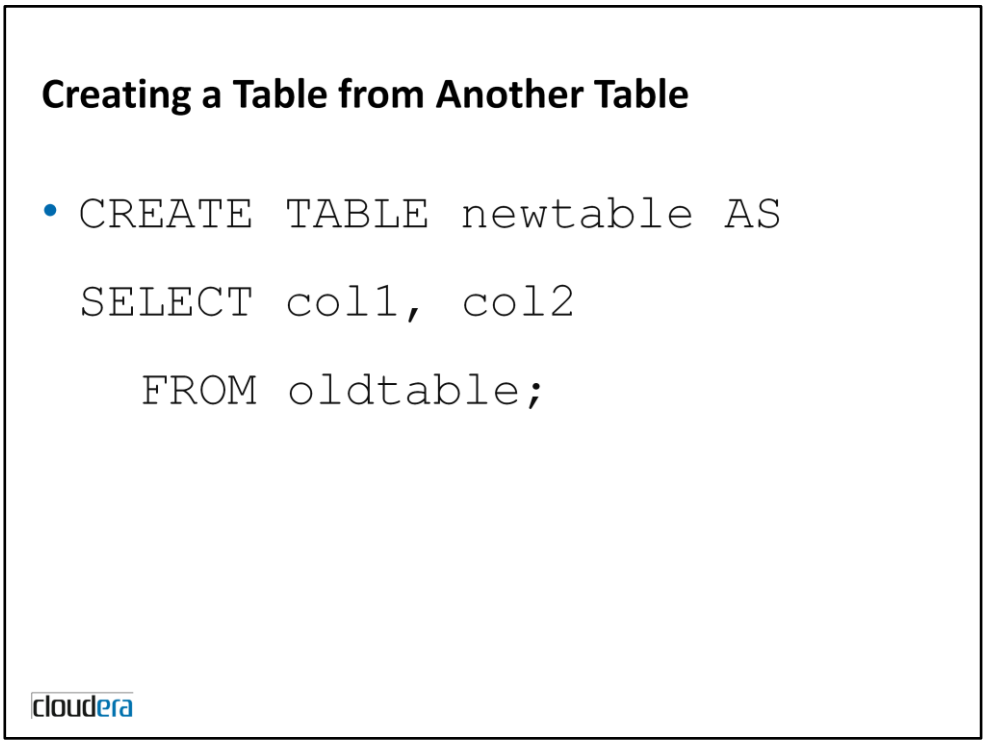
```
CREATE TABLE tablename
   (col1 STRING, col2 ARRAY<STRING>)
ROW FORMAT DELIMITED FIELDS
   TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY ','
STORED AS TEXTFILE;
```

cloudera

To create a table with a complex data type, use `COLLECTION ITEMS TERMINATED BY` in the `CREATE TABLE`. For example, use the above CREATE TABLE if the data had 2 columns (separated by a tab) and the array was comma separated like this:

| | |
|---|---|
| 1 | a,b,c |
| 2 | d,e |
| 3 | f,g,h,i |

For map structures such as JSON-like data, use `MAP KEYS TERMINATED BY` to specify the character between the key and value.

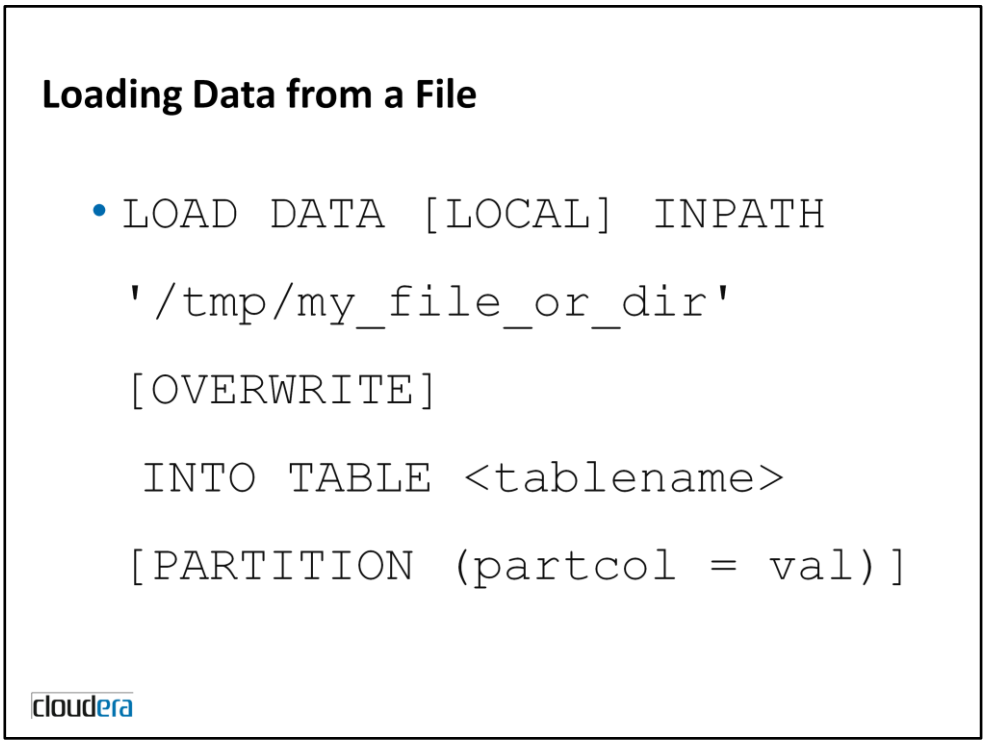



It is possible to create a table from an existing table using `CREATE TABLE..AS SELECT` (i.e., "CTAS"). The `SELECT` statement can be any valid `SELECT` statement in Hive. The keyword `AS` is required.

The new table derives the column definitions from the `SELECT`ed data. For example, if col1 is an INT and col2 is a String in the *oldtable* table, then *newtable* will have two columns of type `INT` and `STRING` respectively. The `SELECT` can contain column aliases, which will become the column names in the new table.

The CTAS statement is atomic, which means other sessions will not see this table unless is completes successfully.

This feature was added in Hive 0.5.



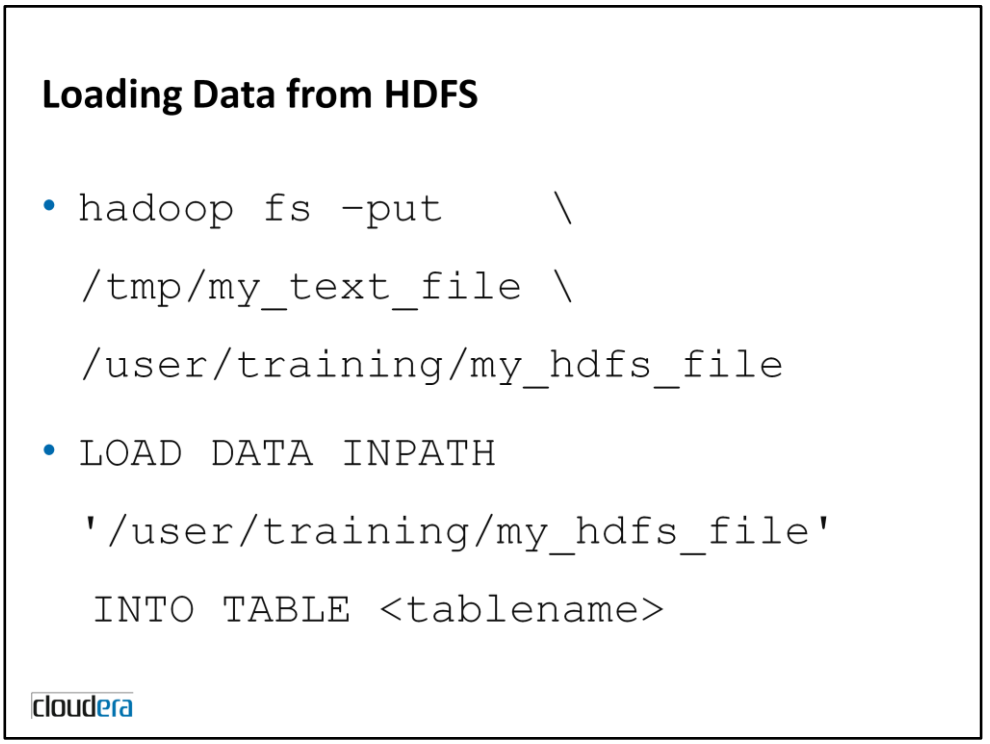

To load a file from the local hard drive into a Hive table, use the `LOAD DATA LOCAL INPATH`. This command will copy the specified file (e.g., /tmp/my_file_or_dir) into the Hive table. The filepath may be a filename or a directory. If a directory is given, then all files in that directory will be loaded into the Hive table. Note, the directory can have files, but not subdirectories. If the path is a relative path, it attempts to locate the file or directory relative to the user's current working directory.

The format of the file must match the table definition, which was specified when the table was created.

The keyword OVERWRITE can also be specified: `LOAD DATA LOCAL INPATH 'file' OVERWRITE INTO TABLE <tablename>`. This will remove any data that was in the table prior to loading the file. Without the OVERWRITE keyword, the file or files are added to the table.

The data can be loaded into a specific partition if the target table is partitioned. In fact, if the target table is partitioned, it is mandatory to specify a partition clause in the `LOAD DATA`. (Partitioning will be covered later)

Hive does not modify the file or attempt to parse it during the load. It is merely copying the file into Hive's HDFS directory (/user/hive/warehouse) unless it is an `EXTERNAL` table.




If the file has already been loaded into the Hadoop Distributed FileSystem (HDFS), then the LOCAL keyword should be omitted. One way to copy a file to HDFS is the `hadoop fs -put` command. It takes a local file (`/tmp/my_text_file`) and copies the data in that file into the HDFS local (`/user/training/my_hdfs_file`).

The `LOAD DATA INPATH` command will **move** the file from `/user/training/my_hdfs_file` into the Hive table. Since Hive also stores data in HDFS (in `/user/hive/warehouse`), this "move" operation is cheap since only the HDFS metadata needs to be updated.

## Overwriting a Table

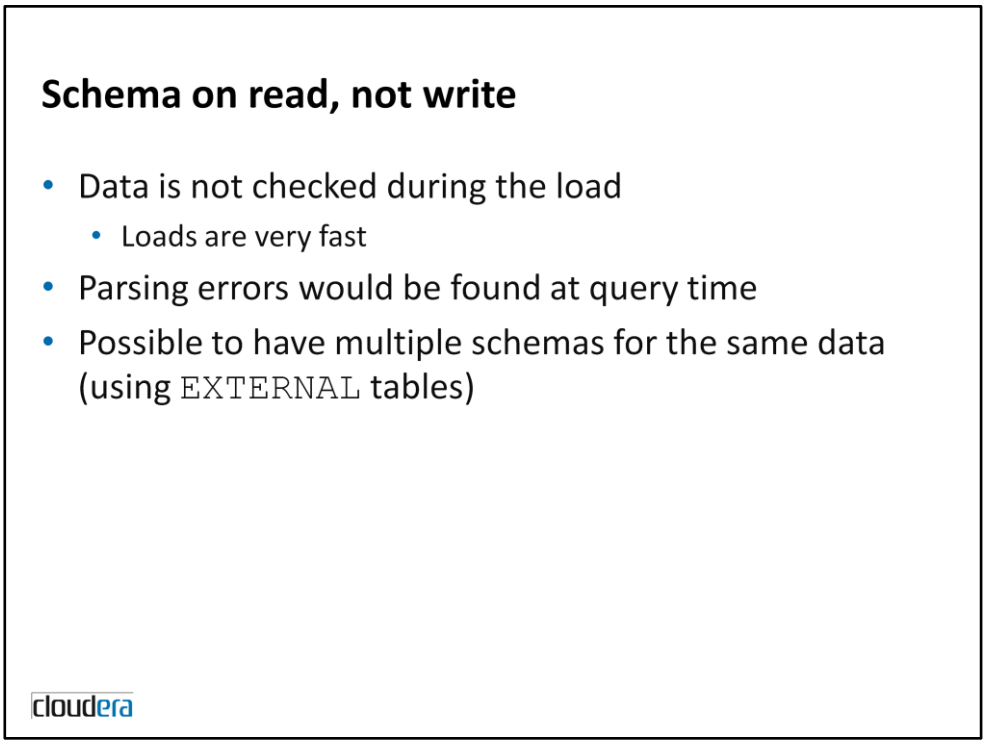
- Replace the rows of a table
- Example:

```
INSERT OVERWRITE TABLE table_foo
[PARTITION (partcol = val)]
SELECT * FROM table_bar;
```

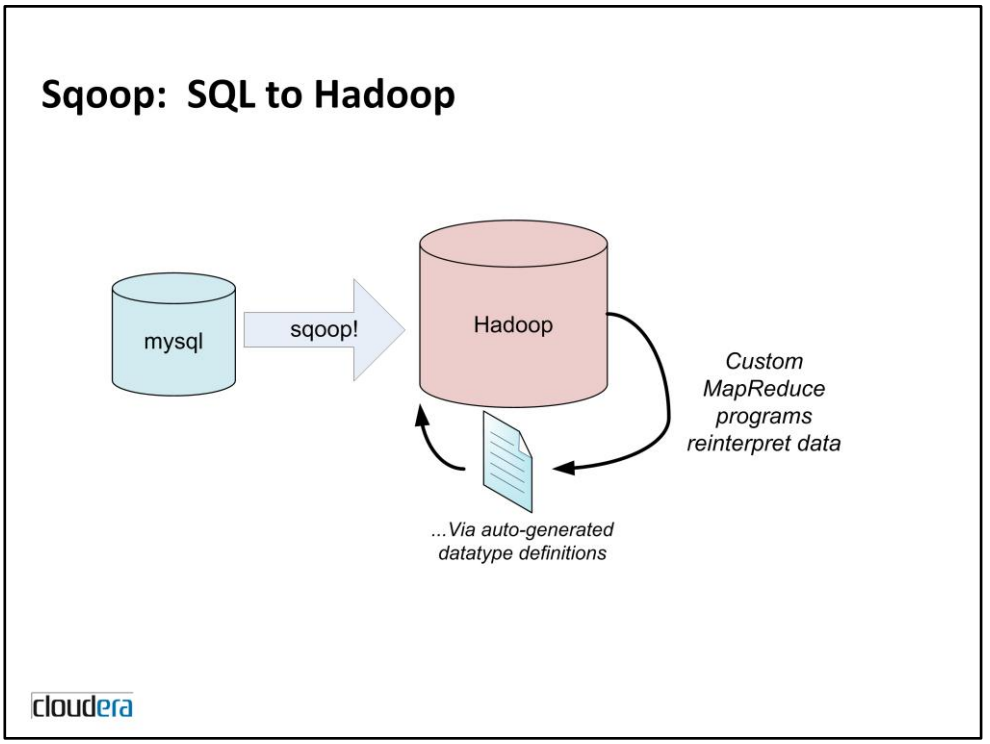
cloudera

It is possible to insert rows into a table by selecting from an existing table. However, the OVERWRITE keyword is required. The inserted rows replace any prior data in this table. The `SELECT` can be any valid `SELECT` statement in Hive.

If the table is partitioned, then a partition clause must be specified. This would only create/replace that particular partition.

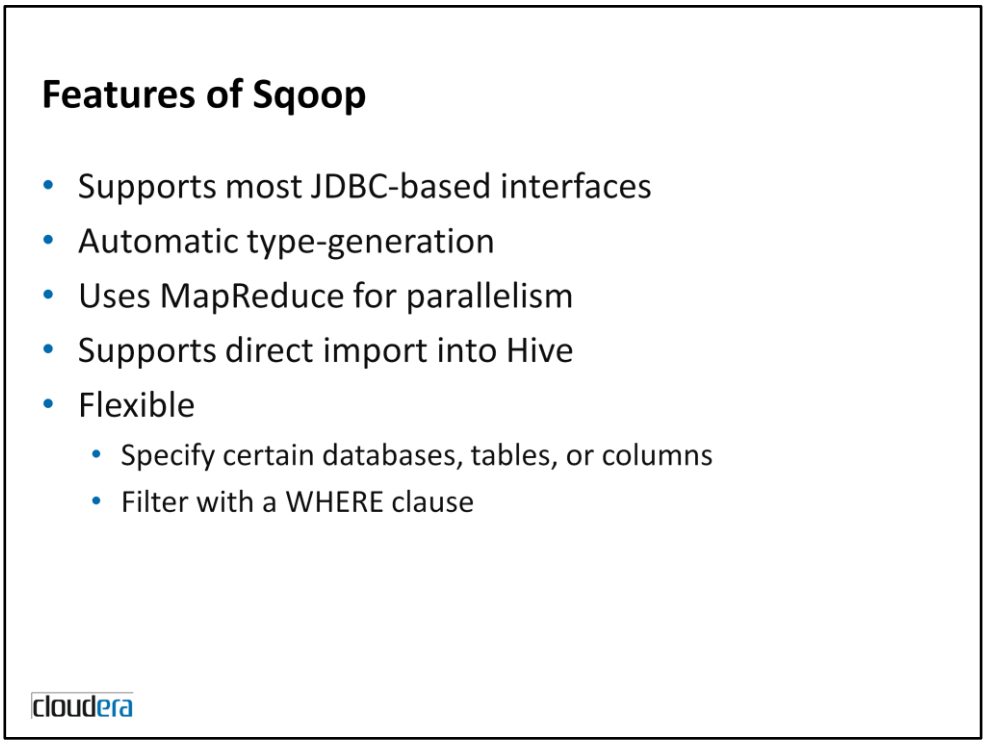



It is interesting that Hive uses a "schema on read" approach, whereas many traditional databases use a "schema on write" technique. This means that when you `LOAD` data into Hive, it does not parse, verify and serialize the data into its own format. Instead, Hive merely moves files (or copies from the local filesystem). Or in the case of an external table, it merely points to the files. This has some benefits such as very fast loads. It is also very flexible; it is possible to have two schemas for the same underlying data using `EXTERNAL` tables.




Sqoop is an open-source project supported by Cloudera for importing data into Hadoop. Sqoop automatically generates the necessary Java classes, then runs a MapReduce job to read data in parallel and creates a set of files in HDFS.

Full documentation for Sqoop can be found at:
http://archive.cloudera.com/docs/sqoop/



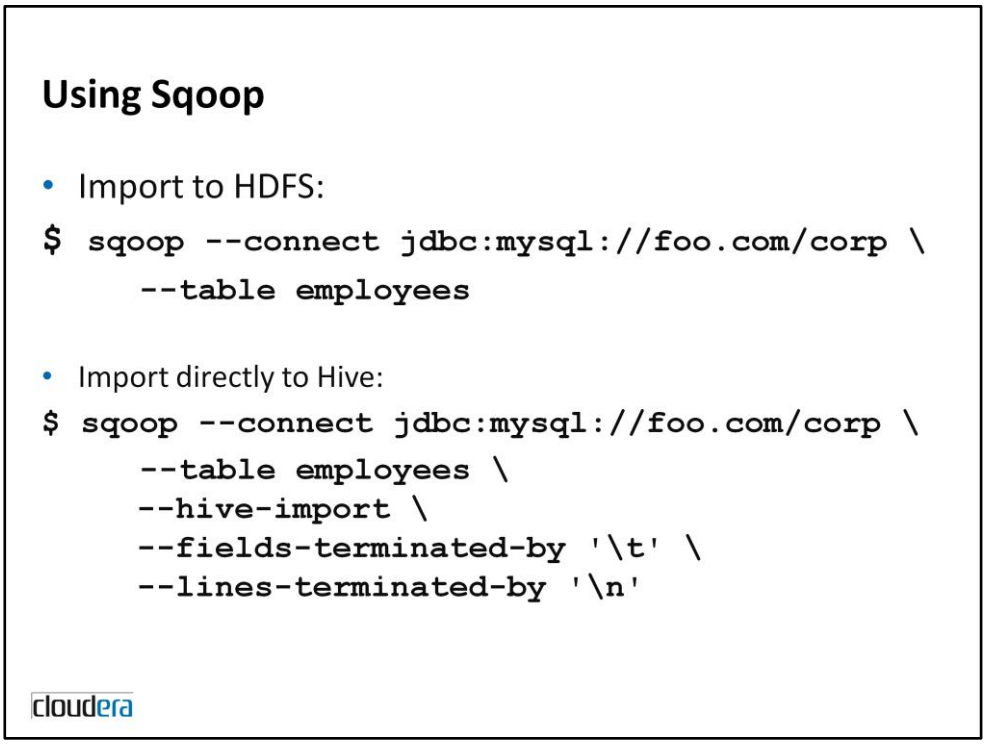

Sqoop uses JDBC to connect to databases. JDBC is a compatibility layer that allows a program to access many different databases through a common API. Slight differences in the SQL language spoken by each database, however, may mean that Sqoop can't use every database out of the box.

Databases currently supported by Sqoop:

| | |
|---|---|
| HSQLDB | v1.8.0+ |
| MySQL | v5.0+ |
| Oracle | v10.2.0+ |
| PostgreSQL | v8.3+ |

Sqoop will inspect the target tables and automatically convert from SQL types to Java types using the JDBC specification type-mapping. Sqoop leverages Hadoop's MapReduce to import data in parallel. In addition, Sqoop can import data into HDFS files or Hive. In order to import into Hive, Sqoop will invoke the correct `CREATE TABLE` and `LOAD DATA INPATH` commands.

Sqoop has many options for controlling what gets imported and how it is done. For example, it is possible to specify all tables or a subset of tables to be imported. If only certain columns or rows are desired, there are options for supplying column names or a `WHERE` clause.




Use the `sqoop` command-line program to connect to a database using a JDBC connection URL. The option --table indicates which table to import. Sqoop can also import directly to Hive. The `--hive-import` option tells Sqoop to import the table or tables into HDFS and then run a script that executes the appropriate `CREATE TABLE` and `LOAD DATA INPATH` commands.

The `--username` and `--password` options may be required to authenticate to the database.

## Conclusion

In this chapter, you have learned:

- How to create tables in Hive
- How Hive uses InputFormats and SerDes to parse data
- How to remove tables from Hive
- How to load data into Hive
- How to import data with Sqoop

cloudera