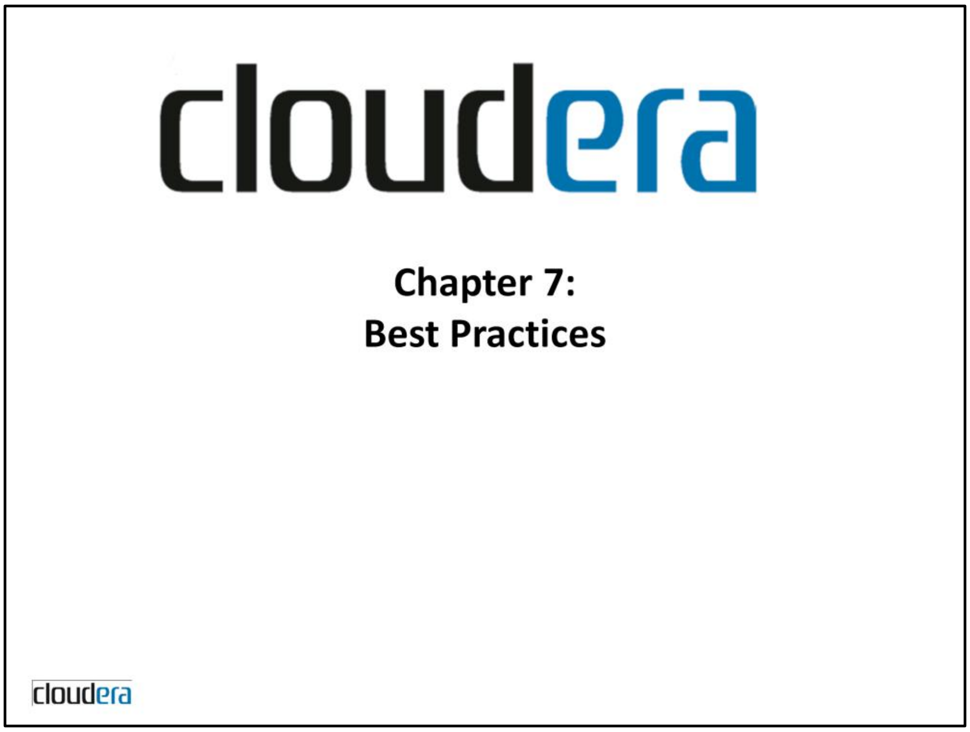
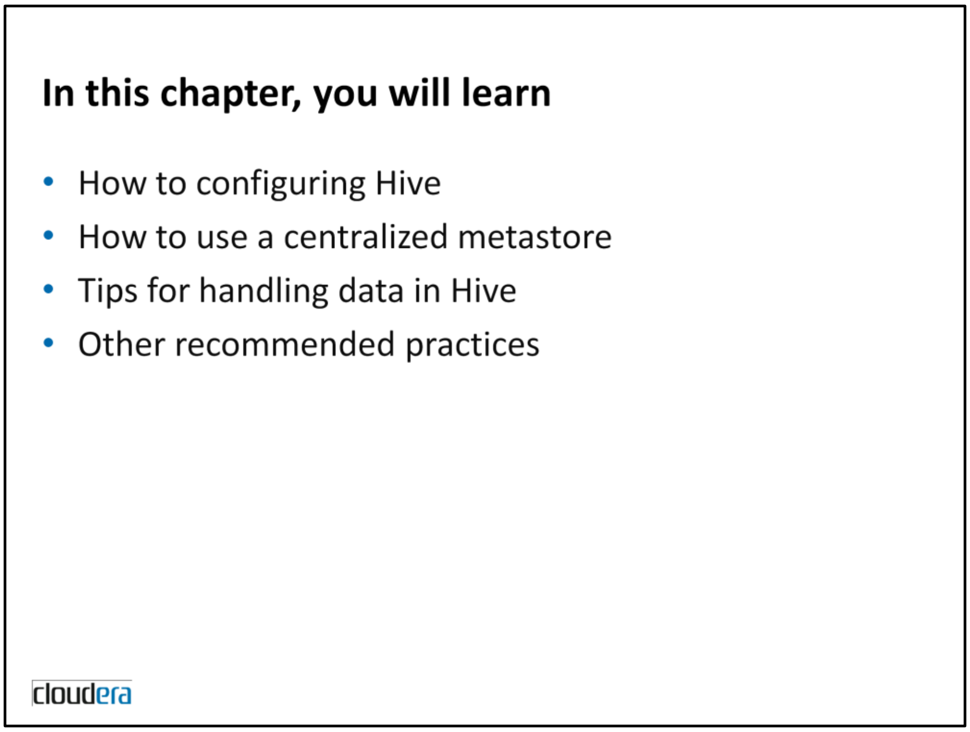


cloudera

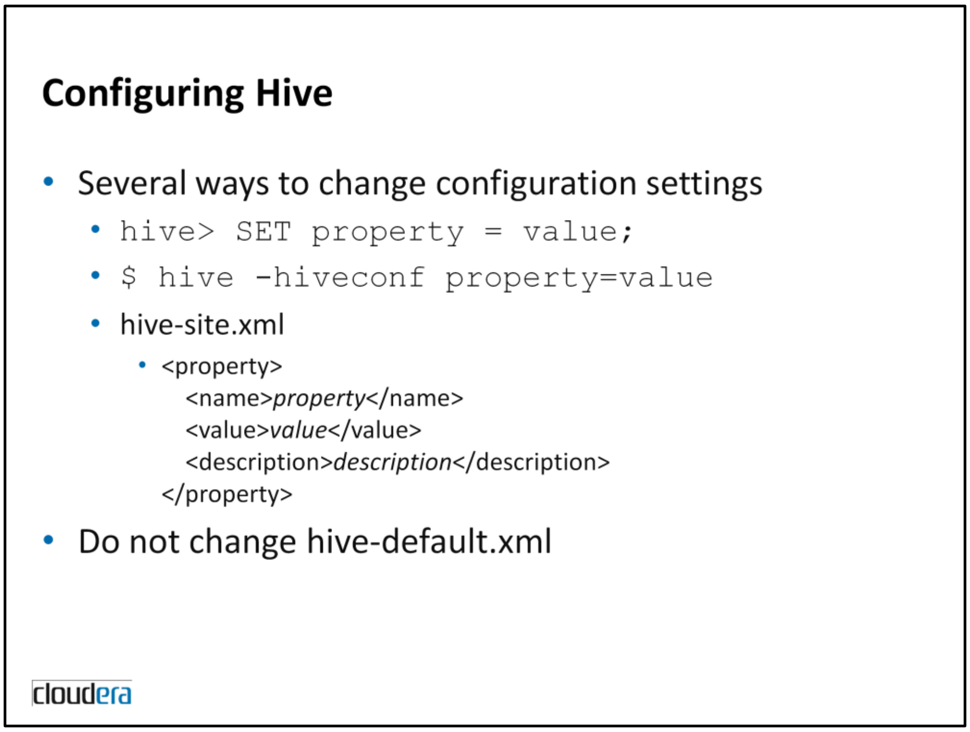
# Chapter 7: Best Practices

cloudera



## In this chapter, you will learn

- How to configuring Hive
- How to use a centralized metastore
- Tips for handling data in Hive
- Other recommended practices



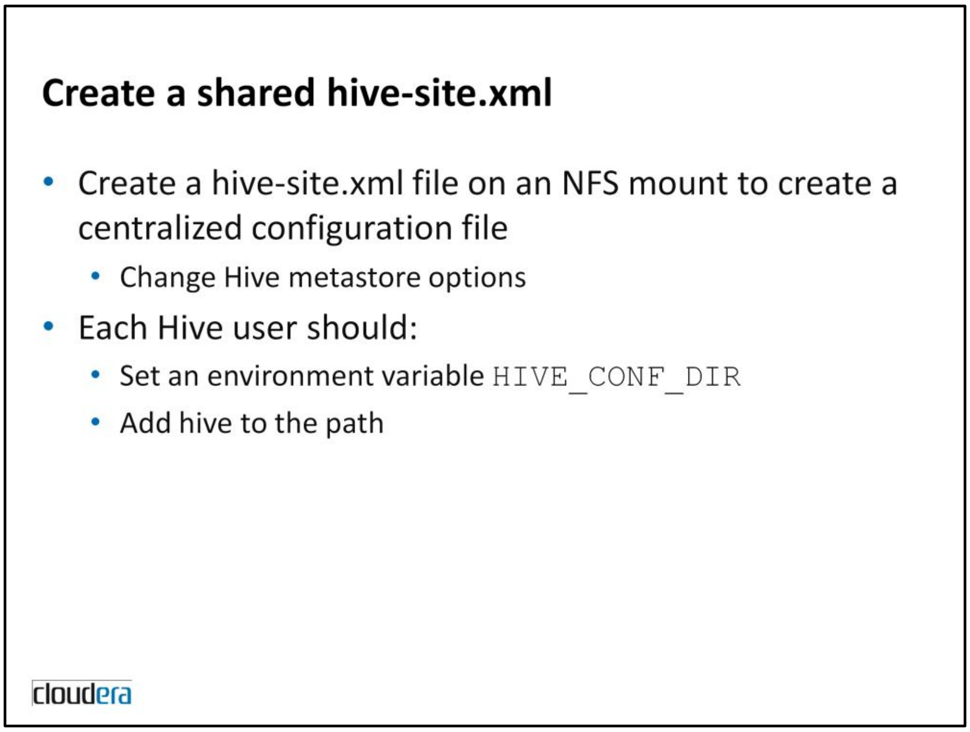
## Configuring Hive

- Several ways to change configuration settings
  - `hive> SET property = value;`
  - `$ hive -hiveconf property=value`
  - hive-site.xml

    -
      ```
      <property>
        <name>property</name>
        <value>value</value>
        <description>description</description>
      </property>
      ```
- Do not change hive-default.xml

cloudera

There are several ways to change configuration settings for Hive. One technique is the "`SET`" keyword which can be used from within the Hive shell. Alternatively, properties can be given at the time Hive is invoked by using the `–hiveconf` option. A third possibility is putting properties in a file called `hive-site.xml`. This file is intended for users to override settings in hive-default.xml. It is not recommended to edit `hive-default.xml` directly.



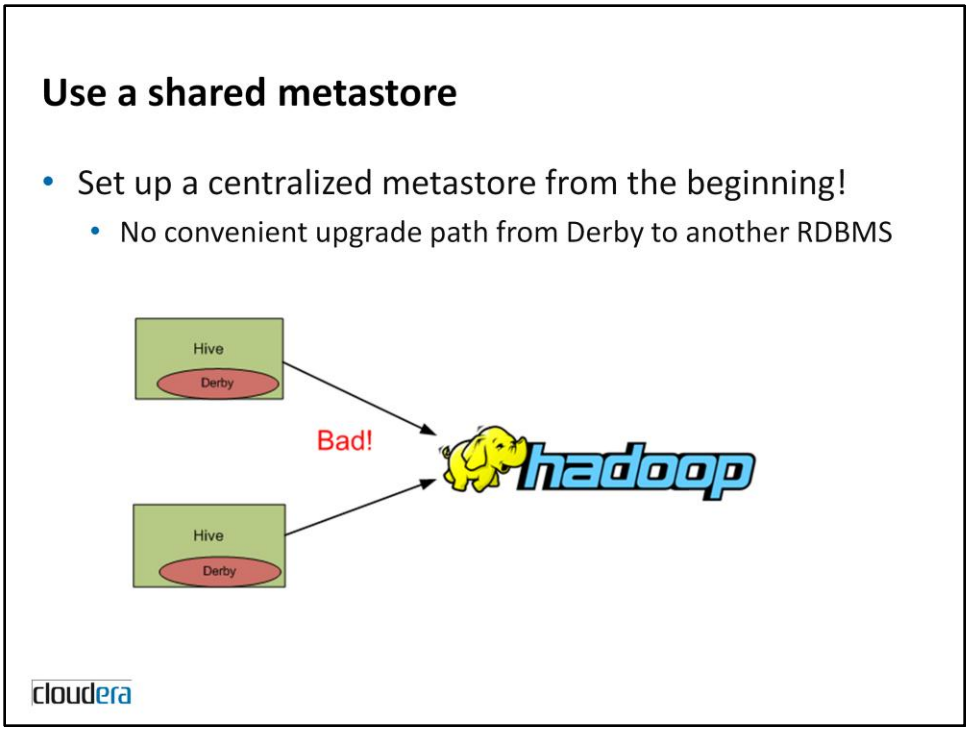
## Create a shared hive-site.xml

- Create a hive-site.xml file on an NFS mount to create a centralized configuration file
  - Change Hive metastore options
- Each Hive user should:
  - Set an environment variable `HIVE_CONF_DIR`
  - Add hive to the path

cloudera

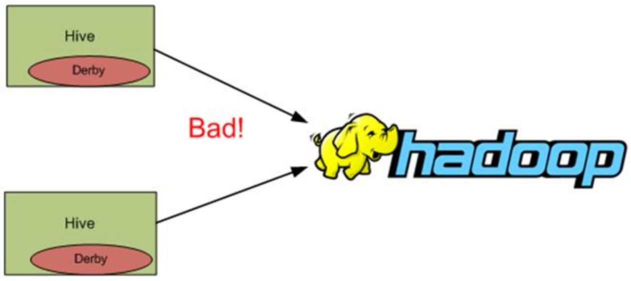
It is recommended that all Hive users share some base configuration options. The default options (in hive-default.xml) are not optimal for production environments. In particular the default metastore should be changed. The optimal way to do this is to create one hive-site.xml file on an NFS mount and have each user point to it.

Each Hive user would then want to set a local environment variable (e.g., using their .bashrc file) called `HIVE_CONF_DIR` which has the path to the hive-site.xml file. Additionally, each user would want to add Hive to their path.



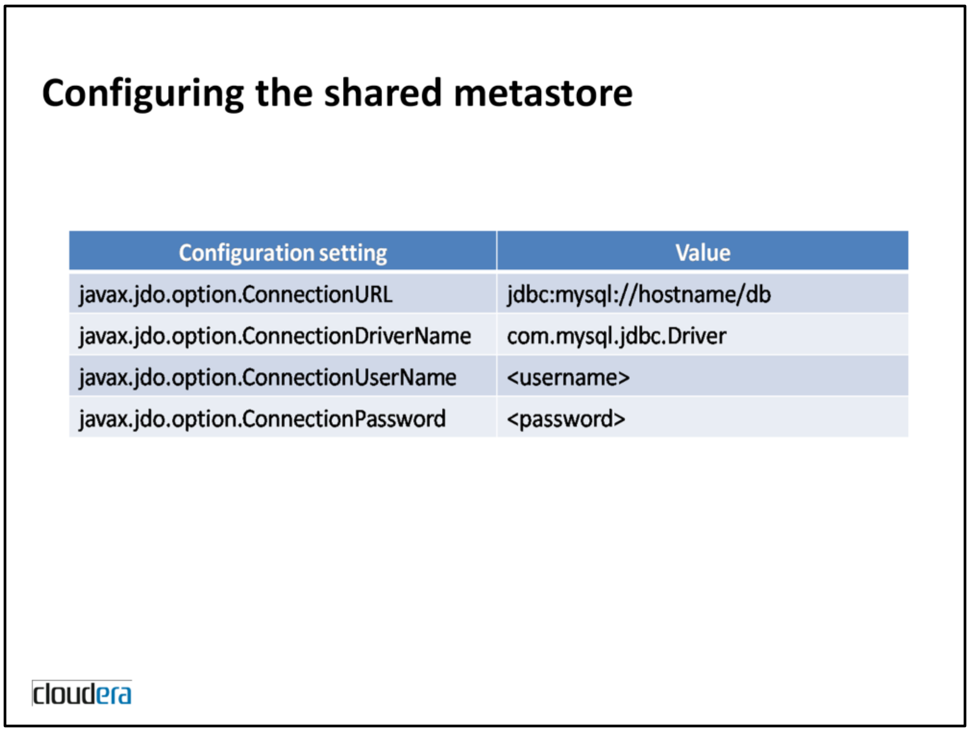
## Use a shared metastore

- Set up a centralized metastore from the beginning!
  - No convenient upgrade path from Derby to another RDBMS



By default, Hive uses an embedded Derby database to store its metadata. This means that multiple Hive clients would have their own metastores. This can cause problems.

It is important to set up a shared metastore from the start. There is currently no convenient upgrade path from the embedded Derby metastore to a centralized one. So plan ahead!

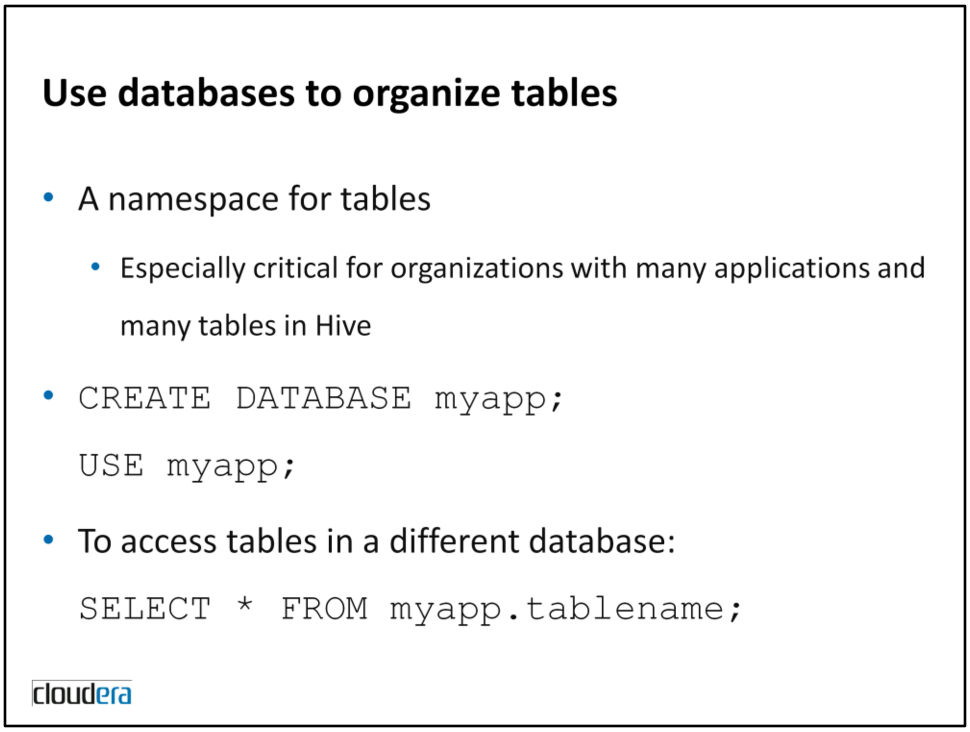


## Configuring the shared metastore

| Configuration setting | Value |
|---|---|
| javax.jdo.option.ConnectionURL | jdbc:mysql://hostname/db |
| javax.jdo.option.ConnectionDriverName | com.mysql.jdbc.Driver |
| javax.jdo.option.ConnectionUserName | <username> |
| javax.jdo.option.ConnectionPassword | <password> |

Most JDBC-compliant databases can be used for Hive's metastore (MySQL and PostgreSQL are popular choices). There are important settings that need to be configured to point Hive to the correct database. The option `javax.jdo.option.ConnectionURL` specifies a URL for the database. This can also include a parameter that tells Hive to create the database (aka schema) if it does not exist:
`jdbc:mysql://hostname/db?createDatabaseIfNotExist=true`

The option `javax.jdo.option.ConnectionDriverName` gives the name of the JDBC driver class. It is also important that the JDBC library is in Hive's classpath.



## Use databases to organize tables

- A namespace for tables
  - Especially critical for organizations with many applications and many tables in Hive
- `CREATE DATABASE myapp;`
  `USE myapp;`
- To access tables in a different database:
  `SELECT * FROM myapp.tablename;`

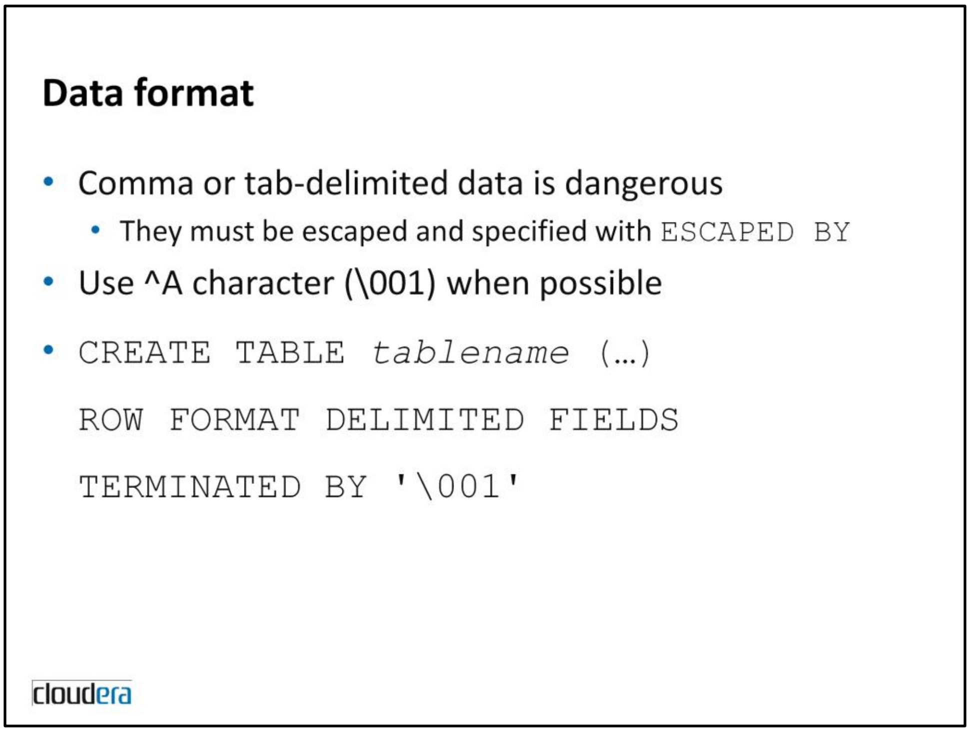
cloudera

A database is a namespace for a collection of tables. Similar to how directories in a file system help you organize files, databases can be useful for organizing tables. A certain user or application may choose to create a database for the tables related to that user or application.

By default Hive uses a default database. To access tables in a user-defined database, either "`USE` *`databasename`*" or qualify the tables with a "dot" notation such as *`db.tablename`*.

An entire database can be deleted with the command `DROP DATABASE` *`databasename`*. Caution: this command removes all objects (such as tables) that are contained in this database.

This feature is currently being developed (https://issues.apache.org/jira/browse/HIVE-675) and is likely coming in Hive 0.6.



## Data format

- Comma or tab-delimited data is dangerous
  - They must be escaped and specified with `ESCAPED BY`
- Use ^A character (\001) when possible
- `CREATE TABLE` *`tablename`* `(…)`

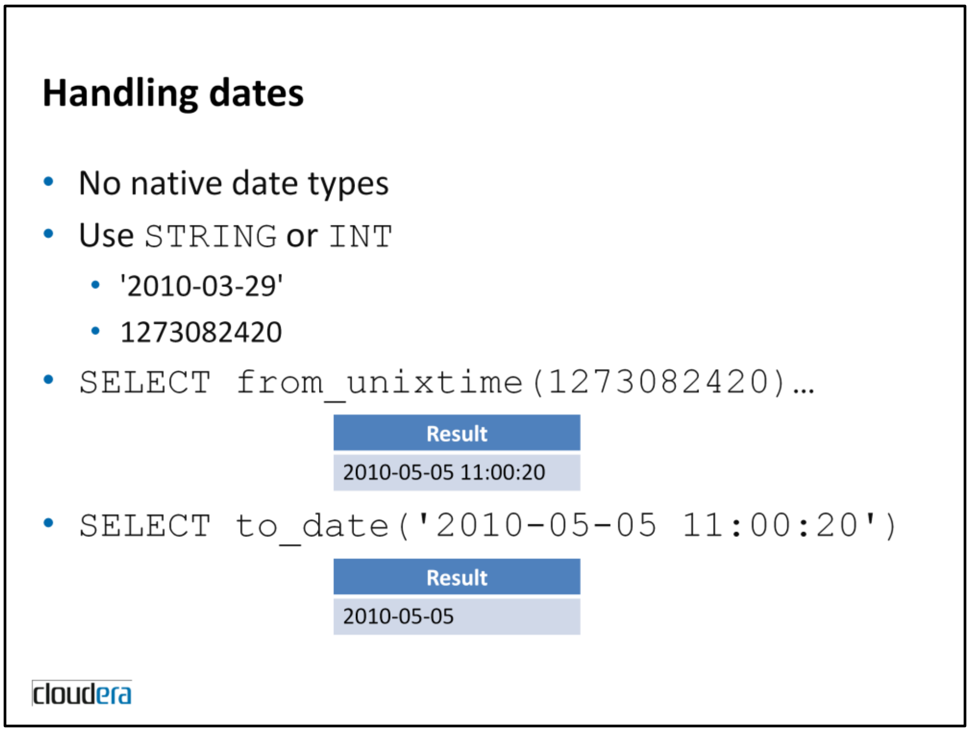
  ```
  ROW FORMAT DELIMITED FIELDS
  TERMINATED BY '\001'
  ```

cloudera

Comma- or tab-delimited data formats are fairly common. However, in Hive they can be problematic if the data could contain a tab or comma that is not meant to be a delimiter. The characters would need to be escaped in the file (e.g., by a backslash) and the `ESCAPED BY` clause specified in the `CREATE TABLE`.

A good practice is to use a ^A (ctrl-A) to separate fields in a text file. In the `CREATE TABLE` statement, use the ascii code for the field terminator:

```
CREATE TABLE tablename (…)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\001'
```



## Handling dates

- No native date types
- Use `STRING` or `INT`
  - '2010-03-29'
  - 1273082420
- `SELECT from_unixtime(1273082420)…`

| Result |
|---|
| 2010-05-05 11:00:20 |

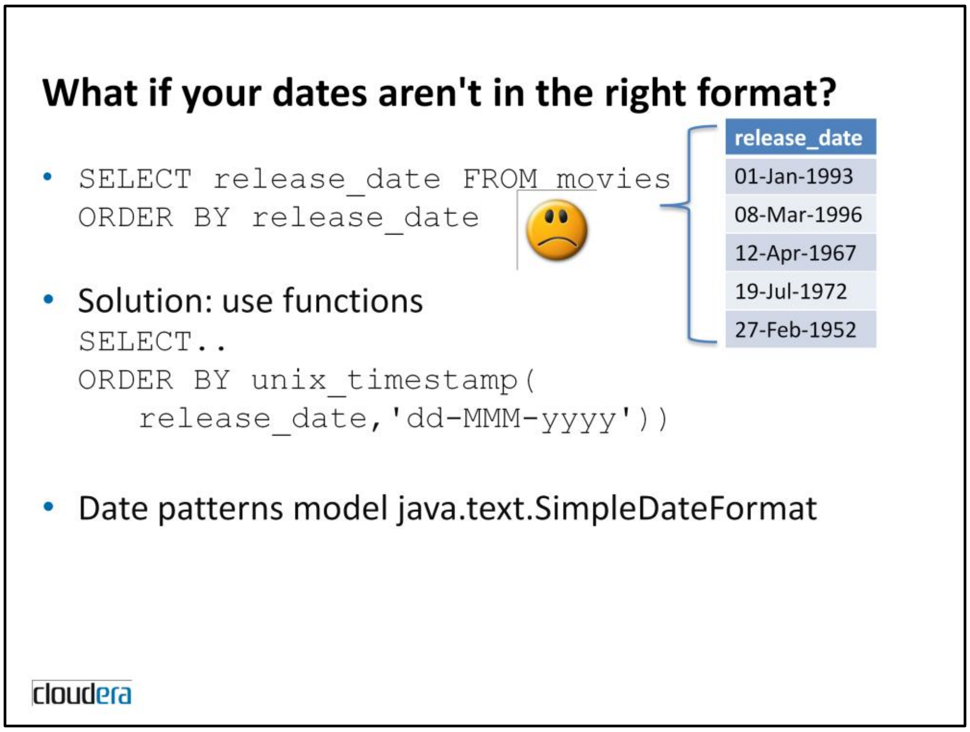
- `SELECT to_date('2010-05-05 11:00:20')`

| Result |
|---|
| 2010-05-05 |

cloudera

There is no native date or time data type in Hive. The recommended approach for storing dates is to use a string or integer type. As a string, store the data in ISO format ("year-month-day"). As an integer, store the date or datetime as a the number of seconds since 1970-01-01 (Unix epoch).

There are useful functions for converting unix timestamps to date format and vice versus. There are also functions for extracting or transforming dates.



## What if your dates aren't in the right format?

- `SELECT release_date FROM movies`
  `ORDER BY release_date`

| release_date |
|---|
| 01-Jan-1993 |
| 08-Mar-1996 |
| 12-Apr-1967 |
| 19-Jul-1972 |
| 27-Feb-1952 |

- Solution: use functions

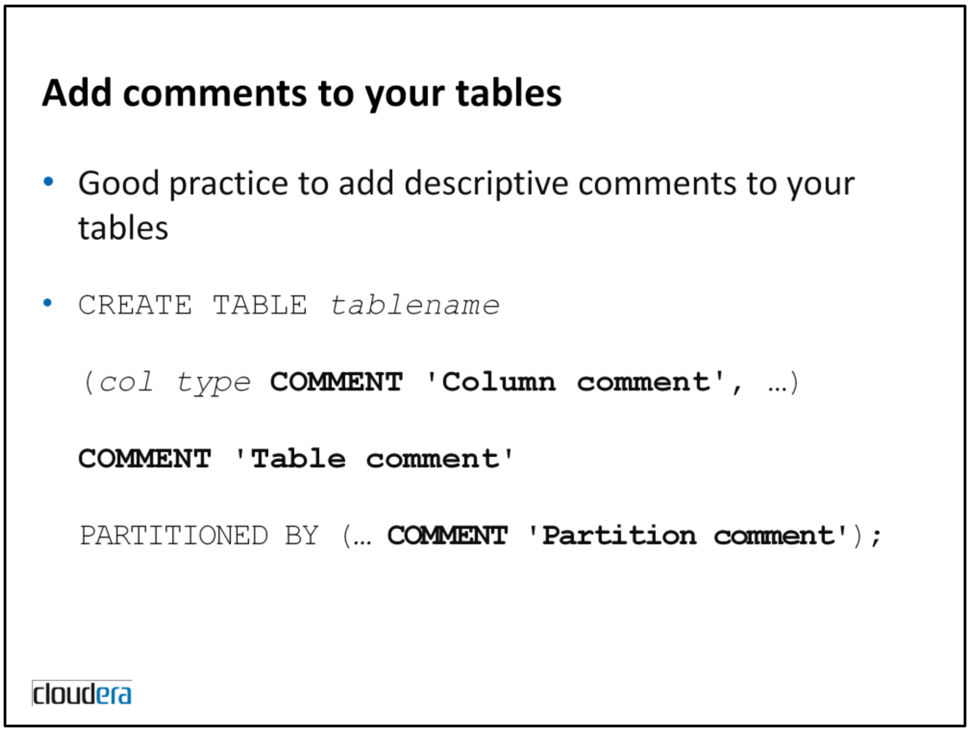
```
SELECT..
ORDER BY unix_timestamp(
   release_date,'dd-MMM-yyyy'))
```

- Date patterns model java.text.SimpleDateFormat

cloudera

If your data file contains a date representation that is not a Unix timestamp or a string in ISO format, then you will probably need a function:
`unix_timestamp(STRING date, STRING pattern)`

The pattern can be any pattern used by java.text.SimpleDateFormat.

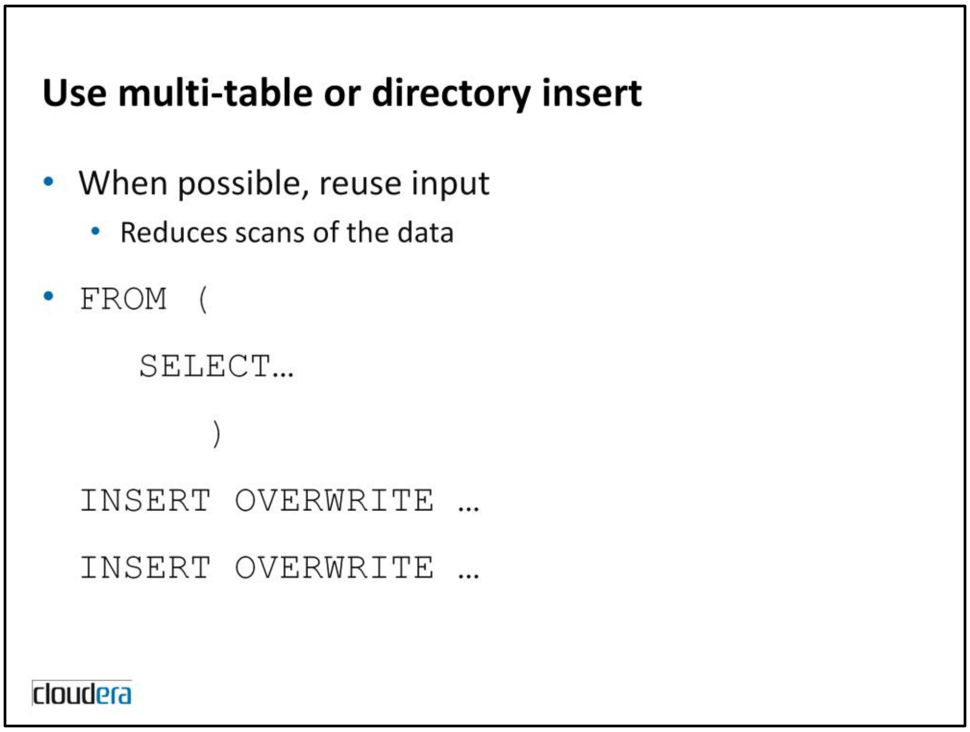


## Add comments to your tables

- Good practice to add descriptive comments to your tables

-
  ```
  CREATE TABLE tablename
  (col type COMMENT 'Column comment', …)
  COMMENT 'Table comment'
  PARTITIONED BY (… COMMENT 'Partition comment');
  ```

Adding comments to your table definitions is a good practice. These comments are saved in the metastore and are visible in `DESCRIBE EXTENDED`. A recommended practice is to use column comments to describe the format of the data (e.g., if the column is a `STRING` that stores a date). It may also be helpful to include the creator's name in the table comment.



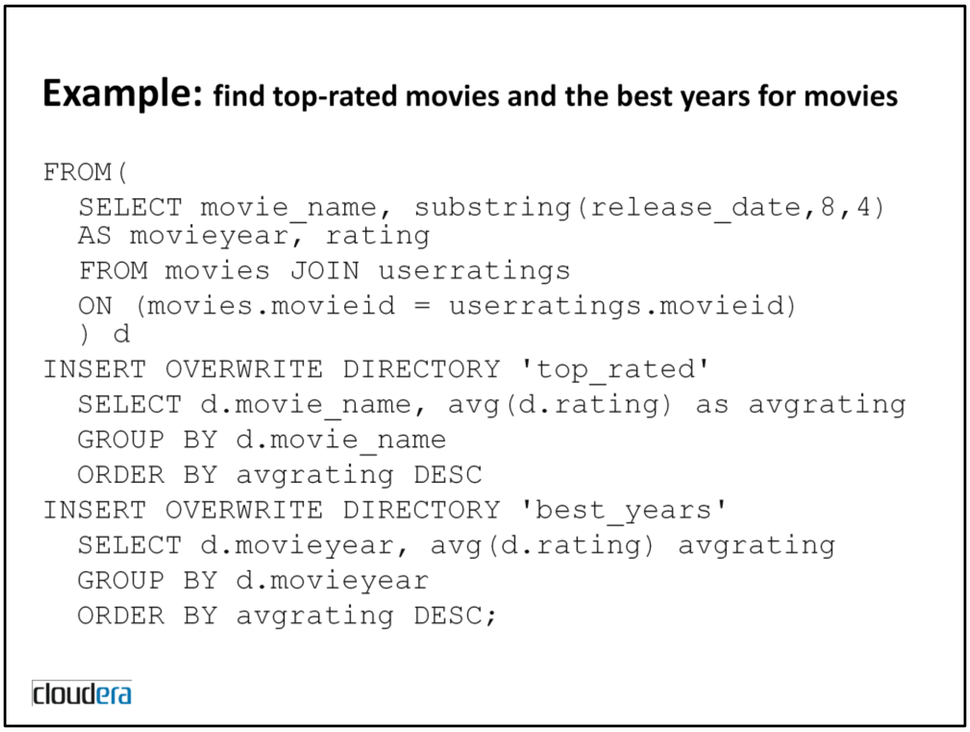
## Use multi-table or directory insert

- When possible, reuse input
  - Reduces scans of the data

-
  ```
  FROM (
     SELECT…
         )
  INSERT OVERWRITE …
  INSERT OVERWRITE …
  ```

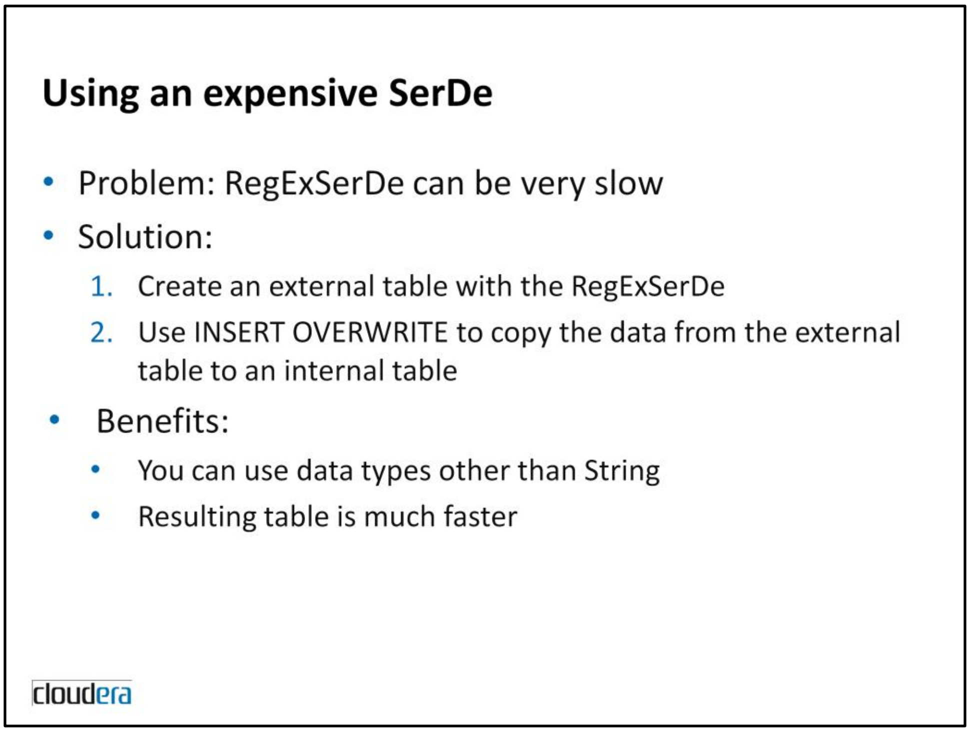
cloudera

Use the multi-table insert syntax whenever possible to reduce the number of times data is scanned.


## Example: find top-rated movies and the best years for movies

```
FROM(
  SELECT movie_name, substring(release_date,8,4)
  AS movieyear, rating
  FROM movies JOIN userratings
  ON (movies.movieid = userratings.movieid)
  ) d
INSERT OVERWRITE DIRECTORY 'top_rated'
  SELECT d.movie_name, avg(d.rating) as avgrating
  GROUP BY d.movie_name
  ORDER BY avgrating DESC
INSERT OVERWRITE DIRECTORY 'best_years'
  SELECT d.movieyear, avg(d.rating) avgrating
  GROUP BY d.movieyear
  ORDER BY avgrating DESC;
```

This example calculates two things using the same base data. First a `SELECT` joins the movies table to the userratings table. Then that joined data is used to: find the top-rated movies and write the results to an HDFS directory named `top_rated`, and find the years which had the highest rated movies overall and write the data to an HDFS directory named `best_years`.

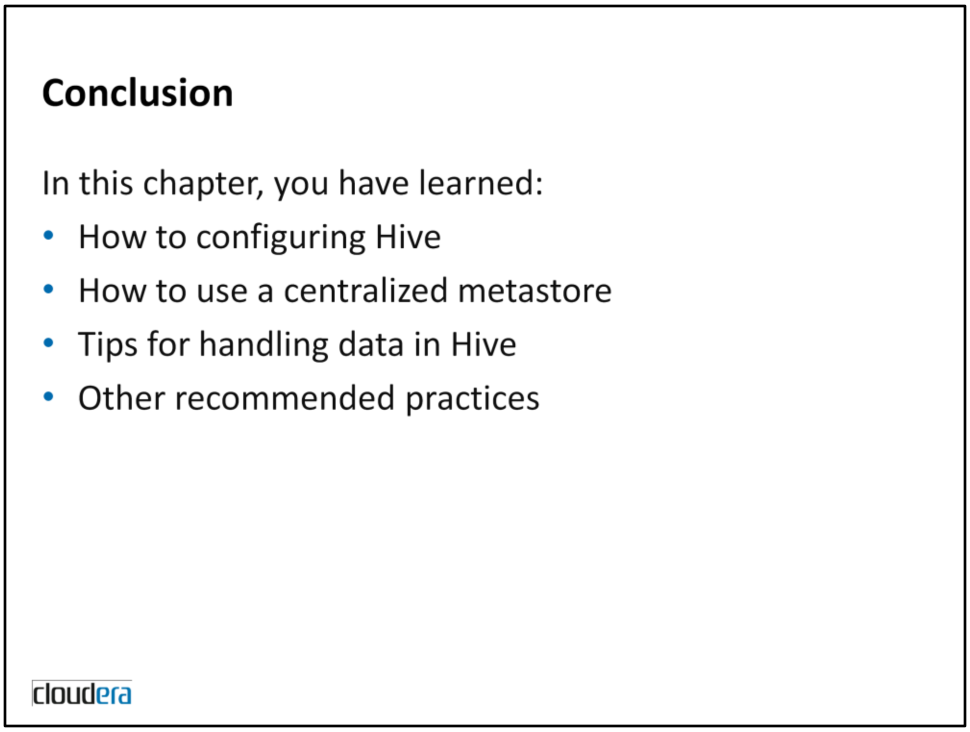


## Using an expensive SerDe

- Problem: RegExSerDe can be very slow
- Solution:
  1. Create an external table with the RegExSerDe
  2. Use INSERT OVERWRITE to copy the data from the external table to an internal table
- Benefits:
  - You can use data types other than String
  - Resulting table is much faster

cloudera

Sometimes you need a custom SerDe. However, these can be very slow if they need to do fancy parsing of the rows into columns. A better approach is to create an external table with the RegExSerDe. Instead of querying from that table regularly, use `INSERT OVERWRITE` to do a one-time copy of the data into a new table. There are several benefits to this technique. The RegExSerDe only returns `STRING` columns, but you may prefer to use other types, such as integers for numeric data or dates. This resulting table will also be faster since it no longer relies on the RegExSerDe for parsing rows.



## Conclusion

In this chapter, you have learned:

- How to configuring Hive
- How to use a centralized metastore
- Tips for handling data in Hive
- Other recommended practices

cloudera