Hive

Hive

You've now had some experience writing mapper and reducer programs and getting them to run as a MapReduce job on Hadoop. You've also had some experience using Python's MRJob class, which simplifies the process of writing and running mappers and reducers.

As you've probably noticed, figuring out how to extract the information that you want from the data using mappers and reducers can take quite a bit of ingenuity and work. Wouldn't it be nice if you could use something like SQL to get those answers instead? Writing SQL statements is much more intuitive and much simpler than writing mappers and reducers.

As it turns out, you can.

By 2007, Facebook had a lot of programmers with experience using SQL to query data, but not much experience using Java or writing MapReduce jobs. So they developed a way to convert SQL queries into MapReduce jobs. The result was some software called **Hive**. Hive has been so popular that it is now an open source application within the Apache Software Foundation, and is one of their top-level projects. It is now used by companies such as Netflix and Amazon.

Hive doesn't quite use SQL - it uses its own version, called **Hive Query Language** (**HiveQL**, **HQL**). HQL is more limited than SQL - it has a data definition language and a data manipulation language, but each is more limited than those of SQL. For example, the DML does not allow row-level inserts, updates, or deletes, and there is limited support for sub queries.

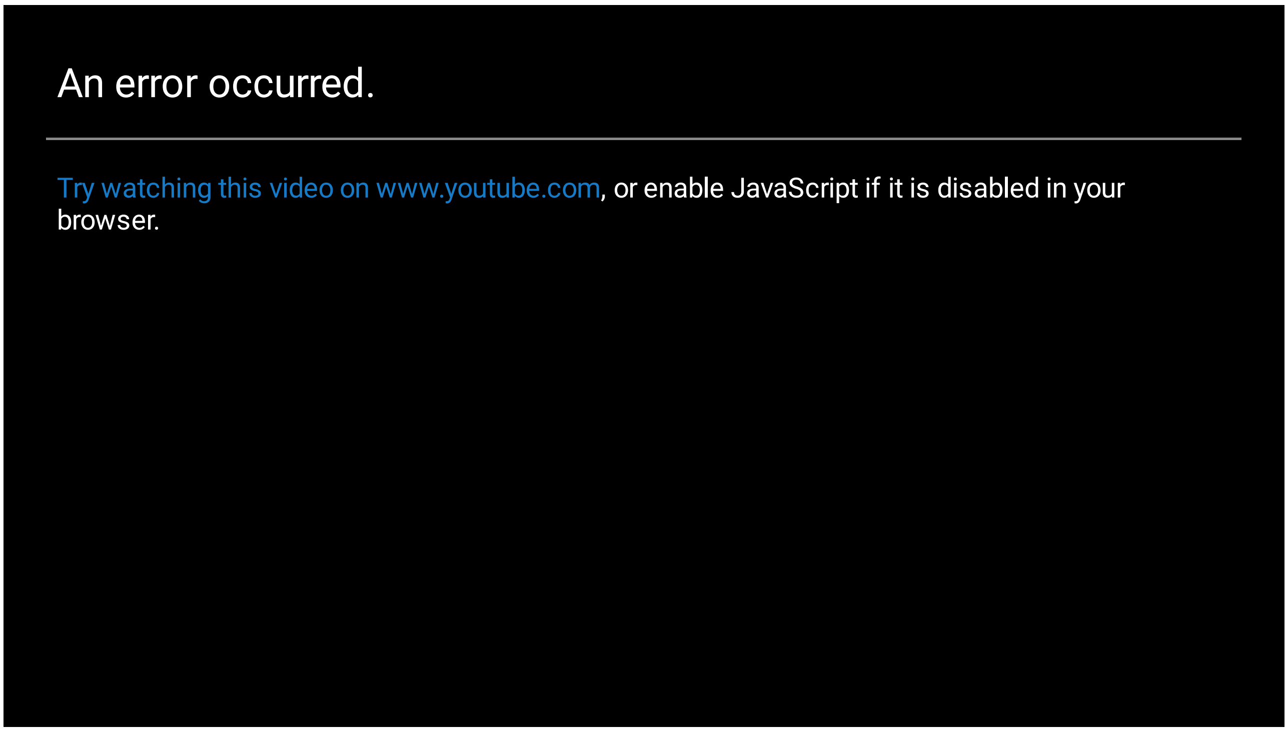
Hive allows you to think about the data you have stored in HDFS as though it were in a relational database, by projecting table-like structure onto it. This structure is stored in a relational database known as the metastore; you can think of the metastore as being like the schema of your would-be relational database. Note, however, that your data is not in a relational database, and Hive is not a relational database management system - it might look that way, because of HQL, but behind the scenes is just Hadoop and MapReduce.

Hive has a driver that manages the lifecycle of an HQL statement as it gets processed by Hive:

- Query Compiler: compiles the HQL into MapReduce tasks
- Optimizer: generates the best execution plan
- Execution Engine: executes the tasks produced by the compiler in the appropriate order, by interacting with Hadoop

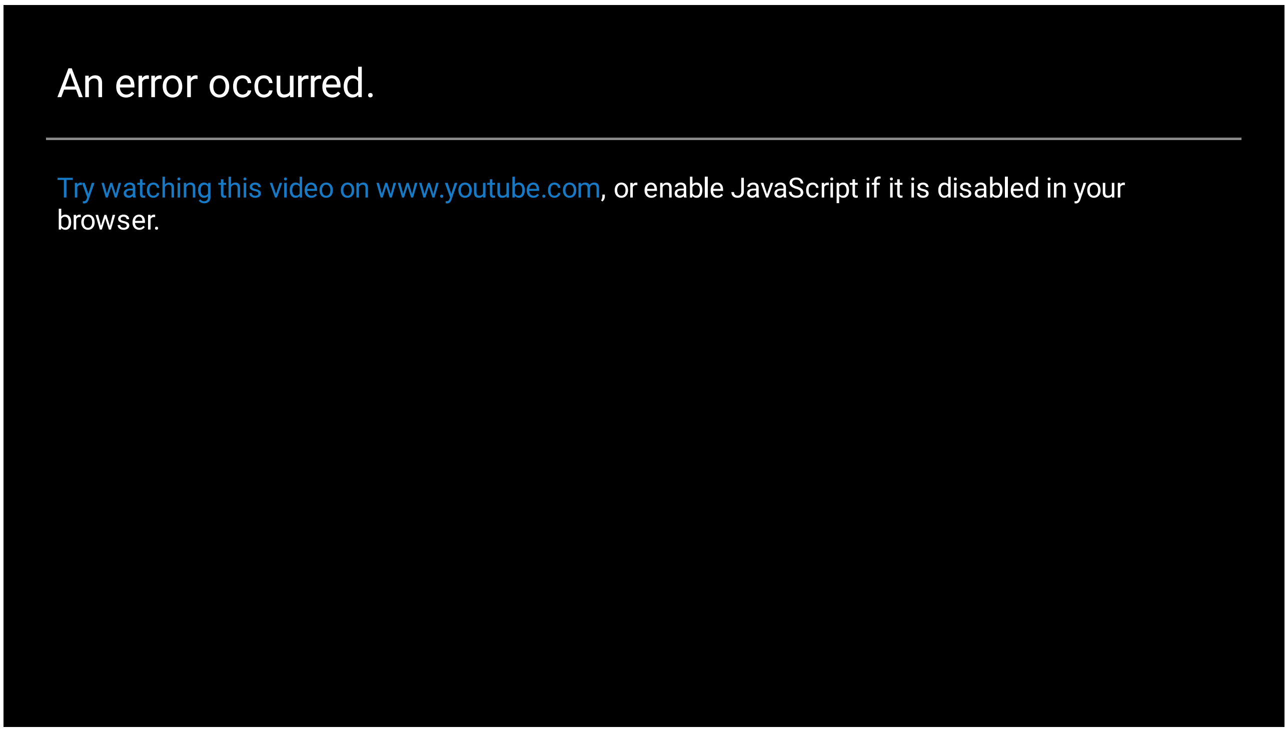
The next two slides contain a couple of video introductions to Hive. Then, in the slides after that, you'll get some hands-on experience using Hive.

Introduction to Hive, 1 (5:24)


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Introduction to Hive, 2 (13:56)


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Working with local text

Let's now use Hive to work with a text file.

When you click the panel on the right you'll get a terminal connection to a server that has Hadoop, YARN, and Hive installed, and has the file "text.txt" in your home directory.

Use the following command to start a Hive shell:

```
$ hive
```

Create a table

Let's create a table in which to put the text file. We need to tell Hive what columns to have in the table, and what type of data is stored in each column (string, integer, date, etc.). We also need to tell it how to interpret an input file as having this structure. Let's think of each line in the text file as being a row in our table, and put the whole line into a single column called "line". So let's create a table using the following command:

```
CREATE TABLE ourText (line STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\n' STORED AS textfil
```

You can check that the table has been created by running the following command:

```
SHOW TABLES;
```

You can check the structure of the table by running the following command:

```
DESCRIBE ourText;
```

Load data

Now let's load text.txt into the table:

```
LOAD DATA LOCAL INPATH 'text.txt' OVERWRITE INTO TABLE ourText;
```

By adding the keyword "LOCAL" we are telling Hive that the file is in the local file system on the server (rather than in HDFS). By adding the keyword "OVERWRITE" we are telling Hive to replace any data that was already in the table (since we've just created it there shouldn't be any).

You can check that the data has been loaded by running the following command:

```
SELECT * FROM ourText;
```

Run queries

Now let's run some HQL queries to answer questions about the text file:

How many lines are in the file?

```
SELECT COUNT(*) FROM ourText;
```

What is the length of each line, from shortest to longest?

```
SELECT LENGTH(line) FROM ourText ORDER BY LENGTH(line);
```

What is the average length of the lines (in characters)?

```
SELECT AVG(LENGTH(line)) FROM ourText;
```

How many words are in each line of the file?

We first need to break the line into words - we can do that using Hive's `split()` function, splitting the line at its spaces. Note that we need to remove the empty lines by checking their length via a where clause. Then we can use Hive's `size()` function to get the number of resulting words:

```
SELECT SIZE(SPLIT(line, ' ')) FROM ourText WHERE LENGTH(line)>0;
```

How many words are in the file?

We can get this by summing the number of words in each line:

```
SELECT SUM(SIZE(SPLIT(line, ' '))) FROM ourText WHERE LENGTH(line)>0;
```

What is the average word length?

To answer this we need the total length of the words, and the number of words, and then divide the former by the latter. We can get the number of words using the previous query. What about the total length of words? We can get the total length of words in a line by removing its spaces (replace them with nothing) and then counting the number of characters that remain, then we can sum these to get the total length of words. So we can use the following query:

```
SELECT SUM(LENGTH(REPLACE(line, ' ', '')))/SUM(SIZE(SPLIT(line, ' '))) FROM ourText;
```

Explode

Hive has a function `EXPLODE()` which turns an array of values into rows. Try the following query:

```
SELECT EXPLODE(SPLIT(line, ' ')) AS word FROM ourText;
```

In effect, this query generates a table of words that we can query to get answers about words. This involves using the above query as a subquery in other queries. For example:

How many words are there?

```
SELECT COUNT(*)
FROM (SELECT EXPLODE(SPLIT(line, ' ')) AS word FROM ourText) AS words;
```

What is the average word length?

```
SELECT AVG(LENGTH(word))
FROM (SELECT EXPLODE(SPLIT(line, ' ')) AS word FROM ourText) AS words;
```

What is the frequency of each word?

```
SELECT word, COUNT(*)
FROM (SELECT EXPLODE(SPLIT(line, ' ')) AS word FROM ourText) AS words
GROUP BY word;
```

Lateral View

Hive has a way of doing this more efficiently, using `LATERAL VIEW`. This gives a way of merging the results of `EXPLODE` back together with the original table.

```
SELECT word
FROM ourText LATERAL VIEW EXPLODE(SPLIT(line, ' ')) words as word;
```

What are the unique words?

```
SELECT DISTINCT word
FROM ourText LATERAL VIEW EXPLODE(SPLIT(line, ' ')) words as word;
```

What is the average word length?

```
SELECT AVG(LENGTH(word))
FROM ourText LATERAL VIEW EXPLODE(SPLIT(line, ' ')) words as word;
```

What is the frequency of each word?

```
SELECT word, COUNT(*)
FROM ourText LATERAL VIEW EXPLODE(SPLIT(line, ' ')) words as word
GROUP BY word;
```

Working with HDFS text

In the previous slide we used Hive to work with a text file in the server's local file system. Now let's now use it to work with a text file in HDFS.

When you click the panel on the right you'll get a terminal connection to a server that has Hadoop, YARN, and Hive installed and running, and has the file "text.txt" in your home directory.

Copy to HDFS

Let's start by copying text.txt to your default directory in HDFS:

```
$ hdfs dfs -mkdir -p /user/user
$ hdfs dfs -put text.txt
```

Start hive

Use the following command to start a Hive shell:

```
$ hive
```

Create the table

Create the table as before:

```
CREATE TABLE ourText (line STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\n' STORED AS textfil
```

Load data

Now let's load text.txt into the table, this time loading it from HDFS:

```
LOAD DATA INPATH 'text.txt' OVERWRITE INTO TABLE ourText;
```

Since we have not added the "LOCAL" keyword Hive will look for the file text.txt in your default HDFS directory.

You can check that the data has been loaded by running the following command:

```
SELECT * FROM ourText;
```

Run queries

Now we can run queries as before. Let's get **the frequency of words in the file**. Last time we did this in two ways. Try each one:

```
SELECT word, COUNT(*)
FROM (SELECT EXPLODE(SPLIT(line, ' ')) AS word FROM ourText) AS words
GROUP BY word;
```

```
SELECT word, COUNT(*)
FROM ourText LATERAL VIEW EXPLODE(SPLIT(line, ' ')) words as word
GROUP BY word;
```

Working with CSV

In the previous slides we were using Hive to query data in a text file. Now let's use Hive to query data a CSV file.

Click the panel on the right. This will give you a terminal connection to a server that has Hadoop, YARN and Hive installed and running, and has a file called "employees.csv" in your home directory.

Run the following command to start a Hive shell:

```
$ hive
```

Create a table

Let's create a table into which we can load the CSV file. Here are the fields in the CSV file:

```
employee_id (integer)
first_name (string)
last_name (string)
email (string)
phone_number (string)
hire_date (date)
salary (integer)
```

Given this structure of the CSV file, we should create our table as follows:

```
CREATE TABLE employees (employee_id INT, first_name STRING, last_name STRING, email STRING, phone_n
```

You can check that the table has been created by running the following command:

```
SHOW TABLES;
```

You can check the structure of the table by running the following command:

```
DESCRIBE employees;
```

Load data

Now let's load the CSV file into the table:

```
LOAD DATA LOCAL INPATH 'employees.csv' OVERWRITE INTO TABLE employees;
```

You can check that the data has been loaded by running the following command:

```
SELECT * FROM employees;
```

Run queries

Now let's run some HQL queries to answers questions about the data in the CSV file.

What is the maximum salary of employees?

```
SELECT MAX(salary) FROM employees;
```

What are the distinct salaries, and which employees have those salaries?

```
SELECT salary, COLLECT_LIST(last_name) FROM employees GROUP BY salary;
```

Note that Hive uses COLLECT_LIST rather than GROUP_CONCAT (which is what MySQL uses).

Which employees were hired in 2006 and have a salary of more than $5,000?

```
SELECT first_name, last_name, hire_date, salary
FROM employees
WHERE hire_date LIKE '2006-%' AND salary > 5000;
```

Writing HIVE scripts

Rather than executing HQL statements one-by-one in a Hive shell, you can bundle them into a script and execute them all together. This is also a good way to save your statements, edit them, and run them easily whenever you like.

Let's see how to do this.

When you click the panel on the right you'll get a terminal connection to a server that has Hadoop, YARN, and Hive installed and running, and has the file "text.txt" in your home directory.

It also has a file called "frequency.hql". This is just a file containing the HQL statements that we ran individually to get the frequency of words in text.txt. You can open the file and inspect its contents.

To execute the statements in the file, just enter the following command in the terminal:

```
$ hive -f frequency.hql
```

This command tells Hive to execute the statements in the file called "frequency.hql".

You should see the commands being executed.

Try it: writing a script

Now try writing a Hive script yourself.

When you click the panel on the right you'll get a terminal connection to a server that has Hadoop, YARN, and Hive installed, and has the file "employees.csv" in your home directory. You can open the file and inspect its contents.

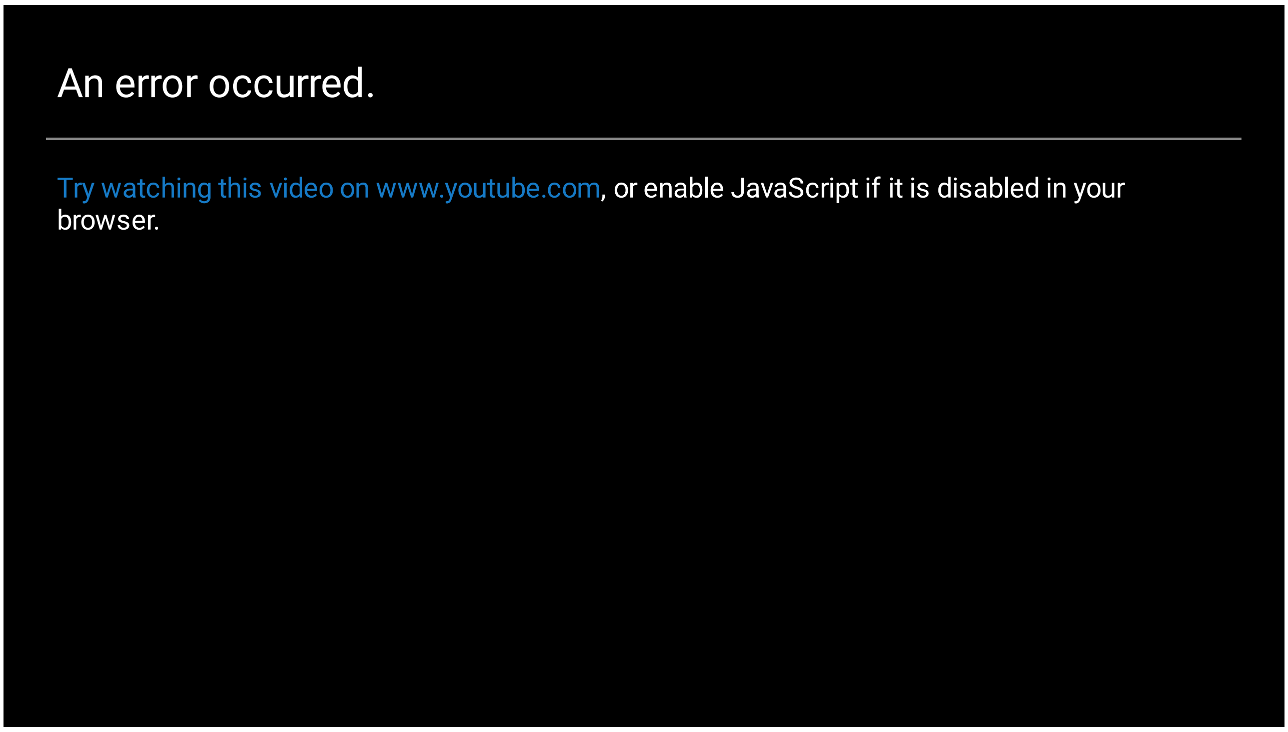
A file called "script.hql" has been created for you. Try adding Hive commands to this script, to **find the average salary of the employees**. You should be able to modify commands that you have seen in previous slides.

You can execute your script using the following command:

```
$ hive -f script.hql
```

You should see the commands in your script being executed.

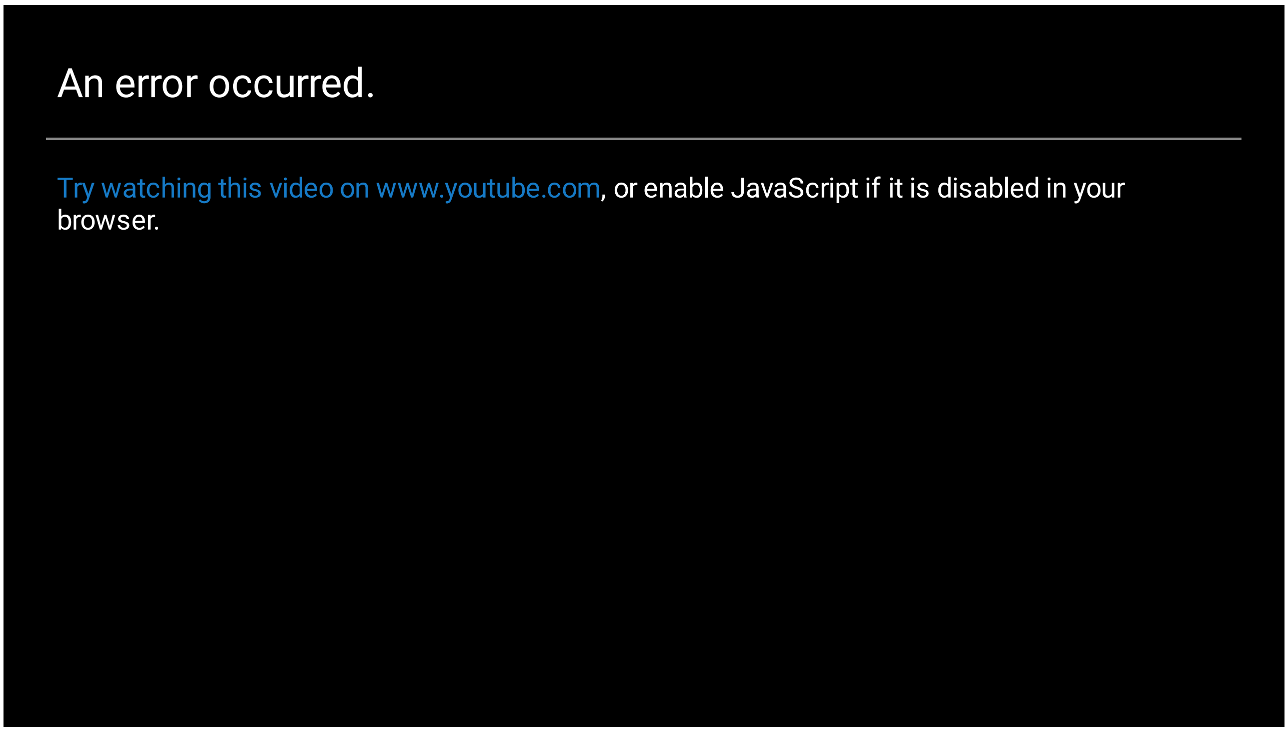
Inserting data (5:34)



An error occurred.

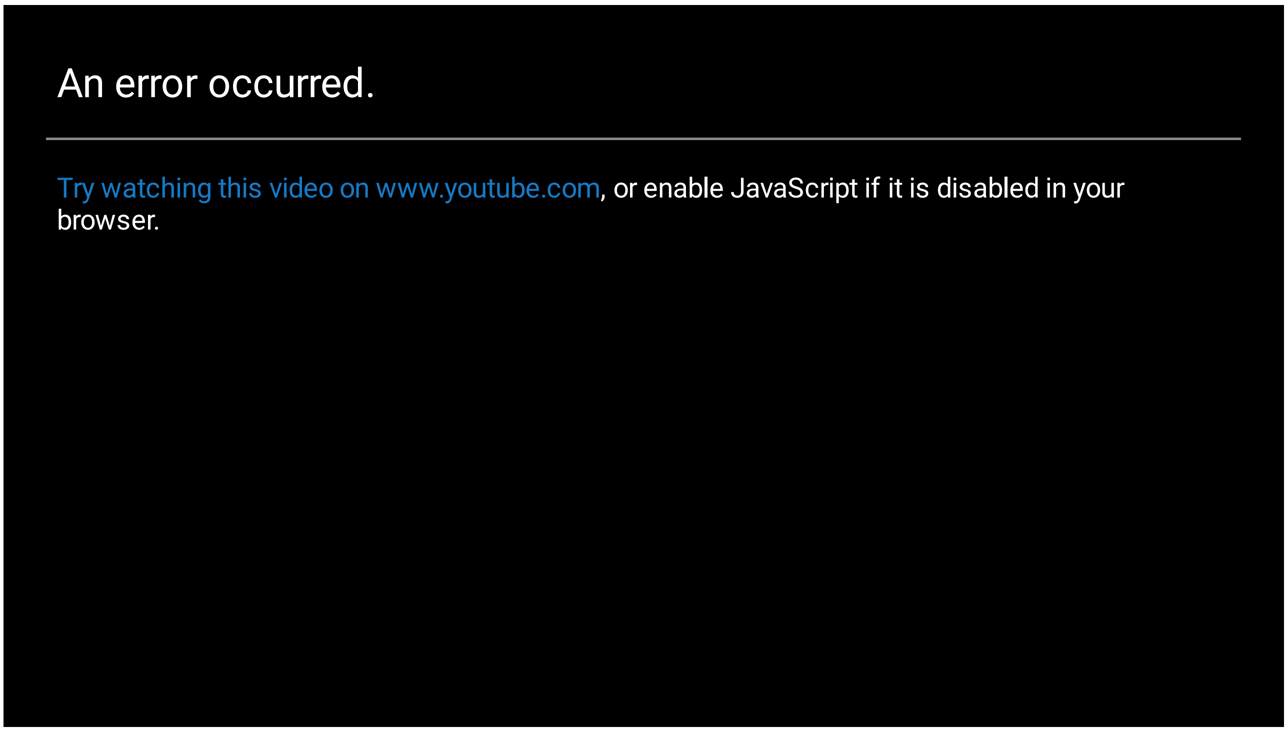
Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Querying data (2:37)


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Explode and Lateral View (7:48)



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Transcript

Explode and Lateral view function in Hive.pdf

Further resources

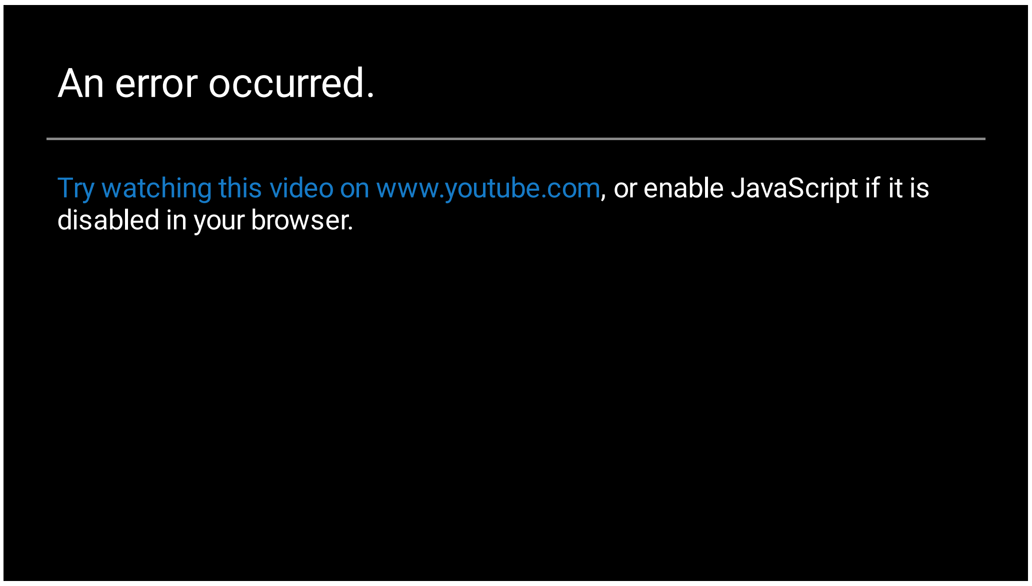
The official Hive website is a good source for further information about Hive:

Official Hive website

You might find this Hive cheatsheet by Hortonworks a convenient reference:

Hive cheatsheet by Hortonworks

The following Hive Tutorial may help you better learn Hive:



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.