

Querying data with Hive

Dr. Villanes

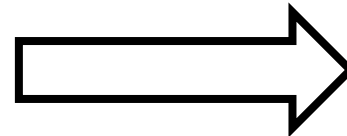
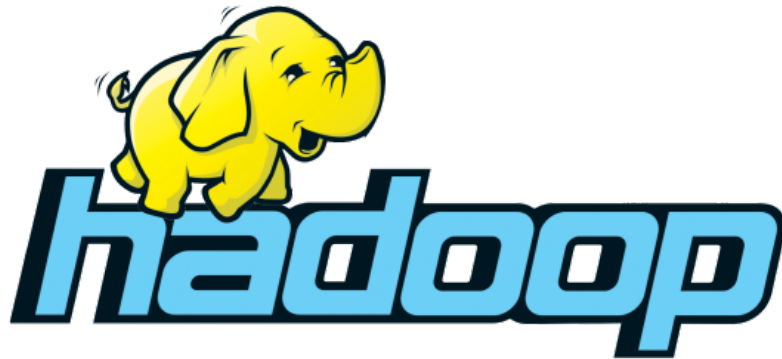
Login to AWS Academy

What are we going to do today? – 2.5 hours

1. Launch AWS EMR Hadoop Cluster
2. If Hadoop/Spark are open source, why do we need commercial distributions?
3. Hive syntax
4. Connect to EMR Master
5. Do Hive Hands-on → Querying Data with Hive (Quiz)

Recap of last class...

Apache Hadoop is a community driven **open-source project** governed by the Apache Software Foundation.





open source


Download it: <http://hadoop.apache.org/>

Open source: advantages/disadvantages

Open source:

-  collaboration between people, communities and projects.
-  enterprise needs are not necessarily the priority of the open source community.

What we need:

-  a software vendor that provides an integrated software package to give enterprises an integrated solution.

Reasons why you need more than open source

1. Open Source doesn't mean it's all designed to work together.

→ *We need a software vendor who is a member of the communities, to compile different projects into a single implementable software package.*

2. Open Source creates an unprecedented pace of innovation.

→ *Access to multiple (possibly more mature) versions of the integrated software package to give enterprises the freedom to update to newer versions at their own pace.*

3. Some requirements – such as enterprise security and governance do not organically manifest themselves in open source communities.

4. Connect in house software deployment, integration and testing with outside experience, expertise, enterprise support, professional services and education.

What we need...



The requirement for Hadoop to be aligned to the needs of individual organizations has resulted in the emergence of many commercial distributions

Examples of Hadoop vendors that provide a Hadoop Distribution



Hadoop Distributions



Hortonworks Data Platform (HDP)



Cloudera CDH



MapR Distribution



Microsoft's Azure HDInsight

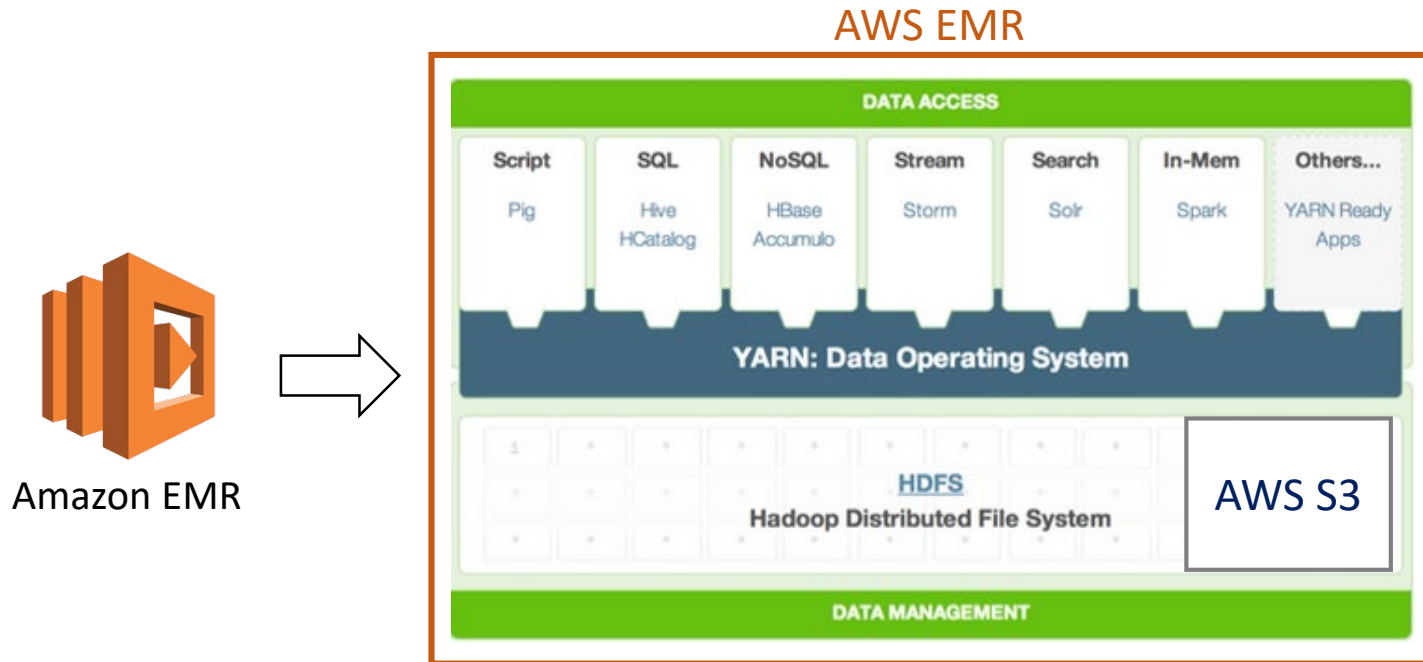


Amazon Elastic MapReduce (EMR)



Cloud Dataproc

AWS Hadoop Distribution: Amazon Elastic MapReduce (EMR)



- **Nodes** in the cluster = **EC2 instances**
- Amazon EMR includes EMRFS, a connector allowing Hadoop to use **S3 as a storage layer**.
- **HDFS** is automatically installed with Hadoop on your EMR cluster, and **you can use HDFS along with Amazon S3 to store your input and output data**.
- Amazon EMR configures Hadoop to use **HDFS for intermediate data** created during MapReduce jobs, even if your input data is located in Amazon S3.

Amazon EMR programmatically installs and configures applications in the Hadoop project, including Hadoop MapReduce (YARN), and HDFS, across the nodes in your cluster.

AWS EMR

- EMR manages **provisioning, management, and scaling** of the EC2 instances.
- You can launch a **10-node EMR cluster** \cong **\$0.15 per hour**
- The **number of instances can be increased or decreased** automatically using **Auto Scaling**. Auto scaling manages cluster sizes based on utilization.
- Amazon EMR integrates with other AWS services. For example:
 - Amazon Virtual Private Cloud (Amazon VPC) to configure the virtual network in which you launch your instances
 - Amazon CloudWatch to monitor cluster performance and configure alarms
 - AWS Identity and Access Management (IAM) to configure permissions
 - AWS CloudTrail to audit requests made to the service

Hive

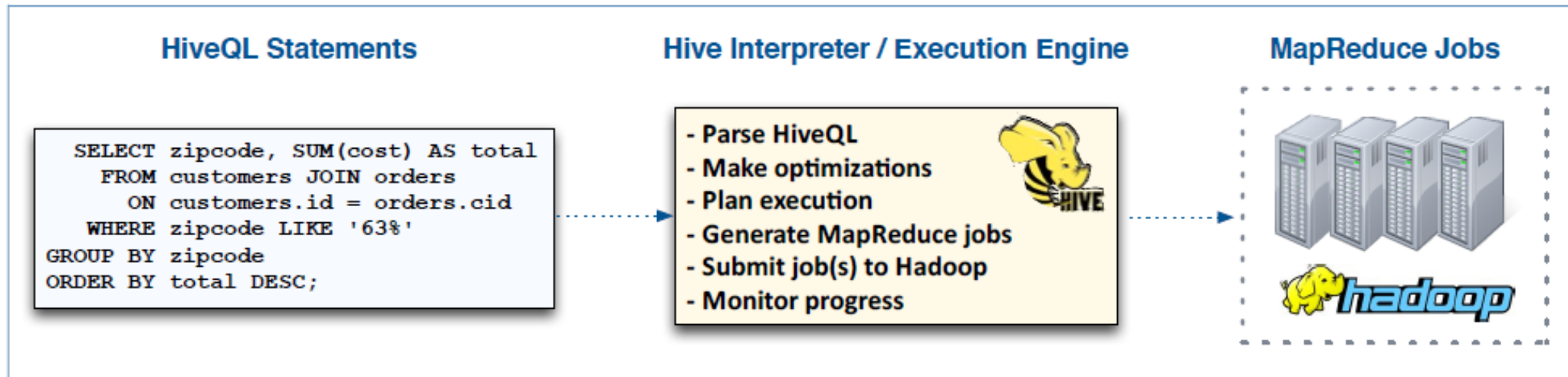
What is Hive?

- Apache Hive is a high-level abstraction on top of MapReduce
- Uses a SQL-like language called HiveQL
- Generates MapReduce jobs that run on the Hadoop Cluster
- Open-source Apache project

```
SELECT zipcode, SUM(cost) AS total
FROM customers
JOIN orders
  ON customers.cust_id = orders.cust_id
WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```

What is Hive?

- Hive turns HiveQL queries into MapReduce jobs



Manual: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual>

Why use Apache Hive?

- Five lines of HiveQL might be equivalent to 100 lines or more of Java
- No software development experience required
- Leverage existing knowledge of SQL
- 100% free and open source

How Hive Loads and Stores Data

- Hive's queries **operate on tables** (just like in an RDBMS)
- A table is simply an HDFS directory containing one or more files
- **The query itself operates on data stored on the HDFS/S3 filesystem**



Define a Hive Table

Example: Creating Table drivers

- First, create a Hive table:

```
>> CREATE TABLE drivers (driverId INT, name STRING, ssn BIGINT,  
location STRING, certified STRING, wageplan STRING) row format  
delimited fields terminated by ',';
```

- Next, load the data file (drivers.csv) into the table drivers using the following query:

```
>> LOAD DATA INPATH 's3a://bucket_name/drivers.csv' OVERWRITE  
INTO TABLE drivers;
```

Basic HiveQL Syntax

Hive Language Manual

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual>

<https://hive.apache.org/>

Exploring Tables

- The DESCRIBE command displays basic structure for a table

```
hive> DESCRIBE orders;
```

- DESCRIBE FORMATTED shows more detailed information

```
hive> DESCRIBE FORMATTED orders;
```

Selecting Data from tables

- The SELECT statement retrieves data from Hive tables

```
hive> SELECT cust_id, fname, lname FROM customers;
```

- An asterisk matches all columns in the table

```
hive> SELECT * FROM customers;
```

Selecting Data from tables

- The LIMIT clause sets the maximum number of rows returned

```
hive> SELECT cust_id, fname, lname FROM customers LIMIT 10;
```

- Caution: no guarantee regarding which 10 results are returned
 - Use ORDER BY for top-N queries

```
hive> SELECT cust_id, fname, lname FROM customers  
      ORDER BY cust_id DESC LIMIT 10;
```

Using the WHERE Clause to restrict Results

- WHERE clause restrict rows to those matching specified criteria

```
hive> SELECT * FROM customers WHERE state = 'CA';
```

```
hive> SELECT * FROM customers WHERE state in ('CA', 'OR');
```

- You can combine expressions using AND or OR

Table Aliases

- Can help simplify complex queries

```
hive> SELECT o.order_date, c.fname  
        FROM customers c JOIN orders o  
        ON c.cust_id = o.cust_id  
        WHERE c.zipcode=90210
```

- Using AS to specify table aliases is ***not*** supported

Combining Query Results with UNION ALL

- Unifies output from SELECTs into a single result set
 - Name, order, and types of columns in each query ***must*** match

```
SELECT emp_id, fname, lname, salary
      FROM employees
      WHERE state='CA'
UNION ALL
SELECT emp_d, fname, lname, salary
      FROM employees
      WHERE salary > 75000;
```

Subqueries in Hive

- Hive only supports subqueries in the FROM clause:

```
SELECT brand, name  
FROM (SELECT *  
      FROM products  
      WHERE price > 100) high_profits  
WHERE cost < 30;
```

- Each subquery must be named (like high_profits above)

Joining Datasets

Joins in Hive

- Hive supports several types of joins
 - Inner joins
 - Outer joins (left, right, and full)
- Only equality conditions are allowed
 - Valid: `customers.cust_id = orders.cust_id`
 - Invalid: `customers.cust_id <> orders.cust_id`

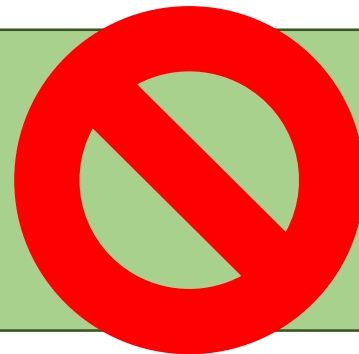
Join Syntax

- Required syntax for joins

```
SELECT c.cust_id, name, total  
FROM customers c  
JOIN orders o on (c.cust_id = o.cust_id);
```

- The following join syntax is not valid in Hive

```
SELECT c.cust_id, name, total  
FROM customers c, orders o  
WHERE c.cust_id = o.cust_id;
```



Common Built-in Functions

Hive Functions

- Hive offers built-in functions
 - Many identical to those in SQL
 - Others Hive-specific
- Function names are not case sensitive
- In order to see information about a function

```
hive> DESCRIBE FUNCTION UPPER;
```


Example Built-in Functions (1)

Function Description	Example Invocation	Input	Output
Rounds to specified # of decimals	<code>ROUND(total_price, 2)</code>	23.492	23.49
Returns nearest integer above	<code>CEIL(total_price)</code>	23.492	24
Returns nearest integer below	<code>FLOOR(total_price)</code>	23.492	23
Return absolute value	<code>ABS(temperature)</code>	-49	49
Returns square root	<code>SQRT(area)</code>	64	8
Returns a random number	<code>RAND()</code>		0.584977

Example Built-in Functions (2)

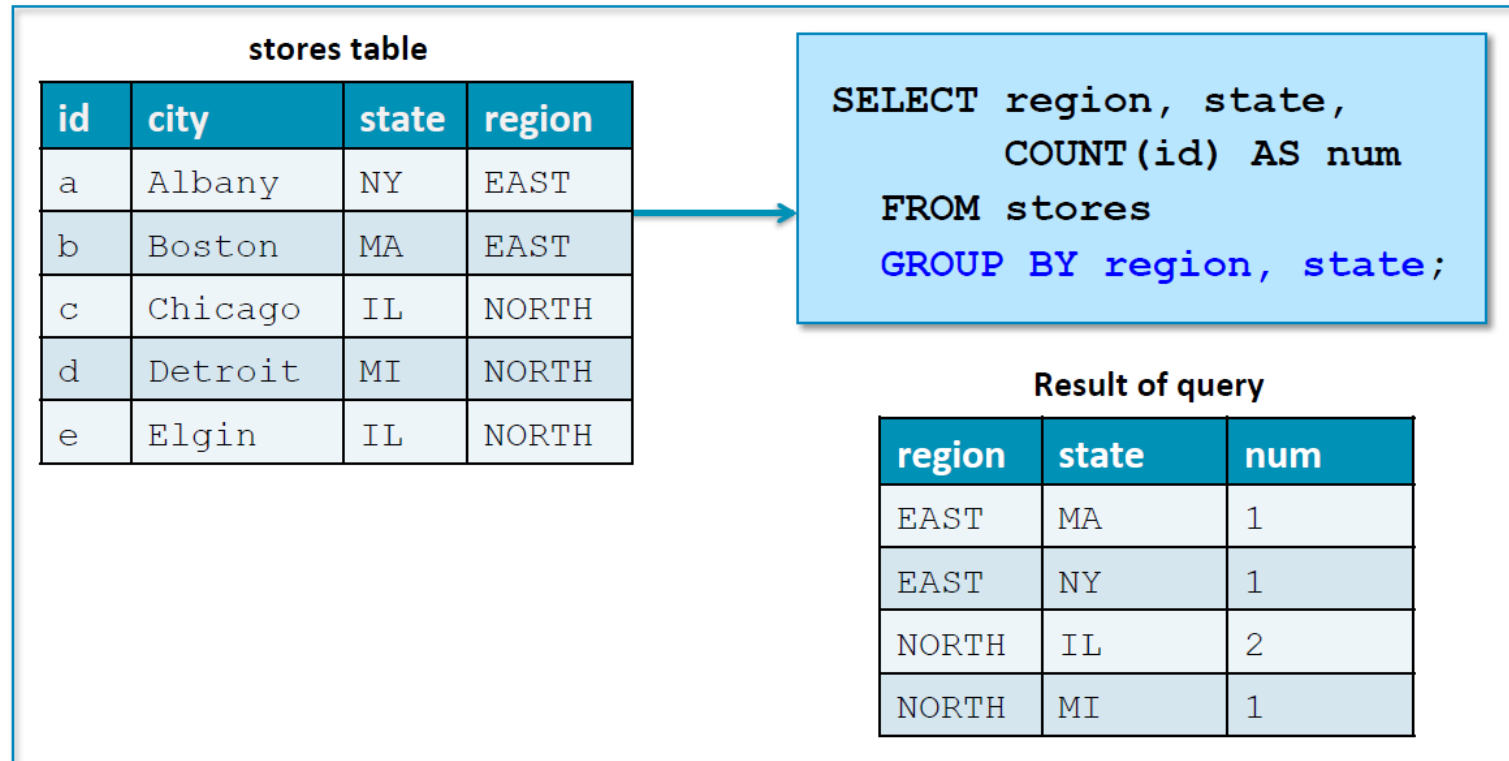
Function Description	Example Invocation	Input	Output
Convert to UNIX format	<code>UNIX_TIMESTAMP (order_dt)</code>	2013-06-14 16:51:05	1371243065
Convert to string format	<code>FROM_UNIXTIME (mod_time)</code>	1371243065	2013-06-14 16:51:05
Extract date portion	<code>TO_DATE (order_dt)</code>	2013-06-14 16:51:05	2013-06-14
Extract year portion	<code>YEAR (order_dt)</code>	2013-06-14 16:51:05	2013
Returns # of days between dates	<code>DATEDIFF (order_dt, ship_dt)</code>	2013-06-14, 2013-06-17	3

Example Built-in Functions (3)

Function Description	Example Invocation	Input	Output
Converts to uppercase	<code>UPPER (fname)</code>	Bob	BOB
Extract portion of string	<code>SUBSTRING (name, 0, 2)</code>	Alice	Al
Selectively return value	<code>IF (price > 1000, 'A', 'B')</code>	1500	A
Convert to another type	<code>CAST (weight as INT)</code>	3.581	3
Returns size of array or map	<code>SIZE (array_field)</code>	N/A	6

Record Grouping and Aggregate Functions

- GROUP BY groups selected data by one or more columns
 - Columns not part of aggregation must be listed in GROUP BY



Built-in Aggregate Functions

Function Description	Example Invocation
Count all rows	<code>COUNT (*)</code>
Count all rows where field is not null	<code>COUNT (fname)</code>
Count all rows where field is unique and not null	<code>COUNT (DISTINCT fname)</code>
Returns the largest value	<code>MAX (salary)</code>
Returns the smallest value	<code>MIN (salary)</code>
Adds all supplied values and returns result	<code>SUM (price)</code>
Returns the average of all supplied values	<code>AVG (salary)</code>

Text Processing with Hive

Basic String Functions

Function Description	Example Invocation	Input	Output
Convert to uppercase	UPPER (name)	Bob	BOB
Convert to lowercase	LOWER (name)	Bob	bob
Remove whitespace at start/end	TRIM (name)	└Bob┐	Bob
Remove only whitespace at start	LTRIM (name)	└Bob┐	Bob┐
Remove only whitespace at end	RTRIM (name)	└Bob┐	└Bob
Extract portion of string *	SUBSTRING (name, 0, 3)	Samuel	Sam

* Unlike SQL, starting position is zero-based in Hive

Splitting and Combining Strings

- CONCAT combines one or more strings
 - The CONCAT_WS variation joins them with a separator
- SPLIT does nearly the opposite
 - SPLIT returns value as an ARRAY

Example Invocation	Output
<code>CONCAT('alice', '@example.com')</code>	<code>alice@example.com</code>
<code>CONCAT_WS(' ', 'Bob', 'Smith')</code>	<code>Bob Smith</code>
<code>CONCAT_WS('/', 'Amy', 'Sam', 'Ted')</code>	<code>Amy/Sam/Ted</code>
<code>SPLIT('Amy/Sam/Ted', '/')</code>	<code>["Amy", "Sam", "Ted"] *</code>

** Representation of **ARRAY<STRING>***

Converting Array to Records with EXPLODE

- The EXPLODE function creates a record for each element in an array

```
hive> SELECT people FROM example;  
Amy, Sam, Ted
```

```
hive> SELECT SPLIT(people, ',') FROM example;  
["Amy", "Sam", "Ted"]
```

```
hive> SELECT EXPLODE(SPLIT(people, ',')) AS x FROM example;  
Amy  
Sam  
Ted
```

Parsing Sentences into Words

- The SENTENCES function parses supplied text into words
- Output is a two-dimensional array of strings

```
hive> SELECT txt FROM phrases WHERE id=12345;  
I bought this computer and really love it! It's very fast and  
does not crash.
```

```
hive> SELECT SENTENCES(txt) FROM phrases WHERE id=12345;  
[["I", "bought", "this", "computer", "and", "really", "love", "it"],  
 ["It's", "very", "fast", "and", "does", "not", "crash"]]
```

N-grams

- An n-gram is a word combination (n=number of words)
 - Bigram is a sequence of two words (n=2)
- NGRAMS function calculates n-grams
- Three input parameters
 - Array of strings (sentences), each containing an array (words)
 - Number of words in each n-gram
 - Desired number of results (top-N, based on frequency)
- The NGRAMS function is often used with the SENTENCES
 - Often used with the LOWER function to normalize case
 - EXPLODE to convert the resulting array to a series of rows

Calculating n-grams in Hive

```
hive> SELECT txt FROM phrases WHERE id=56789;
```

```
This tablet is great. The size is great. The screen is  
great. The audio is great. I love this tablet! I love  
everything about this tablet!!!
```

```
hive> SELECT EXPLODE(NGRAMS(SENTENCES(LOWER(txt)), 2, 5))
```

```
AS bigrams FROM phrases WHERE id=56789;
```

```
{"ngram":["is","great"],"estfrequency":4.0}  
{"ngram":["great","the"],"estfrequency":3.0}  
{"ngram":["this","tablet"],"estfrequency":3.0}  
{"ngram":["i","love"],"estfrequency":2.0}  
{"ngram":["tablet","i"],"estfrequency":1.0}
```

Finding specific n-grams

- CONTEXT_NGRAMS is similar, but considers only specific combinations
 - Additional parameter: array of words used for filtering
 - Any NULL values in the array are treated as placeholders

```
hive> SELECT txt FROM phrases
        WHERE txt LIKE '%new computer%';
My new computer is fast! I wish I'd upgraded sooner.
This new computer is expensive, but I need it now.
I can't believe her new computer failed already.

hive> SELECT EXPLODE(CONTEXT_NGRAMS(SENTENCES(LOWER(txt)),
        ARRAY("new", "computer", NULL, NULL), 4, 3)) AS ngrams
        FROM phrases;
{"ngram":["is","expensive"],"estfrequency":1.0}
{"ngram":["failed","already"],"estfrequency":1.0}
{"ngram":["is","fast"],"estfrequency":1.0}
```

In case you need it one day...

Pig

What is Pig?

- Apache Pig is a platform for data analysis and processing on Hadoop
- Offers an alternative to writing MapReduce code directly
- Generates MapReduce jobs that run on the Hadoop Cluster
- Open-source Apache project

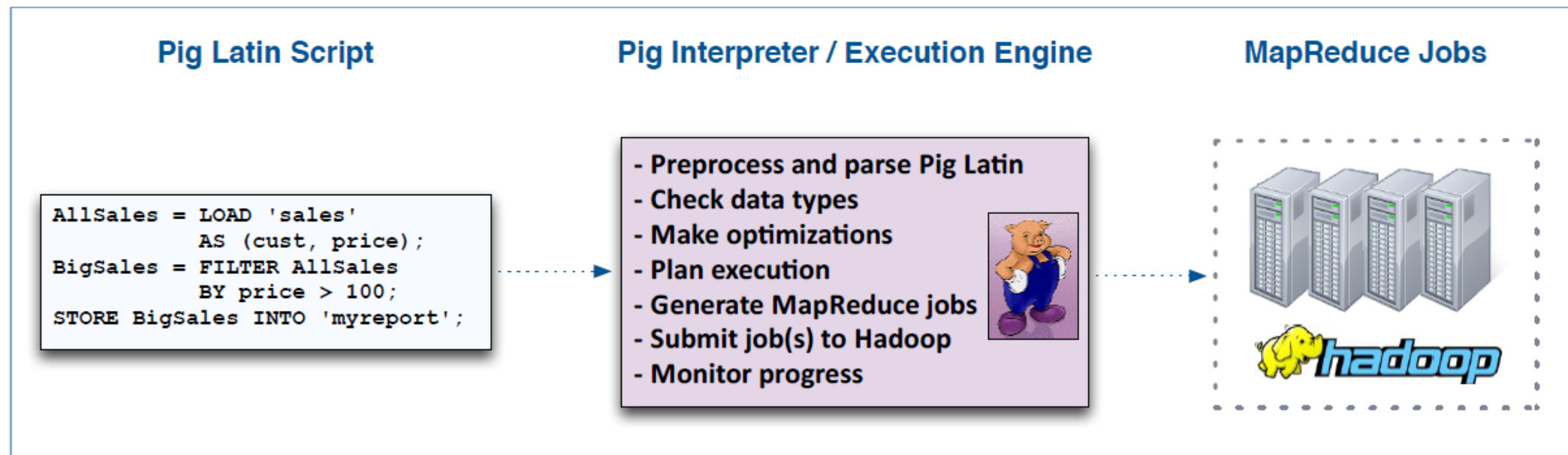
```
allsales = LOAD 'sales' AS (name, price);

bigsales = FILTER allsales BY price > 999; -- in US cents

/*
 * Save the filtered results into a new
 * directory, below my home directory.
 */
STORE bigsales INTO 'myreport';
```

What is Pig?

- Pig interprets Pig Latin code in order to create MapReduce jobs
- It then submits these MapReduce jobs to the Hadoop cluster



Pig Latin Syntax

Pig Latin Overview

- Pig Latin is a *data flow* language: sequence of statement
- The following script loads, filters and stores data

```
allsales = LOAD 'sales' AS (name, price);

bigsales = FILTER allsales BY price > 999; -- in US cents

/*
 * Save the filtered results into a new
 * directory, below my home directory.
 */
STORE bigsales INTO 'myreport';
```

Pig Latin Grammar: Keywords

- Pig Latin keywords are highlighted in blue text
 - Keywords are reserved – you cannot use them to name things
- Keywords are not case-sensitive

```
allsales = LOAD 'sales' AS (name, price);

bigsales = FILTER allsales BY price > 999; -- in US cents

/*
 * Save the filtered results into a new
 * directory, below my home directory.
 */
STORE bigsales INTO 'myreport';
```

Pig Latin Grammar: Identifiers

- Identifiers are the names assigned to fields and other data structures
- Identifiers are case-sensitive

```
allsales = LOAD 'sales' AS (name, price);

bigsales = FILTER allsales BY price > 999; -- in US cents

/*
 * Save the filtered results into a new
 * directory, below my home directory.
 */
STORE bigsales INTO 'myreport';
```

Pig Latin Grammar: Identifiers

- An identifier must always begin with a letter
 - This may only be followed by letters, number or underscores

Valid	x	q1	q1_2013	MyData
Invalid	4	price\$	profit%	_sale

Common Operators in Pig Latin

- Important difference: Pig Latin uses == and != for comparison

Arithmetic	Comparison	Null	Boolean
+	==	IS NULL	AND
-	!=	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

Pig Data Concepts: Fields

- A single element of data is called a field

name	price	country
Alice	2999	us
Bob	3625	ca
Carlos	2764	mx
Dieter	1749	de
Étienne	2368	fr
Fredo	5637	it

Pig Data Concepts: Tuples

- A collection of values is called a tuple

name	price	country
Alice	2999	us
Bob	3625	ca
Carlos	2764	mx
Dieter	1749	de
Étienne	2368	fr
Fredo	5637	it

Pig Data Concepts: Bags

- A collection of tuples is called a bag

name	price	country
Alice	2999	us
Bob	3625	ca
Carlos	2764	mx
Dieter	1749	de
Étienne	2368	fr
Fredo	5637	it

Pig Data Concepts: Relations

- A relation is a bag with an assigned name (alias)
- Most Pig Latin statement create a new relation
- Processing creates new relations instead of modifying existing ones

```
allsales = LOAD 'sales' AS (name, price);  
bigsales = FILTER allsales BY price > 999;  
STORE bigsales INTO 'myreport';
```

Loading Data

Basic Data Loading in Pig

- Pig's default loading function is called PigStorage
 - The name of the function is implicit when calling LOAD
- Consider the following file in HDFS:

Alice	2999
Bob	3625
Carlos	2764

- The following code loads data from the file:

```
allsales = LOAD 'sales' AS (name, price);
```

Simple Data Types

Data Types in Pig

- Pig supports several basic data types

Name	Description	Example Value
int	Whole numbers	2013
long	Large whole numbers	5,365,214,142L
float	Decimals	3.14159F
double	Very precise decimals	3.14159265358979323846
boolean*	True or false values	true
datetime*	Date and time	2013-05-30T14:52:39.000-04:00
chararray	Text strings	Alice
bytearray	Raw bytes (e.g. any data)	N/A

Data Types in Pig

- Pig treats fields of unspecified type as an bytearray type

```
allsales = LOAD 'sales' AS (name, price);
```

- It is better to specify data types explicitly when possible

```
allsales = LOAD 'sales' AS (name:chararray, price:int);
```

Data Output

Data Output in Pig

- Two commands can be used to handle output
- DUMP: sends output to the screen
- STORE: sends output to disk (HDFS)
 - Similar to LOAD, but writes data instead of reading it
 - As with LOAD, the use of PigStorage is implicit

```
allsales = LOAD 'sales' AS (name, price);  
bigsales = FILTER allsales BY price > 999;  
STORE bigsales INTO 'myreport';
```

```
allsales = LOAD 'sales' AS (name, price);  
bigsales = FILTER allsales BY price > 999;  
DUMP bigsales;
```

Viewing the Schema

Viewing the Schema in Pig

- The DESCRIBE command shows the structure of the data, including names and types

```
allsales = LOAD 'sales' AS (name: chararray, price: int);  
DESCRIBE allsales;
```


```
allsales: {name: chararray, price: int}
```

Filtering and Sorting Data

Filtering in Pig

- FILTER extracts tuples matching a specific criteria

```
bigsales = FILTER allsales BY price > 3000;
```

allsales				bigsales		
name	price	country	price > 3000	name	price	country
Alice	2999	us		Bob	3625	ca
Bob	3625	ca		Fredo	5637	it
Carlos	2764	mx				
Dieter	1749	de				
Étienne	2368	fr				
Fredo	5637	it				

Filtering by Multiple Criteria

- The criteria can include AND or OR

```
somesales = FILTER allsales BY name == 'Dieter' OR (price > 3500 AND price < 4000);
```

name	price	country
Alice	2999	us
Bob	3625	ca
Carlos	2764	mx
Dieter	1749	de
Étienne	2368	fr
Fredo	5637	it



name	price	country
Bob	3625	ca
Dieter	1749	de

Name is Dieter, or price is greater than 3500 and less than 4000

Field Selection in Pig Latin

- Filtering extracts rows, but how do we extract columns?
- We do this by using the FOREACH and GENERATE keywords

```
twofields = FOREACH allsales GENERATE amount, trans_id;
```

allsales			twofields	
salesperson	amount	trans_id	amount	trans_id
Alice	2999	107546	2999	107546
Bob	3625	107547	3625	107547
Carlos	2764	107548	2764	107548
Dieter	1749	107549	1749	107549
Étienne	2368	107550	2368	107550
Fredo	5637	107550	5637	107550

Generating New Fields in Pig Latin

- The FOREACH and GENERATE keywords can also be used to create fields

```
t = FOREACH allsales GENERATE price * 0.07;
```

- We can name such fields using the AS keyword

```
t = FOREACH allsales GENERATE price * 0.07 AS tax;
```

- You can also specify the data type

```
t = FOREACH allsales GENERATE price * 0.07 AS tax:float;
```


Eliminating Duplicates

- DISTINCT eliminates duplicate records in a bag
- All fields must be equal to be considered a duplicate

```
unique_records = DISTINCT all_alices;
```

all_alices

firstname	lastname	country
Alice	Smith	us
Alice	Jones	us
Alice	Brown	us
Alice	Brown	us
Alice	Brown	ca



unique_records

firstname	lastname	country
Alice	Smith	us
Alice	Jones	us
Alice	Brown	us
Alice	Brown	ca

Sorting in Pig

- Use ORDER.... BY to sort the records in a bag in ascending order
- Add DESC to sort in descending order instead

```
sortedsales = ORDER allsales BY country DESC;
```

allsales

name	price	country
Alice	29.99	us
Bob	36.25	ca
Carlos	27.64	mx
Dieter	17.49	de
Étienne	23.68	fr
Fredo	56.37	it



sortedsales

name	price	country
Alice	29.99	us
Carlos	27.64	mx
Fredo	56.37	it
Étienne	23.68	fr
Dieter	17.49	de
Bob	36.25	ca



Limiting Results

- You can use `LIMIT` to reduce the number of output records

```
sortedsales = ORDER allsales BY price DESC;  
top_five = LIMIT sortedsales 5;
```

Commonly used Functions in Pig

Function Description	Example Invocation	Input	Output
Convert to uppercase	UPPER (country)	uk	UK
Remove leading/trailing spaces	TRIM (name)	└─Bob┐	Bob
Return a random number	RANDOM ()		0.4816132 6652569
Round to closest whole number	ROUND (price)	37.19	37
Return chars between two positions	SUBSTRING (name, 0, 2)	Alice	Al

- You can use these functions with the FOREACH.. GENERATE keywords

```
rounded = FOREACH allsales GENERATE ROUND (price);
```

Grouping Data

Grouping Records By a Field

- You can group records by a given field using the GROUP... BY statement in Pig Latin

Alice	729
Bob	3999
Alice	27999
Carol	32999
Carol	4999

```
grunt> byname = GROUP sales BY name;
```

- The new relation has one record per unique value in the specified field

Grouping Records By a Field

- Data after grouping

```
grunt> byname = GROUP sales BY name;  
grunt> DUMP byname;  
(Bob, { (Bob, 3999) })  
(Alice, { (Alice, 729), (Alice, 27999) })  
(Carol, { (Carol, 32999), (Carol, 4999) })
```

group
field

sales
field

Input Data (**sales**)

Alice	729
Bob	3999
Alice	27999
Carol	32999
Carol	4999

Using GROUP BY to Aggregate Data

- Aggregate functions create one output value from multiple input values
 - Applied to grouped data
 - For example, to calculate total sales by employee

```
grunt> byname = GROUP sales BY name;
grunt> DUMP byname;
(Bob,{ (Bob,3999) })
(Alice,{ (Alice,729) , (Alice,27999) })
(Carol,{ (Carol,32999) , (Carol,4999) })

grunt> totals = FOREACH byname GENERATE
                                group, SUM(sales.price);
grunt> dump totals;
(Bob,3999)
(Alice,28728)
(Carol,37998)
```


Using GROUP ALL to Aggregate Data

- GROUP... ALL puts all the data into one record
- Use GROUP... ALL when you need to aggregate one or more columns

```
grunt> grouped = GROUP sales ALL;  
grunt> DUMP grouped;  
(all, { (Alice, 729) , (Bob, 3999) , (Alice, 27999) , (Carol, 32999) ,  
(Carol, 4999) })  
  
grunt> totals = FOREACH grouped GENERATE SUM(sales.price);  
grunt> dump totals;  
(70725)
```

Pig's Aggregate Functions

- Pig has built-in support for other aggregate functions besides SUM
- Examples:
 - AVG: calculate the average of all values
 - MIN: Returns the smallest value
 - MAX: Returns the largest value
 - COUNT: Returns the number of non-null elements in the bag
 - COUNT_STAR: Return the number of all elements in the bag