PIG

CAN BE RUN

IN TWO MODES

PIG ON A HADOOP CLUSTER

EVEN IN THIS MODE, PIG ONLY NEEDS TO BE INSTALLED ON YOUR MACHINE, NOT ON THE HADOOP CLUSTER

PIG ON A LOCAL MACHINE

PIG ON A LOCAL MACHINE

THIS MODE IS USEFUL FOR

PROTOTYPING

&

DEBUGGING

PIG ON A LOCAL MACHINE

RUNNING PIG IN LOCAL MODE

DOES NOT REQUIRE HADOOP INSTALLATION

OR

HDFS INSTALLATION

PIG ON A LOCAL MACHINE

FILES WILL BE INSTALLED AND RUN FROM

LOCAL HOST

&

LOCAL FILE SYSTEM

PIG ON A LOCAL MACHINE

PIG SCRIPTS WILL BE RUN USING

```
pig -x local pig_script_file.pig
```

NAME OF THE PIG SCRIPT

PIG ON A LOCAL MACHINE

PIG SCRIPTS WILL BE RUN USING

```
pig -x local pig_script_file.pig
```

SPECIFIES THAT THE MODE IS LOCAL

PIG ON A LOCAL MACHINE

PIG SCRIPTS WILL BE RUN USING

```
pig -x local pig_script_file.pig
```

TO RUN THIS SCRIPT FILE, DON'T FORGET TO COPY THE SCRIPT FILE TO THE PIG DIRECTORY

PIG

CAN BE RUN

IN TWO MODES

PIG ON A HADOOP CLUSTER

EVEN IN THIS MODE, PIG ONLY NEEDS TO BE INSTALLED ON YOUR MACHINE, NOT ON THE HADOOP CLUSTER

PIG ON A LOCAL MACHINE

PIG ON A HADOOP CLUSTER

THIS IS THE MOST COMMON WAY

IN WHICH PIG IS USED

PIG ON A HADOOP CLUSTER

BTW, THIS NOT MEAN THAT

PIG HAS TO BE INSTALLED ON THE HADOOP CLUSTER

PIG CAN BE INSTALLED ON YOUR OWN MACHINE

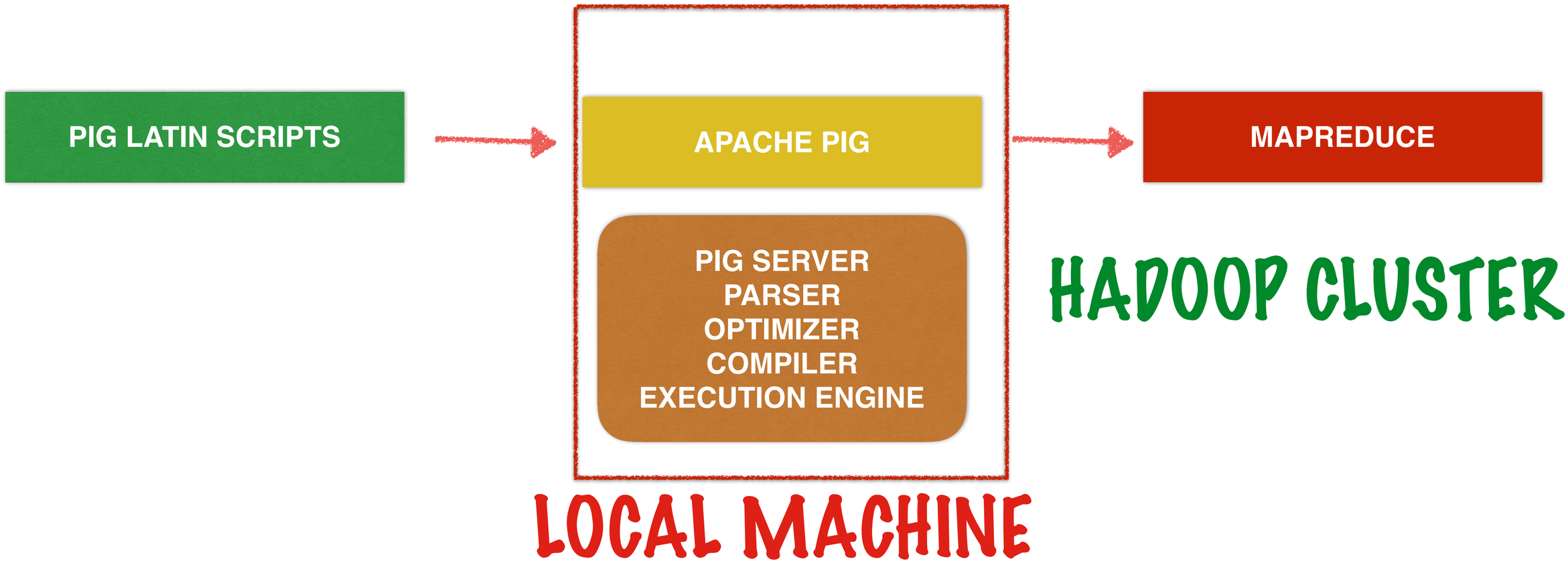
PIG CAN BE INSTALLED ON YOUR OWN MACHINE

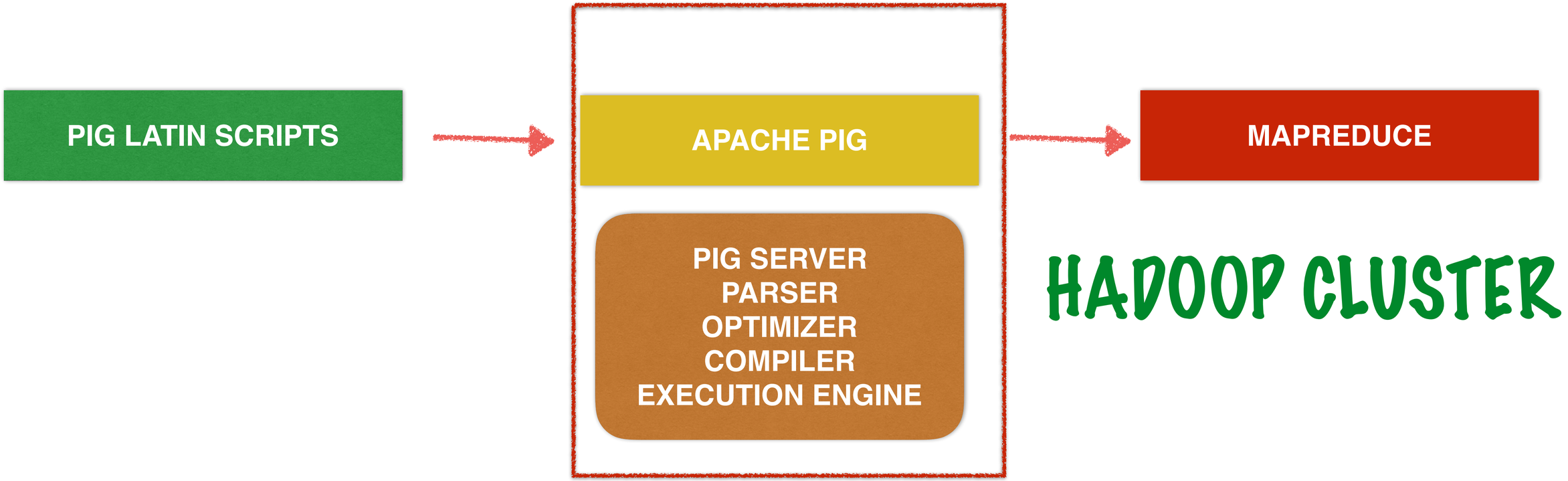
PIG JOBS WILL BE LAUNCHED FROM YOUR MACHINE

PIG WILL CONVERT PIG SCRIPTS TO MAPREDUCE JOBS ON YOUR MACHINE AND THEN RUN THESE JOBS ON HADOOP

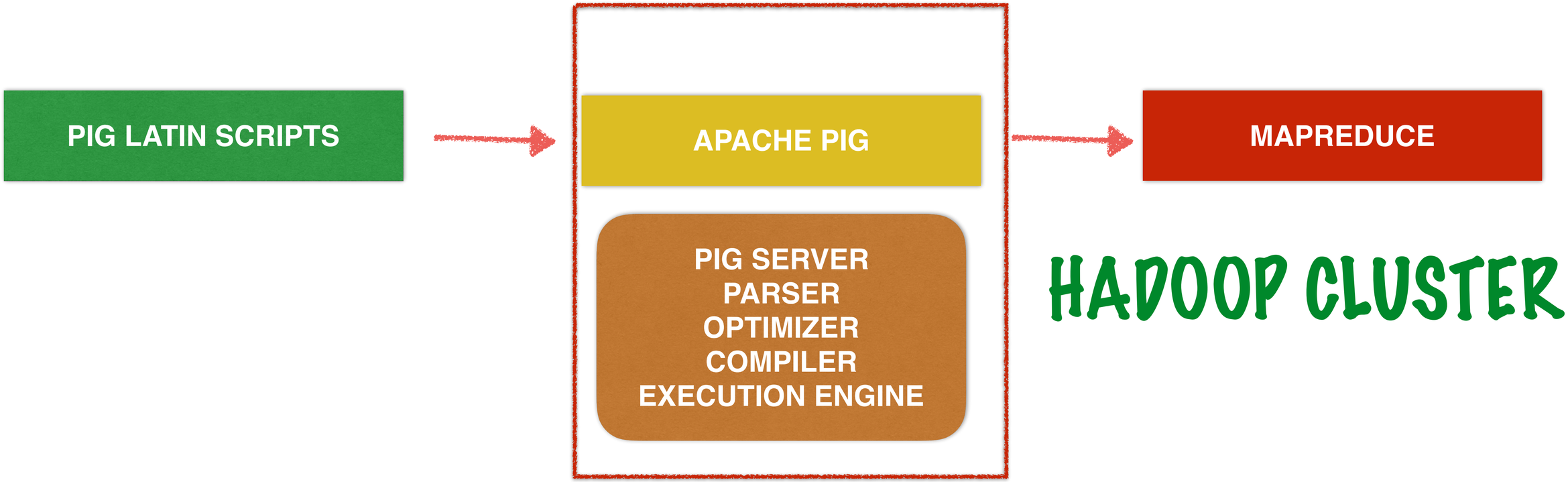
PIG JOBS WILL BE LAUNCHED FROM YOUR MACHINE

PIG WILL CONVERT PIG SCRIPTS TO MAPREDUCE JOBS ON YOUR MACHINE AND THEN RUN THESE JOBS ON HADOOP





USUALLY, SYSTEM ADMINISTRATORS WILL MERELY SET UP PIG ON ONE OR MORE MACHINES THAT HAVE ACCESS TO THE HADOOP CLUSTER



YOU WILL CONNECT TO THOSE GATEWAY MACHINES TO RUN PIG LATIN SCRIPTS

PIG ON A HADOOP CLUSTER

PIG SCRIPTS WILL BE RUN USING

```
pig -x mapreduce pig_script_file.pig
```

NAME OF THE PIG SCRIPT

THIS COMMAND WILL BE RUN FROM YOUR HDFS HOME DIRECTORY

PIG ON A HADOOP CLUSTER

PIG SCRIPTS WILL BE RUN USING

```
pig -x mapreduce pig_script_file.pig
```

NAME OF THE PIG SCRIPT

DO ENSURE THAT THE SCRIPT IS IN YOUR HDFS HOME DIRECTORY

ERRM..BUT WHERE ARE THESE COMMANDS RUN?

ERRM..BUT WHERE ARE THESE COMMANDS RUN?

IN A PIG SHELL NAMED

GRUNT

GRUNT

TO INVOKE GRUNT, TYPE THE FOLLOWING COMMAND AT YOUR TERMINAL

```
pig -x mapreduce
```

THIS IS THE COMMAND WE WOULD USE IF WE ARE USING PIG ON A HADOOP CLUSTER

GRUNT

TO INVOKE GRUNT, TYPE THE FOLLOWING COMMAND AT YOUR TERMINAL

```
pig -x local
```

THIS IS THE COMMAND WE WILL USE IF WE ARE USING PIG IN LOCAL MODE

GRUNT

TO INVOKE GRUNT, TYPE THE FOLLOWING COMMAND IN TERMINAL

```
pig -x local
```

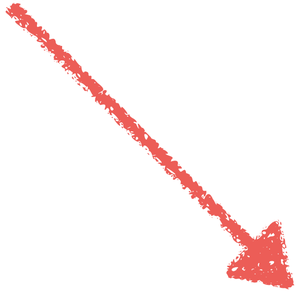
```
pig -x mapreduce
```

BOTH OF THESE COMMANDS WILL RESULT IN THE PROMPT

```
grunt>
```

GRUNT

PIG WILL NOT START EXECUTING UNTILL IT SEES EITHER 'STORE' OR 'DUMP' COMMAND



STORE IS USED TO STORE DATA TO FILES

DUMP IS USED TO PRINT OUTPUT TO THE SCREEN

GRUNT

PIG WILL NOT START EXECUTING UNTILL IT SEES EITHER 'STORE' OR 'DUMP' COMMAND

IT WILL CARRY OUT SYNTAX AND SEMANTIC CHECKS TO HELP CATCH ERRORS

GRUNT

BTW, GRUNT COULD BE USED AS A SHELL FOR HDFS

HIVE'S SHELL DOESN'T GIVE US THIS LUXURY

GRUNT

SOME HDFS COMMANDS THAT WILL BE HANDY IN GRUNT

```
grunt> fs -ls /
```

LISTS ALL THE FILES IN THE HDFS FILESYSTEM

GRUNT

SOME HDFS COMMANDS THAT WILL BE HANDY IN GRUNT

```
grunt> copyFromLocal  localfile  hdfsfile
```

COPIES FILE FROM LOCAL DISK TO YOUR HDFS HOME DIRECTORY

GRUNT

SOME HDFS COMMANDS THAT WILL BE HANDY IN GRUNT

```
grunt> copyToLocal  hdfsfile  localfile
```

COPIES FILE FROM HDFS TO LOCAL DISK

GRUNT

SOME HDFS COMMANDS THAT WILL BE HANDY IN GRUNT

```
grunt>fs -rmr   name_of_the_file
```

THIS REMOVES FILES RECURSIVELY

GRUNT

SOME HDFS COMMANDS THAT WILL BE HANDY IN GRUNT

```
grunt> cat   name_of_the_file
```

PRINTS CONTENT OF THE FILE TO STDOUT

GRUNT

ALSO GIVES THE CAPABILITY TO CONTROL PIG AND MAPREDUCE JOBS

```
grunt> kill jobID
```

KILLS THE MAPREDUCE JOB HAVING JOB ID equal to `jobID`

GRUNT

ALSO GIVES THE CAPABILITY TO CONTROL PIG AND MAPREDUCE JOBS

```
grunt> CTRL + C
```

TERMINATES THE CURRENT PIG PROCESS

GRUNT

IS ALSO WHERE WE USE

PIG LATIN

NOW IS THE TIME TO START LEARNING THIS LANGUAGE

PIG LATIN

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET

WHICH HAS A NAME

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET WHICH HAS A NAME

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

PIG LATIN IS A DATAFLOW LANGUAGE

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

ALL OF THESE COMMANDS ARE DIFFERENT STEPS IN THE DATA FLOW

PIG LATIN IS A DATAFLOW LANGUAGE

ALL OF THESE COMMANDS ARE DIFFERENT STEPS IN THE DATA FLOW

WHERE DATA IS CONVERTED FROM ONE FORM TO ANOTHER

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET WHICH HAS A NAME

```
data_set_1 = pig latin command to fetch data from some file
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH OF THESE NEW DATASETS IS CALLED A RELATION

PIG LATIN IS A DATAFLOW LANGUAGE

data_set_1

data_set_2

data_set_3

data_set_5

data_set_4

EACH OF THESE NEW DATASETS IS CALLED A RELATION

THESE RELATION NAMES ARE LIKE VARIABLES

PIG LATIN IS A DATAFLOW LANGUAGE

data_set_1

data_set_2

data_set_3

data_set_5

data_set_4

THESE RELATION NAMES ARE LIKE VARIABLES

YOU CAN REUSE THEM IF YOU WISH TO,
BUT YOU PROBABLY SHOULD NOT

PIG LATIN IS A DATAFLOW LANGUAGE

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH OF THESE NEW DATASETS IS CALLED A
RELATION

THESE RELATIONS DON'T PERSIST
BEYOND YOUR CURRENT SESSION

PIG LATIN IS A DATAFLOW LANGUAGE

```
data_set_1 = pig latin command to fetch data from some file
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH OF THESE NEW DATASETS IS CALLED A RELATION

THESE RELATIONS ARE NOT COMPUTED AT EVERY STEP

PIG LATIN IS A DATAFLOW LANGUAGE

THESE RELATIONS ARE NOT COMPUTED AT EVERY STEP

THEY ARE COMPUTED ONLY WHEN ONE DECIDES TO PRINT THEM TO SCREEN OR STORE THEM TO A FILE

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH OF THESE NEW DATASETS IS CALLED A RELATION

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

LET US TALK ABOUT THE FIRST STEP

EACH OF THIS NEW DATASET IS CALLED RELATION

```
data_set_1 = pig latin command to fetch data from some file;
```

THE FIRST STEP USUALLY IS TO

FETCH DATA FROM SOME PLACE

THE LOAD COMMAND FETCHES DATA

THE LOAD COMMAND TO FETCH DATA

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS

IN A TAB-DELIMITED FILE

USING DEFAULT LOAD FUNCTION PIGSTORAGE

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

```
grunt> orders = load 'temp.txt' using PigStorage(',');
```

THIS IS THE RELATION NAME. IT WILL HOLD THE DATA SET THAT WILL BE FETCHED

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

```
grunt> orders = load 'temp.txt' using PigStorage(',');
```

PIG WILL LOOK FOR THIS FILE IN YOUR HOME DIRECTORY ON HDFS

/USERS/YOURLOGIN

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

```
grunt> orders = load 'temp.txt' using PigStorage(',');
```

PIGSTORAGE, THE LOAD FUNCTION, CAN TAKE AN ARGUMENT TO INDICATE WHICH CHARACTER TO USE AS A DELIMITER

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

```
grunt> orders = load 'temp.txt' using PigStorage(',');
```

WE HAVE GIVEN ',' AS AN ARGUMENT TO INDICATE THAT WE FETCH DATA FROM A CSV FILE

THE TEMP FILE LOOKS LIKE THIS

```
1,'Gandhi House',110
2,'Akbar Hall',231
3,'Gandhi House',345
4,NULL,NULL
```

OUTPUT OF THE FILE IN PIG

```
(1,'Gandhi House',110)
(2,'Akbar Hall',231)
(3,'Gandhi House',345)
(4,NULL,NULL)
```

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

WE CAN FETCH DATA FROM HBASE USING LOADER FOR HBASE

WE HAVE GIVEN '' AS AN ARGUMENT TO INDICATE THAT WE ARE FETCHING DATA FROM A CSV FILE

HBaseStorage

WE CAN FETCH DATA FROM HBASE USING LOADER FOR HBASE

HBaseStorage

```
grunt> orders = load ‘Hbase_file.txt' using HBaseStorage();
```

YOU OF COURSE HAVE TO HAVE PIG SET UP TO WORK WITH HBASE - IT’S BEYOND THE SCOPE OF THIS COURSE

WE CAN FETCH DATA FROM HBASE USING LOADER FOR HBASE

HBaseStorage

```
grunt> orders = load ‘Hbase_file.txt' using HBaseStorage();
```

THERE ARE OTHER LOADER FUNCTIONS LIKE TEXTLOADER WHICH YOU CAN READ ABOUT

BY DEFAULT

LOAD LOOKS FOR YOUR DATA ON HDFS IN A TAB-DELIMITED FILE USING DEFAULT LOAD FUNCTION PIGSTORAGE

```
grunt> orders = load 'temp.txt' using PigStorage(',');
```

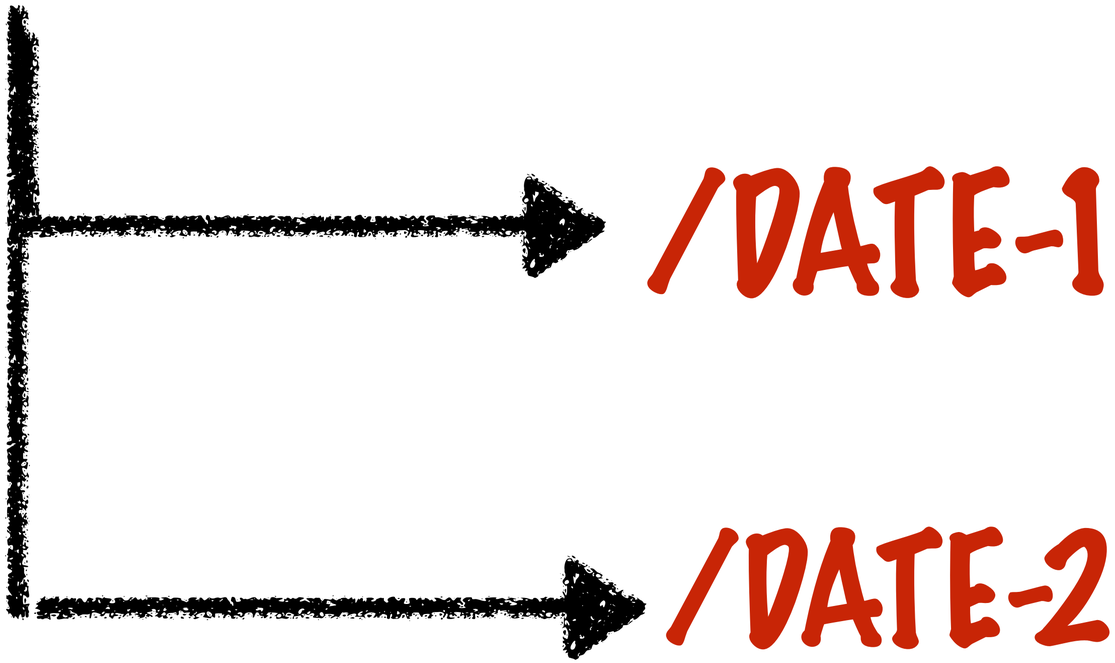
WE CAN FETCH DATA FROM DIRECTORIES AS WELL

WE HAVE GIVEN ',' AS AN ARGUMENT TO INDICATE THAT WE ARE FETCHING DATA FROM A CSV FILE

WE CAN FETCH DATA FROM DIRECTORIES AS WELL

LET US ASSUME THAT DATA IS STORED IN MULTIPLE FILES AND THOSE MULTIPLE FILES ARE IN ONE FOLDER

/USER/YOURLOGIN/ORDER_DATA



PIG WILL READ FROM BOTH THE FILES IF YOU USE THE DIRECTORY NAME

/USER/YOURLOGIN/ORDER_DATA

PIG WILL READ FROM BOTH THE FILES IF YOU USE THE DIRECTORY NAME

```
grunt> orders = load 'order_data' using PigStorage(',');
```

```
grunt> orders = load 'order_data' using PigStorage(',');
```

IN NONE OF THESE LOAD STATEMENTS DID WE MENTION THE SCHEMA OF THE DATA

THIS IS THE POWER OF PIG

IN NONE OF THESE LOAD STATEMENTS DID WE EVER MENTION THE SCHEMA OF THE DATA

THIS IS THE POWER OF PIG

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA GUESSING ALONG THE WAY

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

HOW COULD WE COMMUNICATE SCHEMA INFORMATION TO PIG?

PROCESS THE DATA
GUESSING ALONG THE WAY

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

HOW DO WE COMMUNICATE SCHEMA TO PIG?

WE CAN COMMUNICATE SCHEMA OF THE DATA

DURING LOAD TIME

```
grunt> orders = load 'order_data' using PigStorage(',');
```

WE CAN COMMUNICATE SCHEMA OF THE DATA DURING LOAD TIME

```
grunt> orders = load 'order_data' using PigStorage(',');
```

LET US ASSUME THAT ORDER_DATA LOOK LIKE THIS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders = load 'order_data' using PigStorage(',');
```

THE order_data LOOKS LIKE THIS

```
OD01,1,1,450,2016-01-24,1,Apple iPhone 4S
OD01,1,1,450,2016-01-24,2,Apple iPhone 5S
OD01,1,1,450,2016-01-24,3,Apple iPhone 6
OD02,2,2,400,2016-01-24,1,Apple iPhone 4S
OD02,2,2,400,2016-01-24,2,Apple iPhone 5S
OD02,2,2,400,2016-01-24,3,Apple iPhone 6
OD03,2,1,350,2016-01-25,1,Apple iPhone 4S
OD03,2,1,350,2016-01-25,2,Apple iPhone 5S
OD03,2,1,350,2016-01-25,3,Apple iPhone 6
OD04,3,1,500,2016-01-25,1,Apple iPhone 4S
OD04,3,1,500,2016-01-25,2,Apple iPhone 5S
OD04,3,1,500,2016-01-27,3,Apple iPhone 6
OD05,4,1,650,2016-01-26,1,Apple iPhone 4S
OD05,4,1,650,2016-01-27,2,Apple iPhone 5S
OD05,4,1,650,2016-01-27,3,Apple iPhone 6
```

OUTPUT OF orders IN PIG

```
(OD01,1,1,450,2016-01-24,1,Apple iPhone 4S)
(OD01,1,1,450,2016-01-24,2,Apple iPhone 5S)
(OD01,1,1,450,2016-01-24,3,Apple iPhone 6)
(OD02,2,2,400,2016-01-24,1,Apple iPhone 4S)
(OD02,2,2,400,2016-01-24,2,Apple iPhone 5S)
(OD02,2,2,400,2016-01-24,3,Apple iPhone 6)
(OD03,2,1,350,2016-01-25,1,Apple iPhone 4S)
(OD03,2,1,350,2016-01-25,2,Apple iPhone 5S)
(OD03,2,1,350,2016-01-25,3,Apple iPhone 6)
(OD03,2,1,350,2016-01-25,3,Apple iPhone 6)
(OD04,3,1,500,2016-01-25,1,Apple iPhone 4S)
(OD04,3,1,500,2016-01-25,2,Apple iPhone 5S)
(OD04,3,1,500,2016-01-27,3,Apple iPhone 6)
(OD05,4,1,650,2016-01-26,1,Apple iPhone 4S)
(OD05,4,1,650,2016-01-27,2,Apple iPhone 5S)
(OD05,4,1,650,2016-01-27,3,Apple iPhone 6)
```

WE CAN COMMUNICATE SCHEMA OF THE DATA DURING LOAD TIME

```
grunt> orders = load 'order_data' using PigStorage(',');
```

THERE ARE VARIOUS ROWS AND ALL ROWS HAVE SIMILAR STRUCTURE

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THIS IS THE COLUMN OR FIELD

```
grunt> orders = load 'order_data' using PigStorage(',');
```

THESE COLUMN OR FIELDS HAVE FIXED DATA TYPES

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',')
```

```
as (Order_ID:chararray,
Customer_ID:int,
Order_Quantity:int,
Order_Value:float,
Order_Date:chararray,
Product_ID:int,
Product_Name:chararray);
```

THIS IS HOW YOU COMMUNICATE SCHEMA USING 'as'

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',')
as (Order_ID:chararray,
Customer_ID:int,
Order_Quantity:int,
Order_Value:float,
Order_Date:chararray,
Product_ID:int,
Product_Name:chararray);
```

EVERY FIELD IS REPRESENTED HERE IN THIS MANNER

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',')
as (Order_ID:chararray,
Customer_ID:int,
Order_Quantity:int,
Order_Value:float,
Order_Date:chararray,
Product_ID:int,
Product_Name:chararray);
```

FIELD_NAME: FIELD DATA_TYPE

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',')
as (Order_ID:chararray,
Customer_ID:int,
Order_Quantity:int,
Order_Value:float,
Order_Date:chararray,
Product_ID:int,
Product_Name:chararray);
```

FIELD_NAME: FIELD DATA_TYPE

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

USING THIS INFORMATION, WE WILL REWRITE THE LOAD COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',')
as (Order_ID:chararray,
Customer_ID:int,
Order_Quantity:int,
Order_Value:float,
Order_Date:chararray,
Product_ID:int,
Product_Name:chararray);
```

FIELD_NAME: FIELD DATA_TYPE

FIELD_NAME: FIELD DATA_TYPE

IN PIG LATIN, WE WILL EXPLORE OUR DATA, BUT FIRST -

LET US STUDY THE

DATA TYPES

OFFERED IN PIG

DATA TYPES

DATA EXISTS IN VARIOUS FORMS

NUMBERS, TEXT, IMAGES AND VIDEOS

ids	Rank	bday	Gender	Athlete
26582	.	.	Male	Athlete
46919	.	.	Female	Athlete
20230	Freshman	02-Jan-1996	Male	Athlete
21083	Freshman	22-Dec-1995	Female	Non-athlete
21939	Freshman	12-Dec-1995	Male	Athlete
23017	Freshman	29-Nov-1995	.	Non-athlete
23217	Freshman	26-Nov-1995	Male	Non-athlete
23889	Freshman	18-Nov-1995	Male	Athlete
24985	Freshman	05-Nov-1995	Female	Non-athlete

DATA TYPES

PIG SUPPORTS TWO CATEGORIES OF DATA TYPES

SCALAR

COMPLEX

HIVE ALSO HAD SIMILAR DATA TYPES NAMED PRIMITIVE AND COMPLEX

DATA TYPES

PIG SUPPORTS TWO MAJOR DATA TYPES

SCALAR

COMPLEX

SCALAR

DATA TYPES ARE SIMILAR TO THOSE THAT APPEAR IN ALL COMMON PROGRAMMING LANGUAGE

HIVE'S PRIMITIVE DATA TYPE INCLUDED BOOLEAN, NUMERIC, STRING AND TIMESTAMP TYPES ONLY

SCALAR

SIMPLE DATA TYPES THAT EXIST IN PIG ARE

INT

LONG

FLOAT

DOUBLE

BYTEARRAY

BOOLEAN

CHARARRAY

BIGINTEGER

DATETIME

BIGDECIMAL

SCALAR

SIMPLE DATA TYPES THAT EXIST IN PIG ARE

INT LONG FLOAT DOUBLE

BTW, THESE 3 ARE NOT PRESENT IN HIVE

BYTEARRAY

BOOLEAN

CHARARRAY

BIGINTEGER

DATETIME

BIGDECIMAL

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

INTEGERS

DECIMALS

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

INTEGERS

INT (4-BYTE)	LONG (8-BYTE)	BIGINTEGER (JAVA BIGINTEGER)
$(-2^{31}$ TO $2^{31}-1)$	$(-2^{63}$ TO $2^{63}-1)$	

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

DECIMALS

FLOAT (4-BYTE)	DOUBLE (8-BYTE)	BIGDECIMAL (JAVA BIGDECIMAL)

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

DECIMALS

FLOAT
(4-BYTE)

DOUBLE
(8-BYTE)

DOUBLE HAS A HIGHER PRECISION AND RANGE THAN FLOAT

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

CHARARRAY
(UNBOUNDED
VARIABLE LENGTH
CHARACTER
STRING)

EXAMPLES LIKE “JOHN”, ‘M’, ‘F’

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

CHARARRAY
(UNBOUNDED
VARIABLE LENGTH
CHARACTER
STRING)

HIVE HAD CHAR AND VARCHAR
WHICH ARE NOT PRESENT IN PIG

EXAMPLES LIKE "JOHN", 'M', 'F'

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

DATETIME

TIMESTAMP WITH
NANOSECOND PRECISION

'2001-14-1 T 00:00:00.000+00:00'

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

DATETIME
TIMESTAMP WITH
NANOSECOND PRECISION

'2001-14-1 T 00:00:00.000+00:00'

HIVE HAD THE
DATE DATA TYPE
WHICH IS NOT
PRESENT IN PIG

SCALAR

INCLUDES BOOLEAN, NUMERIC, STRING, TIMESTAMP AND BYTEARRAY TYPES

BYTEARRAY
A BLOB OR ARRAY OF BYTES

THIS IS THE DEFAULT DATATYPE USED IF YOU DON'T SPECIFY THE DATA TYPE FOR A FIELD

BYTEARRAY

A BLOB OR ARRAY OF BYTES

THIS IS THE DEFAULT DATATYPE USED IF YOU DON'T SPECIFY THE DATA TYPE FOR A FIELD

DATA TYPES

PIG SUPPORTS TWO MAJOR DATA TYPES

SCALAR

COMPLEX

COMPLEX DATA TYPE

MAP

TUPLE

BAG

HIVESQL HAD 4
COMPLEX DATA TYPES

ARRAY

MAP

STRUCT

UNION

TUPLE COMPLEX DATA TYPE

SIMILAR TO A ROW IN SQL

TUPLE IS A FIXED-LENGTH, ORDERED COLLECTION OF PIG DATA ELEMENTS

TUPLE COMPLEX DATA TYPE

SIMILAR TO A ROW IN SQL

TUPLE IS A FIXED-LENGTH, ORDERED COLLECTION OF PIG DATA ELEMENTS

A TUPLE IS REPRESENTED AS (FIELD1, FIELD2... FIELDN)

THESE FIELDS CAN BE OF ANY DATA TYPE

TUPLE COMPLEX DATA TYPE

A TUPLE IS REPRESENTED AS (FIELD1, FIELD2... FIELDN)

THESE FIELDS CAN BE OF ANY DATA TYPE

THE TUPLE CAN, OPTIONALLY, HAVE A SCHEMA ASSOCIATED WITH IT

TUPLE COMPLEX DATA TYPE

A TUPLE IS REPRESENTED AS (FIELD1, FIELD2... FIELDN)

THESE FIELDS CAN BE OF ANY DATA TYPE

EXAMPLE

```
(Order_ID: chararray,
  Customer_ID: int)
```

```
(OD01,
1)
```

TUPLE EXAMPLE

(Order_ID: chararray,
Customer_ID: int)

(OD01,
1)

HOW DO YOU LOAD A TUPLE?

```
order_customer = load 'order_customer' as
(order_customer_id:
tuple(Order_ID:chararray,Customer_ID:int));
```

TUPLE

EXAMPLE

```
(Order_ID: chararray,
  Customer_ID: int)
```

```
(OD01,
1)
```

FIELDS IN A TUPLE CAN BE ACCESSED USING THE NAME OF THE FIELD AND "." OPERATOR

TUPLE NAME FIELD NAME

```
order_customer_id.Order_ID = 'OD01'

order_customer_id.Customer_ID = 1
```

```
order_customer = load 'order_customer' as (order_customer_id:tuple(Order_ID:chararray,Customer_ID:int));
```

THE order_customer IN FILE LOOKS LIKE THIS

```
(OD01,1)
(OD01,1)
(OD01,1)
(OD02,2)
(OD02,2)
(OD02,2)
(OD03,2)
(OD03,2)
(OD03,2)
(OD03,2)
(OD04,3)
(OD04,3)
(OD04,3)
(OD05,4)
(OD05,4)
(OD05,4)
```

OUTPUT OF order_customer IN PIG

```
((OD01,1))
((OD01,1))
((OD01,1))
((OD02,2))
((OD02,2))
((OD02,2))
((OD03,2))
((OD03,2))
((OD03,2))
((OD03,2))
((OD04,3))
((OD04,3))
((OD04,3))
((OD05,4))
((OD05,4))
((OD05,4))
```

COMPLEX DATA TYPE

MAP

TUPLE

BAG

TUPLE IS SIMILAR TO STRUCT

HIVESQL HAD 4 COMPLEX DATA TYPES

ARRAY

MAP

STRUCT

UNION

BAG

COMPLEX DATA TYPE

SIMILAR TO A SET IN JAVA

A BAG IS AN UNORDERED COLLECTION OF TUPLES

BAG

COMPLEX DATA TYPE

SIMILAR TO A SET IN JAVA

A BAG IS AN UNORDERED COLLECTION OF TUPLES

BAGS ARE REPRESENTED AS BELOW

{(FIELD1,FIELD2), (FIELD1,FIELD2), (FIELD1,FIELD2)}

BAG

COMPLEX DATA TYPE

{(FIELD1,FIELD2), (FIELD1,FIELD2), (FIELD1,FIELD2)}

CONSIDER THE TUPLE

```
(Order_ID: chararray,
  Customer_ID: int,
Order_Quantity: int,
 Order_Value: float,
Order_Date: chararray,
   Product_ID: int,
Product_Name: chararray)
```

```
(OD01,
  1,
  1,
 450.0
,2016-01-24
   ,1
,Apple iPhone 4S)
```

CONSIDER THE TUPLE

```
(Order_ID: chararray,
  Customer_ID: int,
Order_Quantity: int,
 Order_Value: float,
Order_Date: chararray,
   Product_ID: int,
Product_Name: chararray)
```

```
(OD01,
1,
1,
450.0
,2016-01-24
,1
,Apple iPhone 4S)
```

A BAG OF SUCH TUPLES WILL LOOK LIKE THIS

{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),
(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),
(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6),
(OD02,2,2,400.0,2016-01-25,1,Apple iPhone 4S),
(OD02,2,2,400.0,2016-01-25,2,Apple iPhone 5S),

(OD02,2,2,400.0,2016-01-25,3,Apple iPhone 6)}

```
(Order_ID: chararray,
  Customer_ID: int,
 Order_Quantity: int,
  Order_Value: float,
Order_Date: chararray,
   Product_ID: int,
Product_Name: chararray)
```

```
(OD01,
 1,
 1,
 450.0
 ,2016-01-24
 ,1
 ,Apple iPhone 4S)
```

A BAG OF SUCH TUPLES WILL LOOK LIKE THIS

{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),
(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),
(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6),
(OD02,2,2,400.0,2016-01-25,1,Apple iPhone 4S),
(OD02,2,2,400.0,2016-01-25,2,Apple iPhone 5S),
(OD02,2,2,400.0,2016-01-25,3,Apple iPhone 6)}

THIS IS A RELATION!

CONSIDER THE TUPLE

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

(OD01,
1,
1,
450.0
,2016-01-24
,1
,Apple iPhone 4S)

HOW DO YOU A LOAD A BAG OF TUPLES?

```
load 'order_bag' as
(order_customer_id: bag{tuple_name:
```

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

TUPLE_SCHEMA

```
});
```

THE order_bag IN FILE LOOKS LIKE THIS

```
{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)}
{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)}
{(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)}
{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)}
{(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)}
{(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)}
{(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)}
{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)}
{(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)}
{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)}
{(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)}
{(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)}
{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)}
{(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)}
{(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)}
```

OUTPUT OF order_bag IN PIG

```
({(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)})
({(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)})
({(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
({(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)})
({(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)})
({(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
({(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
({(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)})
({(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)})
({(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)})
({(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)})
({(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
({(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)})
({(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)})
({(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

CONSIDER THE TUPLE

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

(OD01,
1,
1,
450.0
,2016-01-24
,1
,Apple iPhone 4S)

HOW DO YOU A LOAD A BAG OF TUPLE?

```
load 'order_bag' as
(order_customer_id: bag{tuple_name:
```

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

```
});
```

THE TUPLE_NAME IS REQUIRED THOUGH IT CANNOT BE USED!

HOW DO YOU A LOAD A BAG OF TUPLE?

```
load 'order_bag' as
(order_customer_id: bag{tuple_name:
        (Order_ID: chararray,
        Customer_ID: int,
        Order_Quantity: int,
        Order_Value: float,
        Order_Date: chararray,
        Product_ID: int,
        Product_Name: chararray)
});
```

TUPLE_SCHEMA

BAG

IT IS A PRETTY IMPORTANT DATA TYPE IN PIG

BAGS WILL BE USED IN OPERATIONS LIKE

JOIN AND GROUPING

WE WILL SEE THIS IN A BIT

COMPLEX DATA TYPE

MAP

TUPLE

BAG

HIVESQL HAD 4
COMPLEX DATA TYPES

ARRAY

MAP

STRUCT

UNION

MAP COMPLEX DATA TYPE

SIMILAR TO THEIR NAMESAKES IN JAVA

THE MAP IS A DATA TYPE WHERE DATA IS REPRESENTED IN THE FORM OF PAIRS.

A MAP IS REPRESENTED AS A (KEY, VALUE) PAIR

EACH KEY IN MAP NEED NOT BE UNIQUE AND IS MAPPED TO A VALUE

MAP

A MAP IS REPRESENTED AS A (KEY, VALUE) PAIR

IN PIG, THE KEY IS ALWAYS OF CHAR ARRAY DATA TYPE

VALUE CAN BE OF ANY DATA TYPE

RECALL THAT IN HIVE, KEY AND VALUE CAN BOTH BE OF ANY DATA TYPE

MAP

MAP IS REPRESENTED AS A (KEY, VALUE) PAIR
IN PIG, KEY IS ALWAYS OF CHAR ARRAY DATA TYPE
VALUE CAN BE OF ANY DATA TYPE

A LANGUAGE DICTIONARY IS AN EXAMPLE OF A MAP

[WORD# MEANING]

KEY

DATA TYPE =CHARARRAY

VALUE

DATA TYPE =CHARARRAY

MAP

[WORD# MEANING]

VALUE

DATA TYPE =CHARARRAY

MEANINGS ARE TYPICALLY SENTENCES MADE OF WORDS OF DIFFERENT LENGTH

MAP

MAP IS REPRESENTED AS A (KEY, VALUE) PAIR

EACH KEY IN MAP IS MAPPED TO A UNIQUE VALUE

A LANGUAGE DICTIONARY IS AN EXAMPLE OF A MAP

[WORD# MEANING]

THE MAP WILL LOOK LIKE THIS

[sports#‘game’]

[happy#‘jolly’]

MAP

MAP IS REPRESENTED AS A (KEY, VALUE) PAIR

[WORD# MEANING]

HOW WOULD YOU LOAD 'MAP' DATA TYPE?

```
oxford_dictionary  = load 'oxford_dictionary'
as(Word_ID:long,
Page_ID:int,
Dictionary:map[chararray]);
```

YOU ONLY SPECIFY THE DATA TYPE OF THE VALUE

MAP

[WORD# MEANING]

```
oxford_dictionary  = load 'oxford_dictionary'
as(Word_ID:long,
Page_ID:int,
Dictionary:map[chararray]);
```

HOW TO GET 'VALUE' IN MAP USING 'KEY'?

```
MAP_NAME#'KEY'
```

```
oxford_dictionary  = load 'oxford_dictionary'
as(Word_ID:long,
Page_ID:int,
Dictionary:map[chararray]);
```

HOW TO GET 'VALUE' IN MAP USING 'KEY'?

MAP_NAME#'KEY'

[happy#'jolly']

[sports#'games']

Dictionary#'happy' = 'jolly'

Dictionary#'sports' = 'games'

THE oxford_dictionary IN FILE LOOKS LIKE THIS

```
1	1	[sports#'game']
2	1	[happy#'jolly']
```

OUTPUT OF oxford_dictionary IN PIG

```
(1,1,[sports#'game'])
(2,1,[happy#'jolly'])
```

SUMMARY

Data type	Syntax	Example
int	`int`	`as (a:int)`
long	`long`	`as (a:long)`
float	`float`	`as (a:float)`
double	`double`	`as (a:double)`
chararray	`chararray`	`as (a:chararray)`
bytearray	`bytearray`	`as (a:bytearray)`
map	`map[]` or `map[type]`, where *type* is any valid type. This declares all values in the map to be of this type.	`as (a:map[], b:map[int])`
tuple	`tuple()` or `tuple(list_of_fields)`, where *list_of_fields* is a comma-separated list of field declarations.	`as (a:tuple(), b:tuple(x:int, y:int))`
bag	`bag{}` or `bag{t:(list_of_fields)}`, where *list_of_fields* is a comma-separated list of field declarations. Note that, oddly enough, the tuple inside the bag must have a name, here specified as `t`, even though you will never be able to access that tuple `t` directly.	`(a:bag{}, b:bag{t:(x:int, y:int)})`

TAKEN FROM "PROGRAMMING PIG" BY ALAN GATES

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA GUESSING ALONG THE WAY

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA

GUESSING ALONG THE WAY

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA

GUESSING ALONG THE WAY

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA

WE HAVE SEEN HOW WE CAN LOAD USING A SCHEMA

WE CAN ALSO PARTIALLY SPECIFY THE SCHEMA

WE CAN ALSO PARTIALLY SPECIFY THE SCHEMA

WE NEED NOT MENTION THE DATA TYPES IF WE DON'T KNOW them

```
grunt> orders = load 'order_data' using PigStorage(‘,')
as (Order_ID,
Customer_ID,
Order_Quantity,
Order_Value,
Order_Date,
Product_ID,
Product_Name);
```

THIS IS VALID!

WE COULD ALSO PARTIALLY SPECIFY THE SCHEMA

WE NEED NOT MENTION THE DATA TYPES IF WE DON'T KNOW

```
grunt> orders = load 'order_data' using PigStorage(',')
as (Order_ID,
Customer_ID,
Order_Quantity,
Order_Value,
Order_Date,
Product_ID,
Product_Name);
```

THE DATATYPE IS ASSUMED TO BE BYTEARRAY

WE COULD ALSO PARTIALLY SPECIFY THE SCHEMA

WE NEED NOT MENTION THE SCHEMA OF COMPLEX DATA TYPES IF WE DON'T WANT TO

THIS IS OKAY!

```
oxford_dictionary  = load
'oxford_dictionary'
as(Word_ID:long,
Page_ID:int,
Dictionary:map[]);
```

WE COULD ALSO PARTIALLY SPECIFY THE SCHEMA

WE CAN MENTION ONLY FIRST FEW FIELDS IN THE LOAD STATEMENTS

```
grunt> orders = load ‘order_data'using PigStorage(‘,')
as (Order_ID:chararray,
 Customer_ID:int);
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE MENTIONED ONE SCHEMA AND LOADER FOUND ANOTHER ONE

PIG WILL TRY TO ADAPT THE SCHEMA RETURNED BY LOADER TO MATCH

THE SCHEMA SPECIFIED BY US

LET US SAY WE MENTIONED ONE SCHEMA AND LOADER FOUND ANOTHER ONE

PIG WILL TRY TO ADAPT THE SCHEMA RETURNED BY LOADER TO MATCH

THE SCHEMA SPECIFIED BY US

THIS IS ACCOMPLISHED BY CASTING

PIG WILL TRY TO ADAPT THE SCHEMA RETURNED BY LOADER TO MATCH THE SCHEMA SPECIFIED BY US

THIS IS ACCOMPLISHED BY CASTING

IF YOU SPECIFIED A FIELD TYPE TO BE FLOAT AND IT TURNED OUT TO BE INT, INT WILL BE CAST TO FLOAT

PIG WILL TRY TO ADAPT THE SCHEMA RETURNED BY LOADER TO MATCH THE SCHEMA SPECIFIED BY US

THIS IS ACCOMPLISHED BY CASTING

NOT ALL DATATYPE CONVERSIONS ARE PERMITTED

NOT ALL DATATYPE CONVERSIONS ARE PERMITTED

IF PIG CAN'T FIT THE LOADER'S SCHEMA TO THE ONE WE MENTIONED IN THE SCRIPT

IT WILL GIVE AN ERROR

NOT ALL DATATYPE CONVERSIONS ARE PERMITTED

Description

Pig Latin supports casts as shown in this table.

	to								
from	bag	tuple	map	int	long	float	double	chararray	bytearray
bag		error	error	error	error	error	error	error	error
tuple	error		error	error	error	error	error	error	error
map	error	error		error	error	error	error	error	error
int	error	error	error		yes	yes	yes	yes	error
long	error	error	error	yes		yes	yes	yes	error
float	error	error	error	yes	yes		yes	yes	error
double	error	error	error	yes	yes	yes		yes	error
chararray	error	error	error	yes	yes	yes	yes		error
bytearray	yes	yes	yes	yes	yes	yes	yes	yes	

NOTICE CONVERSIONS THAT ARE POSSIBLE

NOT ALL DATATYPE CONVERSIONS ARE PERMITTED

Description

Pig Latin supports casts as shown in this table.

	to								
from	bag	tuple	map	int	long	float	double	chararray	bytearray
bag		error	error	error	error	error	error	error	error
tuple	error		error	error	error	error	error	error	error
map	error	error		error	error	error	error	error	error
int	error	error	error		yes	yes	yes	yes	error
long	error	error	error	yes		yes	yes	yes	error
float	error	error	error	yes	yes		yes	yes	error
double	error	error	error	yes	yes	yes		yes	error
chararray	error	error	error	yes	yes	yes	yes		error
bytearray	yes	yes	yes	yes	yes	yes	yes	yes	

THIS TABLE IS GIVEN ON PIG APACHE WEBSITE

PIG WILL TRY TO ADAPT THE SCHEMA RETURNED BY LOADER TO MATCH THE SCHEMA SPECIFIED BY US

THIS IS ACCOMPLISHED BY CASTING

WHEN WE DON'T MENTION DATATYPES, ALL FIELDS ARE ASSUMED TO HAVE THE BYTEARRAY DATATYPE

THIS IS ACCOMPLISHED BY CASTING

WHEN WE DON'T MENTION DATATYPES, ALL FIELDS ARE ASSUMED TO HAVE BYTEARRAY DATATYPE

OH, AND PIG CAN TYPECAST BYTEARRAY TO ALL OTHER DATATYPES IF NEEDED

WHEN WE DON'T MENTION DATATYPES, ALL FIELDS ARE ASSUMED TO HAVE BYTEARRAY DATATYPE

AND PIG CAN'T PERFORM ARITHMETIC OPERATIONS ON BYTEARRAYS. WE HAVE TO CAST THEM TO OTHER DATATYPES LATER ON

ERRM..HOW DO WE ACCESS COLUMNS IF WE HAVE NOT SPECIFIED A SCHEMA?

ERRM..HOW DO WE ACCESS COLUMNS IF WE HAVE NOT SPECIFIED A SCHEMA?

USING `$(column_number)`

`$0` → FIELD 1

`$1` → FIELD 2

PIG IS OMNIVOROUS

IT WILL CONSUME ANYTHING

IF A SCHEMA IS GIVEN, PIG WILL MAKE USE OF IT

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA

GUESSING ALONG THE WAY

IF A SCHEMA IS NOT GIVEN, PIG WILL STILL PROCESS THE DATA

GUESSING ALONG THE WAY

CONSIDER THE FOLLOWING LOADING COMMAND

```
grunt> orders = load 'order_data' using PigStorage(',');
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

HERE IS THE EQUIVALENT OF A SQL-SELECT COMMAND

```
grunt> relation_1 = foreach orders generate $1,$2*10.0,$3/100;
```

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
grunt> relation_1 = Foreach orders generate $1,$2*10.0,$3/100;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THIS EXPRESSION (`$2*10.0`) WILL HELP PIG GUESS THE DATA TYPE OF FIELD ORDER_QUANTITY

`$2*10.0`

PIG WILL ASSUME THIS IS A DOUBLE AS IT IS MULTIPLIED WITH A DOUBLE

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
grunt> relation_1 = Foreach orders generate $1,$2*10.0,$3/100;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THIS EXPRESSION (`$3/100`) WILL HELP PIG GUESS THE DATA TYPE OF FIELD ORDER_VALUE

`$3/100`

PIG WILL ASSUME THIS IS AN INTEGER AS IT IS OPERATED ON BY AN INTEGER

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

SIMILARLY THESE COMMANDS WILL ALSO HELP PIG GUESS DATA TYPES OF FIELDS

```
grunt> relation_2 = filter orders by $0=='OD01';
```

THE FIRST FIELD IS A CHARARRAY AS IT IS COMPARED WITH A STRING

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

SIMILARLY THESE COMMANDS WILL ALSO HELP PIG GUESS DATA TYPES OF FIELDS

```
grunt> relation_2 = filter orders by $1 > 1 ;
```

FIELD IS AN INTEGER

GUESSING ALONG THE WAY

```
grunt> orders = load 'order_data' using PigStorage(',');
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> relation_2 = filter orders by $0==‘OD01’;

grunt> relation_2 = filter orders by $1 > 1 ;
```

FILTER IS SIMILAR TO THE WHERE CLAUSE IN SQL

GUESSING ALONG THE WAY

PIG CAN'T ALWAYS MAKE INTELLIGENT GUESSES ABOUT DATATYPES OF FIELDS

IN SUCH CASES, PIG WILL ASSUME THE DATATYPE TO BE BYTEARRAY

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET

```
data_set_1 = pig latin command to fetch data from some file;
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH NEW DATASET IS CALLED RELATION

PIG LATIN IS A DATAFLOW LANGUAGE

EACH PROCESSING STEP RESULTS IN A NEW DATA SET

```
data_set_1 = pig latin command to fetch data from some file
data_set_2 = pig latin command ON data_set_1;
data_set_3 = pig latin command ON data_set_2;
data_set_4 = pig latin command ON data_set_3;
data_set_5 = pig latin command ON data_set_4;
```

EACH OF THIS NEW DATASET IS CALLED RELATION

```
data_set_2 = pig latin command ON data_set_1;

data_set_3 = pig latin command ON data_set_2;

data_set_4 = pig latin command ON data_set_3;

data_set_5 = pig latin command ON data_set_4;
```

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

```
data_set_2 = pig latin command ON data_set_1;

data_set_3 = pig latin command ON data_set_2;

data_set_4 = pig latin command ON data_set_3;

data_set_5 = pig latin command ON data_set_4;
```

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

LET US TALK ABOUT DATA DUMPING FIRST

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

THE COMMAND THAT IS USED TO PUT DATA ON SCREEN IS 'DUMP'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> dump orders;
```

PRINTS THE RELATION ORDERS TO THE SCREEN

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into ‘orders’;
```

BY DEFAULT, THE COMMAND STORES THE RELATION ORDERS TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into 'orders';
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

IN A TAB DELIMITED FILE IN A FOLDER WITH NAME 'orders'

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into 'orders';
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

IN A TAB DELIMITED FILE IN A FOLDER WITH NAME 'orders'

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into 'orders';
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

IN A TAB-DELIMITED FILE NAMED orders

YOU CAN SPECIFY A DIFFERENT DIRECTORY

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into ‘/users/new_user/orders’;
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

IN A TAB DELIMITED FILE IN A DIRECTORY WITH NAME 'orders'

YOU CAN SPECIFY A DIFFERENT DIRECTORY

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt>  orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into ‘orders’;
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

IN A TAB SEPARATED FILE NAMED 'orders'

YOU CAN USE DIFFERENT STORE FUNCTION

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt>  orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into ‘orders’ using HBaseStorage();
```

BY DEFAULT, THE COMMAND STORES THE RELATION orders TO YOUR HOME DIRECTORY ON HDFS USING PIGSTORAGE

YOU CAN USE HBaseStorage IF YOU ARE STORING DATA IN HBASE

THE COMMAND THAT IS USED TO PUT DATA INTO A FILE IS CALLED 'STORE'

```
grunt> orders = load 'order_data' using PigStorage(‘,');
grunt> store orders into ‘orders’ using PigStorage(‘,’);
```

YOU CAN STORE DATA INTO COMMA SEPARATED TEXT DATA USING PIGSTORAGE

IN A TAB DELIMITED FILE IN A FOLDER WITH NAME 'orders'

```
data_set_2 = pig latin command ON data_set_1;

data_set_3 = pig latin command ON data_set_2;

data_set_4 = pig latin command ON data_set_3;

data_set_5 = pig latin command ON data_set_4;
```

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

LET'S TALK ABOUT PIG'S DATA TRANSFORMATIONS