UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

BIG DATA
Analytics & Management

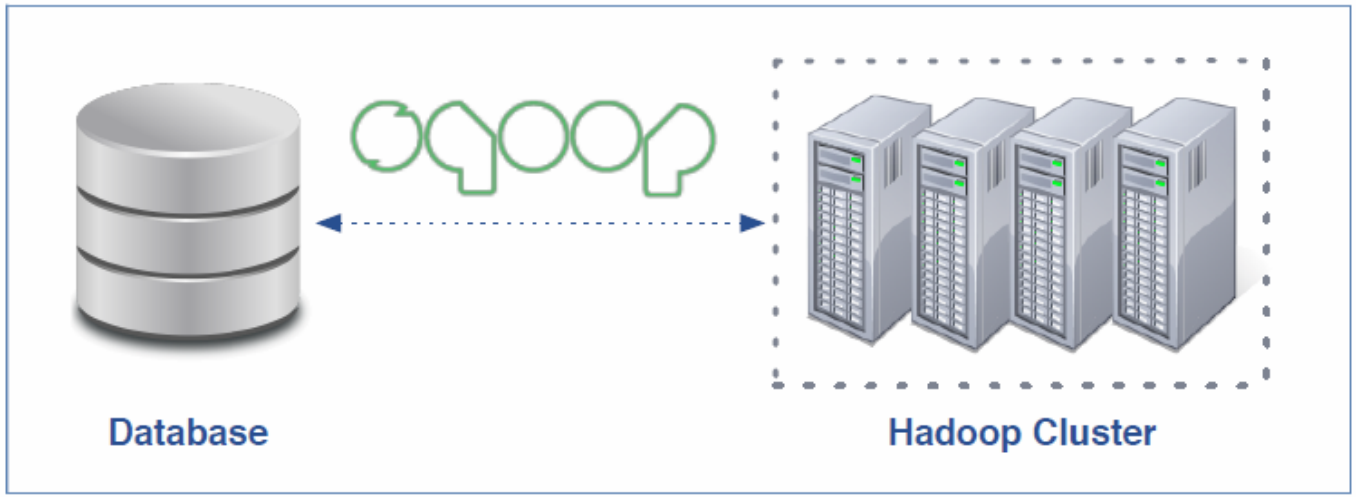
Lecture 7 (03/12, 03/14): Pig and Sqoop

Decisions, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2018

K. Zhang (pic source: mapr.com/blog)

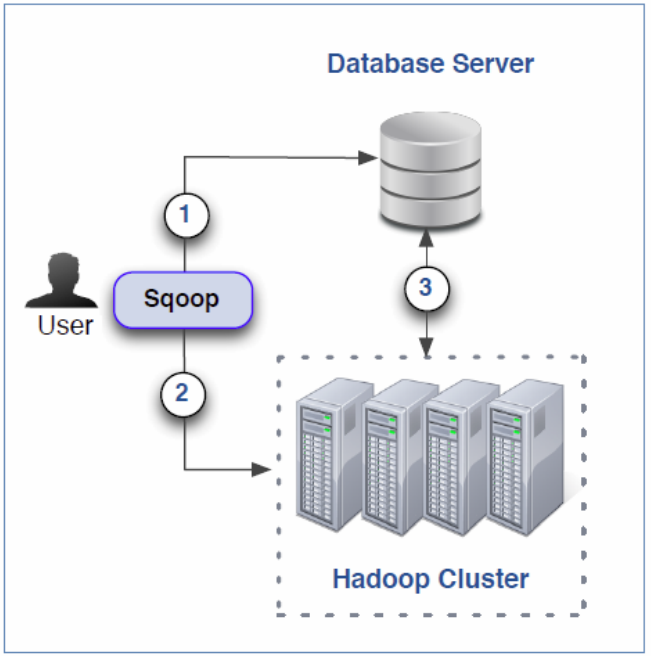
What is Apache Sqoop?

- **Open source Apache project originally developed by Cloudera**
  - The name is a contraction of "SQL-to-Hadoop"
- **Sqoop exchanges data between a database and HDFS**
  - Can import all tables, a single table, or a partial table into HDFS
  - Data can be imported a variety of formats
  - Sqoop can also export data from HDFS to a database



How Does Sqoop Work?

- **Sqoop is a client-side application that imports data using Hadoop MapReduce**
- **A basic import involves three steps orchestrated by Sqoop**
  1. Examine table details
  2. Create and submit job to cluster
  3. Fetch records from table and write this data to HDFS



Basic Syntax

- **Sqoop is a command-line utility with several subcommands, called *tools***
  - There are tools for import, export, listing database contents, and more
  - Run **`sqoop help`** to see a list of all tools
  - Run **`sqoop help`** ***`tool-name`*** for help on using a specific tool
- **Basic syntax of a Sqoop invocation**

```
$ sqoop tool-name [tool-options]
```

- **This command will list all tables in the `loudacre` database in MySQL**

```
$ sqoop list-tables \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser \
    --password pw
```


Overview of the Import Process

- **Imports are performed using Hadoop MapReduce jobs**
- **Sqoop begins by examining the table to be imported**
  - Determines the primary key, if possible
  - Runs a *boundary query* to see how many records will be imported
  - Divides result of boundary query by the number of tasks (mappers)
    - Uses this to configure tasks so that they will have equal loads
- **Sqoop also generates a Java source file for each table being imported**
  - It compiles and uses this during the import process
  - The file remains after import, but can be safely deleted

Importing an Entire Database with Sqoop

- **The `import-all-tables` tool imports an entire database**
  - Stored as comma-delimited files
  - Default base location is your HDFS home directory
  - Data will be in subdirectories corresponding to name of each table

```
$ sqoop import-all-tables \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw
```

- **Use the `--warehouse-dir` option to specify a different base directory**

```
$ sqoop import-all-tables \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    --warehouse-dir /loudacre
```


Importing a Single Table with Sqoop

- The `import` tool imports a single table
- This example imports the `accounts` table
  - It stores the data in HDFS as comma-delimited fields

```
$ sqoop import --table accounts \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw
```

- This variation writes tab-delimited fields instead

```
$ sqoop import --table accounts \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    --fields-terminated-by "\t"
```


Incremental Imports (1)

- **What if records have changed since last import?**
  - Could re-import all records, but this is inefficient
- **Sqoop's `incremental lastmodified` mode imports new and modified records**
  - Based on a timestamp in a specified column
  - You must ensure timestamps are updated when records are added or changed in the database

```
$ sqoop import --table invoices \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    --incremental lastmodified \
    --check-column mod_dt \
    --last-value '2015-09-30 16:00:00'
```


Incremental Imports (2)

- **Or use Sqoop's `incremental append` mode to import only *new* records**
  - Based on value of last record in specified column

```
$ sqoop import --table invoices \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    --incremental append \
    --check-column id \
    --last-value 9478306
```


Exporting Data from Hadoop to RDBMS with Sqoop

- **Sqoop's `import` tool pulls records from an RDBMS into HDFS**
- **It is sometimes necessary to *push* data in HDFS back to an RDBMS**
  - Good solution when you must do batch processing on large data sets
  - Export results to a relational database for access by other systems
- **Sqoop supports this via the `export` tool**
  - The RDBMS table must already exist prior to export

```
$ sqoop export \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    --export-dir /loudacre/recommender_output \
    --update-mode allowinsert \
    --table product_recommendations
```

Controlling Parallelism

- **By default, Sqoop typically imports data using four parallel tasks (called mappers)**
  - Increasing the number of tasks might improve import speed
  - Caution: Each task adds load to your database server
- **You can *influence* the number of tasks using the `-m` option**
  - Sqoop views this only as a hint and might not honor it

```
$ sqoop import --table accounts \
    --connect jdbc:mysql://dbhost/loudacre \
    --username dbuser --password pw \
    -m 8
```

- **Sqoop assumes all tables have an evenly-distributed numeric primary key**
  - Sqoop uses this column to divide work among the tasks
  - You can use a different column with the `--split-by` option

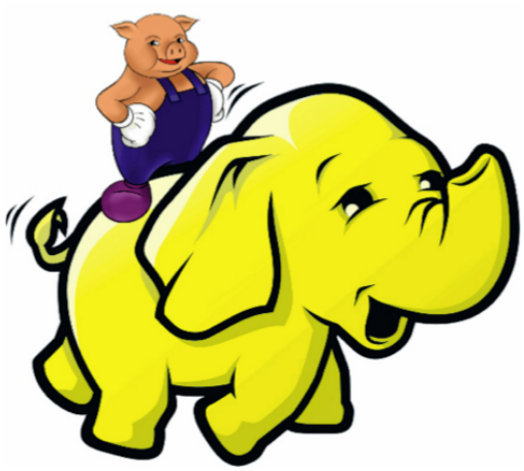
Sqoop lab

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Salesforce Hacker

03 NOVEMBER 2014

Hadoop and Pig come to the Salesforce Platform with Data Pipelines



Event Log Files is big - really, really big. This isn't your everyday CRM data where you may have hundreds of thousands of records or even a few million here and there. One organization I work with does approximately twenty million rows of event data per day using Event Log Files. That's approximately 600 million rows per month or 3.6 billion every half year.

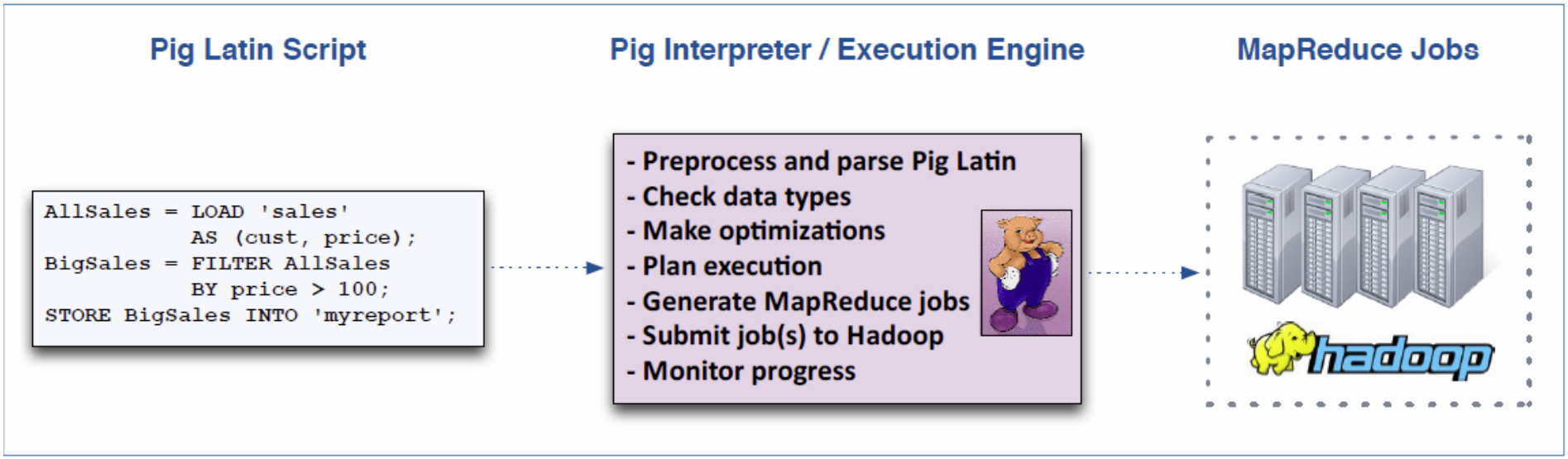
Because the size of the data does matter, we need tools that can orchestrate and process this data for a variety of use cases. For instance, one best practice when working with Event Log Files is to de-normalize Ids into Name fields. Rather than

Apache Pig Overview

- **Apache Pig is a platform for data analysis and processing on Hadoop**
  - It offers an alternative to writing MapReduce code directly
- **Originally developed as a research project at Yahoo**
  - Goals: flexibility, productivity, and maintainability
  - Now an open-source Apache project

The Anatomy of Pig

- **Main components of Pig**
  - The data flow language (Pig Latin)
  - The interactive shell where you can type Pig Latin statements (Grunt)
  - The Pig interpreter and execution engine

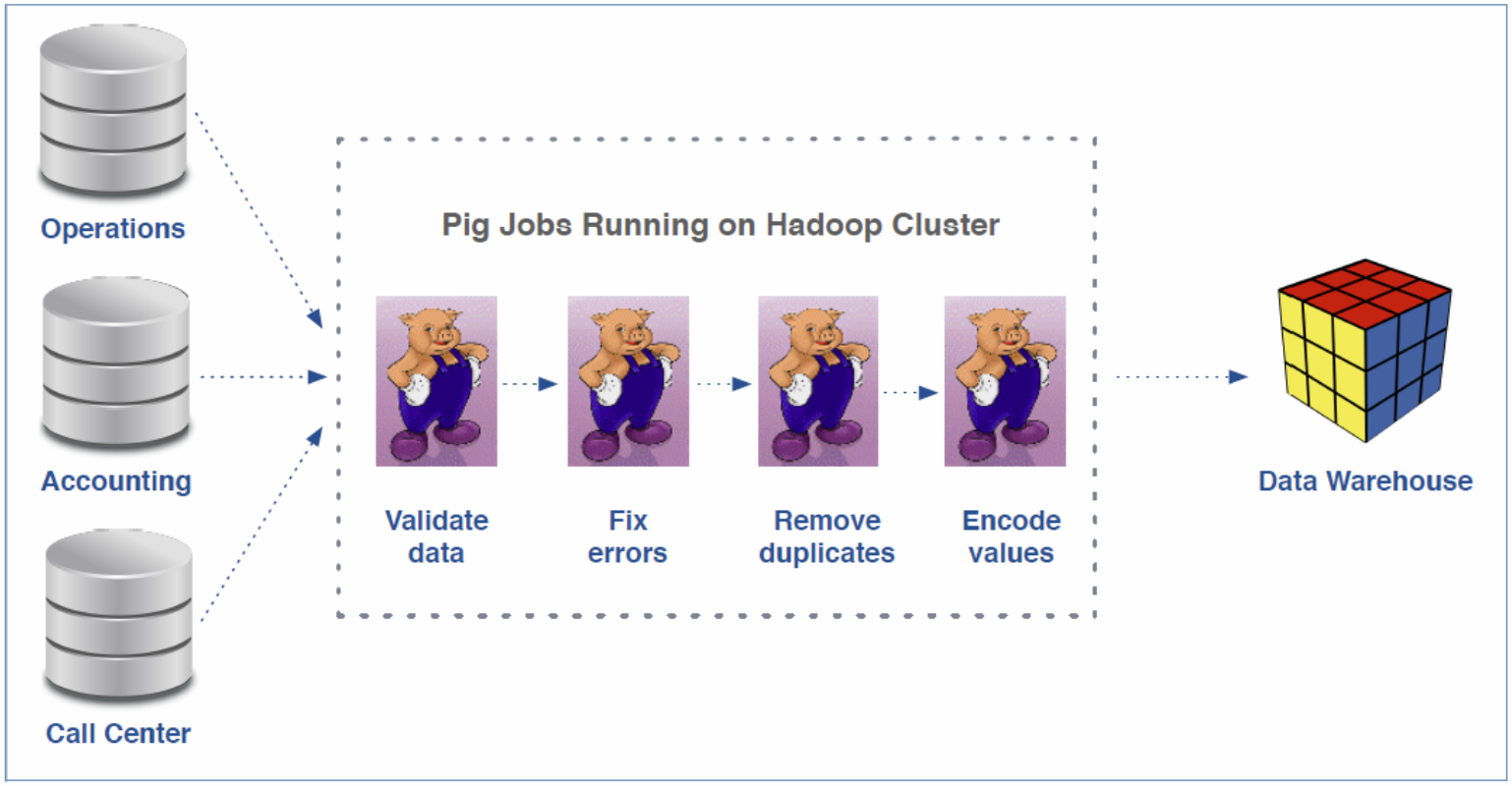


How Are Organizations Using Pig?

- **Many organizations use Pig for data analysis**
  - Finding relevant records in a massive data set
  - Querying multiple data sets
  - Calculating values from input data
- **Pig is also frequently used for data processing**
  - Reorganizing an existing data set
  - Joining data from multiple sources to produce a new data set

Use Case: ETL Processing

- **Pig is also widely used for Extract, Transform, and Load (ETL) processing**



Pig lab (page 7-18)

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Use Pig for ETL processing
  - ❑Local mode
  - ❑Then launch to cluster
- Analyze Ad campaign data with Pig
  - ❑Low cost sites
  - ❑High cost keywords

Go through the corresponding slides first

- Pig-Introduction.pdf
  - ❑Everything needed to complete the Pig Lab
- (optional): Pig-AdvancedFunctions.pdf
  - ❑More advanced functions of Pig
    - Combine, join, analyze sales data
  - ❑Answer more complex question
    - Is the ad campaign effective?

What we will cover next

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- How to run Pig
- Pig Latin syntax
- Loading data, output, etc
- Pig Data concepts
  - Fields, Tuples, Bag, Relation
- Commonly used operations and functions

Using Pig Interactively

- **You can use Pig interactively, via the Grunt shell**
  - Pig interprets each Pig Latin statement as you type it
  - Execution is delayed until output is required
  - Very useful for ad hoc data inspection
- **Example of how to start, use, and exit Grunt**

```
$ pig
grunt> allsales = LOAD 'sales' AS (name, price);
grunt> bigsales = FILTER allsales BY price > 100;
grunt> STORE bigsales INTO 'myreport';
grunt> quit;
```

Running Pig Scripts

- **A Pig script is simply Pig Latin code stored in a text file**
  - By convention, these files have the `.pig` extension
- **You can run a Pig script from within the Grunt shell via the `run` command**
  - This is useful for automation and batch execution

```
grunt> run salesreport.pig;
```

- **It is common to run a Pig script directly from the UNIX shell**

```
$ pig salesreport.pig
```

MapReduce and Local Modes

- **As described earlier, Pig turns Pig Latin into MapReduce jobs**
  - Pig submits those jobs for execution on the Hadoop cluster
- **It is also possible to run Pig in 'local mode' using the -x flag**
  - This runs MapReduce jobs on the *local machine* instead of the cluster
  - Local mode uses the local filesystem instead ofHDFS
  - Can be helpful for testing before deploying a job to production

```
$ pig -x local                      -- interactive

$ pig -x local salesreport.pig  -- batch
```

Pig Latin Syntax

Pig Latin Overview

- **Pig Latin is a *data flow* language**
  - The flow of data is expressed as a sequence of statements
- **The following is a simple Pig Latin script to load, filter, and store 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';
```

Case-Sensitivity in Pig Latin

- **Whether case is significant in Pig Latin depends on context**
- **Keywords (shown here in blue text) are *not* case-sensitive**
  - Neither are operators (such as `AND`, `OR`, or `IS NULL`)
- **Identifiers and paths (shown here in red text) are case-sensitive**
  - So are function names (such as `SUM` or `COUNT`) and constants

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

bigsales = FILTER allsales BY price > 999;

STORE bigsales INTO 'myreport';
```


Pigs are lazy and …smart

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Optimize your codes
  - Ex1: define a field but never use it ➔ Pig won't bother to load data for that field
  - Ex2: step 1, step 2, step 3 ➔ Pig might change it to step1, step3, step 2 if that is more efficient.
- Will not do anything until output is required
  - DUMP
  - STORE
- Now you understand why LOAD data in Pig seems so amazingly fast
  - Even for terabytes of data
  - That's because it only caches the command, rather than execute it.

Basic Data Loading in Pig

- **Pig's default loading function is called `PigStorage`**
  - The name of the function is implicit when calling `LOAD`
  - `PigStorage` assumes text format with tab-separated columns
- **Consider the following file in HDFS called `sales`**
  - The two fields are separated by tab characters

```
Alice      2999
Bob        3625
Carlos     2764
```

- **This example loads data from the above file**

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

Data Sources: File and Directories

- **The previous example loads data from a file named `sales`**

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

- **Since this is not an absolute path, it is relative to your home directory**
  - Your home directory in HDFS is typically `/user/youruserid/`
  - Can also specify an absolute path (e.g., `/dept/sales/2012/q4`)
- **The path can also refer to a directory**
  - In this case, Pig will recursively load all files in that directory
  - File patterns ("globs") are also supported

```
allsales = LOAD 'sales_200[5-9]' AS (name, price);
```

Specifying Column Names During Load

- **The previous example also assigns names to each column**

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

- **Assign column names is not required**
  - This can be useful when exploring a new dataset
  - Refer to fields by position (`$0` is first, `$1` is second, `$53` is 54th, etc.)

```
allsales = LOAD 'sales';
```

Using Alternate Column Delimiters

- **You can specify an alternate delimiter as an argument to `PigStorage`**
- **This example shows how to load comma-delimited data**
  - Note that this is a single statement

```
allsales = LOAD 'sales.csv' USING PigStorage(',') AS
(name, price);
```

- **Or to load pipe-delimited data without specifying column names**

```
allsales = LOAD 'sales.txt' USING PigStorage('|');
```

List of Simple Data Types

- **There are eight data types in Pig for simple values**

Name	Description	Example Value
`int`	Whole numbers	`2013`
`long`	Large whole numbers	`5,365,214,142L`
`float`	Decimals	`3.14159F`
`double`	Very precise decimals	`3.141592653589793238 46`
`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*

* Not available in older versions of Pig

Specifying Data Types in Pig

- **Pig will do its best to determine data types based on context**
  - For example, you can calculate sales commission as `price * 0.1`
  - In this case, Pig will assume that this value is of type `double`
- **However, it is better to specify data types explicitly when possible**
  - Helps with error checking and optimizations
  - Easiest to do this upon load using the format *fieldname*`:`*type*

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

- **Choosing the right data type is important to avoid loss of precision**
- **Important: Avoid using floating point numbers to represent money!**

Viewing the Schema with DESCRIBE

- The **DESCRIBE** command shows the structure of the data, including names and types
- The following Grunt session shows an example

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

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

How Pig Handles Invalid Data

- **When encountering invalid data, Pig substitutes `NULL` for the value**
  - For example, an `int` field containing the value `Q4`
- **The `IS NULL` and `IS NOT NULL` operators test for null values**
  - Note that `NULL` is not the same as the empty string `''`
- **You can use these operators to filter out bad records**

```
hasprices = FILTER Records BY price IS NOT NULL;
```

Pig Data Concepts

Key Data Concepts in Pig

- **Relational databases have tables, rows, columns, and fields**
- **We will use the following data to illustrate Pig's equivalents**
  - Assume this data was loaded from a tab-delimited text file as before

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

Pig Data Concepts: Fields

- **A *single* element of data is called a field**
  - It corresponds to one of the eight data types seen earlier

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***
  - Fields within a tuple are ordered, but need not all be of the same type

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***
- **Tuples within a bag are unordered by default**
  - The field count and types may vary between tuples in 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 simply a bag with an assigned name (alias)**
  - Most Pig Latin statements create a new relation
- **A typical script loads one or more datasets into relations**
  - Processing creates new relations instead of modifying existing ones
  - The final result is usually also a relation, stored as output

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

Data Output in Pig

- **The command used to handle output depends on its destination**
  - `DUMP`: sends output to the screen
  - `STORE`: sends output to disk (HDFS)
- **Example of `DUMP` output, using data from the file shown earlier**
  - The parentheses and commas indicate tuples with multiple fields

```
(Alice,2999,us)
(Bob,3625,ca)
(Carlos,2764,mx)
(Dieter,1749,de)
(Étienne,2368,fr)
(Franco,5637,it)
```

Storing Data with Pig

- **The `STORE` command is used to store data to HDFS**
  - Similar to `LOAD`, but *writes* data instead of reading it
  - The output path is the name of a directory
    - The directory must not yet exist
- **As with `LOAD`, the use of PigStorage is implicit**
  - The field delimiter also has a default value (tab)

```
STORE bigsales INTO 'myreport';
```

  - You may also specify an alternate delimiter

```
STORE bigsales INTO 'myreport' USING PigStorage(',');
```

Extract and re-order columns

Field Selection in Pig Latin

- **Filtering extracts rows, but sometimes we need to extract columns**
  - This is done in Pig Latin using the FOREACH and GENERATE keywords

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

allsales

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

twofields

amount	trans_id
2999	107546
3625	107547
2764	107548
1749	107549
2368	107550
5637	107550

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

Controlling Sort Order

- **Use `ORDER . . . BY` to sort the records in a bag in ascending order**
  - Add **`DESC`** to sort in descending order instead
  - Take care to specify a schema – data type affects how data is sorted!

```
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

- **As in SQL, you can use `LIMIT` to reduce the number of output records**

```
somesales = LIMIT allsales 10;
```

- **Beware! Record ordering is random unless specified with `ORDER BY`**
  - Use `ORDER BY` and `LIMIT` together to find top-N results

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

Built-in Functions

- **These are just a sampling of Pig's many built-in functions**

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`

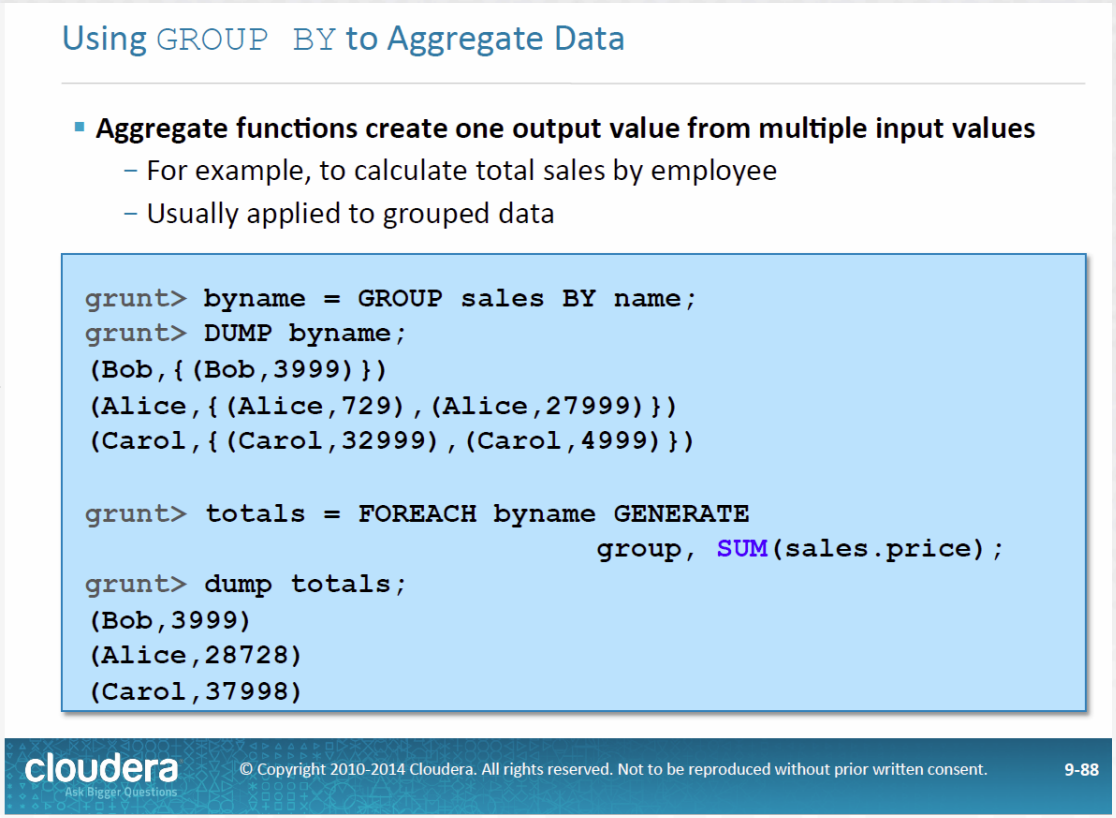
- **You can use these with the `FOREACH..GENERATE` keywords**

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

Group

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Slides: "Pig-Introduction.pdf" pages 9-85 to 9-88
- esp. 9-88



Grouping Records By a Field (1)

- **Sometimes you need to group records by a given field**
  - For example, so you can calculate commissions for each employee

```
Alice       729
Bob         3999
Alice       27999
Carol       32999
Carol       4999
```

- **Use `GROUP BY` to do this in Pig Latin**
  - The new relation has one record per unique value in the specified field

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

Grouping Records By a Field (2)

- **The new relation always contains two fields**

```
grunt> byname = GROUP sales BY name;
grunt> DESCRIBE byname;
byname: {group: chararray,sales: {(name:
chararray,price: int)}}
```

- **The first field is *literally* named `group` in all cases**
  - Contains the value from the field specified in `GROUP BY`
- **The second field is named after the relation specified in `GROUP BY`**
  - It's a bag containing one tuple for each corresponding value

Grouping Records By a Field (3)

- **The example below shows the 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**
  - For example, to calculate total sales by employee
  - Usually applied to grouped data

```
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)
```


Grouping Everything Into a Single Record

- We just saw that `GROUP BY` creates one record for each unique value
- `GROUP ALL` puts *all* data into one record

```
grunt> grouped = GROUP sales ALL;
grunt> DUMP grouped;
(all,{(Alice,729),(Bob,3999),(Alice,27999),
(Carol,32999),(Carol,4999)})
```

Removing Nesting in Data

- **Some operations in Pig, like grouping, produce nested data structures**

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

- **Grouping can be useful to supply data to aggregate functions**
- **However, sometimes you want to work with a "flat" data structure**
  - The `FLATTEN` operator removes a level of nesting in data

An Example of FLATTEN

- The following shows the nested data and what **FLATTEN** does to it

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

grunt> flat = FOREACH byname GENERATE group,
       FLATTEN(sales.price);
grunt> DUMP flat;
(Bob,3999)
(Alice,729)
(Alice,27999)
(Carol,32999)
(Carol,4999)
```

Hope you are now a Pig-Master

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS