cloudera®
Ask Bigger Questions

Choosing the Best Tool for the Job

Chapter 17

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ Querying With Impala and Hive ▪ Impala and Hive Data Management ▪ Data Storage and Performance	Introduction to Impala and Hive
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ **Choosing the Best Tool for the Job** ▪ Conclusion	**Course Conclusion**

Choosing the Best Tool for the Job

In this chapter, you will learn

- **How MapReduce, Pig, Hive, Impala, and RDBMSs compare to one another**
- **Why a workflow might involve several different tools**
- **How to select the best tool for a given job**

Chapter Topics

Choosing the Best Tool for the Job | **Course Conclusion**

- **Comparing MapReduce, Pig, Hive, Impala, and Relational Databases**
- Which to Choose?
- Conclusion
- Hands-On Exercise: Analyzing Abandoned Carts

Recap of Data Analysis/Processing Tools

- **MapReduce**
  - Low-level processing and analysis
- **Pig**
  - Procedural data flow language executed using MapReduce
- **Hive**
  - SQL-based queries executed using MapReduce
- **Impala**
  - High-performance SQL-based queries using a custom execution engine

Comparing Pig, Hive, and Impala

Feature	Pig	Hive	Impala
SQL-based query language	No	Yes	Yes
Optional schema and metastore	Yes	No	No
User-defined functions (UDFs)	Yes	Yes	Yes
Process data with external scripts	Yes	Yes	No
Extensible file format support	Yes	Yes	No
Complex data types	Yes	Yes	No
Query latency	High	High	Low
Built-in data partitioning	No	Yes	Yes
Accessible via ODBC / JDBC	No	Yes	Yes

Do These Replace an RDBMS?

- **Probably not if the RDBMS is used for its intended purpose**
- **Relational databases are optimized for**
  - Relatively small amounts of data
  - Immediate results
  - In-place modification of data (`UPDATE` and `DELETE`)
- **Pig, Hive, and Impala are optimized for**
  - Large amounts of read-only data
  - Extensive scalability at low cost
- **Pig and Hive are better suited for batch processing**
  - Impala and RDBMSs are better for interactive use

Comparing an RDBMS to Hive and Impala

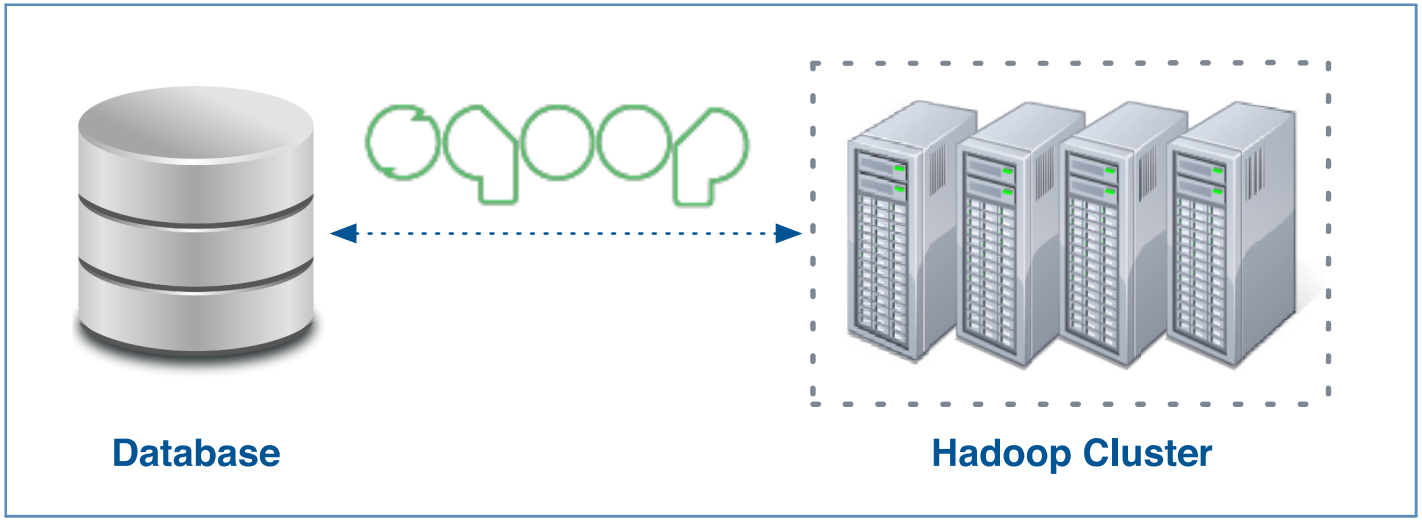
Feature	RDBMS	Hive	Impala
Insert individual records	Yes	No	Yes
Update and delete records	Yes	No	No
Transactions	Yes	No	No
Role-based authorization	Yes	Yes (Sentry)	Yes (Sentry)
Stored procedures	Yes	No	No
Index support	Extensive	Limited	None
Latency	Very low	High	Low
Data size	Terabytes	Petabytes	Petabytes
Complex data types	No	Yes	No
Storage cost	Very high	Very low	Very low

Hive Features Currently Unsupported in Impala

- **Impala does not currently support some features in Hive**
- **Examples**
  - Complex data types (`ARRAY`, `MAP`, or `STRUCT`)
  - `BINARY` data type
  - External transformations
  - Custom SerDes
  - Indexing
  - Bucketing and table sampling
- **Many of these are being considered for future releases**

Recap: Apache Sqoop

- **Sqoop helps you integrate Hadoop tools with relational databases**
- **It exchanges data between RDBMSs and Hadoop**
  - Can import all tables, a single table, or a portion of a table into HDFS
  - Supports incremental imports
  - Can also export data from HDFS back to the database



Chapter Topics

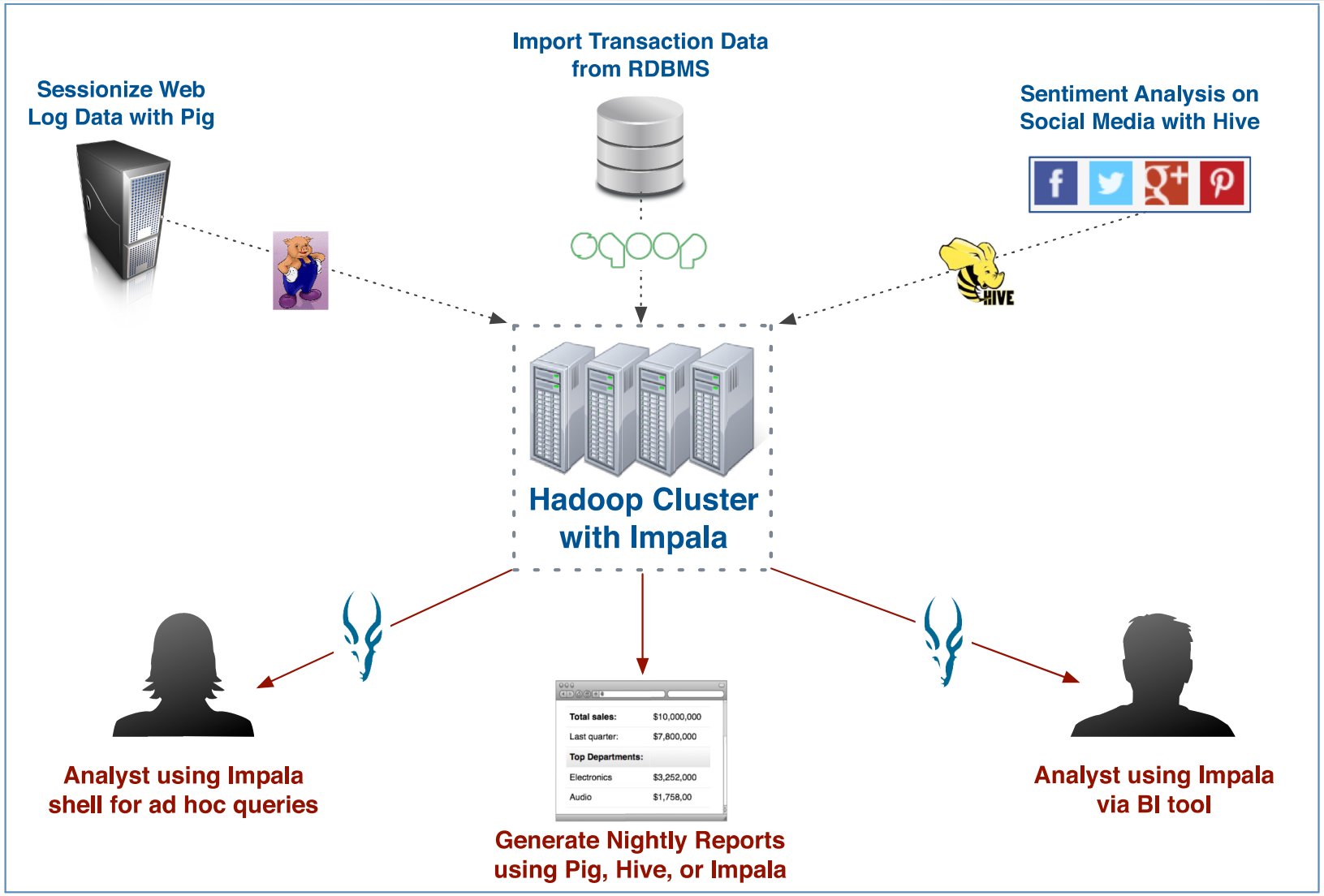
Choosing the Best Tool for the Job | **Course Conclusion**

- Comparing MapReduce, Pig, Hive, Impala, and Relational Databases
- **Which to Choose?**
- Conclusion
- Hands-On Exercise: Analyzing Abandoned Carts

Which to Choose?

- **Choose the best one for a given task**
  - Mix and match them as needed
- **MapReduce and Spark**
  - Low-level approach offers great flexibility
  - More time-consuming and error-prone to write
  - Best when control matters more than productivity
- **Pig, Hive, and Impala offer more productivity**
  - Faster to write, test, and deploy than MapReduce
  - Better choice for most analysis and processing tasks

Analysis Workflow Example



Chapter Topics

Choosing the Best Tool for the Job | **Course Conclusion**

- Comparing MapReduce, Pig, Hive, Impala, and Relational Databases
- Which to Choose?
- **Conclusion**
- Hands-On Exercise: Analyzing Abandoned Carts

Essential Points

- **You have learned about several tools for data analysis**
  - Each is better at some tasks than others
  - Choose the best one for a given job
  - Workflows may involve exchanging data between them
- **Selection criteria include scale, speed, control, and productivity**
  - MapReduce and Spark offer control at the cost of productivity
  - Pig and Hive offer productivity but not necessarily speed
  - Relational databases offer speed but not scalability
  - Impala offers scalability and speed but less control

Chapter Topics

Choosing the Best Tool for the Job | **Course Conclusion**

- Comparing MapReduce, Pig, Hive, Impala, and Relational Databases
- Which to Choose?
- Conclusion
- **Hands-On Exercise: Analyzing Abandoned Carts**

Hands-On Exercise: Analyzing Abandoned Carts

- **In this Hands-On Exercise, you will use your preferred tool to analyze data about abandoned orders in order to determine if a Free Shipping promotion would be profitable**
- **Please refer to the Hands-On Exercise Manual for instructions**

cloudera®
Ask Bigger Questions

Conclusion

Chapter 18

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ Querying With Impala and Hive ▪ Impala and Hive Data Management ▪ Data Storage and Performance	Introduction to Impala and Hive
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ **Conclusion**	**Course Conclusion**

Conclusion (1)

During this course, you have learned

- **The purpose of Hadoop and its related tools**
- **The features that Pig, Hive, and Impala offer for data acquisition, storage, and analysis**
- **How to identify typical use cases for large-scale data analysis**
- **How to load data from relational databases and other sources**
- **How to manage data in HDFS and export it for use with other systems**
- **How Pig, Hive, and Impala improve productivity for typical analysis tasks**
- **The language syntax and data formats supported by these tools**

Conclusion (2)

- How to design and execute queries on data stored in HDFS
- How to join diverse datasets to gain valuable business insight
- How to analyze structured, semi-structured, and unstructured data
- How Hive and Impala can be extended with custom functions and scripts
- How to store and query data for better performance
- How to determine which tool is the best choice for a given task

Which Course to Take Next?

Cloudera offers a range of training courses for you and your team

- **For developers**
  - *Cloudera Developer Training for Apache Hadoop*
  - *Cloudera Developer Training for Apache Spark*
  - *Designing and Building Big Data Applications*
  - *Cloudera Training for Apache HBase*
- **For system administrators**
  - *Cloudera Administrator Training for Apache Hadoop*
- **For data scientists**
  - *Introduction to Data Science: Building Recommender Systems*
- **For architects, managers, CIOs, and CTOs**
  - *Cloudera Essentials for Apache Hadoop*

cloudera®
Ask Bigger Questions

Extending Pig

Appendix A

Course Chapters

▪ Introduction ▪ Hadoop Fundamentals	Course Introduction
▪ Introduction to Pig ▪ Basic Data Analysis with Pig ▪ Processing Complex Data with Pig ▪ Multi-Dataset Operations with Pig ▪ Pig Troubleshooting and Optimization	Data ETL and Analysis With Pig
▪ Introduction to Impala and Hive ▪ Querying With Impala and Hive ▪ Impala and Hive Data Management ▪ Data Storage and Performance	Introduction to Impala and Hive
▪ Relational Data Analysis With Impala and Hive ▪ Working with Impala ▪ Analyzing Text and Complex Data with Hive ▪ Hive Optimization ▪ Extending Hive	Data Analysis With Impala and Hive
▪ Choosing the Best Tool for the Job ▪ Conclusion ▪ **Appendix A: Extending Pig**	**Course Conclusion**

Extending Pig

In this appendix, you will learn

- **How to use parameters in your Pig Latin to increase its flexibility**
- **How to define and invoke macros to improve the reusability of your code**
- **How to call user-defined functions from your code**
- **How to write user-defined functions in Python**
- **How to process data with external scripts**

Chapter Topics

Extending Pig

- **Adding Flexibility with Parameters**
- Macros and Imports
- UDFs
- Contributed Functions
- Using Other Languages to Process Data with Pig
- Conclusion
- Hands-On Exercise: Extending Pig with Streaming and UDFs

The Need for Parameters (1)

- **Some processing is very repetitive**
  - For example, creating sales reports

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

bigsales_alice = FILTER bigsales BY name == 'Alice';
STORE bigsales_alice INTO 'Alice';
```

The Need for Parameters (2)

- **You may need to change the script slightly for each run**
  - For example, to modify the paths or filter criteria

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

bigsales_alice = FILTER bigsales BY name == 'Alice';
STORE bigsales_alice INTO 'Alice';
```

Making the Script More Flexible with Parameters

- **Instead of hardcoding values, Pig allows you to use parameters**
  - These are replaced with specified values at runtime

```
allsales = LOAD '$INPUT' AS (name, price);
bigsales = FILTER allsales BY price > $MINPRICE;

bigsales_name = FILTER bigsales BY name == '$NAME';
STORE bigsales_name INTO '$NAME';
```

  - Then specify the values on the command line

```
$ pig -p INPUT=sales -p MINPRICE=999 \
  -p NAME='Jo Anne' reporter.pig
```

Two Tricks for Specifying Parameter Values

- **You can also specify parameter values in a text file**
  - An alternative to typing each one on the command line

```
INPUT=sales
MINPRICE=999
# comments look like this
NAME='Alice'
```

  - Use `-m filename` option to tell Pig which file contains the values

- **Parameter values can be defined with the output of a shell command**
  - For example, to set `MONTH` to the current month:

```
MONTH=`date +'%m'`    # returns 03 for March, 05 for May
```

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- **Macros and Imports**
- UDFs
- Contributed Functions
- Using Other Languages to Process Data with Pig
- Conclusion
- Hands-On Exercise: Extending Pig with Streaming and UDFs

The Need for Macros

- **Parameters simplify repetitive code by allowing you to pass in values**
  - But sometimes you would like to reuse the actual code too

```
allsales = LOAD 'sales' AS (name, price);
byperson = FILTER allsales BY name == 'Alice';

SPLIT byperson INTO low IF price < 1000,
      high IF price >= 1000;

amt1 = FOREACH low GENERATE name, price * 0.07 AS amount;
amt2 = FOREACH high GENERATE name, price * 0.12 AS amount;

commissions = UNION amt1, amt2;
grpd = GROUP commissions BY name;

out = FOREACH grpd GENERATE SUM(commissions.amount) AS total;
```

Defining a Macro in Pig Latin

- **Macros allow you to define a block of code to reuse easily**
  - Similar (but not identical) to a function in a programming language

```
define calc_commission (NAME, SPLIT_AMT, LOW_PCT, HIGH_PCT)
returns result {
  allsales = LOAD 'sales' AS (name, price);
  byperson = FILTER allsales BY name == '$NAME';

  SPLIT byperson INTO low if price < $SPLIT_AMT,
        high IF price >= $SPLIT_AMT;

  amt1 = FOREACH low GENERATE name, price * $LOW_PCT AS amount;
  amt2 = FOREACH high GENERATE name, price * $HIGH_PCT AS amount;

  commissions = UNION amt1, amt2;
  grouped = GROUP commissions BY name;

  $result = FOREACH grouped GENERATE SUM(commissions.amount);
};
```

Invoking Macros

- **To invoke a macro, call it by name and supply values in the correct order**

```
define calc_commission (NAME, SPLIT_AMT, LOW_PCT, HIGH_PCT)
returns result {
  allsales = LOAD 'sales' AS (name, price);

  ... (other code removed for brevity) ...

  $result = FOREACH grouped GENERATE SUM(commissions.amount);
};
```

```
alice_comm = calc_commission('Alice', 1000, 0.07, 0.12);
carlos_comm = calc_commission('Carlos', 2000, 0.08, 0.14);
```

Reusing Code with Imports

- **After defining a macro, you may wish to use it in multiple scripts**
- **You can include one script within another, starting with Pig 0.9**
  - This is done with the `import` keyword and path to file being imported

```
-- We saved the macro to a file named commission_calc.pig

import 'commission_calc.pig';

alice_comm = calc_commission('Alice', 1000, 0.07, 0.12);
```

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- Macros and Imports
- **UDFs**
- Contributed Functions
- Using Other Languages to Process Data with Pig
- Conclusion
- Hands-On Exercise: Extending Pig with Streaming and UDFs

User-Defined Functions (UDFs)

- **We have covered many of Pig's built-in functions already**
- **It is also possible to define your own functions**
  - Pig allows writing UDFs in several languages

Language	Supported in Pig Versions
Java	All
Python	0.8 and later
JavaScript (experimental)	0.9 and later
Ruby (experimental)	0.10 and later
Groovy (experimental)	0.11 and later

- **In the next few slides, you will see how to use UDFs in Java, and how to write and use UDFs in Python**

Using UDFs Written in Java

- **UDFs are packaged into Java Archive (JAR) files**
- **There are only two required steps for using them**
  - Register the JAR file(s) containing the UDF and its dependencies
  - Invoke the UDF using the fully-qualified classname

```
REGISTER '/path/to/myudf.jar';
...
data = FOREACH allsales GENERATE com.example.MYFUNC(name);
```

- **You can optionally define an alias for the function**

```
REGISTER '/path/to/myudf.jar';
DEFINE FOO com.example.MYFUNC;
...
data = FOREACH allsales GENERATE FOO(name);
```

Writing UDFs in Python (1)

- **Now we will see how to write a UDF in Python**
- **The data we want to process has inconsistent phone number formats**

```
Alice     (314) 555-1212
Bob       212.555.9753
Carlos    405-555-3912
David     (202) 555.8471
```

- **We will write a Python UDF that can consistently extract the area code**

Writing UDFs in Python (2)

- **Our Python code is straightforward**
- **The only unusual thing is the optional `@outputSchema` decorator**
  - This tells Pig what data type we are returning
  - If not specified, Pig will assume `bytearray`

```
@outputSchema("areacode:chararray")
def get_area_code(phone):
   areacode = "???" # return this for unknown formats

   if len(phone) == 12:
      # XXX-YYY-ZZZZ or XXX.YYY.ZZZZ format
      areacode = phone[0:3]
   elif len(phone) == 14:
      # (XXX) YYY-ZZZZ or (XXX) YYY.ZZZZ format
      areacode = phone[1:4]

   return areacode
```

Invoking Python UDFs from Pig Latin

- **Using this UDF from our Pig Latin is also easy**
  - We saved our Python code as `phonenumber.py`
  - This Python file is in our current directory

```
REGISTER 'phonenumber.py' USING jython AS phoneudf;

names = LOAD 'names' AS (name:chararray, phone:chararray);

areacodes = FOREACH names GENERATE
            phoneudf.get_area_code(phone) AS ac;
```

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- Macros and Imports
- UDFs
- **Contributed Functions**
- Using Other Languages to Process Data with Pig
- Conclusion
- Hands-On Exercise: Extending Pig with Streaming and UDFs

Open Source UDFs

- **Pig ships with a set of community-contributed UDFs called Piggy Bank**
- **Another popular package of UDFs, called DataFu, has been open-sourced by LinkedIn**

Piggy Bank

- **Piggy Bank ships with Pig**
  - You will need to register the `piggybank.jar` file
  - The location may vary depending on source and version
  - In CDH on our VMs, it is at `/usr/lib/pig/piggybank.jar`
- **Some UDFs in Piggy Bank include (package names omitted for brevity)**

Class Name	Description
`ISOToUnix`	Converts an ISO 8601 date/time format to UNIX format
`UnixToISO`	Converts a UNIX date/time format to ISO 8601 format
`LENGTH`	Returns the number of characters in the supplied string
`HostExtractor`	Returns the host name from a URL
`DiffDate`	Returns number of days between two dates

DataFu

- **DataFu does not ship with Pig, but is part of CDH 4.1 and later**
  - You will need to register the DataFu JAR file
  - In VM, located in `/usr/lib/pig/`
  - Some UDFs in DataFu include (package names omitted for brevity)

Class Name	Description
`Quantile`	Calculates quantiles for a data set
`Median`	Calculates the median for a data set
`Sessionize`	Groups data into sessions based on a specified time window
`HaversineDistInMiles`	Calculates distance in miles between two points, given latitude and longitude

Using A Contributed UDF

- **Here is an example of using a UDF from DataFu to calculate distance**

Input data

```
37.789336    -122.401385    40.707555    -74.011679
```

Pig Latin

```
REGISTER '/usr/lib/pig/datafu-*.jar';
DEFINE DIST datafu.pig.geo.HaversineDistInMiles;

places = LOAD 'data' AS (lat1:double, lon1:double,
         lat2:double, lon2:double);

dist = FOREACH places GENERATE DIST(lat1, lon1, lat2, lon2);
DUMP dist;
```

Output data

```
(2564.207116295711)
```

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- Macros and Imports
- UDFs
- Contributed Functions
- **Using Other Languages to Process Data with Pig**
- Conclusion
- Hands-On Exercise: Extending Pig with Streaming and UDFs

Processing Data with An External Script

- **While Pig Latin is powerful, some tasks are easier in another language**
- **Pig allows you to stream data through another language for processing**
  - This is done using the `STREAM` keyword
- **Similar concept to Hadoop Streaming**
  - Data is supplied to the script on standard input as tab-delimited fields
  - Script writes results to standard output as tab-delimited fields

STREAM Example in Python (1)

- **Our example will calculate a user's age given that user's birthdate**
  - This calculation is done in a Python script named `agecalc.py`
- **Here is the corresponding Pig Latin code**
  - Backticks used to quote script name following the alias
  - Single quotes used for quoting script name within `SHIP`
  - The schema for the data produced by the script follows the `AS` keyword

```
DEFINE MYSCRIPT `agecalc.py` SHIP('agecalc.py');
users = LOAD 'data' AS (name:chararray, birthdate:chararray);

out = STREAM users THROUGH MYSCRIPT AS (name:chararray, age:int);

DUMP out;
```

STREAM Example in Python (2)

- **Python code for `agecalc.py`**

```
#!/usr/bin/env python

import sys
from datetime import datetime

for line in sys.stdin:
   line = line.strip()
   (name, birthdate) = line.split("\t")

   d1 = datetime.strptime(birthdate, '%Y-%m-%d')
   d2 = datetime.now()

   age = int((d2 - d1).days / 365)

   print "%s\t%i" % (name, age)
```

STREAM Example in Python (3)

- **The Pig script again, and the data it reads and writes**

```
DEFINE MYSCRIPT `agecalc.py` SHIP('agecalc.py');
users = LOAD 'data' AS (name:chararray, birthdate:chararray);

out = STREAM users THROUGH MYSCRIPT AS (name:chararray,
age:int);

DUMP out;
```

Input data

```
andy     1963-11-15
betty    1985-12-30
chuck    1979-02-23
debbie   1982-09-19
```

Output data

```
(andy,49)
(betty,27)
(chuck,34)
(debbie,30)
```

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- Macros and Imports
- UDFs
- Contributed Functions
- Using Other Languages to Process Data with Pig
- **Conclusion**
- Hands-On Exercise: Extending Pig with Streaming and UDFs

Essential Points

- **Pig supports several extension mechanisms**
- **Parameters and macros can help make your code more reusable**
  - And easier to maintain and share with others
- **Piggy Bank and DataFu are two examples of open source UDFs**
  - You can also write your own UDFs
- **It is also possible to embed Pig within another language**

Bibliography

The following offer more information on topics discussed in this chapter

- **Documentation on Parameter Substitution in Pig**
  - `http://tiny.cloudera.com/dac07a`
- **Documentation on Macros in Pig**
  - `http://tiny.cloudera.com/dac07b`
- **Documentation on User-Defined Functions in Pig**
  - `http://tiny.cloudera.com/dac07c`
- **Documentation on Piggy Bank**
  - `http://tiny.cloudera.com/dac07d`
- **Introducing Data Fu**
  - `http://tiny.cloudera.com/dac07e`

Chapter Topics

Extending Pig

- Adding Flexibility with Parameters
- Macros and Imports
- UDFs
- Contributed Functions
- Using Other Languages to Process Data with Pig
- Conclusion
- **Hands-On Exercise: Extending Pig with Streaming and UDFs**

Hands-On Exercise: Extending Pig with Streaming and UDFs

- **In this Hands-On Exercise, you will process data with an external script and a user-defined function.**
- **Please refer to the Hands-On Exercise Manual for instructions**