



Designing and Building Big Data Applications: Hands-On Exercises

General Notes	3
Hands-On Exercise: Natural Language Processing in MapReduce	7
Hands-On Exercise: Working with Avro Schemas	18
Hands-On Exercise: Transforming Data with Kite	22
Hands-On Exercise: Importing Customer Account Data	27
Hands-On Exercise: Collecting Web Server Logs with Flume	31
Hands-On Exercise: Collecting Application Data with Flume	36
Hands-On Exercise: Writing a Custom Flume Interceptor	40
Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie	45
Hands-On Exercise: File Type Processing with Crunch	52
Hands-On Exercise: Log Processing with Crunch	55

Hands-On Exercise: Running Queries in Hive	59
Hands-On Exercise: Using the RegexSerDe in Hive	64
Hands-On Exercise: Implementing a User-Defined Function in Hive	68
Optional Hands-On Exercise: Running Queries in Impala	71
Hands-On Exercise: Batch Indexing of Knowledge Base Articles	73
Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time.....	78
Hands-On Exercise: Building the Application UI.....	82

General Notes

Cloudera's training courses use a Virtual Machine running the CentOS Linux distribution. This VM has CDH (Cloudera's Distribution, including Apache Hadoop) installed in pseudo-distributed mode. Pseudo-Distributed mode is a method of running Hadoop whereby all Hadoop daemons run on the same machine. It is, essentially, a cluster consisting of a single machine. It works just like a larger Hadoop cluster, the only key difference (apart from speed, of course!) being that the block replication factor is set to 1, since there is only a single DataNode available.

Points to note while working in the VM

1. The VM is set to automatically log in as the user `training`. Should you log out, you can log back in as the user `training` with the password `training`.
2. Should you need it, the root password is `training`. You may be prompted for this if, for example, you want to change the keyboard layout. In general, you should not need this password since the `training` user has unlimited `sudo` privileges.
3. In some command-line steps in the exercises, you will see lines like this:

```
$ hadoop fs -put shakespeare \  
/user/training/shakespeare
```

The dollar sign (\$) at the beginning of each line indicates the Linux shell prompt. The actual prompt will include additional information (e.g., `[training@dev workspace]$`) but this is omitted from these instructions for brevity.

The backslash (\) at the end of the first line signifies that the command is not completed, and continues on the next line. You can enter the code exactly as shown (on two lines), or you can enter it on a single line. If you do the latter, you should *not* type in the backslash.

4. In order to better simulate a development cluster, the VM's hostname is set to `dev.loudacre.com` and we use this (rather than `localhost`) in all exercises.
5. If you need to log in to Archiva (the local Maven repository management system), the web interface is `http://dev.loudacre.com:4861/archiva`. The username is 'admin' and the password is 'admin1'.

Points to note during the exercises

1. For most exercises, three folders or Java packages are provided. Which you use will depend on how you would like to work on the exercises:
 - a. `stubs`: contains minimal skeleton code for the Java classes you will need to write. These are best for those with extensive Java experience.
 - b. `hints`: contains Java class stubs that include additional hints about what is required to complete the exercise. Often, these are marked with a TODO that describes what needs to be done. These are best for developers with intermediate Java experience.
 - c. `solution`: Fully implemented Java code which may be run “as-is”, or compared with your own solution.

Whenever you see a portion of a command or package name in italicized red text, it indicates that the value should be replaced with the one corresponding to the version of the exercise you implemented. For example, you might replace `solution` with `hints` in the package name below.

```
$ hadoop jar example.jar com.loudacre.solution.MyJob
```

2. As the exercises progress, and you gain more familiarity with the tools and environment, we provide fewer step-by-step instructions; as in the real world, we merely give you a requirement and it's up to you to solve the problem! You should feel free to refer to the hints or solutions provided, ask your instructor for assistance, or consult with your fellow students.
3. There are additional challenges for many of the Hands-On Exercises. If you finish the main exercise, please attempt the additional steps.
4. If you are unable to complete an exercise, we have provided a script to catch you up automatically. Each exercise has instructions for running the catch-up script, where applicable.
5. The VM has certain environment variables set. These are used in the terminal command and start with a dollar sign then some text and look like `$BDDIR`.

These are used to shorten the commands in the exercise guide and make it easier to enter a command.

If you want to see the value of an environment variable, you can use the `echo` command:

```
$ echo $BDDIR
```

Hands-On Exercise: Natural Language Processing in MapReduce

Files Used in This Exercise:

Eclipse projects:

nlp-mr

nlp-mr-bonus

Data files (local)

~/training_materials/bigdata/data/kb/

In this exercise you will use a Natural Language Processing (NLP) algorithm called 'stemming' to count the number of similar keywords found in Loudacre's Knowledge Base articles. Before this, however, you will learn how to create and import a Maven project using Eclipse.

Maven

Most of the exercises in this class use Maven to compile, test, and package code.

You will execute Maven from the command line. Try it now by running the following command in a terminal window:

```
$ mvn
```

This displays an error because Maven requires you to specify a *goal*. A goal tells Maven which tasks you want to perform, such as compiling code or running unit tests. You can also use a Maven goal to create a fully-configured Eclipse project that you can then import into the IDE.

Step 1: Maven Eclipse Configuration

Note: Skip this step if it has already been run during a previous exercise. If Eclipse is running, you must close it before proceeding.

1. Add the Maven repository to Eclipse by running this command:

```
$ mvn -Declipse.workspace=/home/training/workspace/ \
eclipse:add-maven-repo
```

If you skip this step, you will see compilation errors in Eclipse because the necessary JARs will not be found. Remember, this is a one-time step and is only required when setting up your copy of Eclipse to use Maven for the first time.

Step 2: Eclipse Project Creation

The `nlp-mr` project is the base for the MapReduce project. You will need to use Maven to create the project. Except for the specific directory paths, you will repeat exactly the same steps whenever you need to create and import Eclipse projects during later exercises.

1. Change directories to the local filesystem directory containing the `nlp-mr` project.

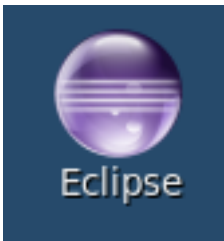
```
$ cd $BDDIR/nlp-mr/
```

2. Create the Eclipse project by running:

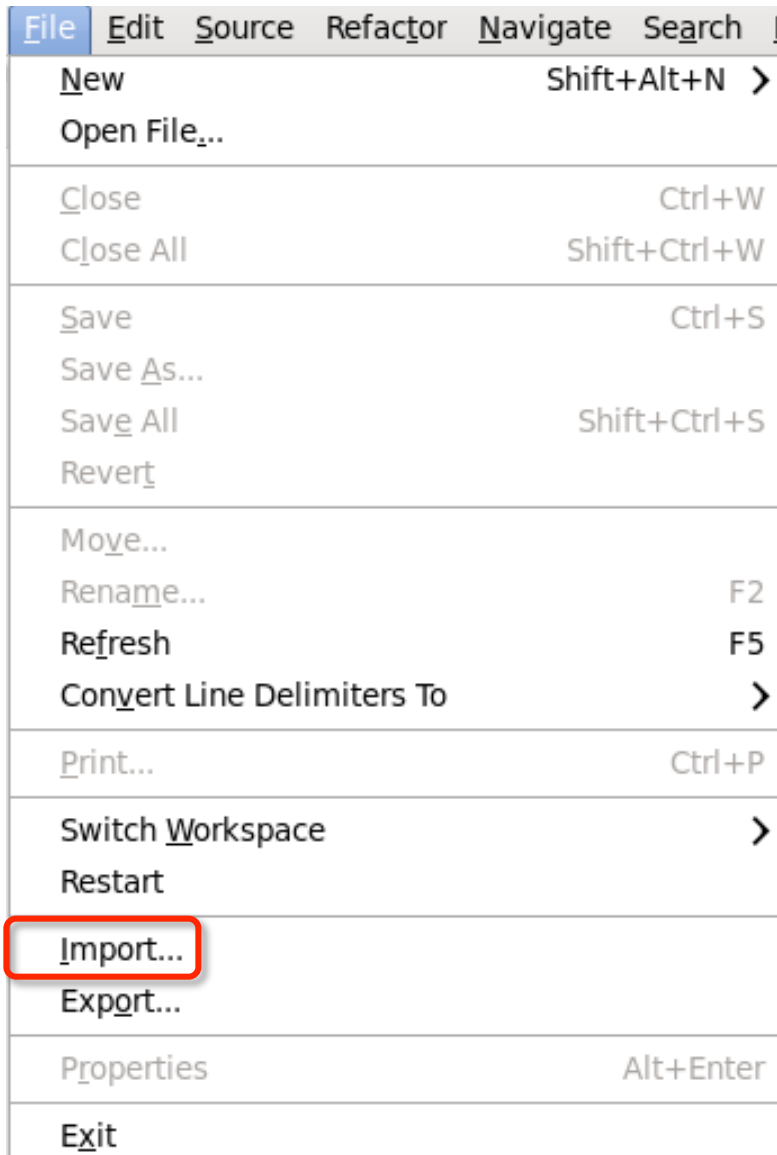
```
$ mvn eclipse:eclipse
```

This downloads the necessary JAR files and creates the directory structure and configuration files needed for you to import the project in Eclipse.

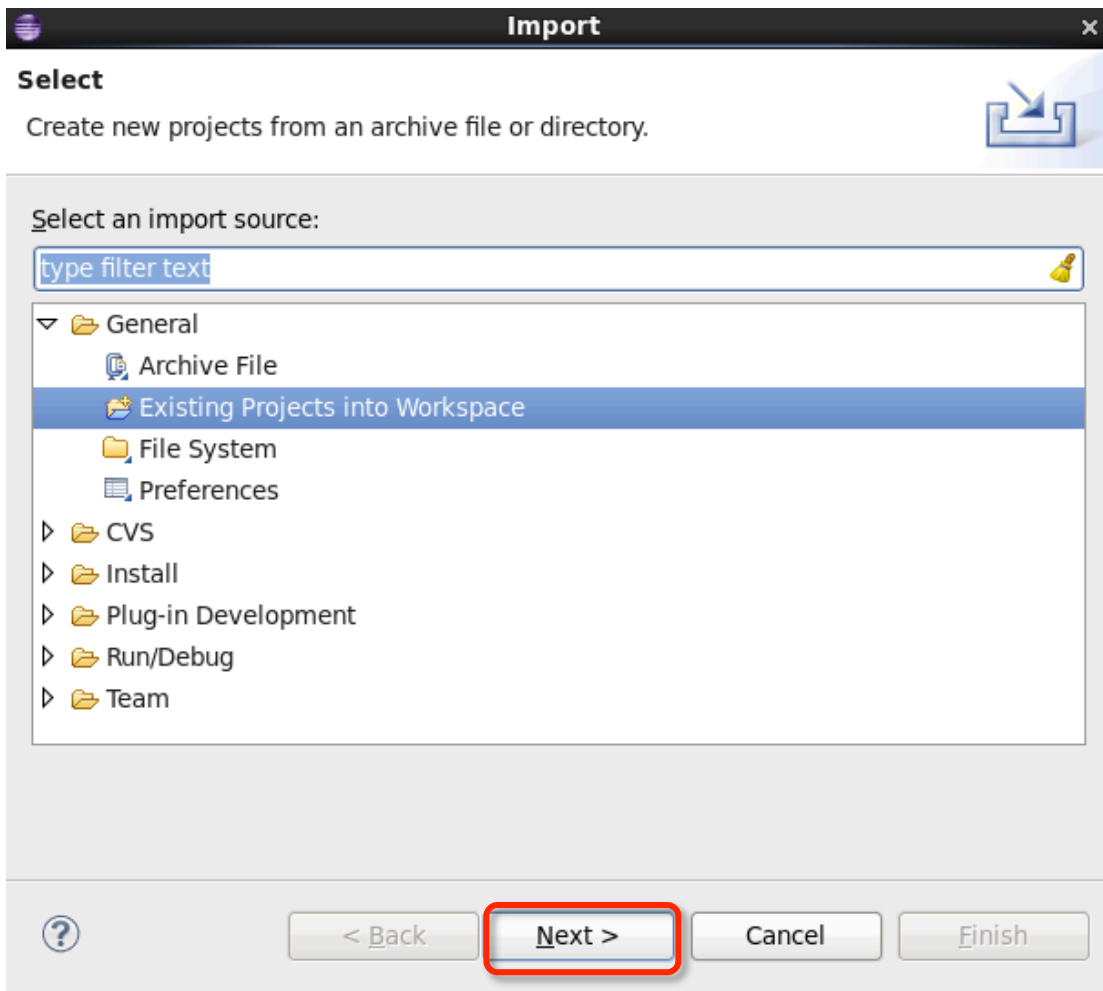
3. Start Eclipse by double clicking on its icon on the desktop.



4. Go to 'File', then 'Import'.



5. Click on 'General', then 'Existing Projects Into Workspace'.




6. Click the 'Next' button.
7. In 'Select root directory', click on 'Browse'.

Import [X]

Import Projects

Select a directory to search for existing Eclipse projects.



☒ Select root directory: [v] **Browse...**

☐ Select archive file: [v] Browse...

Projects:

[Select All](#)
[Deselect All](#)
[Refresh](#)

Options

☐ Search for nested projects

☐ Copy projects into workspace

Working sets

☐ Add project to working sets

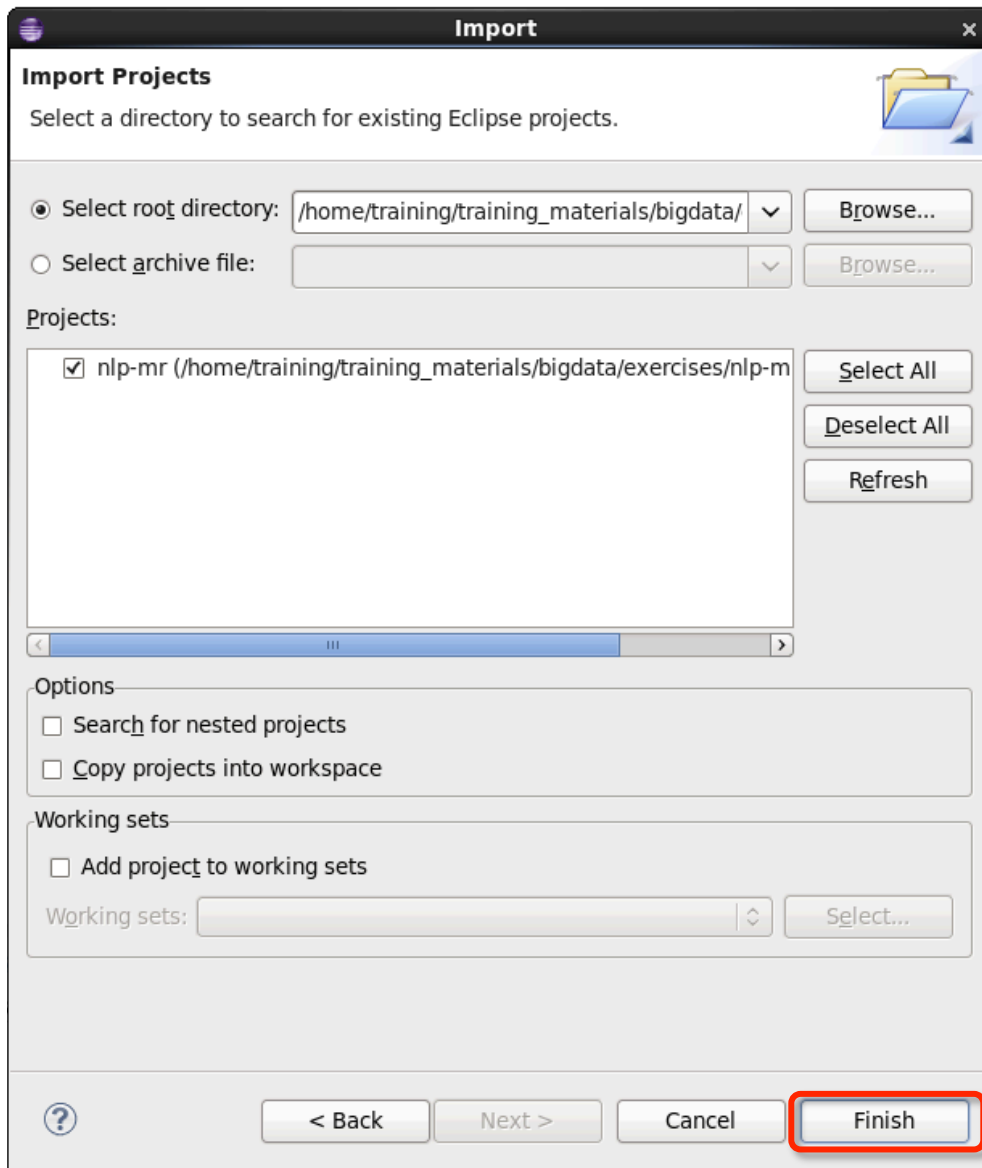
Working sets: [v] [Select...](#)

[?] < Back Next > Cancel Finish

8. Browse to:

```
/home/training/training_materials/bigdata/exercises/nlp  
-mr
```

Click the 'OK' button. The 'Projects' section will contain the `nlp-mr` project, as shown below:



9. Click on 'Finish'.

The `nlp-mr` project will be imported in to Eclipse.

Note: Any time the exercise instructions call for you to work with code in a Maven project, you should repeat these instructions in the appropriate starting directory.

Step 3: Stem Counting

You will write a MapReduce job to process all documents in the Knowledge Base directory. These files use a heading to list relevant keywords, or tags, like this:

```
<h4>Tags: changing, contact, the</h4>
```

The `KBHelper` class provides methods to help you parse and stem this content. You can use the `getTags` method to check whether a line of input contains this heading. The method returns `null` if it does not, otherwise it returns a comma-delimited list of tags. The `getWords` method takes that list and returns an array of individual words. The `getStem` method takes a single word and returns its stem.

You will edit the `StemMapper`, `StemSumReducer`, and `KnowledgeBaseDriver` classes in this project.

The Mapper needs to:

- Check each line of input for tags using the `getTags` method described above
- Split the line containing tags into individual words
- Stem each word, and then emit the stem as the key and a 1 as the value

The Reducer needs to:

- Count the number of times each stem was found in Knowledge Base articles
- Output the stem as the key and the count as the value

Step 4: Unit Testing KBHelper

You need to write two test cases for the `KBHelper` class to verify that it behaves correctly with both valid and invalid input, as described below.

1. The `getTags` method in `KBHelper` class must return `null` when given a line that does not contain tags in the format shown above. Write a test for this scenario.
2. When given a line in the format shown above, the `getTags` method must strip the HTML and `Tags :` from the String that it returns. Write a test for this scenario using:

```
<h4>Tags: changing, contact, the</h4>
```

as the input, and assert that the output is:

```
changing, contact, the
```

We encourage you to use unit tests to speed up your development. You can either write your own or use the ones provided with the sample solution. If you use those from the sample solution, you must change the package names to match your own.

Testing MapReduce with MRUnit

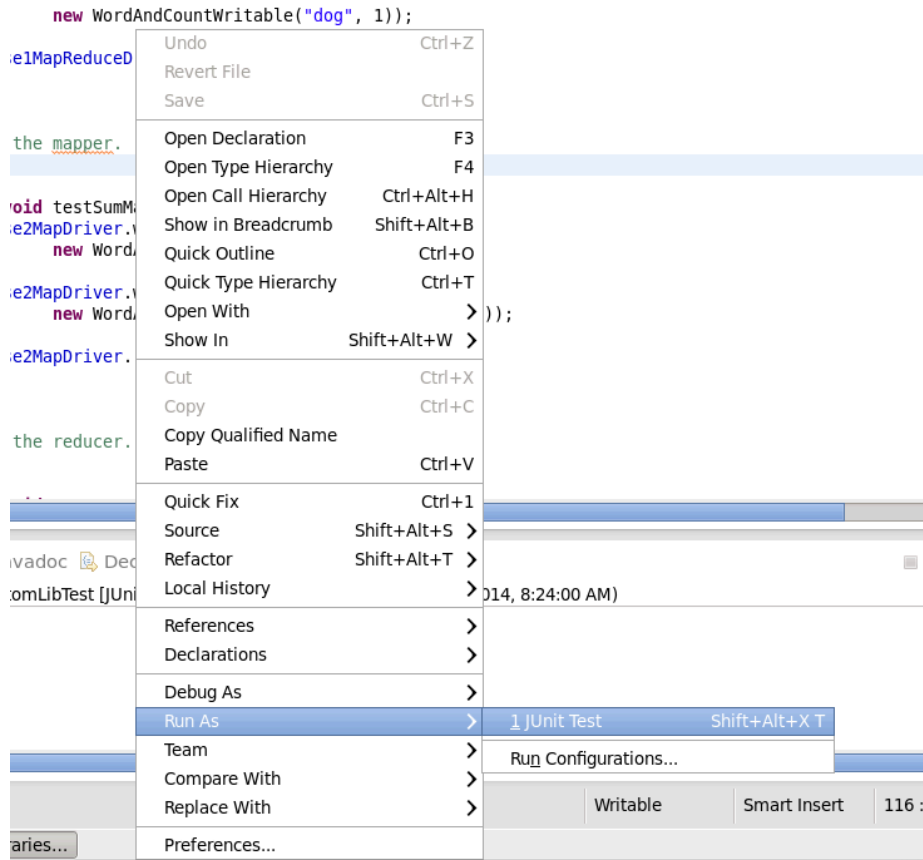
Although these unit tests validate code used in the MapReduce job, they don't actually test the `map` or `reduce` methods themselves. That can be done using a specialized unit testing library called MRUnit.

Since MRUnit is already in your project's classpath, you can add tests for your mappers and reducers if you wish. If you do not know how to use MRUnit, look at the `com.loudacre.nlpmr.StemMapReduceTest` class for an example.

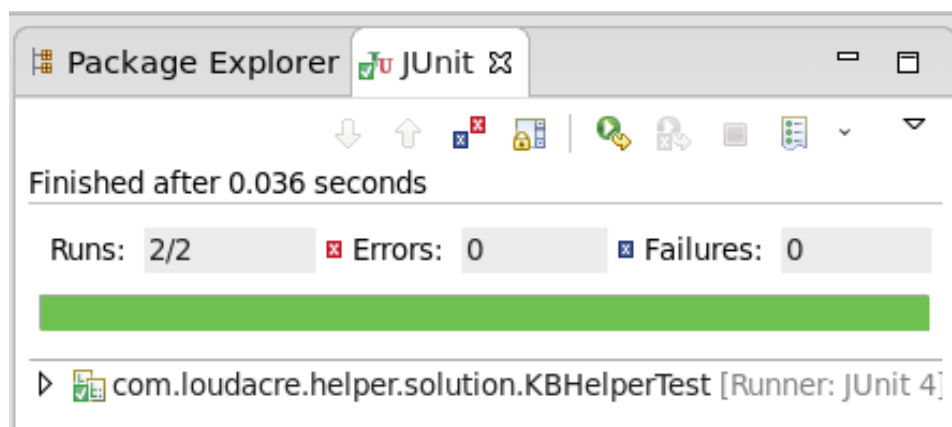
Step 5: Running Unit Tests in Eclipse

Unit tests can be run directly in Eclipse. These can even be run in debug mode to allow you step through code quickly.

1. Open the unit test class
2. Right click in the editor, click 'Run As', and then select 'JUnit Test'



3. This will display the JUnit tab, which shows the results of the test execution.



If the bar displays green, all unit tests passed. If it shows red, one or more unit tests has failed.

Running the Code

While developing the code, you can run it in Eclipse or from the command line. Once you are done writing MapReduce code, you should always test it again on a cluster. The next step will show you how to test on a cluster.

To run code in Eclipse, click on the 'Run' button (circle with triangle inside of it) or click on 'Debug' (an icon with a six-legged insect) while the class with the `main` method is opened and its editor tab is selected.



To run Java code in Maven from the command line, use the `exec:java` goal and specify the fully-qualified class name for a class with a `main` method, as shown here.

```
$ mvn exec:java \
-Dexec.mainClass="full.package.name.MyClass"
```

Step 6: Running the MapReduce Job

Note: Only continue with this step after you have finished writing the MapReduce program.

1. Create a directory in HDFS called `/loudacre`.

```
$ hadoop fs -mkdir /loudacre
```

2. HDFS does not perform well with many small files. Run the following command to concatenate all of the Knowledge Base files together and place them in HDFS:

```
$ cat $BDDATADIR/kb/* | hadoop fs -put \
- /loudacre/allkb.txt
```


3. Run the Maven command to create a JAR for the `nlp-mr` project:

```
$ mvn package
```

This will create the JAR file in the `target` directory, at the following path:

```
/home/training/training_materials/bigdata/exercises/nlp  
-mr/target/nlp-mr-1.0-SNAPSHOT.jar
```

4. After creating the JAR, run the MapReduce job using a command like this:

```
$ hadoop jar target/nlp-mr-1.0-SNAPSHOT.jar \  
com.loudacre.nlpmr.solution.KnowledgeBaseDriver \  
/loudacre/allkb.txt /loudacre/kboutput
```

Note: You must change the driver class's package to match your package name; for example, by replacing `solution` with `hints` or `stubs`.

Bonus Exercises

After looking at the output file, the Documentation team requested more help. They want a list of all of the words that have the same stem and the count of each.

The `nlp-mr-bonus` project contains a custom Writable called `WordAndCountWritable` in the `com.loudacre.data` package to make it easier to pass the word and count. Create an Eclipse project using Maven, import the project into your IDE, and then use this Writable for the value from the Mapper and Reducer. The new code needs to:

- Emit the stem, the word and the count from the Mapper
- Emit the stem, the count of all words with the same stem and the comma-separated list of all words with the same stem from the Reducer

This is the end of the Exercise

Hands-On Exercise: Working with Avro Schemas

Files Used in This Exercise:

Eclipse projects:

avro

In this exercise, you will create and test your own Avro schema.

Step 1: Creating the Schema

Before you can serialize data with Avro, you must define a schema for that data.

1. Perform the Eclipse project initialization in Maven as shown in the previous exercise. The Eclipse project name will always match the directory name in the `exercises` directory:

```
~/training_materials/bigdata/exercises/
```

In this exercise, the subdirectory is named `avro`, so it is located at:

```
~/training_materials/bigdata/exercises/avro
```

Change to the directory above, and then create the Eclipse project by running:

```
$ mvn eclipse:eclipse
```

2. Import the new project into Eclipse, and then open the `activation.avsc` file under `src/main/resources` in Eclipse. The Avro schema has a comment showing where to start filling in the fields.

Give the Avro schema the following fields:

- A field named `activationTimestamp` of type `long`
- A field named `deviceType` of type `string`
- A field named `acct_num` of type `int`
- A field named `device_id` of type `string`
- A field named `phone_number` of type `string`
- A field named `deviceModel` of type `string`

If you need help, the `activation.avsc_solution` file shows the sample solution. The `activation.avsc_hint` file shows the hints. To use either of these, you will need to copy and paste their entire contents in to the `activation.avsc` file.

Step 2: Generating the Avro Class

The Maven POMs in this and subsequent exercises are configured to automatically create Java classes from the Avro schemas in your project.

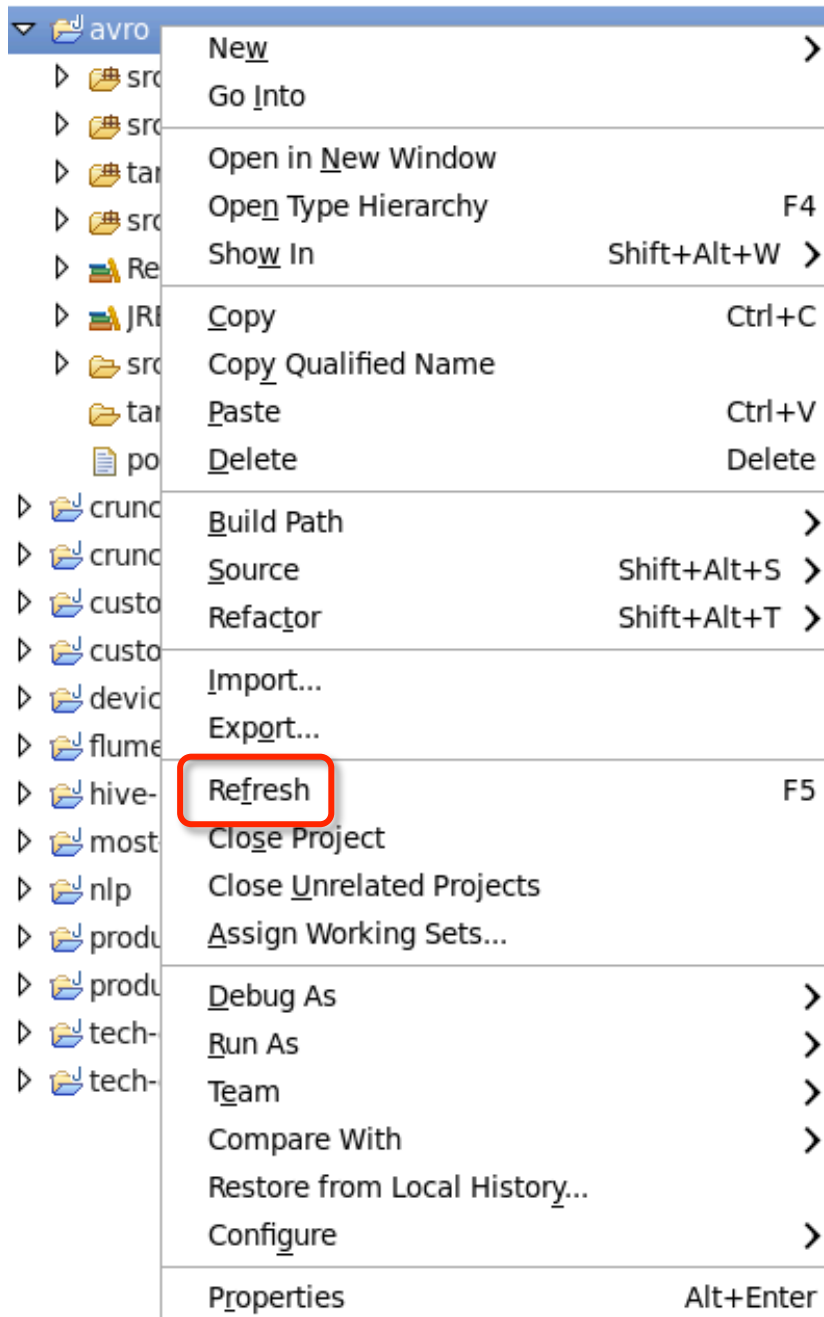
1. To see how this is configured, open the `pom.xml` file and examine the content of the `build` element near the bottom. It uses the 'avro-maven-plugin' plugin to generate Java source code from Avro schemas. In this case, it expects to find Avro schemas in the `src/main/resources` directory and generates Java code to the `target/generated-sources` directory.

You can use this as a guide for Avro projects you might create later.

2. Run the Maven command below to generate Java source that corresponds to the Avro schema you just created:

```
$ mvn generate-sources
```

3. Refresh the `avro` project in Eclipse by right-clicking its icon and selecting 'Refresh'.



4. In the `target/generated-sources` folder in Eclipse, open the `Activation.java` file below the `com.loudacre.data` package. Inspect the file and observe that Avro generated this Java class from your Avro schema. Do not edit this file.
5. In the `src/test/java` folder, open the `AvroTest.java` file below the `com.loudacre.avro` package and uncomment the unit test method.
6. Run the unit test by right clicking in `AvroTest.java` and clicking on run. Verify that all unit tests passed. If the tests fail to compile, you will need to edit

your Avro schema and run the Maven command to regenerate the `Activation.java` file.

Step 3: Using the Avro Tools

Avro comes with tools for viewing the contents or schema of an Avro data file.

3. From the command line, run the `AvroDeviceWriter` program:

```
$ mvn exec:java -Dexec.mainClass=\
"com.loudacre.writers.AvroDeviceWriter"
```

This creates an Avro data file named `devices.avro` containing several records.

4. Use the Avro tools JAR to display the data file's records in JSON format:

```
$ java -jar $AVROTOOLSJAR tojson devices.avro
```

5. Use the Avro tools JAR to view the schema of the Avro file:

```
$ java -jar $AVROTOOLSJAR getschema devices.avro
```

Bonus Exercises

Add a field of type `int` named `battery` to `device.avsc`, and then regenerate the `Device` class. Modify the `AvroDeviceWriter` class to add the missing field to the constructor (choose any number between 1,000 and 3,000 for the battery size). Finally, run the `AvroDeviceWriter` program and view the data with Avro tools.

If you need help, the `device.avsc_bonussolution` file shows the sample solution. The `device.avsc_bonushint` file shows the hints. To use either file, you must copy and paste its entire contents into `device.avsc`.

This is the end of the Exercise

Hands-On Exercise: Transforming Data with Kite

Files Used in This Exercise:

Eclipse projects:

device-activation

products

products-phase2

Data files (local)

~/training_materials/bigdata/data/activations/

In this exercise, you will use Kite to transform phone activation data from XML to Avro format and store the result as a data set in HDFS.

Kite

The phone activations data set is in XML format, which is cumbersome and inefficient to process. Kite makes it easy to transform this data from XML to Avro, which is more efficient and better supported by Hadoop and related tools.

Step 1: Add Device Activation Files to HDFS

1. Before continuing, run the following command to become acquainted with the input data format:

```
$ head -20 $BDDATADIR/activations/2014-03.xml
```

2. Next, put the ~/training_materials/bigdata/data/activations directory in HDFS under the /loudacre directory.

3. Run the `install-kite-javadoc.sh` script.

```
$ $BDDIR/device-activation/install-kite-javadoc.sh
```

This script will install the JavaDoc for Kite. Don't be concerned by any warnings you might see, since we intentionally do not install Javadoc for several libraries that you will not directly use in your code.

4. Perform the Eclipse project initialization in Maven as described earlier. In this exercise, the project directory is located at:

```
~/training_materials/bigdata/exercises/device-activation
```

Step 2: Using Kite

You will use Kite to write device activation records to a new data set. The `com.loudacre.data` package contains the Avro-generated `Activation` class, which is used to store an activation element's data.

The `ActivationsReader` class, in the `com.loudacre.readers` package, parses the XML from a specified input file and populates `Activation` objects based on its content. You will construct an `ActivationsReader` by supplying a `Hadoop Path` object that specifies the location of the XML file, so you must create a new `ActivationsReader` object for each XML file you process.

You can then access all available `Activation` objects by using the `ActivationsReader` class in a `foreach` loop, like this:

```
for (Activation activation : activationsReader) {  
    // write the activation object to the dataset  
}
```

Like Kite's `DatasetWriter`, the `ActivationsReader` also requires that you call its `close()` method to free up resources once you are finished with the object.

You must modify the `ActivationLoader` class so that it:

- Opens a data set repository with a URI that uses the Hive Metastore and stores data sets under `/loudacre` in HDFS
- Creates a `DatasetDescriptor` with a schema URI for the `activation.avsc` file in the `resources` folder of your project
- Uses the `ActivationReader` class to iterate through every device activation file and write out all the `Activation` objects to the data set

Step 3: Verifying the Output

1. Run the `ActivationLoader` program:

```
$ mvn exec:java -Dexec.mainClass=\
"com.loudacre.deviceactivation.solution.ActivationLoader" \
-Dexec.args="/loudacre/activations"
```

2. List the contents of the `/loudacre/deviceactivations` directory:

```
$ hadoop fs -ls /loudacre/deviceactivations
```

3. Use the `hcat` program to list all tables:

```
$ hcat -e "show tables"
```

4. Use the `hcat` program to describe the `deviceactivations` table:

```
$ hcat -e "describe deviceactivations"
```


Bonus Exercises

Note: There are two separate projects for the bonus exercises.

Phase 1

Loudacre's product data can only be accessed using a Java class, so you will use the Kite SDK to create a data set from this data.

In the `products` Maven project, the `ProductsReader` class is the only program that can read the product data. It creates `Product` objects. Use Kite to write out all of the `Product` data.

The program needs to:

- Open a data set repository with a URI that uses the Hive metastore and stores data sets under `/loudacre` in HDFS
- Create a `DatasetDescriptor` with a schema URI for the `product.avsc` file in your project
- Write all `Product` records from the `ProductsReader` class to the new data set

Phase 2

The product data set is being used in production now. The Data Warehousing team realized that they need to make two changes to the format of the data and add some products that were missed. The changes are:

- Change the product ID from an `int` to a `long` to support future growth
- Add an `int` field that reflects the wholesale price of the product (in cents)

In the `products-phase2` Maven project, there is a new class called `ProductsV2Reader` that creates `Product` objects with the changes described above. You must modify the `ProductLoader` class to:

- Load the products dataset and get its existing descriptor

- Create a new descriptor based on the updated (local) schema. This will be used to update the repository using the new descriptor.
- Write out each record to standard output, and prefix it with either 'V1' or 'V2' to identify the version. Hint: If the product's wholesale price is -1 cent, then it is an original (version 1) record, otherwise it is a new (version 2) record
- Update the Kite data to use the new data format for new records
- Add the products that were missed by iterating over the `ProductsV2Reader` object

The code should output all of the `Product` records, and for each, display a message to show the phase (V1 or V2) in which they were originally written

This is the end of the Exercise

Hands-On Exercise: Importing Customer Account Data

Projects and Directories Used in this Exercise

MySQL Database: loudacre

MySQL Table: accounts

Exercise directory: ~/training_materials/bigdata/exercises/sqoop/

In this exercise, you will import tables from MySQL with Sqoop.

Step 1: Sqoop

You can use Sqoop to look at the table layout in MySQL. With Sqoop, you can also import the table from MySQL to HDFS.

1. Run the `sqoop help` command to familiarize yourself with the options in Sqoop:

```
$ sqoop help
```

2. List the tables in the `loudacre` database:

```
$ sqoop list-tables \  
--connect jdbc:mysql://dev.loudacre.com/loudacre \  
--username training --password training
```

3. Run the `sqoop import` command to see its options:

```
$ sqoop import
```

4. Use Sqoop to import the `accounts` table in the `loudacre` database and save it in HDFS under `/loudacre`:

```
$ sqoop import \  
--connect jdbc:mysql://dev.loudacre.com/loudacre \  
--username training --password training \  
--table accounts \  
--target-dir /loudacre/accounts \  
--null-non-string '\\N'
```

The `--null-non-string` option tells Sqoop to represent null values as `\N`, which makes the imported data compatible with Hive and Impala.

Step 2: Exploring the Output

Sqoop uses MapReduce to import the contents of the `accounts` table to HDFS. You can use Hadoop's built-in tools to view the files and their contents.

1. List the contents of the `accounts` directory:

```
$ hadoop fs -ls /loudacre/accounts
```

2. Use Hadoop's `tail` command to view the last part of the file for each of the MapReduce part files:

```
$ hadoop fs -tail /loudacre/accounts/part-m-00000  
$ hadoop fs -tail /loudacre/accounts/part-m-00001  
$ hadoop fs -tail /loudacre/accounts/part-m-00002  
$ hadoop fs -tail /loudacre/accounts/part-m-00003
```

3. The first six digits in the output are the account ID. Note the number of the largest account ID because you will use it in the next step.

Step 3: Incremental Updates

As Loudacre adds new accounts, the account data in HDFS must be updated as accounts are created. You can use Sqoop to append these new records.

1. Run the `add_new_accounts.py` script to add the latest accounts to MySQL.

```
$ $BDDIR/sqoop/add_new_accounts.py
```

2. Incrementally import and append the newly added accounts to the `accounts` directory. Use Sqoop to import on the last value on the `acct_num` column largest account ID:

```
$ sqoop import \  
--connect jdbc:mysql://dev.loudacre.com/loudacre \  
--username training --password training \  
--incremental append \  
--null-non-string '\\N' \  
--table accounts \  
--target-dir /loudacre/accounts \  
--check-column acct_num \  
--last-value <largest_acct_num>
```

Note: replace `<largest_acct_num>` with the largest account number.

3. List the contents of the `accounts` directory to verify the Sqoop import:

```
$ hadoop fs -ls /loudacre/accounts
```

4. You should see three new files. Use Hadoop's `cat` command to view the entire contents of these files.

```
$ hadoop fs -cat '/loudacre/accounts/part-m-0000[456]'
```

Bonus Exercises

Note: The following bonus exercises must use a different directory for each step. Otherwise, the original table will be overwritten and you will need to delete the directory and import the table again.

Sqoop uses comma-delimited text format when importing tables into HDFS by default, but it can also import tables in alternate or more efficient formats.

Run the following Sqoop commands to different directories. Use Hadoop's `tail` command to verify the output of each command.

1. Import the `accounts` table using a tab character as the delimiter.
2. Import the `accounts` table to a SequenceFile format.
3. Import the `accounts` table to an Avro data format.
4. Use Sqoop to import only accounts where the person lives in California (`state = "CA"`) and has an active account (`acct_close_dt IS NULL`).

If you need a hint or would like to check your work, you can find a solution in the `bonus-sqoop.sh` file.

This is the end of the Exercise

Hands-On Exercise: Collecting Web Server Logs with Flume

Files and Directories Used in this Exercise

Data (local):

```
~/training_materials/bigdata/data/weblogs
```

Exercise directory:

```
~/training_materials/bigdata/exercises/flume
```

In this exercise, you will configure Flume to ingest web log data from a local directory to HDFS.

Step 1: Directory Creation

Both the local and HDFS directories must exist before using the spooling directory source.

1. Create a directory in HDFS under `/loudacre` called `weblogs`:

```
$ hadoop fs -mkdir /loudacre/weblogs
```

This is the output directory in HDFS where Flume will store the web log files.

2. Create a path on the local filesystem at `/flume/weblogs_spooldir`:

```
$ sudo mkdir -p /flume/weblogs_spooldir
```

3. Give all users the permissions to write to the `/flume/weblogs_spooldir` directory:

```
$ sudo chmod a+w -R /flume
```

Step 2: Configuration

In `~/training_materials/bigdata/exercises/flume` under `stubs`, `hints` or `solutions`, create a Flume configuration file with the following characteristics:

- The source is a spooling directory source that pulls from `/flume/weblogs_spooldir`
- The sink is an HDFS sink that:
 - Writes files to the `/loudacre/weblogs` directory
 - Disables time-based file rolling by setting the `hdfs.rollInterval` property to 0
 - Disables event-based file rolling by setting the `hdfs.rollCount` property to 0
 - Sets the `hdfs.rollSize` property as 524288 to enables size-based file rolling at 512KB
- The channel is a Memory Channel that:
 - Can store 10,000 events using the `capacity` property
 - Has a transaction capacity of 10,000 events using the `transactionCapacity` property

Flume Performance Note

Flume's performance on a VM will vary. Sometimes, Flume will exhaust the memory capacity of its channel and give the following error:

```
Space for commit to queue couldn't be acquired. Sinks are
likely not keeping up with sources, or the buffer size is
too tight
```

If you get this error, you will need to increase the `capacity` and `transactionCapacity` properties to a higher value. Then, you will need to restart the agent, clean up any files in HDFS, and repeat the Flume operation.

Step 3: Running the Agent

Once you have created the configuration file, you need to start the agent and copy the files to the spooling directory.

1. Change directories to the to the `~/training_materials/bigdata/exercises/flume` directory.
2. Start the Flume agent using the configuration you just created.

```
$ flume-ng agent --conf /etc/flume-ng/conf \
--conf-file solution/spooldir.conf \
--name agent1 -Dflume.root.logger=INFO,console
```

3. In a separate terminal window, run the script to place the web log files in the `/flume/weblogs_spooldir` directory:

```
$ ./copy-move-weblogs.sh /flume/weblogs_spooldir
```

This script will create a temporary copy of the web log files and move them to the `spooldir` directory.

4. Return to the terminal that is running the Flume agent and watch the logging output. The output will give information about the files Flume is putting into HDFS.
5. Once the Flume agent has finished, press CTRL+C to terminate the process.
6. List the files in HDFS that were added by the Flume agent.

```
$ hadoop fs -ls /loudacre/weblogs
```

Bonus Exercises

Loudacre has a data source that comes in via the network. You want to see some examples since the format is undocumented.

Flume has a `netcat` source, which allows Flume to listen on a port and process the contents received. The address to which to bind is specified by the `bind` property, and the port to bind to is specified by the `port` property.

Flume also has a `logger` sink, which logs any incoming data at the `INFO` level in Log4J. This is helpful for debugging and testing Flume configuration. This sink does not require any configuration properties.

Create a Flume configuration file with the following characteristics:

- Uses the `netcat` source to capture data from host `dev.loudacre.com` on port `44444`.
- Uses the `logger` sink
- Uses the `memory` channel

Start the Flume agent using the new configuration file.

In another terminal window, start the script using the command:

```
$ $BDDIR/flume/script/rng_feed.py
```

This script will instantiate the feed. Observe the log messages in the Flume agent's terminal and stop the agent after you have seen a few example records.

This is the end of the Exercise

Hands-On Exercise: Collecting Application Data with Flume

Files and Directories Used in this Exercise

Exercise directory:

```
~/training_materials/bigdata/exercises/tech-diagnostic-gui/
```

Eclipse Projects:

```
tech-diagnostic-bonus
```

In this exercise, you will log to a Kite data set from a legacy application without modifying that application's source code.

Loudacre's field service technicians use a Java Swing application to check the status of the cell phone base stations. This application writes current sensor readings to the terminal using Log4J, and the technicians use this information to fill out reports. The Data Warehousing team wants to collect this data and store it in HDFS for long-term trend reporting. Unfortunately, nobody can find the application's source code.

Step 1: Running the Application

1. Change directories to the exercise directory:

```
$ cd $BDDIR/tech-diagnostic-gui/
```

2. In the terminal, start the application:

```
$ ./startgui.sh $TDGURL/stubs/log4j.properties
```

It may take a few seconds before the GUI is visible. The argument provided to the `startgui.sh` script specifies the URI to the Log4J properties file that the application will use.

3. In the GUI, press the 'Dump Diagnostic Data' button several times, then choose another station and repeat.

4. Observe the data that Log4J displays in the terminal after you click the button, and then close the GUI.

Step 2: Configuration

The messages from Log4J are currently only displayed in the terminal, but we want to save them as Avro files in HDFS.

In the `~/training_materials/bigdata/exercises/tech-diagnostic-gui/`, you will find the stubs, hints and solutions for the `log4j.properties` file.

Modify the Log4J configuration file to have the following characteristics:

- Uses Flume's Log4J appender
- Outputs the data to `dev.loudacre.com` over port 41415
- Uses the following conversion pattern:

```
TIME:%d{yyyy-MM-dd HH:mm:ss};%m
```

Create a Flume configuration file with the following characteristics:

- Uses an Avro source to pull data from `dev.loudacre.com` on port 41415
- Uses an HDFS sink that writes data to `/loudacre/diagnosticdata`
- Uses a Memory Channel

Step 3: Running with Flume

1. In one terminal, start the Flume agent:

```
$ flume-ng agent --conf /etc/flume-ng/conf \  
--conf-file solution/flume_kite.conf \  
--name agent1 -Dflume.root.logger=INFO,console
```

2. In a second terminal, start the GUI and specify the URI of the desired Log4J properties file:

```
$ ./startgui.sh $TDGURL/stubs/log4j.properties
```

3. In the GUI, press the 'Dump Diagnostic Data' button several times and then choose the next station. Repeat this for every station in the list.
4. Verify the output is being written to HDFS by listing the files in the data set and using `cat` to output the contents.

Bonus Exercises

The team just found the source code to the application, so you can load data using Kite. The Data Warehousing team has already created an Avro schema for the station information. They also updated the Maven files to include the relevant Kite dependencies.

In the `~/training_materials/bigdata/exercises/tech-diagnostic-gui-bonus/`, you will need to create the Eclipse project and modify the code.

Make the following changes to the legacy program:

- Change the program to use the Avro object instead of the POJO
- You cannot modify the format or information in the logging output because the Data Warehouse uses it. Make sure that the logging output of the unaltered program matches the new program
- Use the data set name `stationinfo`
- Modify the `MainPanel` class's `StationDataWorker` class's `doInBackground` method to write to a Kite data set and output to Log4J
- Make sure you close your Kite `DatasetWriter` object before the program exits by adding the following code to the `TechDiagnosticGui` class:

```
frame.addWindowListener(new WindowAdapter() {  
    public void windowClosing(WindowEvent e) {  
        panel.closeWriter();  
    }  
});
```

This code will invoke the `closeWriter` method in the `MainPanel` class and close any Kite resources on exit. This is important to make sure all data is flushed to disk on exit.

You can run the program directly from Eclipse or with Maven using the command:

```
$ mvn exec:java -Dexec.mainClass=\  
"com.loudacre.techdiagnosticgui.solution.TechDiagnosticGui"
```

This is the end of the Exercise

Hands-On Exercise: Writing a Custom Flume Interceptor

Projects Used in this Exercise

Eclipse project:

`flume-interceptor`

`flume-interceptor-bonus`

In this exercise, you will write a custom interceptor for Flume that processes incoming device status.

Data Formats and Helpers

All devices on Loudacre's mobile network send periodic status updates, which include the unique device ID and various sensor readings. Each record is sent over the network as a single line of data.

You need to use a custom interceptor to convert this data to a tab-delimited format. Although each device sends the same data, each manufacturer uses a slightly different format. The provided `DeviceDataHelper` class parses the record text and returns a normalized `DeviceData` object.

The device temperature is sent as a delta, or difference, between the device temperature and the ambient temperature. This makes it difficult to query with Hive or Impala. The device temperature needs to be reconstituted to a whole number by adding the device temperature and ambient temperature together.

Step 1: Write the Interceptor

In the `~/training_materials/bigdata/exercises/flume-interceptor/` directory, you will find the `stubs`, `hints`, and `solutions` packages for the Flume Interceptor.

Using your knowledge of the incoming data and formats, write a Custom Interceptor using Flume. This class will need to implement the

`org.apache.flume.interceptor.Interceptor` interface and have an inner class that implements the `org.apache.flume.interceptor.Interceptor.Builder` interface.

The `Interceptor` class needs to:

- Intercept the incoming device status events
- Convert the text representation of a device status event to use the `DeviceData` object with the `DeviceDataHelper` class
- Change the device temperature from a delta to a whole number by adding the ambient temperature to it
- Replace the event body with the tab-delimited string representation of the `DeviceStatus` object using the `toTabDelimitedString` method

We highly recommend using unit tests to speed up the development of this exercise. Create a unit test for each manufacturer according to its device format. Use the unit test's assertions to verify that the data was parsed correctly and the `DeviceData` object contains the correct information.

You can use the solution's unit tests as well. Be sure to copy the class to the stubs package and change all package names to the stub's package names.

Proceed to the next step once all of the unit tests pass. Moving on before the unit tests are passing will make the device status data set invalid. This data set is analyzed later in the course.

Step 2: Create the Flume Configuration

In the `~/training_materials/bigdata/exercises/flume-interceptor/` directory, you will find the stubs, hints, and solutions for the Flume configuration that uses the `Interceptor` that you just wrote. Create a Flume configuration file with the following characteristics:

- Uses a `netcat` source to listen on port `44444` of `dev.loudacre.com` and uses the `interceptor` that you just wrote

- The sink is an HDFS sink that:
 - Saves to the `/loudacre/devicedata` directory
 - Creates a new file every 10,000 events or 60 seconds
 - Has a file type of `DataStream`
- The channel is a Memory Channel that:
 - Can store 10,000 events using the `capacity` property
 - Has a transaction capacity of 10,000 events using the `transactionCapacity` property

Remember to watch for any memory issues with the Flume agent.

Step 3: Running the Interceptor

1. Create the plugin directory for the Flume Interceptor at `/etc/flume-ng/plugins.d/flume-interceptor/lib`:

```
$ sudo mkdir -p \  
/etc/flume-ng/plugins.d/flume-interceptor/lib
```

2. Change the permissions for the Flume Interceptor directory to allow all users to read, write, and execute (note that in a production environment, you would instead set permissions to limit access so that only a few trusted people could deploy or modify Flume plugins):

```
$ sudo chmod -R 777 /etc/flume-ng/plugins.d
```

3. Create the JAR artifact in Maven:

```
$ mvn package
```

4. Symlink the JAR artifact to the Flume Interceptor plugins directory:

```
$ ln -s \
$FLUMEINT/target/flume-interceptor-1.0-SNAPSHOT.jar \
/etc/flume-ng/plugins.d/flume-interceptor/lib/
```

Note: You would not create a symlink in production; you would copy the file to the plugins directory. Symlinking allows you to make code changes in the exercises directory without having to copy the file each time.

5. In one terminal, start the Flume agent:

```
$ flume-ng agent --conf /etc/flume-ng/conf \
--conf-file solution/flume_interceptor.conf \
--name agent1 -Dflume.root.logger=INFO,console \
--plugins-path /etc/flume-ng/plugins.d/
```

6. In a second terminal, start the device status script:

```
$ $FLUMEINT/script/device_status_feed.py
```

This program will send the device status data over the network to the Flume agent. The Flume agent will save the data to HDFS.

7. Verify there are no errors being logged by the Flume agent or the script.
8. In a third terminal, go to HDFS and list the files created by the program in `/loudacre/devicedata`.
9. Let the script run for 5-10 minutes before closing both programs. Press CTRL+C in each terminal to stop each program.

Bonus Exercises

The code for the bonus exercises is found in the `~/training_materials/bigdata/exercises/flume-interceptor-bonus/` directory. Before continuing, you must first change to this directory, create the Eclipse project using Maven, and import that project into your IDE. Like the

main exercise, the this project also contains `stubs`, `hints`, and `solutions` packages.

Hard coding delimiters and workarounds is a bad idea. Making them properties in the Flume configuration is a better idea. Update the interceptor to use the `Context` object such that these configuration settings can be retrieved from the Flume properties file and then used in your code:

- Use the value of the `delimiter` configuration property, and the `toDelimitedString` method in the `DeviceData` object, to allow a user to specify a different delimiter.
- If the `fixDelta` configuration property is set to `true`, combine the device temperature with the ambient temperature so that we store the complete value rather than just the difference.

This is the end of the Exercise

Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Files and Directories Used in this Exercise

MySQL Database: loudacre

MySQL Table: basestation

Exercise directory:

```
~/training_materials/bigdata/exercises/oozie-workflow
```

In this exercise, you will create a workflow with Hue for Oozie that imports data from MySQL using Sqoop, processes it with a MapReduce, and exports the result to a table for access by other systems.

IMPORTANT: This exercise builds on the previous one. If you were unable to complete the previous exercise or think you may have made a mistake, run the following command to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh flumeCustomInterceptor
```

In this exercise we will be working with the device status data set created in the previous exercise. This data set will be combined with base station (cell phone tower) data to see which base station handled the call and when the call occurred.

Step 1: Configuring and Compiling the Job

There is a prewritten MapReduce class that you will add to the workflow. Before you can add it, you will need to configure certain pieces and compile the JAR. First, you will configure Oozie.

1. In the terminal, change to the exercise directory

```
~/training_materials/bigdata/exercises/oozie-workflow.
```
2. Run the Oozie setup script:

```
$ ./setup_oozie.sh
```

This script sets up the shared libraries that Oozie needs to run certain actions.

3. Create the JAR:

```
$ mvn package
```

Step 2: Creating a New Oozie Job with Hue

1. In the Web browser, go to `http://dev.loudacre.com:8888`.
2. Maximize your browser window so you can see the entire Hue UI.
3. If prompted, use 'training' for both the username and password, and then click the 'Sign Up' button to log in. If you see a warning about a potential misconfiguration of the `desktop.ldap.ldap_url` property, you may ignore it and click the 'Next' button.
4. Click the Oozie icon, which looks like a blue box with a yellow circle inside it.



This will bring up the listing of all Oozie jobs.

5. Click the 'Workflows' link just below the row of icons at the top of the page.
6. Click the 'Create' button.
7. Name the new job 'Most Active Base Station'.
8. Click the 'Save' button.

Step 3: Importing the Base Station Table

You can use a Sqoop action in Oozie to import database table.

Sqoop Command Syntax in Hue

The syntax of the Sqoop command in Hue differs from the one used on the command line.

On the command line, you may put single quotes around the character specified in the `--fields-terminated-by` argument. In Hue, however, you must either omit the single quotes around this character or use double quotes instead.

Sqoop commands in Hue also do not begin with `sqoop` as they do on the command line. A Sqoop import command in Hue might therefore begin like this:

```
import --connect jdbc:mysql://dev.loudacre.com/loudacre
```

In Hue, commands must all be on one line, without any newlines or backslashes.

Click and drag the Sqoop action button to the dashed rectangle between the 'start' and 'end' nodes. This will add it to your new workflow, after which you should configure the action with the following characteristics:

- Name the action 'BaseStationImport'
- Type the command (`import`) in the 'Command' textarea, followed by all arguments listed below on the same line:

Argument	Value
<code>--connect</code>	<code>jdbc:mysql://dev.loudacre.com/loudacre</code>
<code>--username</code>	<code>training</code>
<code>--password</code>	<code>training</code>
<code>--table</code>	<code>basestations</code>
<code>--target-dir</code>	<code>\${baseStationDir}</code>
<code>--fields-terminated-by</code>	<code>"\t"</code>

- In the 'Prepare' section, add a delete action for `${baseStationDir}`
- Click the 'Done' button to return to the workflow editor

Step 4: Running the MapReduce Job

Now that you have defined a Sqoop action to import the Base Station table from MySQL, you will define a new action to run a MapReduce job that processes this data.

1. Click on 'Workspace'. This opens the Hue File Browser in a new tab and displays the files associated with the Oozie Workflow.
2. Create a directory to hold your workflow's libraries by clicking the 'New' button on the far right and selecting 'Directory'. After specifying 'lib' as the directory name, click the 'Create' button.
3. Click the 'lib' link to navigate into the new subdirectory.
4. Next, you will upload the database driver so that the Sqoop action can access MySQL. First, click the 'Upload' button on the far right and select 'Files' from the menu. Then, click the 'Select files' button, browse to locate the `/usr/lib/sqoop/lib/mysql-connector-java-5.1.17.jar` file, and then click the 'Open' button.
5. Use the 'Upload' button again to upload the `target/most-active-station-1.0-SNAPSHOT.jar` JAR that you packaged with Maven earlier in this exercise. It contains the MapReduce code the job will run to process the data.
6. Close the new tab in your browser and return to the Workflow Editor.

In this step, we will add another action to process the base station data from Sqoop and process it with a MapReduce job.

Add a MapReduce action to the workflow after the `BaseStationImport` action and configure it as follows:

- Has the name 'MostActiveStation'.
- Click the button adjacent to 'Jar Name' and select the `most-active-station-1.0-SNAPSHOT.jar` by browsing to it in the `lib` directory.
- In the 'Prepare' section, add a delete action for `${mostActiveDir}`

- The Job Properties should be configured with the following properties:
 - Set `maxstations` to 5. This controls the number of output results.
 - Set `timewindow` to 60. This controls the number of seconds in each grouping of activity.
 - Set `mapreduce.map.class` to `com.loudacre.mostactivestation.DistanceMapper`
 - Set `mapreduce.reduce.class` to `com.loudacre.mostactivestation.ActivityByStationReducer`
 - Set `mapred.input.dir` to `${deviceDataDir}`
 - Set `mapred.output.dir` to `${mostActiveDir}`
 - Set `basestationlist` to `${baseStationDir}`
 - Set `mapred.mapper.new-api` to `true`
 - Set `mapred.reducer.new-api` to `true`
 - Set `mapred.mapoutput.value.class` to `com.loudacre.data.BaseStation`
- Click the 'Done' button to return to the workflow editor.

Step 5: Exporting the Data

Finally, we will take the output from the MapReduce job and export it back to the MySQL database so the results are available to other applications. The table in MySQL is named `mostactivestations`, which already exists but is empty. This table has three columns: `station_num`, `activity`, and `activity_time`.

The format of the output from the MapReduce job is:

<code>{activity_time}</code>	<code>{activity}</code>	<code>{station_num}</code>
------------------------------	-------------------------	----------------------------

Add another Sqoop action after the `MostActiveStation` action and configure it with the following characteristics:

- Name the action 'MostActiveExport'
- Type the command (`export`) in the 'Command' textarea, followed by all arguments listed below on the same line:

Argument	Value
<code>--connect</code>	<code>jdbc:mysql://dev.loudacre.com/loudacre</code>
<code>--username</code>	<code>training</code>
<code>--password</code>	<code>training</code>
<code>--table</code>	<code>mostactivestations</code>
<code>--export-dir</code>	<code>\${mostActiveDir}</code>
<code>--input-fields-terminated-by</code>	<code>"\t"</code>
<code>--columns</code>	<code>activity_time,activity,station_num</code>
<code>--update-mode</code>	<code>allowinsert</code>

Afterwards, click the 'Done' button to return to the workflow editor.

Step 6: Running the Workflow

Save your changes to the Workflow by clicking the blue 'Save' button at the bottom, and then run the workflow in Hue by clicking the 'Submit' button on the left.

You will be prompted to give paths to the three parameters you used in the actions.

- The `baseStationDir` should be set to `/loudacre/basestation`
- The `mostActiveDir` should be set to `/loudacre/mostactive`
- The `deviceDataDir` should be set to `/loudacre/devicedata`

After submitting the workflow, check the progress and success using Hue. You can verify that the export succeeded by executing the following command:

```
$ mysql -h dev.loudacre.com -u training -ptraining \  
loudacre -e "select count(*) from mostactivestations where  
activity_time='2014-03-15 12:25:00'"
```

The command should show that five records were exported for this time window.

Bonus Exercises

Create a schedule for the newly created workflow using Hue. Make the workflow run once an hour. After you have verified this change, remove the schedule so that it does not continue to run in the background.

This is the end of the Exercise

Hands-On Exercise: File Type Processing with Crunch

Files and Directories Used in this Exercise

Eclipse projects:

`crunch-filetypes`

`crunch-filetypes-bonus`

Data directory (local):

`~/training_materials/bigdata/data/weblogs/`

In this exercise, you will use Crunch to count the file types in web logs.

Step 1: Creating the Crunch Pipeline

The web log data has a specific format. Here is an example line:

```
165.32.101.206 - 8 [15/Sep/2013:23:59:50 +0100] "GET
/KBDOC-00001.html HTTP/1.0" 200 17446
"http://www.loudacre.com" "Loudacre CSR Browser"
```

From this line, the program will need to extract the file type. In this example, the page is `KBDOC-00001.html` and the file type is `html`.

Create a Crunch pipeline with the following characteristics:

- Reads in all log lines
- Parses the file type from the request in the log line
- Emits the file type
- Counts the number of times each file type was requested
- Writes out each file type and the number of times it was accessed

Step 2: Creating the JAR

Once the coding is done and the unit tests pass, you will need to create a JAR with Maven. Creating a JAR with Maven for Crunch is slightly different to creating a standard MapReduce JAR.

1. Open the `pom.xml` file for the project.
2. Find the element that says `mainClass`.
3. Change the value to the fully qualified name of your class with the main method. This is currently configured to use the solution package.
4. In a terminal window, run:

```
$ mvn package
```

This will create a JAR with a slightly different name than previous JARs. Note that the JAR's name has '-job' at the end.

5. HDFS does not perform well with many small files. Run the following command to concatenate all of the web log files together and put them in HDFS:

```
$ cat $BDDATADIR/weblogs/* | hadoop fs -put \  
- /loudacre/allweblogs.log
```

6. Run the JAR with Crunch and MapReduce using the following command:

```
$ hadoop jar \  
target/crunch-filetypes-1.0-SNAPSHOT-job.jar \  
/loudacre/allweblogs.log /loudacre/filetypecounts
```

Bonus Exercises

Counting amounts in Crunch can be done with counters instead of Crunch's `count` method. Counters in Crunch work exactly the same way as MapReduce counters. Using counters is a more efficient way of counting smaller groups.

To use a counter in Crunch, use the `increment` method in the `DoFn` class. The `increment` method takes a group name and a counter name as arguments.

Change your `DoFn` to use the `increment` method instead of emitting and using the `count` method. Use the `CounterHelper.COUNTER_GROUP_NAME` string as the group name and the extension as the counter name.

Use the `outputCounterValues` method in the `CounterHelper` class to output the counter values at the end of the job. The `outputCounterValues` method takes a `PipelineResult` as a parameter. To get the `PipelineResult` object, run the Crunch job by calling:

```
PipelineResult result = done();
```

Counters in Crunch

This version of Crunch does not output counter values automatically, but newer versions can be configured to do so.

This is the end of the Exercise

Hands-On Exercise: Log Processing with Crunch

Files and Directories Used in this Exercise

Eclipse projects:

`crunch-logs`

`crunch-sessionization`

Data directory (local):

`~/training_materials/bigdata/data/weblogs/`

In this exercise, you will use Crunch to process web logs.

IMPORTANT: This exercise builds on the previous one. If you were unable to complete the previous exercise or think you may have made a mistake, run the following command to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh crunchFileTypeProcessing
```

The web log data contains valuable information about what Loudacre's customers are looking at on the site. Processing this data will show how users interact with the Loudacre web site.

Step 1: Creating the Crunch Pipeline

The web log data has a specific format. Here is an example line:

```
165.32.101.206 - 8 [15/Sep/2013:23:59:50 +0100] "GET
/KBDOC-00001.html HTTP/1.0" 200 17446
"http://www.loudacre.com" "Loudacre CSR Browser"
```

From this line, the program will need several pieces of data. You will need to extract the page that was accessed. In this example, the page is `KBDOC-00001.html`. All

Knowledge Base pages start with 'KBDOC-'. The string 'KBDOC-' will not appear anywhere else in the log line.

Create a Crunch pipeline with the following characteristics:

- Reads in all log lines
- Only processes log lines for Knowledge Base articles hits
- Emits the full name of the Knowledge Base article that was accessed
- Counts the number of times each Knowledge Base article was accessed
- Writes out the name of each Knowledge Base article that was accessed, along with the number of times it was accessed

Step 2: Finding the Top Requested Articles

The Web Development team needs to use the output from the Crunch job. They want to display the most requested pages. You will need to change the Crunch pipeline to meet their needs.

Modify the Crunch pipeline so that it outputs the top N most requested pages. The maximum number of pages to output should be passed in with a configuration parameter. If the maximum number of pages is not passed in, it should default to 20.

We encourage you to use unit tests to speed up your development. You can either write you own or use the pre-written one from the sample solution. If you use the one from the sample solution, you will need to change the packages to point to your class's packages.

Step 3: Creating the JAR

Once the coding is done and the unit tests pass, you will need to create a JAR with Maven. Creating a JAR with Maven for Crunch is slightly different to creating a standard MapReduce JAR.

1. Open the `pom.xml` file for the project.
2. Find the element that says `mainClass`.
3. Change the value to the fully qualified name of your class with the main method. This is currently configured to use the solution package.
4. In a terminal window, run:

```
$ mvn package
```

This will create a JAR with a slightly different name than previous JARs. Note that the JAR's name has '-job' at the end.

5. Run the JAR with Crunch and MapReduce using the following command:

```
$ hadoop jar target/crunch-logs-1.0-SNAPSHOT-job.jar \  
/loudacre/allweblogs.log /loudacre/mostpopulararkbs
```

Bonus Exercises

The Data Science team decided that they want to change the report's algorithm. They explained that the last page a customer looks is usually the one that solved their problem. The Data Science team needs you to change the algorithm to only output the last page a customer accessed during a session.

To track a user, you will need the user's account ID. The user's account ID is part of the web log. Using the following line as an example (the account ID here is '8'):

```
165.32.101.206 - 8 [15/Sep/2013:23:59:50 +0100] "GET KBD0C-  
00001.html HTTP/1.0" 200 17446 "http://www.loudacre.com"  
"Loudacre CSR Browser"
```

The session time is defined by the Data Science team as being less than 30 minutes from the last access by a user. If more than 30 minutes has passed since the last request, it should be considered a new session.

Create a Crunch pipeline with the following characteristics:

- Reads in all log lines, but only process log lines for Knowledge Base articles
- When a Knowledge Base hit is found, use the Avro class `WebHit` to represent a user access
- Use a Secondary Sort to group on account IDs and sort on access timestamps
- Process all data from the Secondary Sort to calculate sessions and output only the last Knowledge Base article that was accessed for the session
- Write out the top 20 most helpful Knowledge Base articles

We encourage you to use unit tests to speed up your development. You can either write you own or use the pre-written one from the sample solution. If you use the one from the sample solution, you will need to change the packages to point to your class' packages.

Once the coding is done and all unit tests pass, you will need to create a JAR with Maven as described above. Run the job and place the output in HDFS at `/loudacre/mosthelpfulkb`.

This is the end of the Exercise

Hands-On Exercise: Running Queries in Hive

Files and Directories Used in this Exercise

Data Directory (HDFS):

```
/loudacre/accounts
```

Exercise directory:

```
~/training_materials/bigdata/exercises/hive-queries
```

In this exercise, you run Hive queries on the data you have imported.

IMPORTANT: This exercise builds on the previous ones. If you were unable to complete the previous exercise or think you may have made a mistake, run the following command to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh crunchLogProcessing
```

Step 1: Creating the Accounts Table

In a previous exercise, you used Sqoop to import the `accounts` table from MySQL. Sqoop dumped all of the data from the table into HDFS, but did not create a corresponding table in Hive. Before querying the data in Hive, you will need to create this Hive table.

1. Re-familiarize yourself with the table's schema in MySQL:

```
$ sudo mysql -e 'describe loudacre.accounts;'
```

2. Look at the format of the data as it was output by Sqoop:

```
$ hadoop fs -tail /loudacre/accounts/part-m-00000
```

3. Start the Beeline shell and connect to HiveServer2 on `dev.loudacre.com`.

```
$ beeline -u jdbc:hive2://dev.loudacre.com \  
-n training -p training
```

4. Create an external table in Hive for the `accounts` table called `accounts` located at `/loudacre/accounts`. Since the data in this directory has fields separated by commas, you need to specify the comma as the field terminator in your `CREATE TABLE` statement.

Step 2: Querying the Accounts Table

1. Write a query to select the row for account ID 42.
2. Write a query to find how many customers live in the state of 'OR' (Oregon).

Step 3: Creating the Device Status Table

In a previous exercise, you created a custom Flume Interceptor that captured device status data. This data was written out using a tab-delimited format that you placed in HDFS at:

```
/loudacre/devicedata
```

The table's column names and types are:

```
time BIGINT,  
name STRING,  
device_id STRING,  
device_temp INT,  
ambient_temp INT,  
battery_pct INT,  
signal_pct INT,  
cpu_load INT,  
ram_usage INT,  
gps_status STRING,  
bluetooth_status STRING,  
wifi_status STRING,  
longitude FLOAT,  
latitude FLOAT
```

1. Create an external table in Hive called `devicestatus` for the device data.
2. Write a query to select the first device status entry that came in from a device ID that starts with '123'.

Step 4: Using Hue

1. In a web browser, go to `http://dev.loudacre.com:8888`.
2. If you are prompted for a username and password, use 'training' for both.
3. Click on the Beeswax icon, which looks like an elephant and a bee. This will bring up Hue's Hive interface.



4. You can execute Hive queries by typing them into the Query Editor and clicking 'Execute'.
5. Using Hue, find how many customers live in the state of 'OR' (Oregon).

Step 5: Creating the Most Helpful Table

In a previous exercise, you created a most popular knowledge base article data set using Crunch. This data was written in HDFS to:

```
/loudacre/mostpopulararkbs
```

The data set is tab-separated and contains two fields. The first field is the name of the name article and the second field is the number of hits.

1. Create an external table in Hive called `mostpopulararkbs` for this data.
2. Execute a Hive query to select all of the records.

Step 6: Describing the Device Activations Table

In a previous exercise, you created a Hive table with Kite called `deviceactivations`. This table contains the account ID and device ID information for every device that was activated on Loudacre's network.

1. In the beeline shell, run the `DESCRIBE` command to display the structure of the `deviceactivations` table.

Step 7: Joining Tables

Using the foreign key relationships between the `accounts`, `deviceactivations`, and `devicestatus` tables, you can join them together with Hive. The `accounts` and `deviceactivations` tables can be joined together using the `acct_num` column. The `devicestatus` and `deviceactivations` tables can be joined together using the `device_id` column.

Beeline can accept a file as a parameter to run the HiveQL query inside of it.

1. Using your favorite text editor, create a file that contains a Hive query. This query should join all three tables together with a limit of 20 rows.
2. Run the file with Beeline:

```
$ beeline -u jdbc:hive2://dev.loudacre.com \  
-n training -p training -f joins.hql
```

Using your favorite query creation and run method, write Hive queries with the following characteristics:

- Create a query to select all distinct account numbers whose state is 'CA' (California) from the `accounts` table and have a device whose model name is 'Sorrento F41L' from the `devicestatus` table.

Bonus Exercises

Write and execute queries to retrieve the following information:

- Get the count of devices grouped by the state where they are used
- Get the twenty most popular devices when grouped by state
- Calculate the average ambient temperature of devices grouped by state

This is the end of the Exercise

Hands-On Exercise: Using the RegexSerDe in Hive

Files and Directories Used in this Exercise

Exercise Directory:

```
~/training_materials/bigdata/exercises/hive-serde
```

Data (local):

```
~/training_materials/bigdata/data/calllogs
```

In this exercise, you will use the RegexSerDe to work with fixed-width data.

Call Log Format

The format of the call log data is fixed-width. Each line is a separate call made from one phone to another phone. The lines are a maximum of 104 characters and contains the following six columns:

Field	Output Data Type	Input Description
call_id	STRING	Uniquely identifies a call
call_begin	TIMESTAMP	Date and time at which the call began, formatted as yyyy-MM-dd:HH:mm:ss
call_end	TIMESTAMP	Date and time at which the call ended, formatted as yyyy-MM-dd:HH:mm:ss
status	STRING	Status of a phone call, which is one of: SUCCESS, DROPPED, or FAILED
from_phone	STRING	Phone number that initiated the call
to_phone	STRING	Phone number that received the call

Here is an example line from the call logs:

```
997f743f-ac3b-4e0d-83e4-5210c657a1092014-03-15
00:00:142014-03-15 00:06:55SUCCESS 50366436189285597836
```


Each column contains a specific number of characters. The following table shows the width (character count) of each column, and values for the example line above.

Field	Width	Example Value
call_id	36	997f743f-ac3b-4e0d-83e4-5210c657a109
call_begin	19	2014-03-15 00:00:14
call_end	19	2014-03-15 00:06:55
status	10	SUCCESS
from_phone	10	5036643618
to_phone	10	9285597836

Step 1: Preparing the Data

The call logs need to be placed in HDFS before creating the table.

1. Put the call log directory in HDFS:

```
$ hadoop fs -put \  
~/training_materials/bigdata/data/calllogs \  
/loudacre
```

Step 2: Using the Regex SerDe

The Operations team wants to analyze these call logs in order to identify network problems. You will need to use Hive's built-in RegexSerDe to create a table in Hive. This will allow Operations to query the incoming call logs with Hive.

You will need to:

- Use the RegexSerDe to create an external table called `calldata` with its data stored at `/loudacre/calllogs`
- Include all fields in the call log
- All columns should be of type `STRING`

- The regular expression will need to handle extra space in the status field and not include the whitespace in the regular expression group

Verify that the table creation worked by running a `SELECT` query with a `LIMIT`.

Step 3: Using the Table

The Operations team also needs a data product containing all calls that are not successful. Use a CTAS (create table as `SELECT`) to create a table named `calldatafail` and be saved to HDFS at `/loudacre/calldatafail`.

Verify that the table creation worked by running a `SELECT` query with a `LIMIT`.

Step 4: Non-Fixed-Width Data Products

The Data Warehousing team wants the table saved in the default Hive format instead of the fixed-width format. It also needs to have the correct types for the columns instead of every column being a `STRING`. Use the CTAS technique to create a new table with the following characteristics:

- Named `calldetails` with its data stored at `/loudacre/calldetails`
- The `call_begin` and `call_end` columns should be converted to timestamps

Verify that the table creation worked by running a `SELECT` query with a `LIMIT`.

Bonus Exercises

Use the `RegexSerDe` to create a Hive table for the web log data in `/loudacre/weblogs`. Make the regex parse the IP address, user id, request time, and request page. Remember that the regex needs to match the entire line even though you only want certain pieces of information.

Write and execute queries to retrieve the following information:

- Get the count of hits grouped by the page
- Get the count of hits grouped by the page, for all user ids under 1000, for hits to the Knowledge Base ('KBDOC-'), ordered by the count of hits in descending order

This is the end of the Exercise

Hands-On Exercise: Implementing a User-Defined Function in Hive

Files and Directories Used in this Exercise

Eclipse project:

hive-udf

hive-udf-bonus

In this exercise you will create a custom UDF for Hive that normalizes and formats phone numbers.

IMPORTANT: This exercise builds on the previous ones. If you were unable to complete the previous exercises or think you may have made a mistake, run the following command to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh hiveRegexSerDe
```

The Customer Service team uses the `calldetails` table for some of its operations. Their other systems format the phone number for them.

Phone numbers are stored as unformatted strings, such as:

```
1238675309
```

This is too hard to read, so the Customer Service team wants them formatted as:

```
(123) 867-5309
```

Some phone numbers have some embedded formatting. Normalize the phone numbers before formatting them.

Step 1: Creating the Hive UDF

Create a Hive UDF with the following characteristics:

- Extract the input phone number and re-format using the pattern (XXX) XXX-XXXX
- Add a `@Description` annotation describing the function

We encourage you to use unit tests to speed up your development. You can either write your own or use the pre-written one from the sample solution. If you use the one from the sample solution, you will need to change the packages to point to your class' packages.

Step 2: Using the UDF

1. Create a directory in HDFS at `/loudacre/udfs` to store the UDF.

```
$ hadoop fs -mkdir /loudacre/udfs
```

2. Create the JAR package

```
$ mvn package
```

3. Put the UDF JAR in HDFS under `/loudacre/udfs`.

```
$ hadoop fs -put target/hive-udf-1.0-SNAPSHOT.jar \
/loudacre/udfs/hive-udf-1.0-SNAPSHOT.jar
```

UDF Placement

Although this step places the UDF in HDFS, this is not always required. When using a UDF with Beeline, the UDF can be placed in the local filesystem of the node running HiveServer2. Placing the UDF in HDFS is required when using Java UDF with Impala.

4. Run the `udf-setup.sh` script.

```
$ $BDDIR/hive-udf/udf-setup.sh
```

This script modifies `/etc/hive/conf/hive-site.xml` to add the `hive.aux.jars.path` property and modifies `/etc/default/hive-server2` to add the `AUX_CLASSPATH` environment. Finally, it restarts the `HiveServer2` service.

5. In the Beeline shell, create the function with the fully qualified path to the UDF class.
6. From the Beeline shell, display the standard and extended descriptions of this function.
7. From the `calldetails` table, select the `to_phone` and `from_phone` columns and format them using the UDF.

Bonus Exercises

Modify the UDF to add another evaluate method that lets you specify the format of the phone number as another parameter in the function. This way, a query could look like:

```
SELECT phonenumberformatterbonus(from_phone,  
"(XXX) XXX-XXXX") from calldetails;
```

Once you are done coding, use the `udf-setup.sh` script in the `hive-udf-bonus` project to add the new UDF JAR to `HiveServer2`.

Write some queries that use the new method and specify your own phone number formats.

This is the end of the Exercise

Optional Hands-On Exercise: Running Queries in Impala

Files and Directories Used in this Exercise

Exercise directory:

```
~/training_materials/bigdata/exercises/impala-queries
```

In this optional exercise, you will query data with Impala.

Step 1: Writing a Basic Query

Impala caches table metadata to speed up queries. As you created and made changes to the tables with Hive, those changes were not communicated to Impala. You will need to manually refresh the table metadata cache.

1. Start the Impala shell.

```
$ impala-shell
```

2. Run the following command:

```
impala> invalidate metadata;
```

3. Write a query to select the row for account ID 42 in the `accounts` table.

Step 2: Using Hue

1. In a web browser, go to `http://dev.loudacre.com:8888`.
2. If prompted for a username and password, use 'training' for both.
3. Click on the Impala icon, which looks like a stylized impala:



4. This displays Hue's Impala interface. You can execute queries by typing them into the Query Editor and clicking 'Execute'.
5. Using Hue, write a query to find how many customers live in the state of 'OR' (Oregon) using the `accounts` table.

Step 3: Running a File Query

The Impala shell can accept a file as a parameter to run the statements it contains.

1. Using your favorite text editor, open the `status.sql` file to write an Impala query. The query should group all device statuses together for accounts that are in the state of 'NV' (Nevada) and count the number of times each status appeared.
2. Run the file with the Impala shell:

```
$ impala-shell -f status.sql
```

Bonus Exercises

Using the foreign key relationships between the `accounts`, `deviceactivations`, and `devicestatus` tables, you can join them together with Impala. The `accounts` and `deviceactivations` tables can be joined together using the `acct_num` column. The `devicestatus` and `deviceactivations` tables can be joined together using the `device_id` column.

Write and execute queries to retrieve the following information:

- Get the count of calls where their call failed, grouped by state and device used
- Get the five most popular devices
- Get the count of devices grouped by status of GPS, WiFi, and Bluetooth

This is the end of the Exercise

Hands-On Exercise: Batch Indexing of Knowledge Base Articles

Files and Directories Used in this Exercise

Data (HDFS):

`/loudacre/kb`

Exercise directory:

`~/training_materials/bigdata/exercises/bulk-search-indexing`

In this exercise, you will index Knowledge Base articles using Cloudera Search and MapReduce.

Step 1: Preparation

1. Close every program except for your terminal and browser.
2. Run the service shutdown script:

```
$ search-shutdown-unused-services.sh
```

3. Copy the Knowledge Base articles to HDFS:

```
$ hadoop fs -put \  
~/training_materials/bigdata/data/kb /loudacre
```

Step 2: Knowledge Base Article Indexing

Before indexing can be run, the schema must be created. The data can then be loaded.

If you make a mistake during this step, run the cleanup script and then start over:

```
$ bigdata-advance-labs.sh bulkSearchCleanup
```

1. If you plan to use the stubs, run the schema generation program to create a template for the Knowledge Base articles. Otherwise, the hints and solution already have the generated files and you can skip running this command.

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \  
--generate kb_search_config
```

Removing Elements in `schema.xml`

When the `schema.xml` file is generated, there are many example `field` and `dynamicField` elements that are created. You can use these as examples to follow, but you will need to remove all of them before creating the collection.

There are also `copyField` elements just below the `fields` section. All of those elements must be removed before creating the collection.

2. Modify the `schema.xml` file in the `kb_search_config` directory to have the following fields, all of which should be both indexed and stored:

- A field of type `string` named `id`
- A content field of type `text_general` named `kb_content`
- A content field of type `text_general` named `kb_title`
- A content field of type `text_general` named `device`
- A date field of type `date` named `created_date`
- A date field of type `date` named `updated_date`
- A version field named `_version_`

3. Create the `instancedir`:

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \  
--create kb_collection kb_search_config
```

4. Create the collection:

```
$ solrctl --zk dev.loudacre.com:2181/solr collection \
--create kb_collection
```

5. Change to either the hints or stubs directory below

~/training_materials/bigdata/exercises/bulk-search-indexing and then complete the `kb-morphlines.conf` Morphline by filling in the TODOs with the following characteristics:

- Complete the `SOLR_LOCATOR` by adding the collection name and the ZooKeeper host
- Place all new Morphline commands after the XQuery command
- Add a Morphline command to convert the `created_date` field to a timestamp from the format `yyyy-MM-dd` to the format `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`
- Add a Morphline command to convert the `updated_date` field to a timestamp from the format `yyyy-MM-dd` to the format `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`
- Add Morphline commands to sanitize and load the data into Solr

6. Create the directory in HDFS to store the Solr files at

`/loudacre/search/kbindex`

```
$ hadoop fs -mkdir /loudacre/search/kbindex
```

7. Run the MapReduce program to index the Knowledge Base articles stored at `/loudacre/kb`. Because we are running on a virtual machine with limited RAM, we specify a single Mapper and a single Reducer.

```
$ hadoop jar \  
/usr/lib/solr/contrib/mr/search-mr-1.1.0-job.jar \  
org.apache.solr.hadoop.MapReduceIndexerTool \  
--morphline-file kb-morphlines.conf --output-dir \  
hdfs://dev.loudacre.com/loudacre/search/kbindex \  
--go-live \  
--zk-host dev.loudacre.com:2181/solr \  
--collection kb_collection \  
hdfs://dev.loudacre.com/loudacre/kb \  
--mappers 1 --reducers 1
```

8. If the MapReduce job fails, you will need to look at the MapReduce program's log output to view the error via the JobTracker's web UI. The URL is <http://dev.loudacre.com:50030>

Step 3: Searching Knowledge Base Articles

1. Open the browser and go to <http://dev.loudacre.com:8983/solr>.
2. In the 'Core Selector' drop down, choose `kb_collection_shard`.
3. Click on 'Query'.
4. In the 'q' box, query for all Knowledge Base articles that have the word 'reboot'.

```
kb_content:reboot
```

5. Click the 'Execute Query' button.
6. Verify that the query returned results. If it does not, check that the MapReduce command completed successfully.

IMPORTANT: Later exercises build on this one. If you were unable to complete this exercise or think you may have made a mistake, run the following command to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh bulkSearchIndex
```

This is the end of the Exercise

Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Files and Directories Used in this Exercise

Data (local):

```
~/training_materials/bigdata/data/chatlogs
```

Exercise directory:

```
~/training_materials/bigdata/exercises/nrt-search-indexing
```

In this Exercise, you will index chat logs using Flume and Cloudera Search.

IMPORTANT: This exercise builds on previous ones. If you were unable to complete the previous exercise or think you may have made a mistake, run the following command (if you have not already done so) to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh bulkSearchIndex
```

Step 1: Chat Log Indexing

Before indexing can be run, the schema must be created and the data loaded.

If you make a mistake during this step, run the cleanup script and start over:

```
$ bigdata-advance-labs.sh nrtSearchCleanup
```

1. If you plan to use the stubs, run the schema generation program to create a template for the Knowledge Base articles. Otherwise, the hints and solution already have the generated files and you do not need to run this command.

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \  
--generate chat_search_config
```

2. Modify the `schema.xml` file in the `chat_search_config` directory to have the following fields, all of which should be both indexed and stored:

- A field of type `string` named `id`
- A field of type `int` named `account_num`
- A field of type `string` named `agent_name`
- A field of type `string` named `category`
- A multivalued field of type `string` named `sender`
- A multivalued field of type `date` named `timestamp`
- A multivalued field of type `string` named `chat_text`
- A version field named `_version_`

3. Remember to remove the example `field`, `dynamicfield`, and `copyfield` elements from `schema.xml`.

4. Create the `instancedir`:

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \  
--create chat_collection chat_search_config
```

5. Create the collection:

```
$ solrctl --zk dev.loudacre.com:2181/solr collection \  
--create chat_collection
```

6. Change into either the `hints` or `stubs` directory below

`~/training_materials/bigdata/exercises/nrt-search-indexing` and complete the `chat-morphlines.conf` Morphline by filling in the TODOs with the following characteristics:

- Complete the `SOLR_LOCATOR` by adding the collection name and the ZooKeeper host

- Add a Morphline command to read in the entire line as JSON
- Add a Morphline command to convert the `timestamp` field to a timestamp from the format `yyyy-MM-dd HH:mm:ss` to the format `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`
- Add Morphline commands to sanitize and load the data into Solr

7. Create a Flume configuration with the following characteristics:

- The source is a spooling directory source that pulls from `/flume/chatlogs_spooldir`.
- The sink is to `org.apache.flume.sink.solr.morphline.MorphlineSolrSink` and is configured to use the Morphline you just created.
- The channel is a Memory Channel with a capacity of 10,000 events and a transaction capacity of 10,000 events.

Remember to watch for any memory issues with the Flume agent.

8. Create the pool directory:

```
$ sudo mkdir -p /flume/chatlogs_spooldir
```

9. Give all users the permissions to write to the `/flume/chatlogs_spooldir` directory:

```
$ sudo chmod a+w -R /flume
```

10. Start the Flume agent:

```
$ flume-ng agent --conf /etc/flume-ng/conf \
--conf-file chatlog-flume.conf \
--name agent1 -Dflume.root.logger=INFO,console
```

11. In another terminal, move the chat files to the spool directory:


```
$ mv ~/training_materials/bigdata/data/chatlogs/* \  
/flume/chatlogs_spooldir
```

Step 2: Searching the Chat Logs

1. Open the browser and go to `http://dev.loudacre.com:8983/solr`.
2. In the 'Core Selector' drop down, choose `chat_collection_shard`.
3. Click on 'Query'.
4. In the 'q' box, query for all Knowledge Base articles that have the word 'battery'.

```
chat_text:battery
```

5. Click on the 'Execute Query' button.
6. Verify that the query returned results. If it does not, check that the Flume and Morphlines processing completed successfully.

IMPORTANT: The next exercise builds on this and all previous exercises. If you were unable to complete any previous exercise or think you may have made a mistake, run the following command to prepare for the next exercise:

```
$ bigdata-advance-labs.sh nrtSearchIndex
```

This is the end of the Exercise

Hands-On Exercise: Building the Application UI

Files and Directories Used in this Exercise

Eclipse project:
application-ui

In this Exercise, you will build the web user interface that ties all of the processed data together.

IMPORTANT: This exercise builds on the previous ones. If you were unable to complete the previous exercises or think you may have made a mistake, run the following command (if you have not already done so) to prepare for this exercise before continuing:

```
$ bigdata-advance-labs.sh nrtSearchIndex
```

The Web Development team has been working to create a dashboard that consumes the data sets you have created. They are using JSP, HTML, JavaScript, and CSS for the dashboard. They need you to write the queries that bring in the data from Impala and MySQL to the web application.

Step 1: Setting Up the Web Server

1. Run the following command:

```
$ sudo mount --bind \  
$BDDIR/application-ui/src/main/webapp \  
~/apache-tomcat-7.0.52/webapps/ROOT
```

This command mounts the exercise as the root Web application for Tomcat. This allows you to work around some issues with symlinking on our VM.

2. Start the web server:

```
$ ~/apache-tomcat-7.0.52/bin/startup.sh
```

JSP Pages and Data Sources

All changes to JSP pages are recompiled automatically. The Impala data source is already configured for you.

Step 2: Setting Up Impala

In a previous exercise, you created a Hive UDF to format phone numbers. This Hive UDF can also be used in Impala, but must first be registered.

We will use this UDF to format the phone numbers in the web application.

1. In the terminal, change to the exercise directory
`~/training_materials/bigdata/exercises/application-ui.`
2. Run the following command to register the UDF:

```
$ impala-shell -f add-phonenummer-formatter.sql
```

Note: If the Impala daemon is restarted, you may need to re-run this command.

3. The command will run a query using the UDF. Verify that the query did not generate an error, and that the formatting follows the pattern: (XXX) XXX-XXXX.

The UDF will be registered as `PHONENUMBERFORMATTER` and takes a single string, the phone number, as the parameter.

Step 3: Becoming Acquainted with the Application

Open your browser to `http://dev.loudacre.com:8080/solution/` and spend a few minutes becoming acquainted with the capabilities of the finished application before you continue with the exercise. We suggest that you begin by searching for account number 1, since this is an active account with multiple devices and a transcript from a chat session with a Loudacre support representatives. We

also recommend searching the Knowledge Base for common problems related to mobile phones, such as “reboot” or “battery”.

Step 4: Completing the Application

Below the `src/main/webapp` subdirectory of this project are the `stubs`, `hints`, and `solutions` directories for the web application. Depending on which directory you modify, you will need to use the corresponding URL in the browser. For example, if you modify the `hints` directory, you will need to use the `http://dev.loudacre.com:8080/hints/index.jsp` URL.

The web application uses JSP pages, many of which run queries with Impala. For example, the following code will select all records from the `accounts` table:

```
<sql:query var="activations" dataSource="jdbc/impala">
SELECT * FROM accounts
</sql:query>
```

Some pages make use of a parameter in the `WHERE` clause. The parameter values come from data sent when the browser requests the page. For example, the following code will select a single account from the `accounts`:

```
<sql:query var="activations" dataSource="jdbc/impala">
SELECT *
FROM accounts
WHERE acct_num=<%= request.getParameter("acct_num") %>
</sql:query>
```

Each main page in the application uses JSP include statements to include smaller, more modularized pages (in the `tiles` subdirectories) that perform a specific function such as displaying the account details, device status, or a chart illustrating call status.

While the majority of the application code is provided, you must complete a few unfinished sections. Open each of the files below, in your choice of either the `hints`

or `stubs` subdirectories, and complete the steps directed by the TODO comments in those files. As with earlier exercises, you may look at the corresponding files in the `solution` subdirectory for guidance.

1. `accounts/tiles/account-info.jsp` (this is the “Account Details” section you see in the upper-left corner when looking at a specific account)
2. `accounts/tiles/device-status-info.jsp` (this is the “Status Info” section you see on the left side of the page when looking at the details of a customer’s device)
3. `accounts/tiles/device-account-summary.jsp` (this is the “Account Summary” section you see in the upper-left corner when looking at the details of a customer’s device)
4. `articles/tiles/article-display.jsp` (this is the page you see when looking at a specific Knowledge Base article)
5. `transcripts/tiles/transcript-display.jsp` (this is the page you see when looking at a specific chat transcript)

If you worked on the `stubs` version, you must complete the tasks described in the TODO comments of the following files before continuing on to “Step 5: Testing the Changes”. If you worked on the `hints` version of the exercise, you may skip to “Step 5: Testing the Changes” now.

- `tiles/most-popular-devices.jsp`
- `chart-data/call-status-json.jsp`
- `articles/tiles/article-search.jsp`

Step 5: Testing Your Changes

1. Start the browser and go to the URL that corresponds to the directory you modified. For example, the `stubs` directory would be `http://dev.loudacre.com:8080/stubs/index.jsp`.

2. Go to the account search results page by typing in '1' in the 'Account #' field and clicking 'Search'.
3. Click the link for the first account in the list.
4. Verify that the data shows up as expected and that no errors are shown.
5. Click on one of the devices associated with this account.
6. Verify that the data shows up as expected and that no errors are shown.
7. You should perform additional checks, such as searching the chat transcripts or Knowledge Base Articles, to verify the specific changes you made to files earlier.

Creating a WAR File

Java web applications are distributed as a WAR (Web Archive) file. To create a WAR file with Maven, you would run the command:

```
$ mvn war:war
```

This will create a WAR file in the target directory, which you could then deploy to an application server as you would for any other WAR file.

Step 5: Visualizing the Issues

Loudacre has recently experienced a high volume of support requests. The initial impression was that Loudacre had just expanded into new markets and added more customers, but further analysis showed that the increase in support traffic far outpaced the increase in new customers.

The web application can reveal a pattern that may explain this increase.

1. From the main page, look at the 'Most Popular Devices' table to see which devices are most popular based on recent activations.
2. Click one of the links in the table to view the dashboard and see visualizations of aggregate device statistics for each model:

- 'Battery Percent' shows average battery power remaining
- 'Call Breakdown' shows the number of successful, dropped, and failed calls
- 'CPU Load' shows the average processor load
- 'Device Temp' chart shows the internal temperature, in Celsius

Use the data presented in these charts to find any anomalies for a single device. Remember that these are averages for every device, so anomalies for a single device at a single moment in time are balanced out in aggregate.

- Do you have a theory about why Loudacre's support call volume has recently increased? Discuss your idea with the instructor following the exercise.

This is the end of the Exercise