Lecture 12 - Sqoop

BDAT 1002

Apache Sqoop

Motivation

- Assume you work for a company and the your database is growing too fast
- You decide to migrate to Hadoop because you do not have enough storage
- But at the same time, you do not want to start "fresh"
- You want to migrate your structured data into Hadoop
  - To be able to do analysis

Motivation

- So we need a way to extract data from RDBSM
- Sqoop is an open source tool designed to move data from structured database to Hadoop and vice versa
  - Example MySQL, Oracle ←→ Hadoop

Start MySQL Database

- You can use Sqoop with any JDBC compliant database
  - Sqoop can work with any database that has the supported JDBC driver
  - All the major RDBMS have JDBC support
- We will use MySQL
  - Already installed in our Cloudera version
  - Similar to Hive

Interview Question

- MySQL already has tools like mysqldump to extra data from a table to a delimited text file

  • Oracle expdp / Impdp datapump

  - Similar tools exist in other RDBMS
- In this case, what is the use of Sqoop?
  - Why not use the tools already available?

Benefits of Sqoop

- With Sqoop you can achieve parallelism with your extract Oracle
  - Leverage MapReduce framework to extract data
  - Sqoop will create a map only MapReduce job with multiple mappers ⇒ default 4 (m=4)
  - Each mapper extracts a portion of the table and puts the contents directly into HDFS or even in a Hive table

Benefits of Sqoop

- We can also create Sqoop jobs to import data from your database in an ***incremental fashion***
  - for example, create jobs to import data every night into your Hadoop cluster
  - More details later in the lecture

Exercise 1: Sqoop Imports

Basic Sqoop import

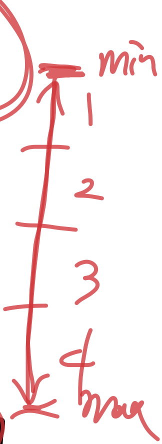
- Here is a basic import, by default it will run 4 mappers

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root -
-password cloudera --table stocks
```

"HDFS"

Interview Question

- How is data distributed between mappers?
  - Sqoop import looks at the primary key column in the table
  - Sqoop then does a min and max value on the column
  - Data is then divided into 4 parts
- Try to find this process in the output log of your Sqoop job



where is the local log file ?

HDFS or local filesystem

Basic Sqoop import

- By default, the output will be same in a directory with the name same as the table name in the user home directory
- Check out the output

```
[cloudera@quickstart ~] hadoop fs -ls stocks
[cloudera@quickstart ~] hadoop fs -cat stocks/part-m-00001
```

- Note that all the column values are delimited by commas by default

Basic Sqoop import

- Let's say you have a very big table and the 4 mappers is not enough parallelism
- You can increase the number of mappers by using the -m option
- And in fact we may want to reduce the number of mappers as in our case for a small table

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks -m 2
```

Basic Sqoop import

- We can also change the default directory using -target-dir option
- Run the command again

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks -m 2 --target-dir
/BDAT1002/sqoop/stocks_nmaps
```

(this is a table on the MySQL. option.

- HDFS root directory에 저장이 된다.

Basic Sqoop import

- Take a look under the directory and list the content of one of the files

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks -m 2 --target-dir
/BDAT1002/sqoop/stocks_nmaps
```

Basic Sqoop import - Delimitation

- What if I don't want to use comma as my delimiter?
- What if you want a space or tab delimitation?
- And each column to be surrounded by ""

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera  --table stocks -m 1 --target-dir
/BDAT1002/sqoop/stocks_terminated --fields-
terminated-by '\t' --enclosed-by '"'
```

--fields-terminated-by '\t'
--enclosed-by '"'

Sqoop import - Delimitation

- And take a look at the output to confirm

```
[cloudera@quickstart ~] hadoop fs -ls
/BDAT1002/sqoop/stocks_terminated
[cloudera@quickstart ~] hadoop fs -cat
/BDAT1002/sqoop/stocks_terminated/part-m-00000
```

Sqoop import - Delimitation

- So far we have imported all the columns from the table
- But let's say we want to extract only *selected* columns and *certain* rows
  - Use columns and where option

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks --columns
"symbol,name,trade_date,volume" --where "id > 5" -m
1 --target-dir /BDAT1002/sqoop/stocks_selective
```

Exercise 2: Sqoop File Formats

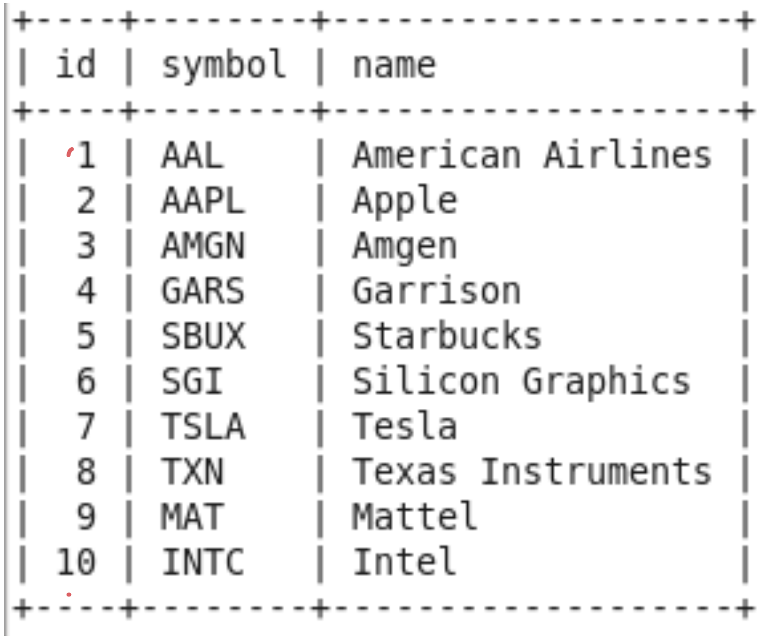
Introduction

- We want to do some costume import instructions and how to use Sqoop to import files in different file formats

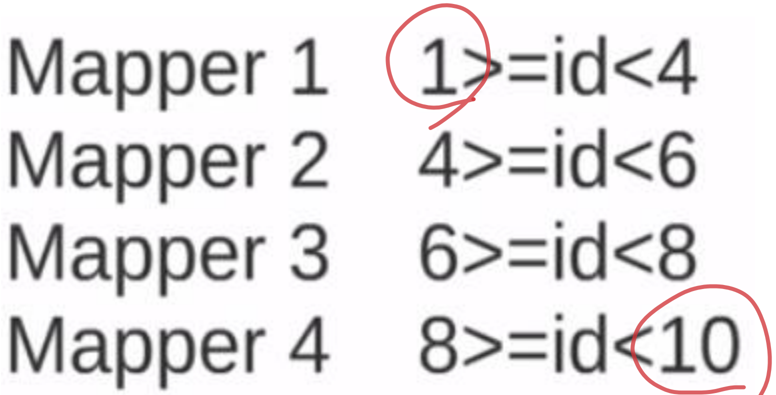
Data boundary

- Remember, Sqoop at runtime looks at the min and max value of a the primary key column

반드시 있어야 Mapper가 되나?



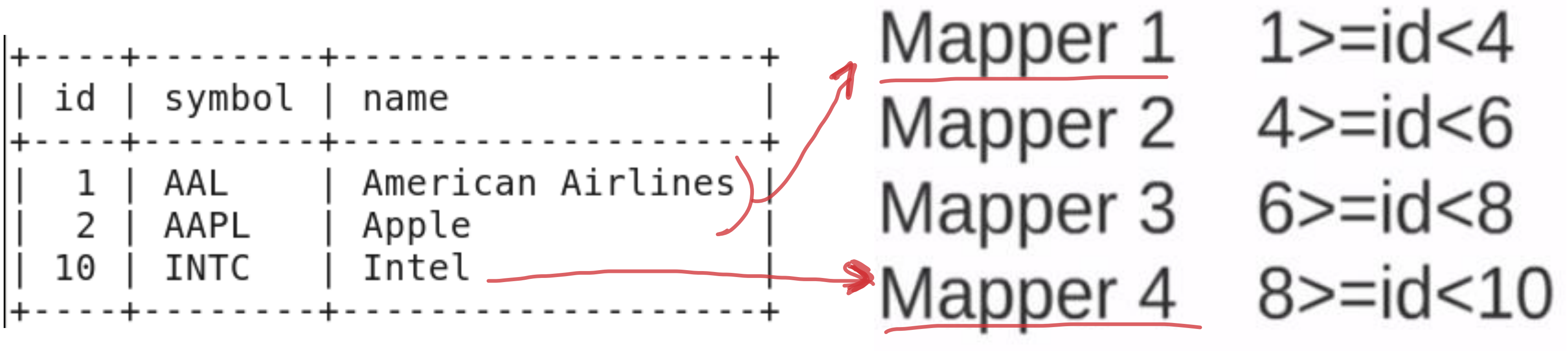
```
+----+--------+--------------------+
| id | symbol | name               |
+----+--------+--------------------+
|  1 | AAL    | American Airlines  |
|  2 | AAPL   | Apple              |
|  3 | AMGN   | Amgen              |
|  4 | GARS   | Garrison           |
|  5 | SBUX   | Starbucks          |
|  6 | SGI    | Silicon Graphics   |
|  7 | TSLA   | Tesla              |
|  8 | TXN    | Texas Instruments  |
|  9 | MAT    | Mattel             |
| 10 | INTC   | Intel              |
+----+--------+--------------------+
```



Mapper 1	1>=id<4
Mapper 2	4>=id<6
Mapper 3	6>=id<8
Mapper 4	8>=id<10

Data boundary

- But what if our table is sparse
  - Sqoop looks at only the min and max values, NOT the actual number of records



```
+----+--------+-------------------+
| id | symbol | name              |
+----+--------+-------------------+
|  1 | AAL    | American Airlines |
|  2 | AAPL   | Apple             |
| 10 | INTC   | Intel             |
+----+--------+-------------------+
```

Mapper 1	1>=id<4
Mapper 2	4>=id<6
Mapper 3	6>=id<8
Mapper 4	8>=id<10

Data boundary

- We need to instruct Sqoop to use a column that has a pretty good data distribution to decide the data boundary
  - Use --split-by option

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password --table stocks --split-by volume --target-
dir /BDAT1002/sqoop/stocks_conds
```

* primary key 대신에 volume 으로 데이터를 분산할 수 있다.

Data boundary

- Note that in the output log, you will see that the condition is now based on volume rather than the default id column
- Look at the output

```
[cloudera@quickstart ~] hadoop fs -ls
/BDAT1002/sqoop/stocks_conds
```

Data boundary

- Note that in the output log, you will see that the condition is now based on volume rather than the default id column
- Look at the output

```
[cloudera@quickstart ~] hadoop fs -ls
/BDAT1002/sqoop/stocks_conds
```

Joining Tables

- Assume you have two tables with a relationship, and you like to import a merged version of the table
  - Use --query option and supply your query
- One caveat to this is that we also have our volume selection for mappers
  - You need to provide a placeholder for Sqoop to inject the additional where condition at runtime

Joining Tables

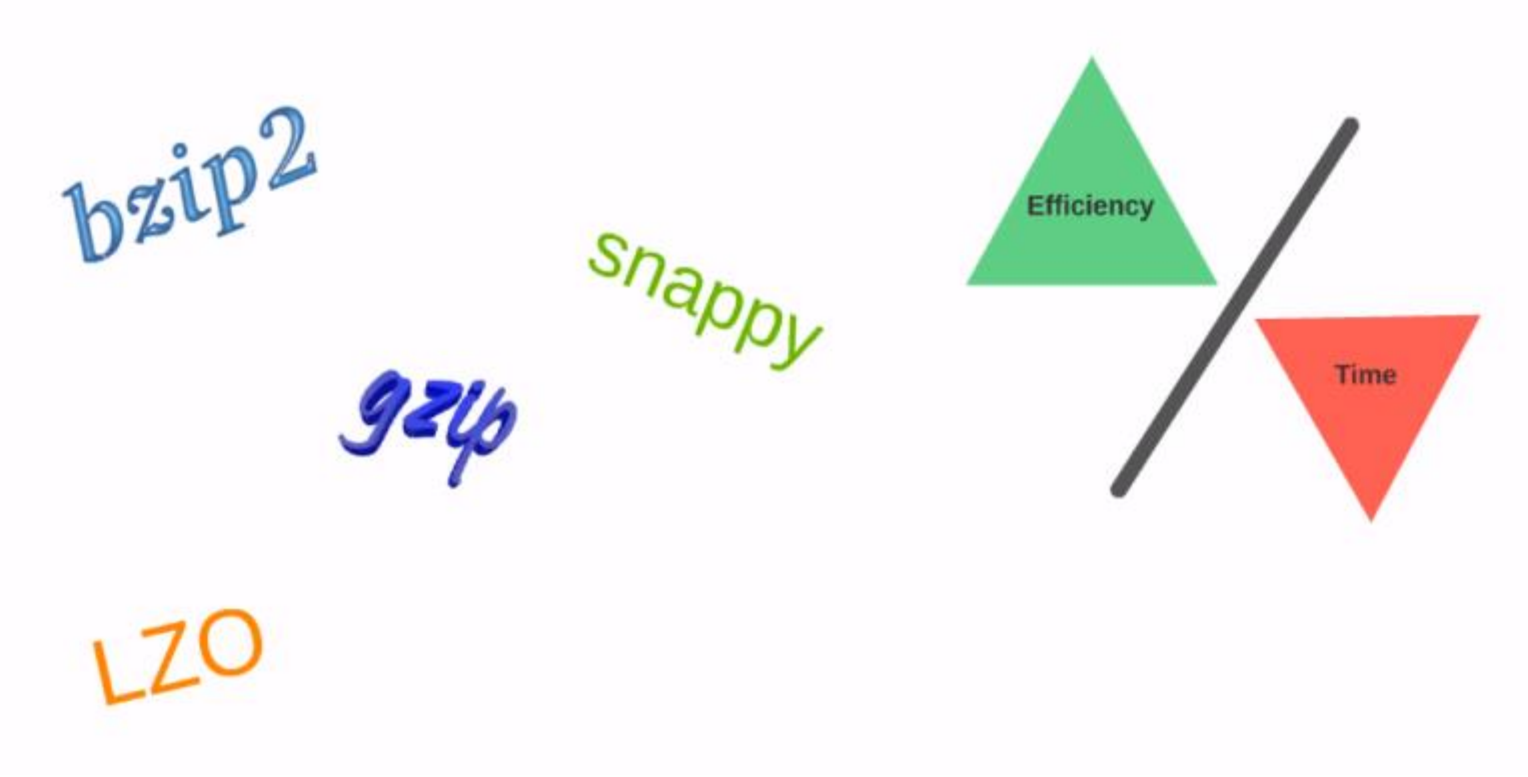
- In the following instruction, $CONDITOINS is a placeholder and it will be replaced by the data boundary restriction for mapper
  - In this case volume

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --query 'SELECT a.id, a.name,
a.trade_date, a.volume, b.dividend_amount FROM stocks
a INNER JOIN dividends b ON a.symbol = b.symbol WHERE
a.id > 2 and $CONDITIONS' --split-by a.volume --
target-dir /BDAT1002/sqoop/stocks_join_conds
```

Compressing Files

- Compressing saves space
- What about performance of compressed files?
  - Uncompressing files is an overhead
  - But smaller files increase bandwidth
- The benefit of network congestion is dwarfed by the downside of overhead

Compression Algorithms



Compressing Files

- To compress files, we need a codec program
  - Stands for **co**mpression, **dec**ompression
- Hadoop has codecs for several algorithms out of the box
- We can do compression on
  - Input dataset
  - Output dataset
  - Output of mappers

Sqoop: Compressed File Format

- What if you want a compressed file
  - Use --compress option
  - Will give you a gzip format file by default

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks --compress -m 2 --
target-dir /BDAT1002/sqoop/stocks_comp
```

Sqoop: Compressed File Format

- Look at the output files

```
[cloudera@quickstart ~] hadoop fs -ls
/BDAT1002/sqoop/stocks_comp
[cloudera@quickstart ~] hadoop fs -cat
/BDAT1002/sqoop/stocks_comp/part-m-00000.gz
```

- Look at the output files

Sequence File Format

- Compressed files cannot be seamlessly “broken“ up for MapReduce jobs
- Sequence files allow you a solution to this problem
- We will not go into details in this lesson

Sequence File Format

- To import a table in sequence-file format use --as-sequencefile format option

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks --as-sequencefile -m
2 --target-dir /user/hirw/sqoop/stocks_seq
```

Sequence File Format

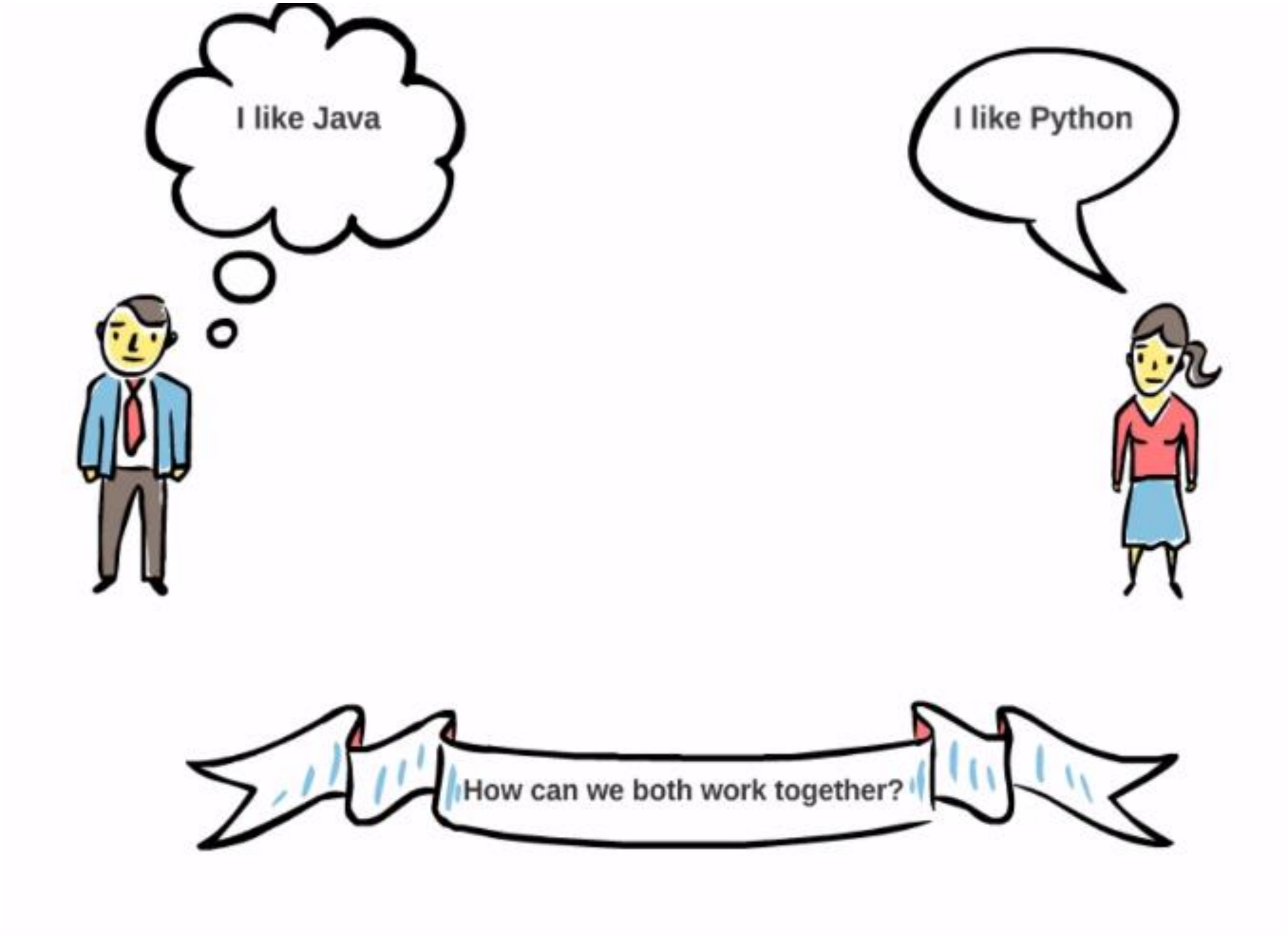
- Look at the target directory, you will see two files
- And again it will not be readable because it is in sequence file format

```
[cloudera@quickstart ~] hadoop fs -ls
sqoop/stocks_seq
[cloudera@quickstart ~] hadoop fs -cat
sqoop/stocks_seq/part-m-00000
```

Avro Format Motivation

- Certain organizations allow their developers to use different languages
  - Others do not
  - Pros and cons to each method
- Let's say you write a MapReduce program in Java and it outputs an object in Sequence Format
- Now someone else wants to use Python to read that
- How would you do that?

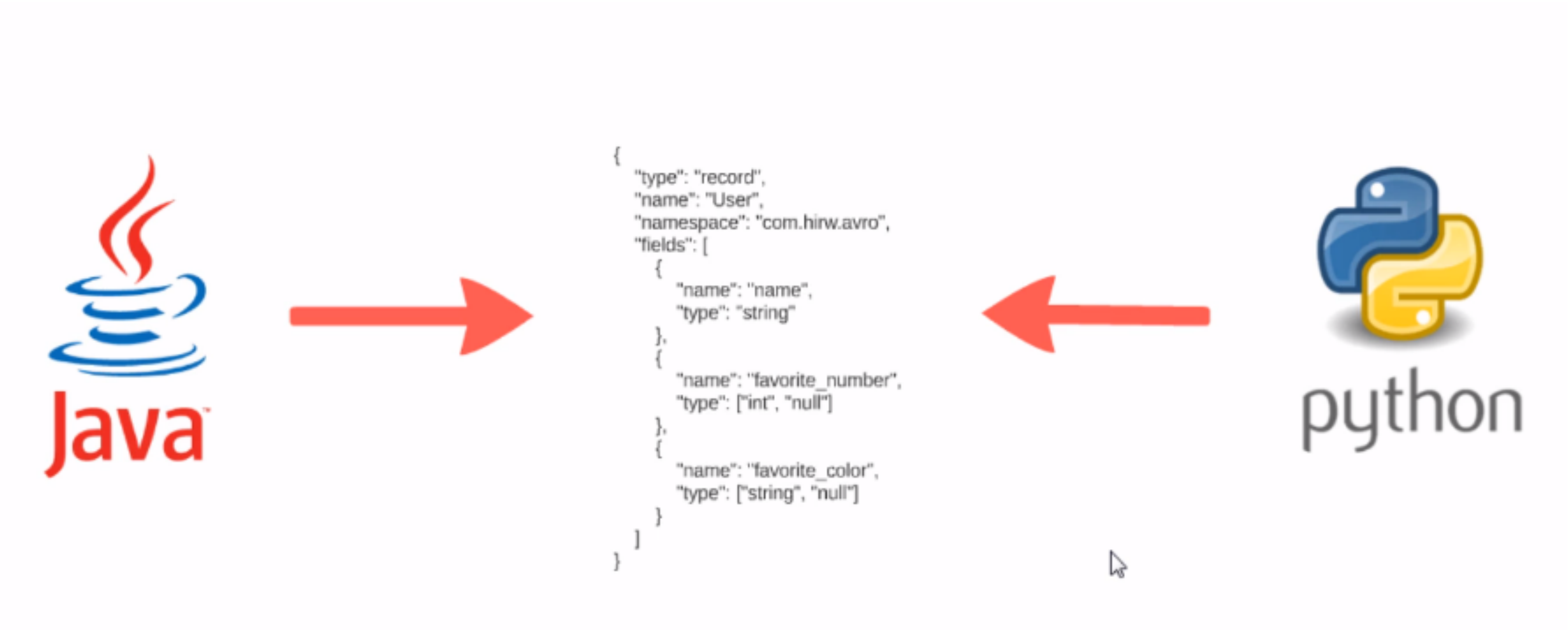
Avro Format Motivation



Avro Format Motivation

- We can use a language neutral data serialization system like Avro
- Avro is a high level Apache project
- It is used in places where you need *inter-language portability*
- The basic idea is that the object to be read inter-changeably is first given a schema or format
  - Usually in JSON format

Avro Format Motivation



```
{
    "type": "record",
    "name": "User",
    "namespace": "com.hirw.avro",
    "fields": [
        {
            "name": "name",
            "type": "string"
        },
        {
            "name": "favorite_number",
            "type": ["int", "null"]
        },
        {
            "name": "favorite_color",
            "type": ["string", "null"]
        }
    ]
}
```

Avro Format Motivation

- Once the schema is known, we can use certain libraries to read and write data files in several different formats for each language

Avro File Format

- For Avro file format use --as-avrodatafile

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username
root --password cloudera --table stocks --as-
avrodatafile -m 2 --target-dir
/BDAT1002/sqoop/stocks_avro
```

- List the directory and look at the file
  - Again file will not be in readable format

Summary

- In this section, we saw
  - how to provide costume join queries to input data from tables
  - How to do a custom split by data to distribute the rows evenly for MapReduce jobs
  - How to import data in compressed, sequence, avro file formats

Exercise 3: Sqoop Jobs and Incremental Imports

Introduction

- So far our inputs were one off inputs
  - This means the imports brought everything in the table
  - There is no *easy* way to import only records from last time you imported or what we call “incremental” imports

Introduction

- One way to do this using classic import is as follows

```
[cloudera@quickstart ~] sqoop import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks --columns
"symbol,name,trade_date,volume" --where "id > 10" -m
1 --target-dir /BDAT1002/sqoop/stocks_selective
```

- This is not ideal
  - You can't automate the process
  - Therefore, error-prone

Incremental Imports

- You can create a Sqoop job to do incremental imports

```
[cloudera@quickstart ~] sqoop job --create
incrementalImportJob -- import --connect
jdbc:mysql://localhost/stocks_db --username root --
password cloudera --table stocks --target-dir
/BDAT1002/sqoop/stocks_append --incremental append --
check-column id
```

- We are saying, only import new rows from last import based on check-column
  - In this case the id column

Incremental Imports - Steps

- Create the sqoop job first
- Now to see a list of Sqoop jobs use the instruction:

```
[cloudera@quickstart ~] sqoop job --list
```

- If you want to know the details of the job execute:

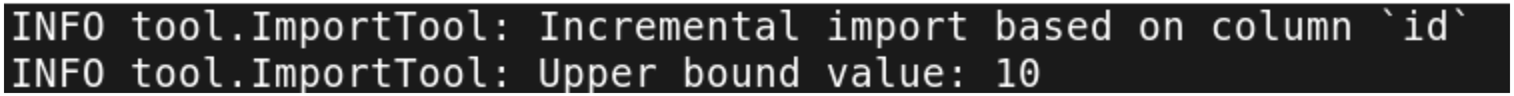
```
[cloudera@quickstart ~] sqoop job --show
incrementalImportJob
```

Incremental Imports

- Now execute the job

```
[cloudera@quickstart ~] sqoop job --exec
incrementalImportJob
```

- Notice the log output



```
INFO tool.ImportTool: Incremental import based on column `id`
INFO tool.ImportTool: Upper bound value: 10
```

Incremental Imports

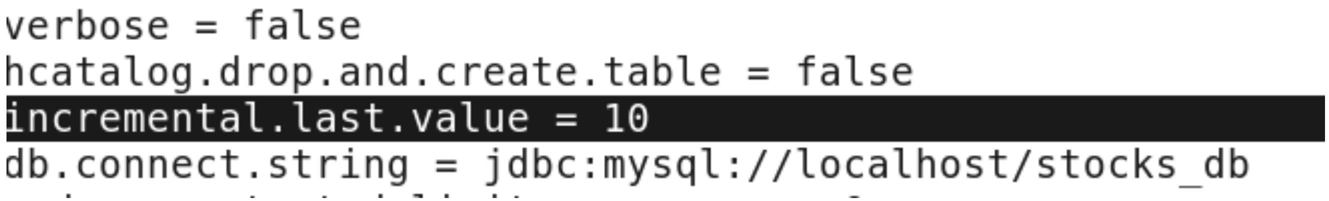
- Look at the output file

```
[cloudera@quickstart ~] hadoop fs -ls
sqoop/stocks_append
[cloudera@quickstart ~] hadoop fs -cat
sqoop/stocks_append/part-m-00000
```

Incremental Imports

- Now look up the details of the job again

```
[cloudera@quickstart ~] sqoop job --show
incrementalImportJob
```



```
verbose = false
hcatalog.drop.and.create.table = false
incremental.last.value = 10
db.connect.string = jdbc:mysql://localhost/stocks_db
```

- Sqoop actually records the maximum value fromm the id column
  - When we execute the job again, only records with id > 10 will be brought

Incremental Imports

- Let’s test this concept by importing a few more records into the stocks table and then running the job again
- You should notice that only three records are retrieved
- If you look up the details (show option), you will that the last value parameter is now 13
- Finally take a look at the output directory

Incremental Imports – Updates

- Note that incremental imports only work when you have an *incremental unique column* in your table
- You will not capture the rows that were updated between incremental imports
- For example, let's say you update three rows → not captured

Incremental Imports - Updates

- What if you want to capture updates?
- Slide modification
  - Use --incremental *lastmodified* option
  - And --check-column with a column that has modification record

```
[cloudera@quickstart ~] sqoop job --create
incrementalImportModifiedJob -- import --connect
jdbc:mysql://localhost/stocks_db --username root -
-password cloudera --table stocks --target-dir
/BDAT1002/sqoop/stocks_modified --incremental
lastmodified --check-column updated_time -m 1 --
append
```

Incremental Imports - Update

- Create the sqoop job first
- Now to see a list of Sqoop jobs use the instruction:

```
[cloudera@quickstart ~] sqoop job --list
```

- Now look at the details of the job

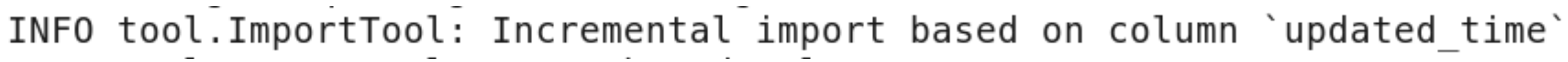
```
[cloudera@quickstart ~] sqoop job --show
incrementalImportModifiedJob
```

- Execute job

Incremental Imports - Update

- Execute job
  - You will notice update is based on column "updated_time"

```
[cloudera@quickstart ~] sqoop job --list
```



```
INFO tool.ImportTool: Incremental import based on column `updated_time`
```

- Look at the directory and files

Incremental Imports - Update

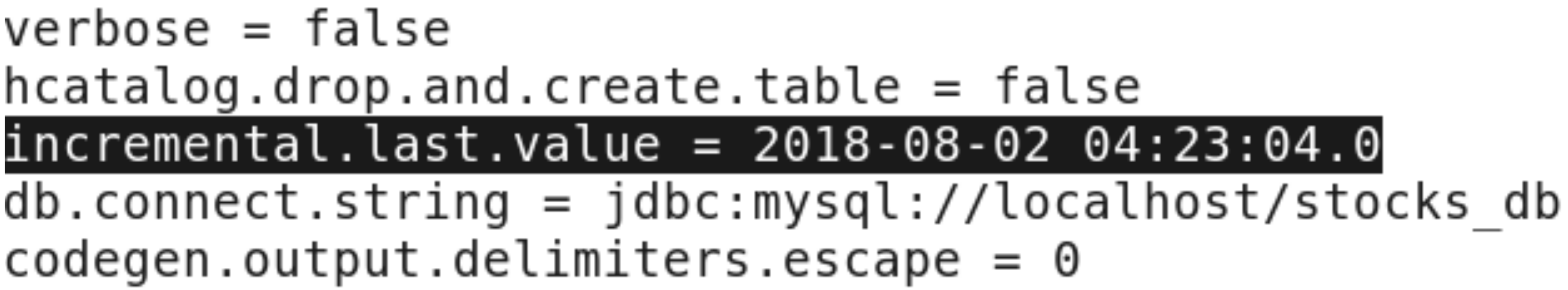
- Now update the stocks table and add three more records

```
[cloudera@quickstart ~] UPDATE stocks SET volume =
volume+100, updated_time=now() WHERE id IN
(10,11,12);
```

Incremental Imports - Update

- Now look at the details of the incremental job

```
[cloudera@quickstart ~] sqoop job --show
incrementalImportModifiedJob
```



```
verbose = false
hcatalog.drop.and.create.table = false
incremental.last.value = 2018-08-02 04:23:04.0
db.connect.string = jdbc:mysql://localhost/stocks_db
codegen.output.delimiters.escape = 0
```

- If we now execute the job, it will import all the jobs with update time greater than above

Incremental Imports - Consolidation

- Note that even though we only had 16 records, we have extra 3 records
- We say that we don't have a consolidated table in HDFS anymore
- How can we consolidate the results?
- We can use the sqoop merge command

Incremental Imports - Merge

- First we need to create a jar file
- Follow instructions in the exercise file

Summary

- In this section, we learned:
  - How to create incremental sqoop jobs with append and last-modified options
  - We also saw how to merge data between two imports

Exercise 4: Hive Imports

Introduction

- We want to see how to import files from HDFS to MySQL
- How to create a Hive table and load it with data directly from a MySQL table using Sqoop
- Follow instructions in exercise file uploaded to blackboard