Overview of Hive, Flume and Sqoop

Agenda

- What is Hive
- Different kinds of table supported in Hive
- What is a Partition table and its benefits
- When to create bucketed table
- Different file format supported in Hive
- How to create a view in Hive
- Importing data into HDFS by using Sqoop and Flume

What is Hive?

- Hive is data warehouse tool for Hadoop
- It facilitates easy data summarization and ad-hoc queries
- It helps in data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis
- provides a mechanism to project structure onto data stored in HDFS and query the data using a SQL-like language called HiveQL
- Hive can not store the actual data, it stores only meta data i.e Schema of the table
- Data will be store in platform such as HDFS

What is Hive?

- Hive is not a full database, Hive tables don’t support update and delete Operations

- However, ACID table in Hive supports update and delete Operations

- Initially, In hive there was no way to define primary and foreign keys

- Generate new tables from queries or output query results to files

- You can connect to Hive using a JDBC command-line tool, such as Beeline, or using an JDBC/ODBC driver with a BI tool, such as Tableau

Query engines in Hive

- There are 3 different query engines supported in Hive such as Tez, MapReduce and Spark

- Tez is of type of MPP while MR and Spark are of type Parallel distributed computing

- Hive queries are translated into corresponding jobs and will be run on the cluster to generate the output.

- So, Hive queries have higher latency, due to the start-up overhead for running jobs.

Hive MetaStore

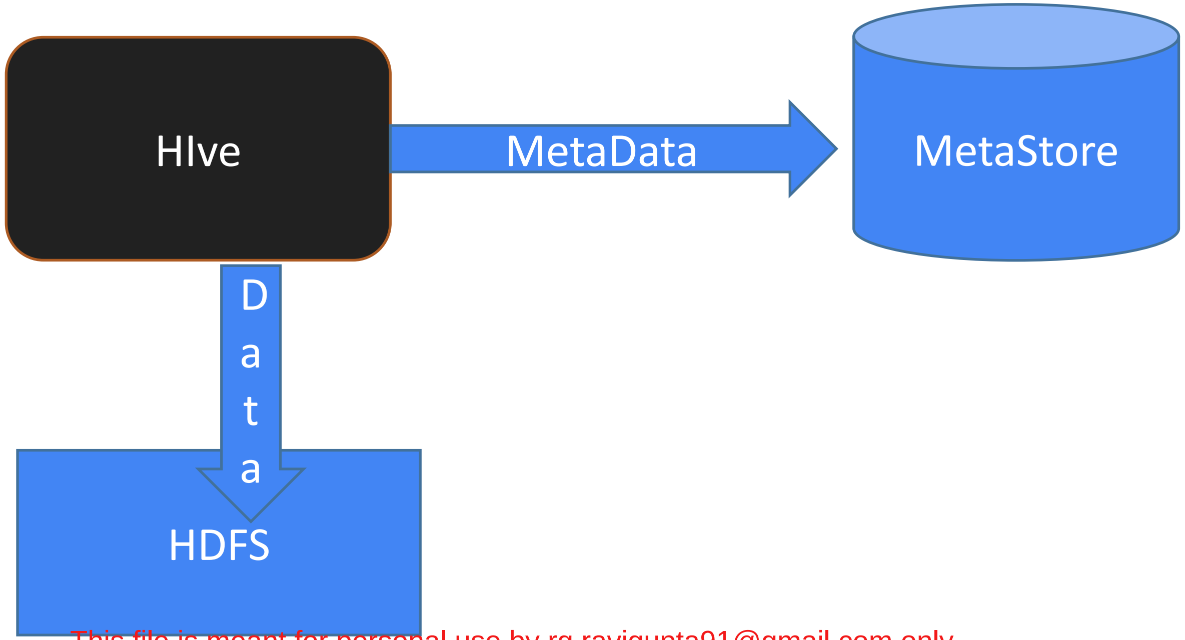
- Hive metastore (HMS) is a service that stores metadata related to Apache Hive and other services

- Hive metastore uses RDBMS, such as MySQL or Postgres

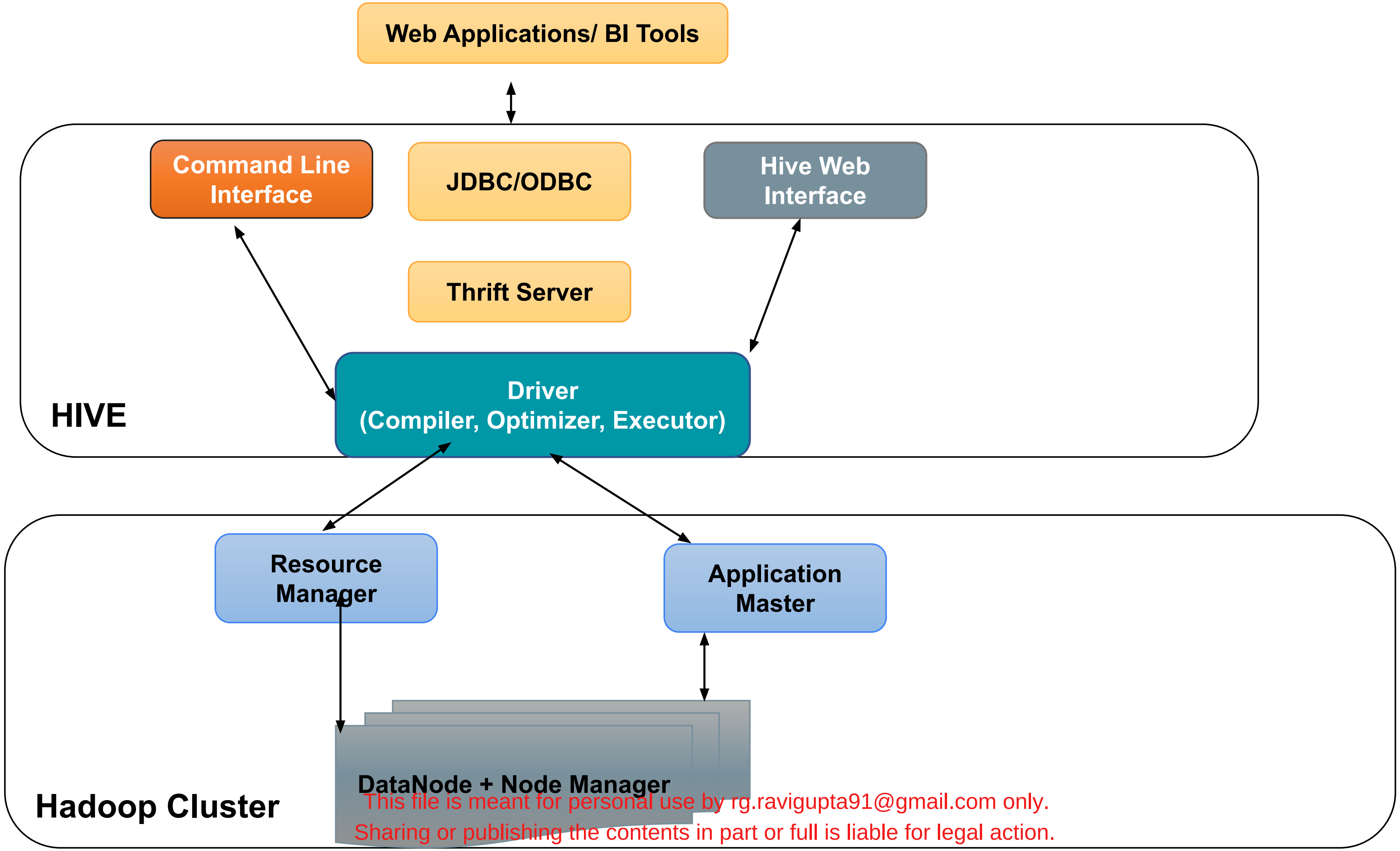
- Impala, Spark, Hive, Presto and other services share the same metastore.

Hive MetaStore

- The HMS includes the following Hive metadata about tables that you create:
- A table definition
- Column names
- Data types
- Comments in a central schema repository



Hive Architecture



- In Hive 3.0 ACID tables have become the default table
- Hive supports primary key and foreign key starting 3.0 version
- In Hive 3.0 , Materialized view is also supported
- Hive also supports creating table for external data sources such as S3, GCS, Azure Data Lake gen2 , DynamoDB, Cassandra, MongoDB, Kafka, Elasticsearch and other data warehouses
- YARN allocates resources for applications across the cluster and enables authorization for Hive jobs in YARN queues.

Need for Partitioned Table

- A Hive query reads the entire dataset even if there is a WHERE clause on it.
- I/O is the main bottleneck when Hive triggered jobs run over data stored on disk.
- Reducing I/O improves query performance.

What is Partitioned Table?

- Data is split across multiple partitions based on or more table columns.

- Hive table data is stored in directories.

- When Partitioned, data will be stored in sub directories.

- **Hive Directory Structure for Partitioned Tables:**
- /user/hive/warehouse/sales.db
- /user/hive/warehouse/sales.db/Year=2021
- /user/hive/warehouse/sales.db/Year=2020
- /user/hive/warehouse/sales.db/Year=2019
- /user/hive/warehouse/sales.db/Year=2018

Create Partition Table Syntex

```
CREATE TABLE Sales_dailypartitioned(
productName STRING,
amount float,
customerid INT
)

PARTITIONED BY (Year String) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t’ STORED AS TEXTFILE;
.
```

Load data into Partitioned Table

Loading data into a Managed Partition Table:

```
LOAD DATA INPATH ‘/sales/sales_daily_a_year_2021 ' OVERWRITE INTO
TABLE sales_dailypartitioned PARTITION (Year=2021);

LOAD DATA INPATH ‘/sales/sales_daily_a_year_2020' OVERWRITE INTO
TABLE sales_dailypartitioned PARTITION (Year=2020);
```

Partitioned Table

Alternate way of inserting data into Partitioned tables:

```
INSERT OVERWRITE TABLE dailypartitioned PARTITION (Year=2008)
SELECT * FROM daily;

FROM daily d
INSERT OVERWRITE TABLE dailypartitioned
PARTITION (Year=2007)
SELECT * LIMIT 10
```

Querying Partitioned Table

```
SELECT * FROM dailypartitioned WHERE Year=2020;
```

- Only contents of one directory (partition 2020) will be read to retrieve results
- By default, Hive runs in “strict” mode.

- Prohibits queries of partitioned tables without a WHERE clause on the partitioned columns.

```
SET hive.mapred.mode=strict;
```

Querying Partitioned Table

```
SELECT * FROM sales_dailypartitioned WHERE Year=2020;
```

```
FAILED: SemanticException [Error 10041]: No partition predicate found for
Alias " sales_dailypartitioned " Table " sales_dailypartitioned “
```

```
SET hive.mapred.mode=nonstrict;
```

```
SELECT * FROM sales_dailypartitioned WHERE Year=2020;
```

Static Partitioned Table

- If data is already segmented and stored in different folders
- Data ingestion tools imports the data into folders
- No need to specify the location of the file during the table creation.

```
CREATE EXTERNAL TABLE dailyexternal(
exchangename STRING,
symbol STRING,
date STRING,
)
PARTITIONED BY (Year String) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t’ LINES TERMINATED BY '\n';
```

Loading data into Static Partitioned Table

ALTER TABLE dailyexternal ADD PARTITION(Year=2010) LOCATION '/hive/nyse/2010';

ALTER TABLE dailyexternal ADD PARTITION(Year=2009) LOCATION '/hive/nyse/2009';

Advantage of External Static Partitioned Table

- Share data between tools.
- Get performance benefits.

- Flexibility of using our own naming conventions for directories.
  - You can store historical data on cheap storage such as Amazon S3
  - Create an External Partitioned table and point its partitions to S3 location and HDFS.
  - You can query active data on HDFS and historical data on S3 separately or together from a single table.

- External Partitioned tables are most common in production scenarios

Dynamic Partitioned Table

- Eliminates ‘hard coding’ partition value in query.

```
INSERT OVERWRITE TABLE dailypartitioned
PARTITION (Year)
SELECT …, Year
FROM daily;
```

- Data has to be loaded from a query itself and cannot use LOAD DATA from directory.

- Intermediate table which has data to be loaded into the partitioned table should be present.

Loading data into Dynamic Partitioned Table

```
INSERT OVERWRITE TABLE dailypartitioned
PARTITION (Year)
SELECT exchangename, symbol, date, opencloseadj, highlow, volume,
YearValue FROM daily_nonPartition;

set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition.mode=nonstrict;
set hive.mapred.mode=nonstrict;
hive.exec.max.dynamic.partitions;
hive.exec.max.dynamic.partitions.pernode;
```

Bucketing Table

- Buckets in hive is used in segregating of hive table-data into multiple files

- The division is performed based on Hash of bucketing columns
- In Hive, we have to enable buckets by using

the set.hive.enforce.bucketing=true;

- You can create buckets on only one column

- You can create Hive buckets on Hive managed tables or hive external tables

- The high cardinality field in the bucketing results in even distribution of data among created buckets.

How to create Bucketing Table

```
CREATE TABLE order_bucketed (
   order_id INT
    order_date STRING
    order_status STRING
    year STRING
    month STRING)

CLUSTERED BY order_id  INTO 4 BUCKETS
```

Benefits of Hive Bucketing Table

- Optimized Hive tables
- Enables more efficient queries
- Optimized access to the table data
- Evenly distribute the data.

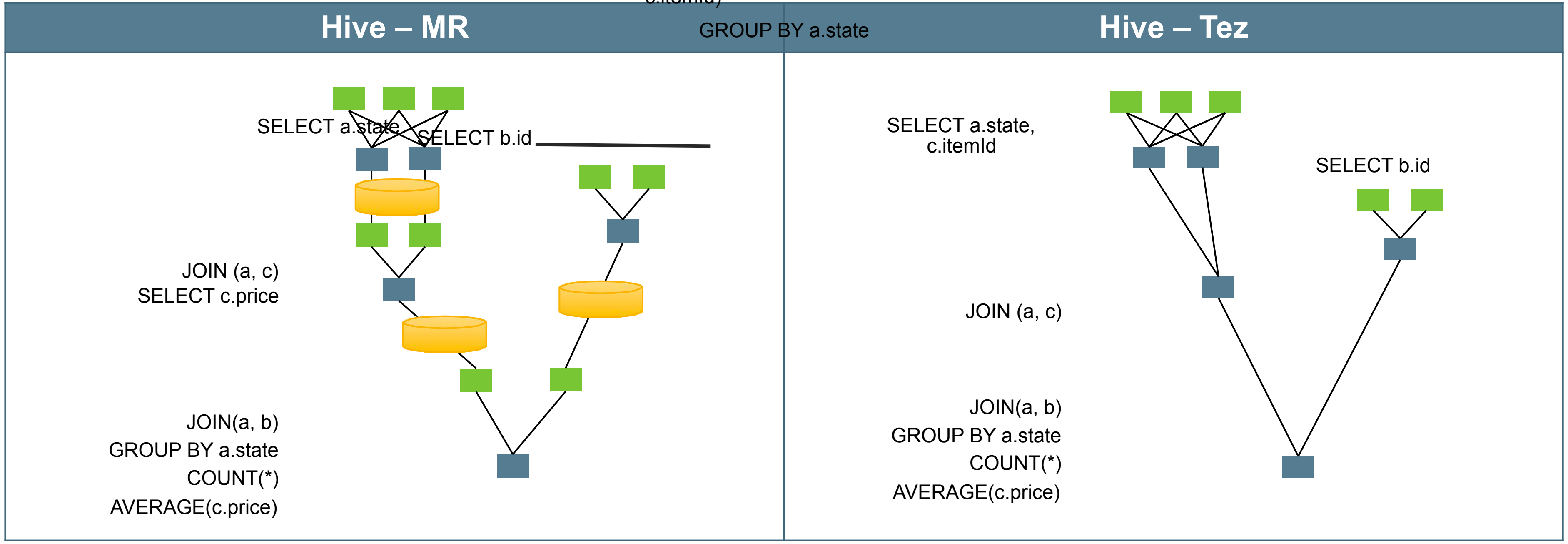
What is Tez

- Alternative data processing framework to MapReduce
- Who is involved?
  - Hortonworks, Facebook, Twitter, Yahoo, Microsoft
- Why does it matter?
  - Widens the platform for Hadoop use cases
  - Crucial to improving the performance of low-latency applications
  - Evidence of Hortonworks leading the community in the evolution of Enterprise Hadoop

Tez Vs MapReduce

```
SELECT a.state, COUNT(*),
        AVERAGE(c.price)  FROM
        a
    JOIN b ON (a.id = b.id)
  JOIN c ON (a.itemId =
  c.itemId)
    GROUP BY a.state
```

Tez avoids unneeded writes to HDFS



File formats supported in Hive

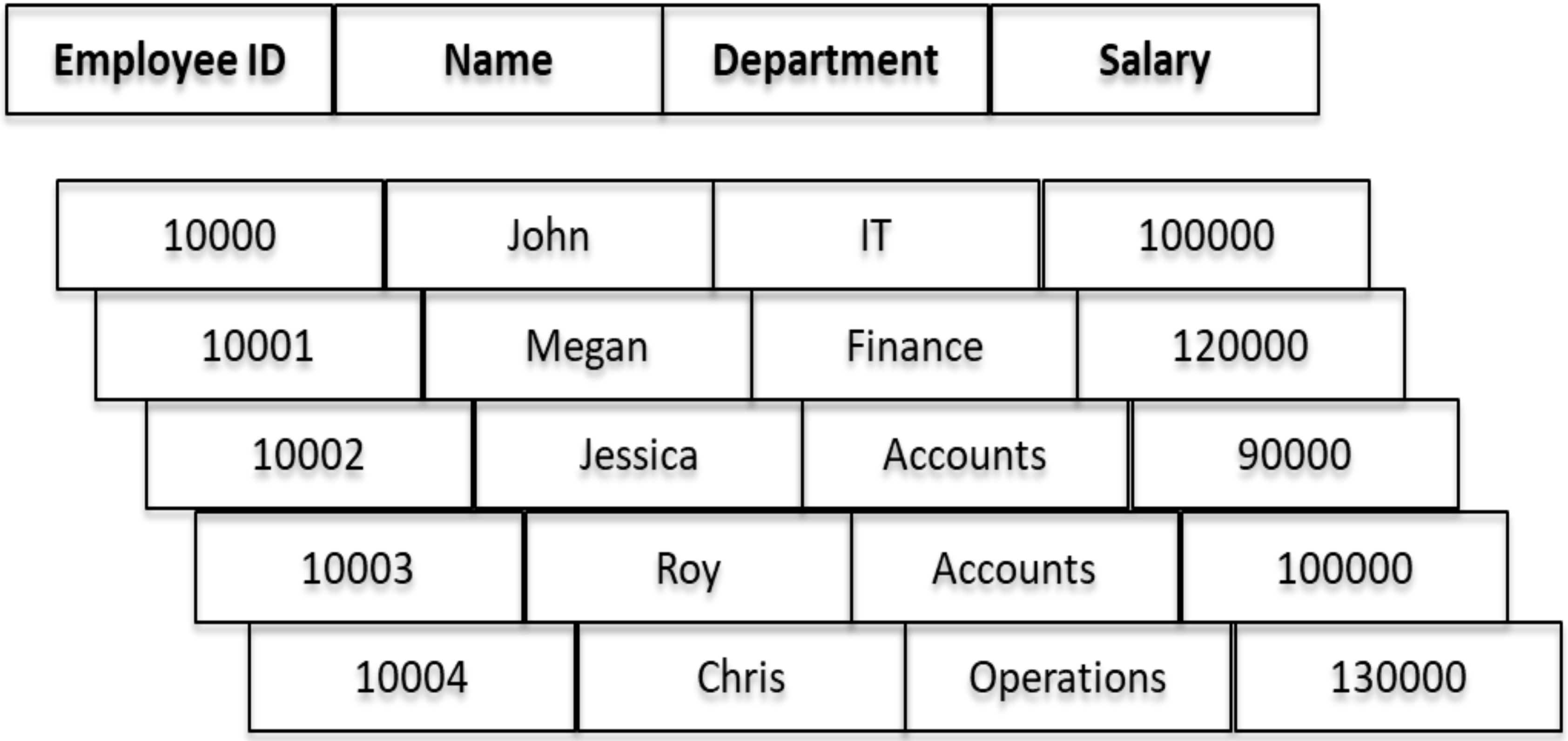
Hive supports various file formats , these file format can be categorized as

- Row format
- Columnar format

Row format : record is stored as row such as sequencefile, avro and json,

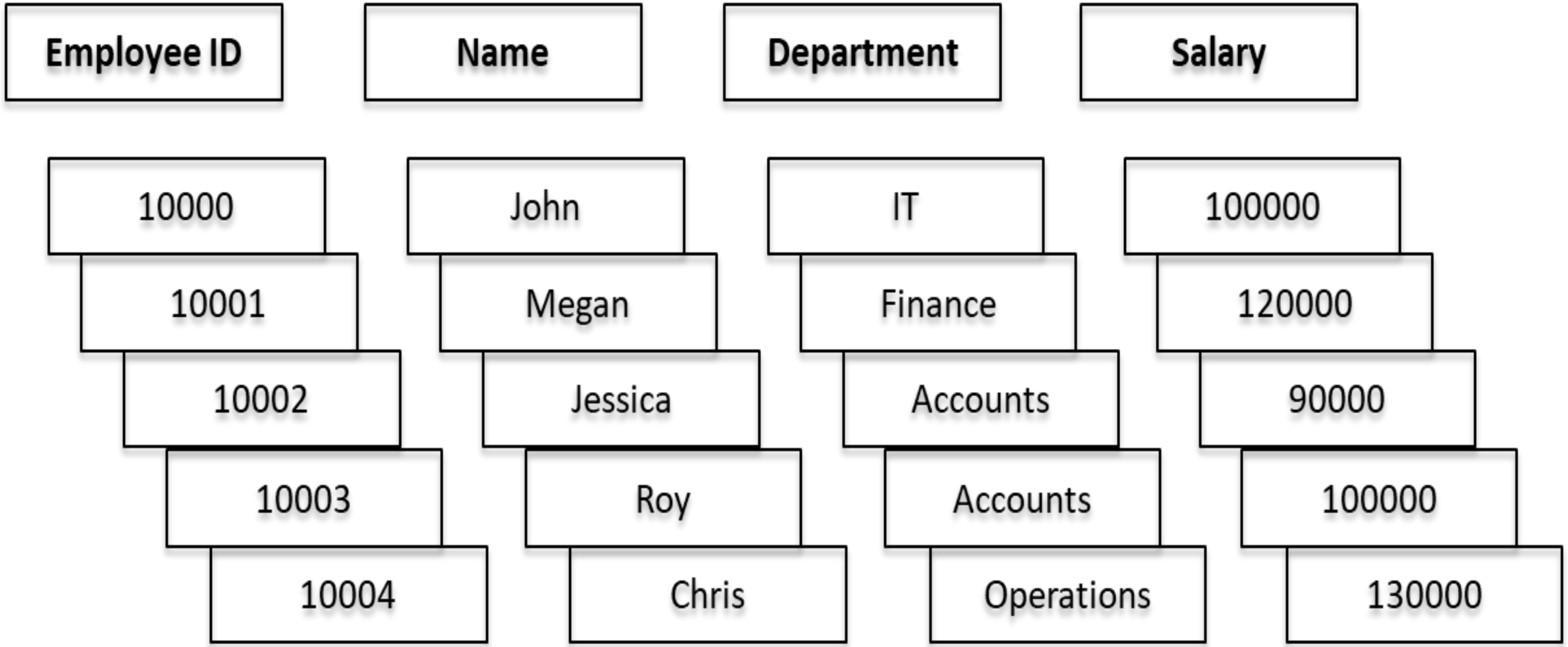
Columnar format : record is stored as column-by-column manner such as ORC and Parquet files,

Row oriented Storage in Hive



Employee ID	Name	Department	Salary
10000	John	IT	100000
10001	Megan	Finance	120000
10002	Jessica	Accounts	90000
10003	Roy	Accounts	100000
10004	Chris	Operations	130000

Column oriented Storage in Hive



Employee ID	Name	Department	Salary
10000	John	IT	100000
10001	Megan	Finance	120000
10002	Jessica	Accounts	90000
10003	Roy	Accounts	100000
10004	Chris	Operations	130000

Advantage of **Columnar Format**

- Columnar storage like ORC and Parquet uses data encoding to bring more efficiency compared to row-based files.

- When querying, columnar storage can only relevant columns from disk and skip over the non-relevant columns.

- With these optimization, aggregation queries take less time and memory compared to row-oriented structures.

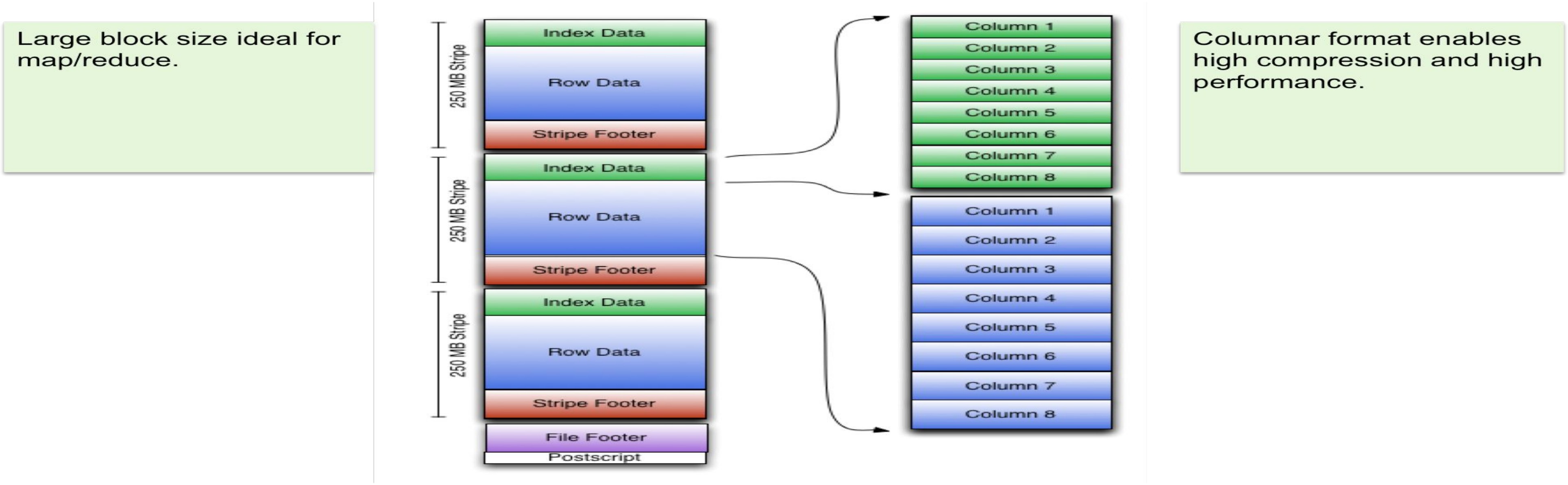
- This kind of storage results into better disk utilization and minimized latency for accessing data.

ORC (Optimized Row Columnar)file format

- Columns stored separately
  - Knows types
  - Uses type-specific encoders
  - Stores statistics (min, max, sum, count)
  - Has light-weight index
  - Skip over blocks of rows that don’t matter
- ORC files are completely self-describing and do not depend on the Hive Metastore or any other external metadata.

ORC Sturcture

- ORC files are divided into stripes that are 64MB by default.
- The stripes in a file are independent of each other and form the natural unit of distributed work.
- Within each stripe, the columns data are separated from each other by delimiter so the reader can read just the columns that are required.



Large block size ideal for map/reduce.

Columnar format enables high compression and high performance.

https://cwiki.apache.org/confluence/display/hive/languagemanual+orc

ORC file options and defaults

Key	Default	Notes
`orc.compress`	ZLIB	High level compression (one of NONE, ZLIB, SNAPPY)
`orc.compress.size`	262, 144 (=256 KiB)	Number of bytes in each compression chunk
`orc.stripe.size`	67,108,864 (=64 MiB)	Number of bytes in each stripe
`orc.row.index.stride`	10,000	Number of rows between index entries (must be >= 1,000)
`orc.create.index`	true	Whether to create row indexes

Note:

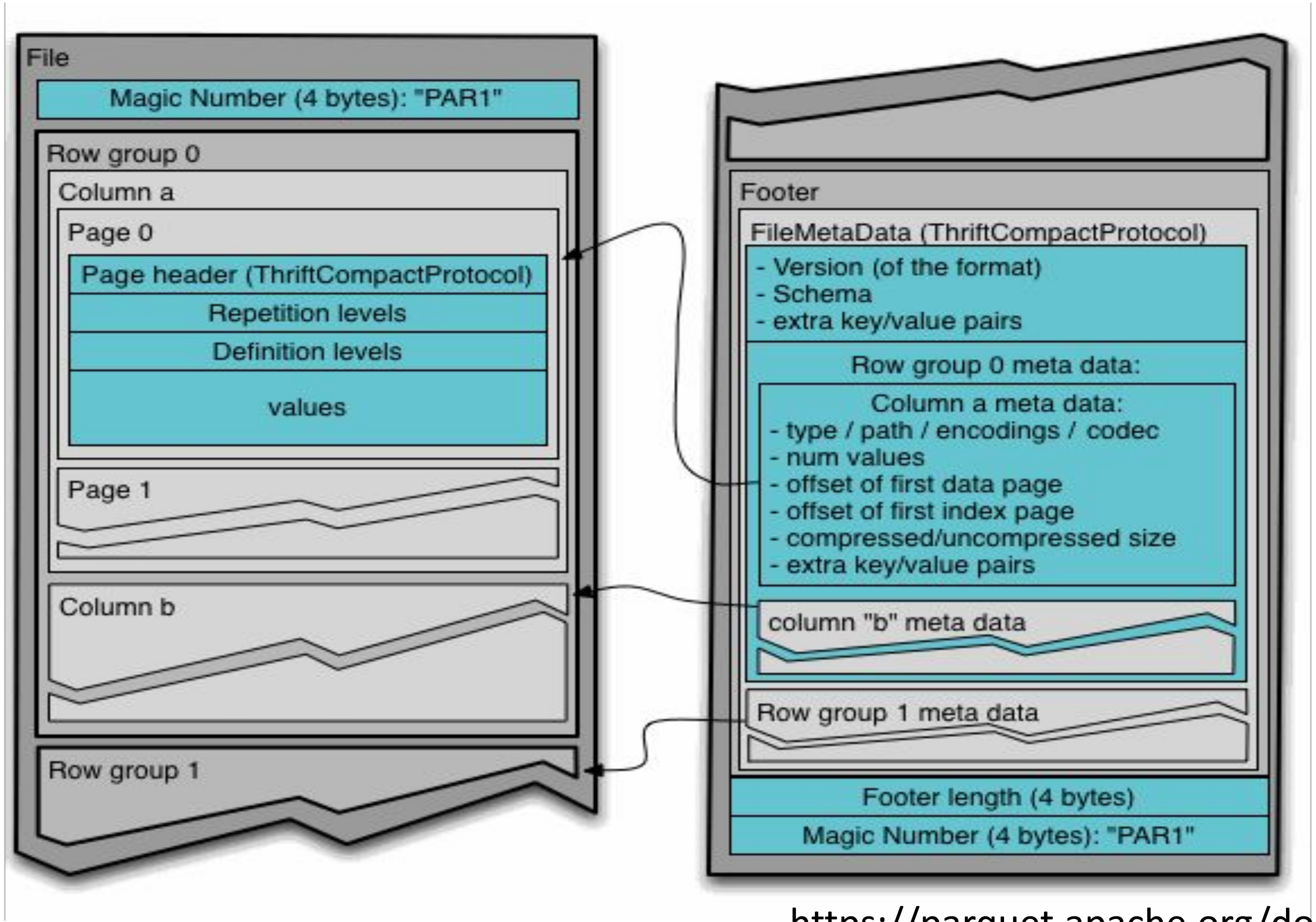
“orc.compress” = “SNAPPY” is a good trade-off between size and performance

https://docs.cloudera.com/HDPDocuments/HDP2/HDP2.3.0/

Parquet file format

- Parquet is an open-source file format available supported in Hadoop and non-Hadoop platform.
- Apache Parquet stores data in an efficient columnar structure.
- Parquet is optimized to work with complex data in bulk and uses different efficient data compression and encoding types.
- Parquet files are efficient only when queries read specific columns data from the file
- Parquet reads only the needed columns therefore greatly minimizing the disk IO.

Parquet Internal Structure


File
Magic Number (4 bytes): "PAR1"
Row group 0
Column a
Page 0
Page header (ThriftCompactProtocol)
Repetition levels
Definition levels
values
Page 1
Column b
Row group 1
Footer
FileMetaData (ThriftCompactProtocol)
- Version (of the format)
- Schema
- extra key/value pairs
Row group 0 meta data:
Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs
column "b" meta data
Row group 1 meta data
Footer length (4 bytes)
Magic Number (4 bytes): "PAR1"

https://parquet.apache.org/documentation/latest/

Benefits of Parquet file format

- Apache Parquet supports advanced nested data structures.

- The layout of Parquet data files is optimized for queries that process large volumes of data, in the gigabyte range for each individual file.

- Parquet is built to support flexible compression options and efficient encoding schemes.

Benefits of Parquet file format

- As the data type for each column is quite similar, the compression of each column is straightforward (which makes queries even faster).

- Data can be compressed by using one of the several codecs available; as a result, different data files can be compressed differently.

- Apache Parquet works best with interactive and serverless technologies like AWS Athena, Amazon Redshift Spectrum, Google BigQuery and Google Dataproc.

Avro file format

- Avro is a file format and data serialization protocol.
- Avro has following features:
  - It provided rich data structures.
  - It is fast and compact binary data format.
  - It is file format to store persistent data.
  - It is Remote procedure call (RPC) to send request to other process.

- Code generation is not needed to read or write Avro data files
- Code generation is also not needed to implement RPC protocols.
- Code generation as an optional optimization, needed for statically typed languages.

Avro Schema

- Avro protocol relies on schemas.
- This permits each record to be written with no per-value overheads, making serialization both fast and small.
- When Avro data is stored in a file, its schema is also stored with it, so that files may be processed later by any program.

- Availability of schema with data make Avro data file fully self-describing.

- Avro file also support backward and forward compatibility of the schema

Benefits of Avro Schema

- When Avro is exchanges in Remote Procedure call, the client and server exchange schemas in the connection handshake.

- With full schema availability with both client and server, confusion regarding same named fields, missing fields, extra fields, etc. can all be easily resolved.

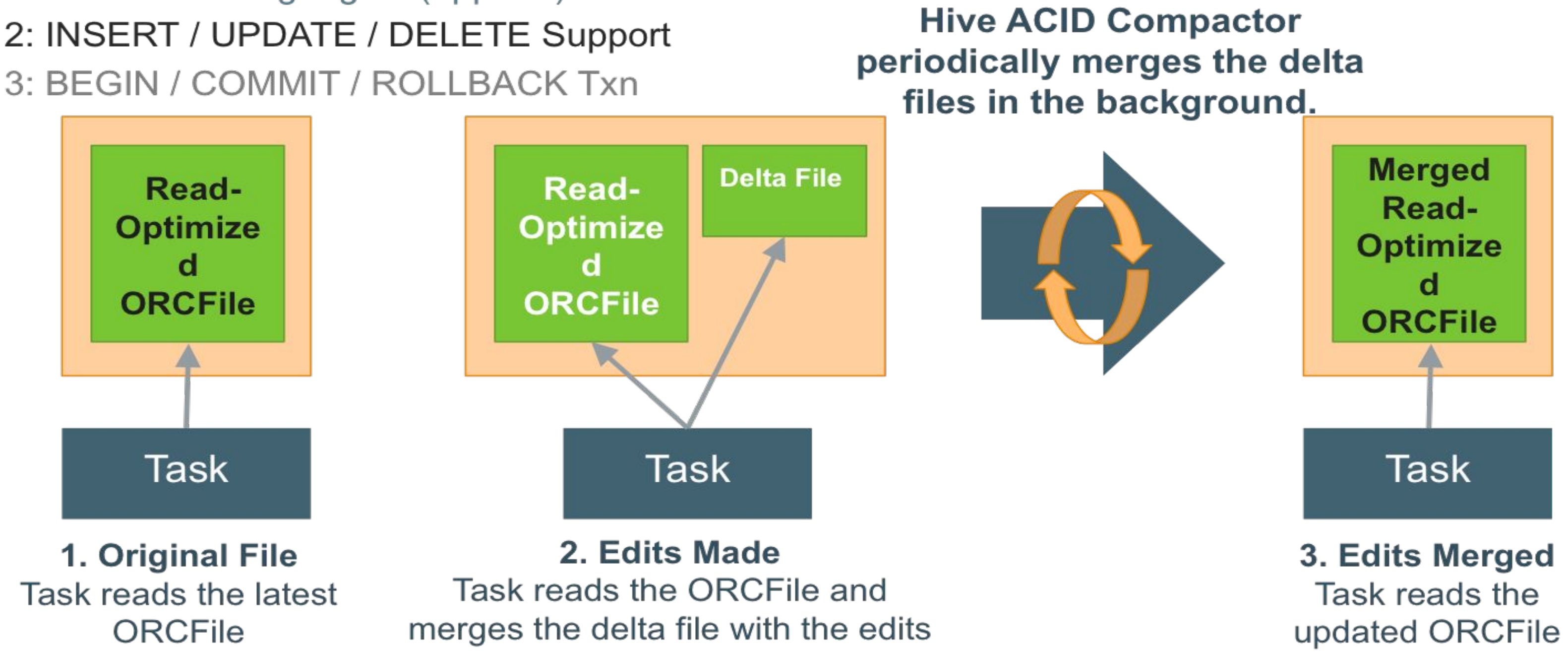
- Avro schemas are defined in a JSON format .

- Avro data can be exchanges by all those application which supports JSON libraries.

Transactions OR ACID Table

Transaction Support in Hive with ACID semantics

- Hive native support for INSERT, UPDATE, DELETE.
- Split Into Phases:
  - [Done] Phase 1: Hive Streaming Ingest (append)
  - [Done] Phase 2: INSERT / UPDATE / DELETE Support
  - [Next] Phase 3: BEGIN / COMMIT / ROLLBACK Txn


Hive ACID Compactor periodically merges the delta files in the background.
Read-Optimized ORCFile
Task
Read-Optimized ORCFile
Delta File
Task
Merged Read-Optimized ORCFile
Task
1. Original File
Task reads the latest ORCFile
2. Edits Made
Task reads the ORCFile and merges the delta file with the edits
3. Edits Merged
Task reads the updated ORCFile

Transactions OR ACID Table Requirments

- Tables must be:
  - Declared as having Transaction Property
  - In ORC format
  - Bucketed

```
hive.support.concurrency=true  hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
hive.compactor.initiator.on=true  hive.compactor.worker.threads=2
hive.enforce.bucketing=true  hive.exec.dynamic.partition.mode=nonstri
```

Hive Views

- Views allow query to be saved and treated just like a table.

- View is a logical construct and does not store data.

- Used to simplify queries.

Syntax to create a View:

- CREATE VIEW ViewName AS SELECT <cols> FROM <table> WHERE <predicate>

- WHERE clause is optional.
- A view's name must be unique compared to all other table and view names in the same database.

Hive Views

- Query a view just like how you query a table.
- If a query references a view, the definition of the view is combined with the rest of the query by Hive's query planner.
- No storage allocated for this logical view.
- However, Hive 3.0 also supports Materialized view which will save view data into a HDFS folder

Nested Queries and CASE Statements

- hive> SELECT * FROM student_info s where s.marks > (SELECT avg(marks) FROM student_info);

- Hive allows use of CASE statements to enable you to classify records depending on various inputs

-
  ```
  SELECT name,CASE WHEN marks >= 66 THEN 'DISTINCTION',
        WHEN marks < 66 AND marks >= 60 THEN 'FIRST CLASS,
        WHEN marks < 60 AND marks >= 55 THEN 'SECOND CLASS,
                      WHEN marks < 55 AND marks >= 50 THEN 'THIRD CLASS,
                      ELSE 'FAIL'
                      END AS class FROM student_info;
  ```

Limit, Like, RLike , Group By and Having

- hive> SELECT * FROM student_info limit 10;
- LIKE and RLIKE operators compare and match strings or substrings from a given set of records. Following is an example:

```
hive>SELECT * FROM student_info WHERE name LIKE 'A%';
Hive>SELECT * FROM student_info WHERE name LIKE '%Jr.%';
```

- Similarly, RLIKE allows you to query data by putting regex

```
hive>SELECT * FROM student_info WHERE name RLIKE '.*(Jr|Sr)*.';
hive>SELECT class, avg(marks) FROM student_info GROUP BY class;
hive> SELECT class, avg(marks) FROM student _info GROUP BY class
HAVING class = 'V' OR class = 'VI;
```

Built-In and User-defined function(UDF)

- Hive supports a variety of built-in functions, such as: Arithmetic functions, Mathematical functions and Aggregated functions
- hive>SELECT avg(percentage) FROM student_info;
- hive>SELECT upper(name) FROM student_info;

- Sometimes, you may require to custom modify certain values in columns, and it may not be feasible to use built-in functions.
- In such a case, Hive allows its users to define their own functions to be used in SELECT statements

- To create a User-Defined Function (UDF), you need to write a Java class that extends org.apache.hadoop.hive.ql.exec.UDF.
- In that class, write the evaluate () method where you can modify the default behavior.

Joins : Inner and Outer Joins

```
•  hive>SELECT o.order_id, c.customer_name FROM order o
              RIGHT OUTER JOIN customer c ON ( o.customer_id =
c.customer_id);

•hive> SELECT o.order_id, c.customer_name FROM order o LEFT OUTER
    JOINcustomer c ON ( o.customer_id = c.customer_id);

•  hive>SELECT o.order_id, c.customer_name FROM order o FULL OUTER
   JOIN
         customer c  ON ( o.customer_id = c.customer_id);
```

did you know?

- Apache Hive was developed by Facebook in 2007

- Before hive, End users had to write map/reduce programs for simple tasks like getting raw counts or averages

Let us perform some hands-on

- ✔ Create Hive internal and external table
- ✔ Executing Hive query using Tez engine
- ✔ Executing Hive query using MR engine
- ✔ Creating Partition table to store data into ORC format

Data Ingestion

- Data ingestion is the first step of any Big data Project.
- It means taking data from various silo databases and files and putting it into Hadoop.

- Data Ingestion can be categories into following categories
  - Database Ingestion
  - Database Dump
  - Streaming Ingestion

Souce of data to be ingested in Hadoop

- We may want to import the data from following sources to HDFS
  - Files
  - Data bases
  - Application Log files
  - Social media data
  - Un-structured data such as Image, Video and Audio files
  - Digital assets
  - Structured business data
  - Data as REST request

What is Flume

- Flume is a distributed, reliable and available system to:
  - Collect
  - Aggregate
  - Move
- High volume, streaming data from many sources to a centralized store.
- Makes data loading easy and efficient.

Application of Flume:

- To ingest real time stream into HDFS or Kafka
- To perform transformation while streaming data into HDFS

Flume Architecture

Event: Singular unit of data that can be transported, like a single log entry or a Tweet.

Client: Produces data in the form of Events (not files).

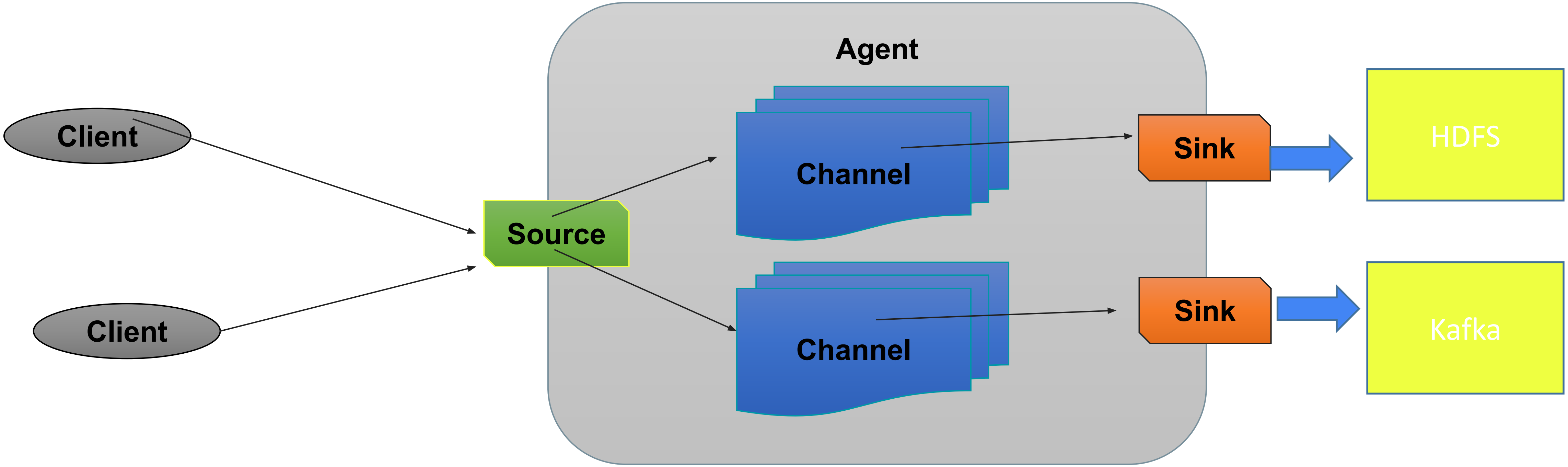
Source: Listens for and consume events.

Channel: Mechanism by which Flume agents transfer events from their sources to their destination.

Sink: Removes events from a channel and transmit them to the next agent in the flow, or to the event's final destination.

Agent: An independent process that hosts flume components such as sources, channels and sinks, and thus has the ability to receive, store and forward events

Flume Architecture



Flume Configuration

- Agent comprises Source, Channel and Sink components.

```
<Agent>.sources    = <Source>
<Agent>.sinks      = <Sink>
<Agent>.channels   = <Channel>
```

<Agent> : Name of our flume Agent.
<Source> : Name of our Source.
<Sink>   : Name of our Sink.
<Channel>   : Name of our Channel.

Flume Source Type

- Avro source
- Netcat, which is a TCP event source.
- Syslog files
- Exec: Execute a given Unix command that continuously produces data on the standard out.
- Custom: Twitter, for example.

Flume Sink Type

- HDFS
- HBase
- Avro
- Custom

Flume Channel Types

- Memory
- File

- JDBC

- Kafka
- Custom

When to use File Channel

File Channel

- Durable – persists events to disk.

- When Operating System crashes/reboots – events not transferred will be there when the Flume Agent restarts.

- In case of downstream failures, ability to buffer events is superior since it writes the events to the disk.

Benefits of Memory Channel

Memory Channel

- Volatile – buffers events in memory.
- When Operating System crashes/reboots – events not transferred will be LOST.
- Due to limited RAM availability, in case of downstream failures, ability to buffer events is limited.
- High throughput compared to File Channel.

When to use Kafka as Channel

Kafka

- Writes data(messages) into multiple partitions which provided fault tolerance
- Data is safe even in the case broker going down.
- Kafka keeps the data for default period of 1 week
- Kafka helps consumer reading the data in exactly once manner

.

Flow Definition: Example

Define the Source, Sink and the Channel

```
log-to-hdfs.sources  = namenode-log-source
log-to-hdfs.sinks          = hdfs-sink
log-to-hdfs.channels      = memchannel
```

Flow Definition: Example

Join the Source and Sink to the Channel.

```
<Agent>.sources.<Source>.channels = <Channel>
<Agent>.sinks.<Sink>.channel = <Channel>
```

Flow Definition: Example

Join the Source and Sink to the Channel.

```
<Agent>.sources.<Source>.channels = <Channel>
<Agent>.sinks.<Sink>.channel = <Channel>
```

Flow Definition: Example

```
# Join Source to Channel

log-to-hdfs-agent.sources.namenode-log-source.channels = memchannel
```

Flow Definition: Example

```
# Join Source to Channel

log-to-hdfs-agent.sources.namenode-log-source.channels = memchannel


#Join Sink to Channel

log-to-hdfs-agent.sinks.hdfs-sink.channel = memchannel
```

Configure Individual Components

General Syntax

<Agent>.sources.<Source>.<Property> = <Value>

<Agent>.sinks.<Sink>.<Property> = <Value>

<Agent>.channels.<Channel>.<Property> = <Value>

Configure Source

```
log-to-hdfs-agent.sources.namenode-log-source.type   = exec

log-to-hdfs-agent.sources.namenode-log-source.command = tail -F /home/hduser/hadoop/logs/hadoop-hduser-namenode-ubuntu.log
```

Configure Sink

```
log-to-hdfs-agent.sinks.hdfs-sink.type = hdfs

log-to-hdfs-agent.sinks.hdfs-sink.path = hdfs://localhost:54310/flume
```

Configure Channel

```
log-to-hdfs-agent.channels.memchannel.type = memory
```

Configuration File

```
# Define the components: Name of the agent, Source, Sink and Channel
log-to-hdfs-agent.sources = namenode-log-source
log-to-hdfs-agent.sinks = hdfs-sink
log-to-hdfs-agent.channels = mem-channel

# Bind the Source and Sink to the channel
log-to-hdfs-agent.sources.namenode-log-source.channels = mem-channel
log-to-hdfs-agent.sinks.hdfs-sink.channel = mem-channel

# Configure the Source
log-to-hdfs-agent.sources.namenode-log-source.type = exec
log-to-hdfs-agent.sources.namenode-log-source.command = tail -F
/home/hduser/hadoop/logs/hadoop-hduser-namenode-ubuntu.log

# Configure the Sink
log-to-hdfs-agent.sinks.hdfs-sink.type = hdfs
log-to-hdfs-agent.sinks.hdfs-sink.hdfs.path = hdfs://localhost:54310/flume

# Configure the Channel
log-to-hdfs-agent.channels.mem-channel.type = memory
```

Before running Flume

- Create a configuration file in flume/conf directory.
- Ensure that the path of the log file is correct.
- Ensure that the user has access to the log file.
- Ensure that the destination directory does not exist in HDFS.

Running a Flume Agent

- Ensure you are in the /flume directory.

bin/flume-ng agent –n log-to-hdfs-agent –f conf/log-to-hdfs-agent.conf

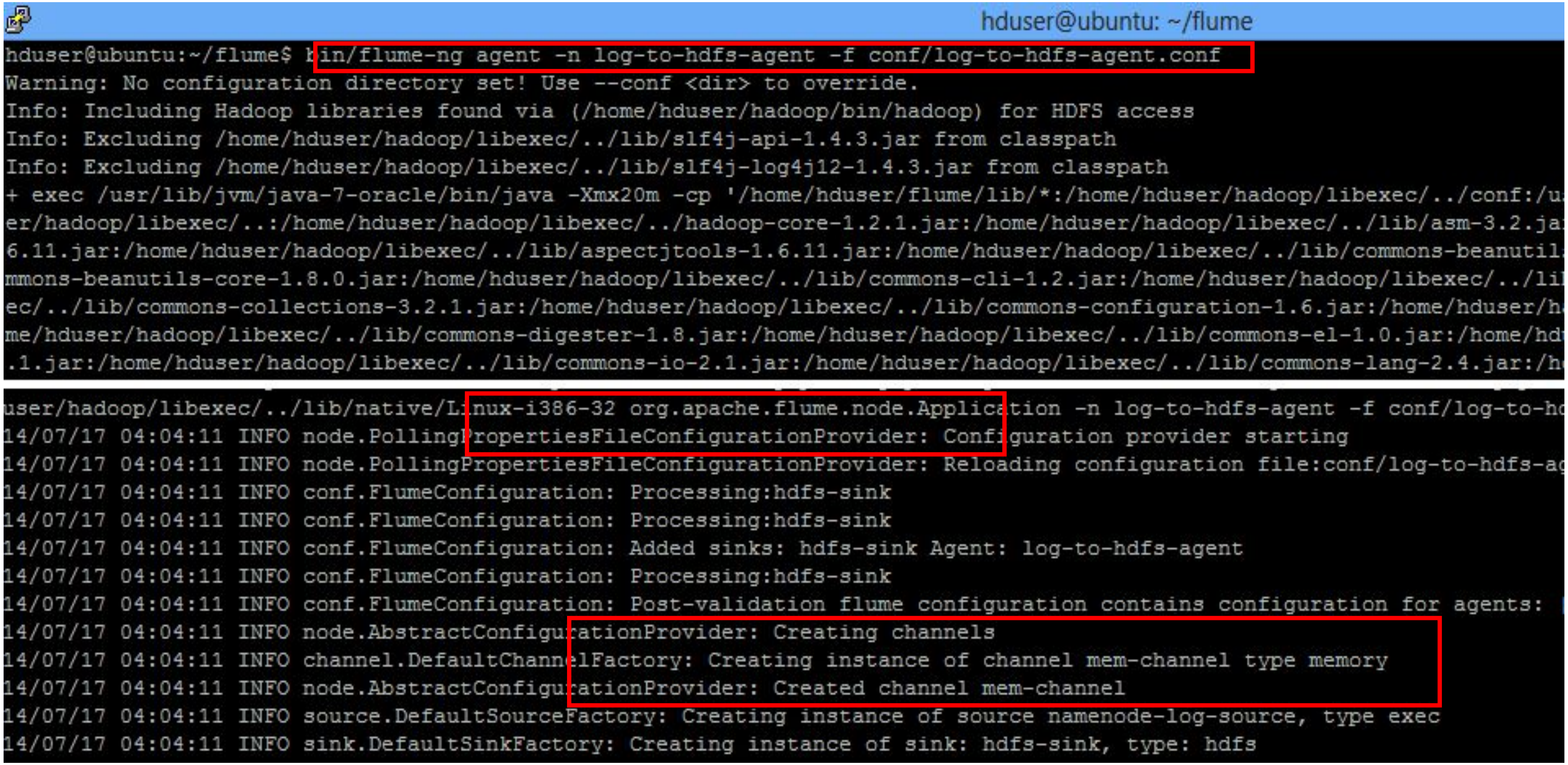
flume-ng: binary executable of flume

- Agent : indicates that you want to run a flume agent

-n : provides the name of the agent to be run. This is the name of the agent we provided in the configuration file.

–f : provides the path of the configuration file that needs to be run

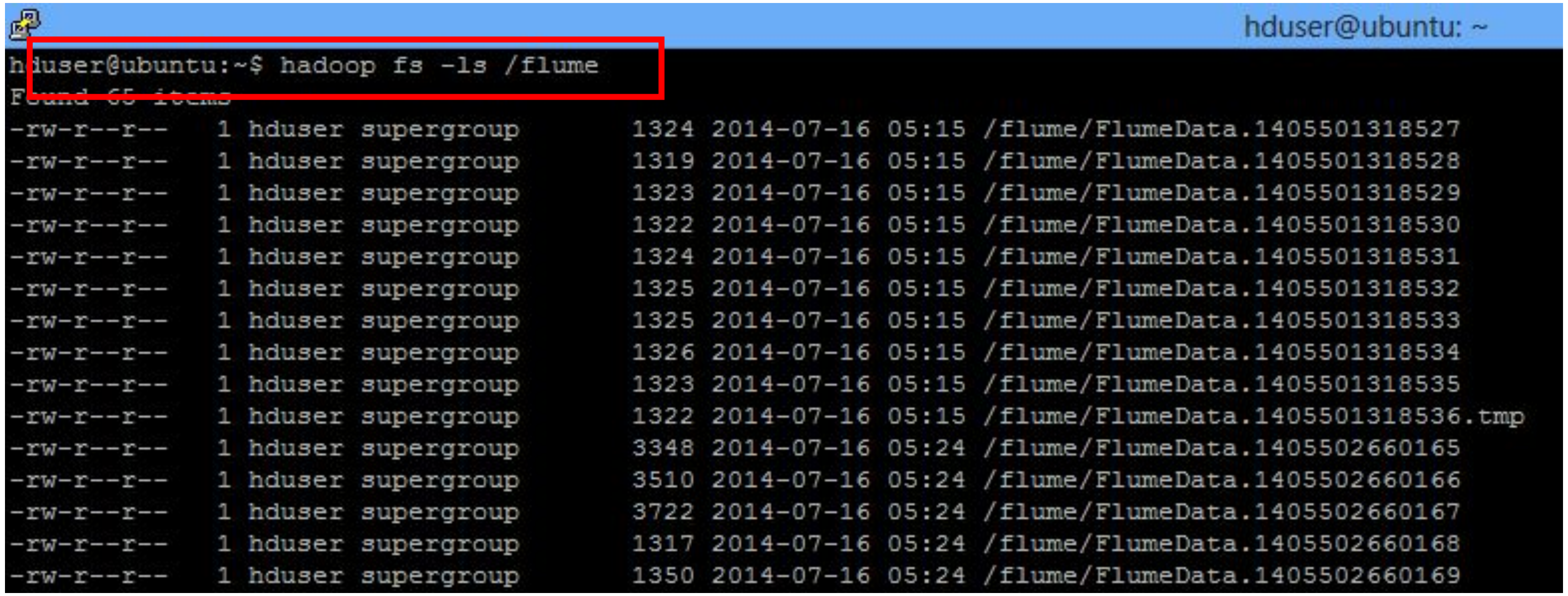
Running a Flume Agent



```
hduser@ubuntu:~/flume$ bin/flume-ng agent -n log-to-hdfs-agent -f conf/log-to-hdfs-agent.conf
Warning: No configuration directory set! Use --conf <dir> to override.
Info: Including Hadoop libraries found via (/home/hduser/hadoop/bin/hadoop) for HDFS access
Info: Excluding /home/hduser/hadoop/libexec/../lib/slf4j-api-1.4.3.jar from classpath
Info: Excluding /home/hduser/hadoop/libexec/../lib/slf4j-log4j12-1.4.3.jar from classpath
+ exec /usr/lib/jvm/java-7-oracle/bin/java -Xmx20m -cp '/home/hduser/flume/lib/*:/home/hduser/hadoop/libexec/../conf:/u
er/hadoop/libexec/..:/home/hduser/hadoop/libexec/../hadoop-core-1.2.1.jar:/home/hduser/hadoop/libexec/../lib/asm-3.2.ja
6.11.jar:/home/hduser/hadoop/libexec/../lib/aspectjtools-1.6.11.jar:/home/hduser/hadoop/libexec/../lib/commons-beanutil
mmons-beanutils-core-1.8.0.jar:/home/hduser/hadoop/libexec/../lib/commons-cli-1.2.jar:/home/hduser/hadoop/libexec/../li
ec/../lib/commons-collections-3.2.1.jar:/home/hduser/hadoop/libexec/../lib/commons-configuration-1.6.jar:/home/hduser/h
me/hduser/hadoop/libexec/../lib/commons-digester-1.8.jar:/home/hduser/hadoop/libexec/../lib/commons-el-1.0.jar:/home/hd
.1.jar:/home/hduser/hadoop/libexec/../lib/commons-io-2.1.jar:/home/hduser/hadoop/libexec/../lib/commons-lang-2.4.jar:/h
```

```
user/hadoop/libexec/../lib/native/Linux-i386-32 org.apache.flume.node.Application -n log-to-hdfs-agent -f conf/log-to-hd
14/07/17 04:04:11 INFO node.PollingPropertiesFileConfigurationProvider: Configuration provider starting
14/07/17 04:04:11 INFO node.PollingPropertiesFileConfigurationProvider: Reloading configuration file:conf/log-to-hdfs-ag
14/07/17 04:04:11 INFO conf.FlumeConfiguration: Processing:hdfs-sink
14/07/17 04:04:11 INFO conf.FlumeConfiguration: Processing:hdfs-sink
14/07/17 04:04:11 INFO conf.FlumeConfiguration: Added sinks: hdfs-sink Agent: log-to-hdfs-agent
14/07/17 04:04:11 INFO conf.FlumeConfiguration: Processing:hdfs-sink
14/07/17 04:04:11 INFO conf.FlumeConfiguration: Post-validation flume configuration contains configuration for agents:
14/07/17 04:04:11 INFO node.AbstractConfigurationProvider: Creating channels
14/07/17 04:04:11 INFO channel.DefaultChannelFactory: Creating instance of channel mem-channel type memory
14/07/17 04:04:11 INFO node.AbstractConfigurationProvider: Created channel mem-channel
14/07/17 04:04:11 INFO source.DefaultSourceFactory: Creating instance of source namenode-log-source, type exec
14/07/17 04:04:11 INFO sink.DefaultSinkFactory: Creating instance of sink: hdfs-sink, type: hdfs
```

Verify Result



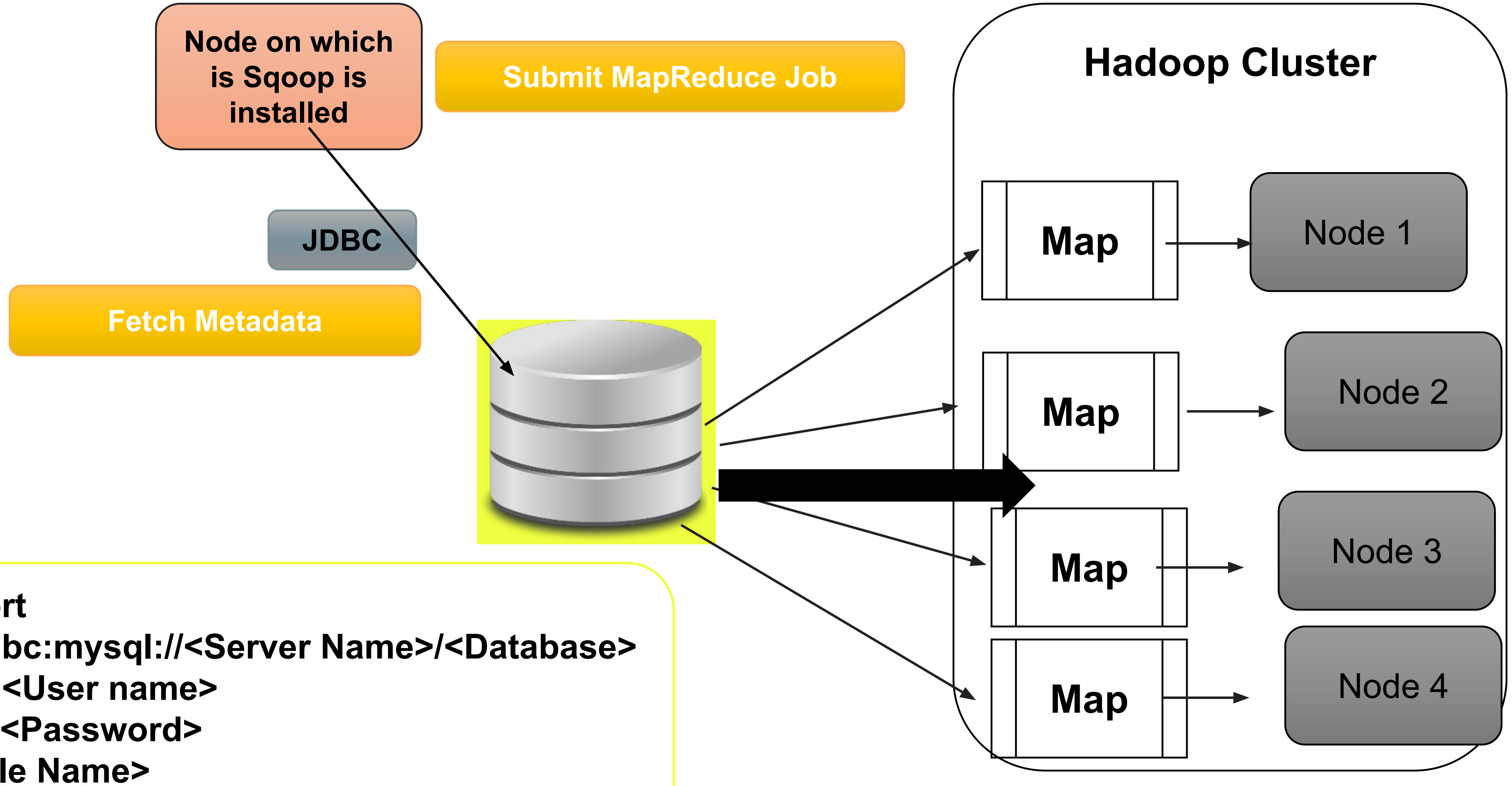
hduser@ubuntu: ~

```
hduser@ubuntu:~$ hadoop fs -ls /flume
Found 65 items
-rw-r--r--   1 hduser supergroup       1324 2014-07-16 05:15 /flume/FlumeData.1405501318527
-rw-r--r--   1 hduser supergroup       1319 2014-07-16 05:15 /flume/FlumeData.1405501318528
-rw-r--r--   1 hduser supergroup       1323 2014-07-16 05:15 /flume/FlumeData.1405501318529
-rw-r--r--   1 hduser supergroup       1322 2014-07-16 05:15 /flume/FlumeData.1405501318530
-rw-r--r--   1 hduser supergroup       1324 2014-07-16 05:15 /flume/FlumeData.1405501318531
-rw-r--r--   1 hduser supergroup       1325 2014-07-16 05:15 /flume/FlumeData.1405501318532
-rw-r--r--   1 hduser supergroup       1325 2014-07-16 05:15 /flume/FlumeData.1405501318533
-rw-r--r--   1 hduser supergroup       1326 2014-07-16 05:15 /flume/FlumeData.1405501318534
-rw-r--r--   1 hduser supergroup       1323 2014-07-16 05:15 /flume/FlumeData.1405501318535
-rw-r--r--   1 hduser supergroup       1322 2014-07-16 05:15 /flume/FlumeData.1405501318536.tmp
-rw-r--r--   1 hduser supergroup       3348 2014-07-16 05:24 /flume/FlumeData.1405502660165
-rw-r--r--   1 hduser supergroup       3510 2014-07-16 05:24 /flume/FlumeData.1405502660166
-rw-r--r--   1 hduser supergroup       3722 2014-07-16 05:24 /flume/FlumeData.1405502660167
-rw-r--r--   1 hduser supergroup       1317 2014-07-16 05:24 /flume/FlumeData.1405502660168
-rw-r--r--   1 hduser supergroup       1350 2014-07-16 05:24 /flume/FlumeData.1405502660169
```

Import and Export data from RDBMS using Sqoop

- Sqoop is a MR tool for importing data from RDBMS to HDFS
- Sqoop can export data from HDFS to RDBMS too
- Sqoop convert import command into Map Only Job to import the data
- Parallel Map tasks help importing the data faster
- Sqoop uses JDBC connection of databases to run the query and fetch the data

Sqoop

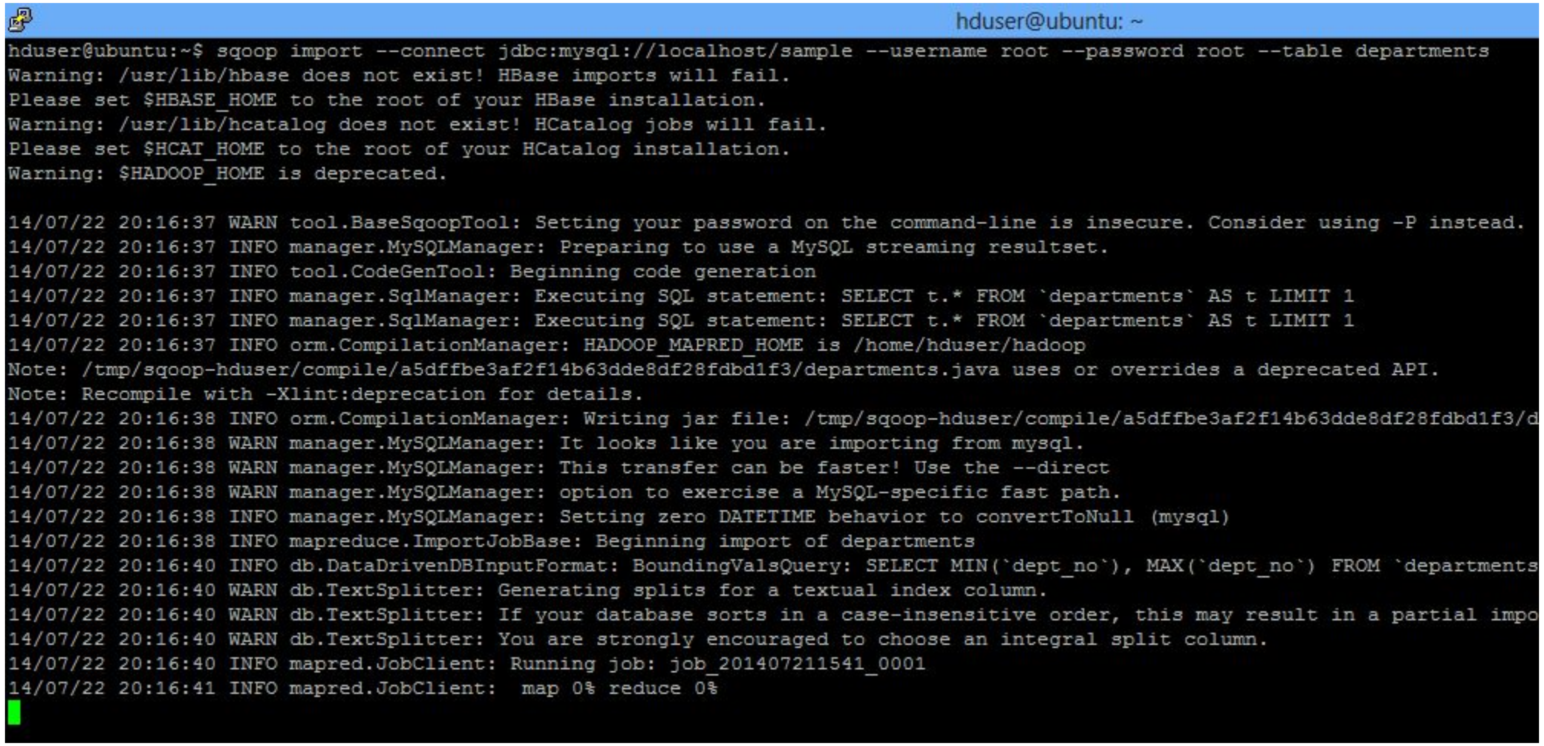


```
sqoop import
--connect jdbc:mysql://<Server Name>/<Database>
--username <User name>
--password <Password>
--table <Table Name>
```

Table Import

sqoop import --connect jdbc:mysql://localhost/sample --username root --password root **--table** departments

sqoop import --connect jdbc:mysql://localhost/sample --username root --password root --table departments



```
hduser@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/sample --username root --password root --table departments
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: /usr/lib/hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: $HADOOP_HOME is deprecated.

14/07/22 20:16:37 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
14/07/22 20:16:37 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
14/07/22 20:16:37 INFO tool.CodeGenTool: Beginning code generation
14/07/22 20:16:37 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `departments` AS t LIMIT 1
14/07/22 20:16:37 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `departments` AS t LIMIT 1
14/07/22 20:16:37 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /home/hduser/hadoop
Note: /tmp/sqoop-hduser/compile/a5dffbe3af2f14b63dde8df28fdbd1f3/departments.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
14/07/22 20:16:38 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hduser/compile/a5dffbe3af2f14b63dde8df28fdbd1f3/d
14/07/22 20:16:38 WARN manager.MySQLManager: It looks like you are importing from mysql.
14/07/22 20:16:38 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
14/07/22 20:16:38 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
14/07/22 20:16:38 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
14/07/22 20:16:38 INFO mapreduce.ImportJobBase: Beginning import of departments
14/07/22 20:16:40 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`dept_no`), MAX(`dept_no`) FROM `departments
14/07/22 20:16:40 WARN db.TextSplitter: Generating splits for a textual index column.
14/07/22 20:16:40 WARN db.TextSplitter: If your database sorts in a case-insensitive order, this may result in a partial impo
14/07/22 20:16:40 WARN db.TextSplitter: You are strongly encouraged to choose an integral split column.
14/07/22 20:16:40 INFO mapred.JobClient: Running job: job_201407211541_0001
14/07/22 20:16:41 INFO mapred.JobClient:  map 0% reduce 0%
```

Incremental Import

- Using Sqoop, we can also incrementally import the data
- Let’s say you want to import data inserted in data after row id 1

```
sqoop import -Dmapreduce.job.user.classpath.first=true \
— connect jdbc:mysql://localhost/hadoop_test \
— username sqoop_test \
— password password \
— table emp -m 1 \
— incremental append \
— check-column id \
— last-value 2;
```

Sqoop’s Facts

- Sqoop spins multiple map tasks and imports data in parallel.

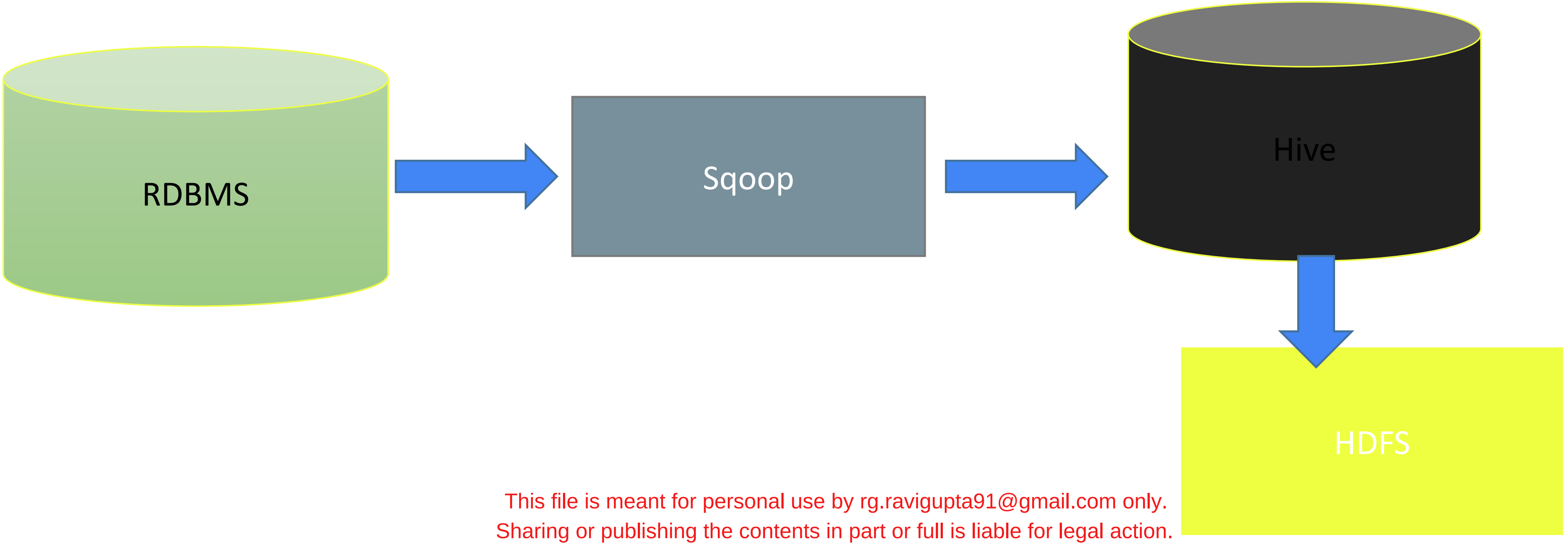
- Each Mapper retrieves a subset of rows.

- Default mappers are 4 hence 4 files.
- Sqoop uses splitting column, by default Primary Key.

- Uses low and high values of the splitting column
- Sqoop can import the whole database as well as output of the query
- Sqoop also support incremental import provided source data table has timestamp or a column which contains sequential number
- Sqoop can import the data into HDFS in different file format such as ORC,Parquet,Avro, SequenceFile etc

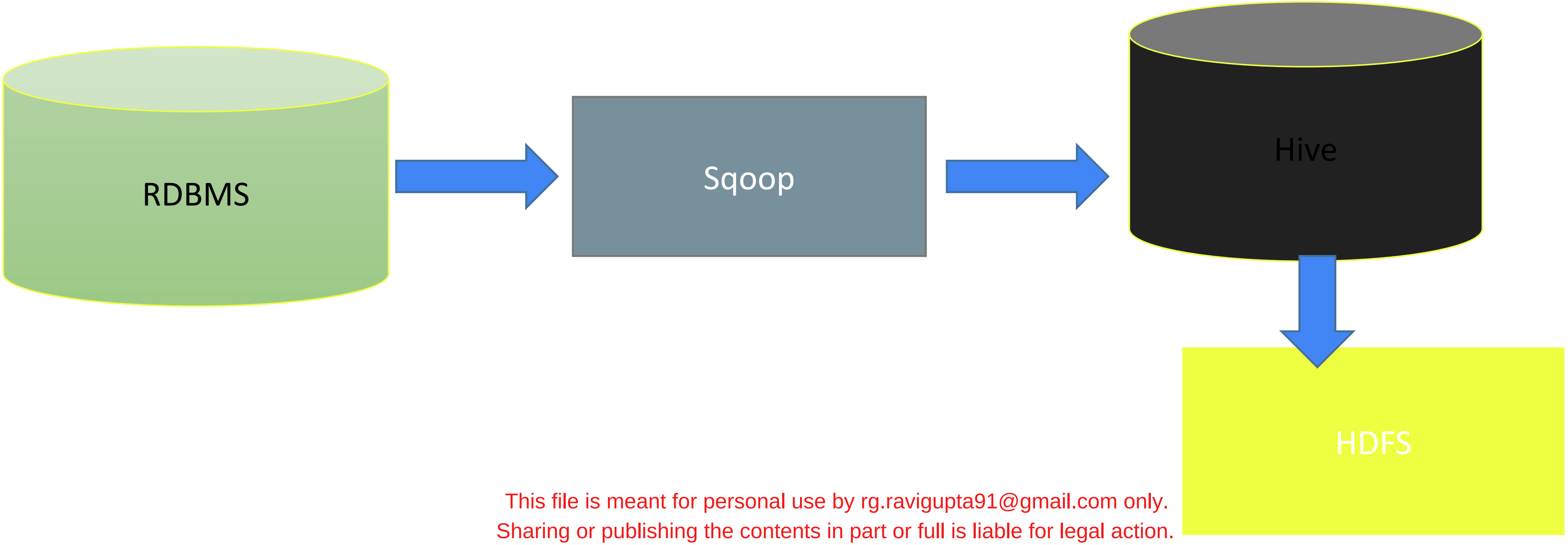
Import to Hive

```
sqoop import --connect jdbc:mysql://localhost:3306/testing --username root
--table greatlearning_employees --hive-import --hive-table
subex.mysql_employees --fetch-size 10 -m 2
```



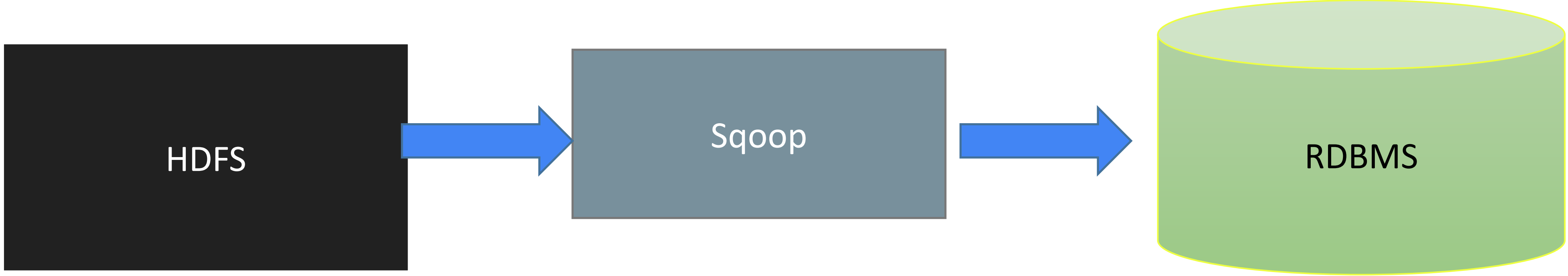
Import to Hive

```
sqoop import --connect jdbc:mysql://localhost:3306/testing --username root
--query 'select A.name,B.role from greatlearning_employees A JOIN
employees_role B ON A.id= B.id where $CONDITIONS' --target-dir
/sqoop_mysql_query --fetch-size 10 -m 2 --split-by A.name
```



Export

```
sqoop export --connect jdbc:mysql://localhost:3306/testing --username root --table emp_duplicate --export-dir /sqoop_mysql_employees1  -m 2 --driver com.mysql.jdbc.Driver
```



Thank You