University of East London

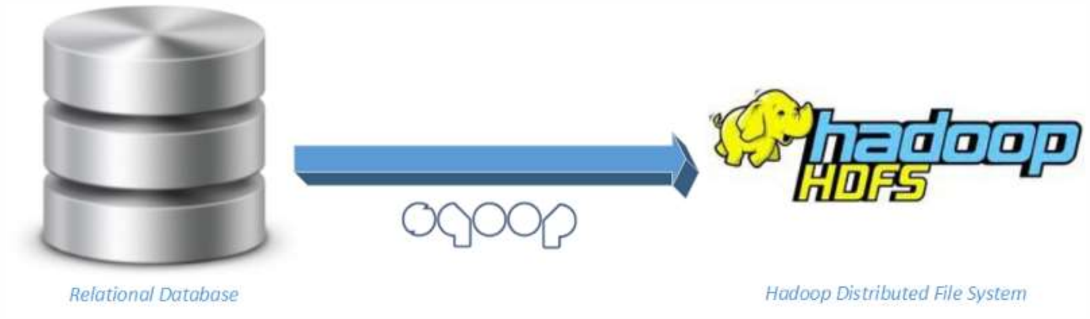
Pioneering Futures Since 1898

Apache Sqoop

Transferring data between relational databases and Hadoop

Dr Fahimeh Jafari

f.jafari@uel.ac.uk



CN7031 – Lecture 3

Outline

- Hadoop Ecosystem
- Apache Sqoop
    - What is Apache Sqoop?
    - Why Sqoop?
    - Sqoop commands
    - Sqoop MetaStore and Sqoop Merge

Learning Outcomes

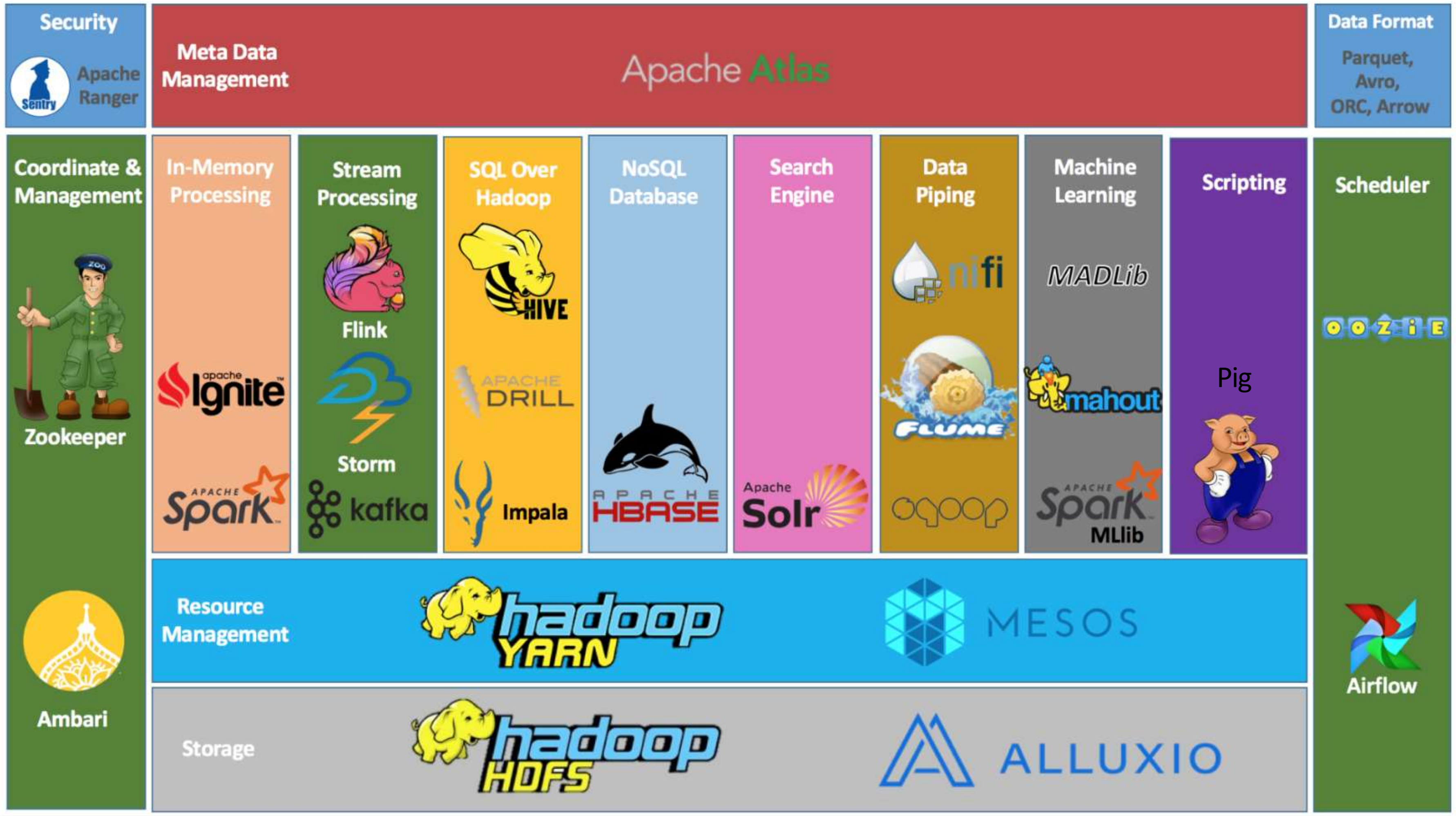
- To be able to explain Sqoop
- Understand Sqoop’s engine
- To be able to write Sqoop commands
- Get familiar with advanced Sqoop components: MetaStore and Merge

Hadoop Ecosystem

- Hadoop has become the kernel of the distributed operating system for Big Data
- No one uses the kernel alone
- A collection of projects at Apache

hulu
amazon.com
facebook
TREND MICRO
IBM
bing
ebay
LinkedIn
阿里巴巴 Alibaba.com
Bai du 百度

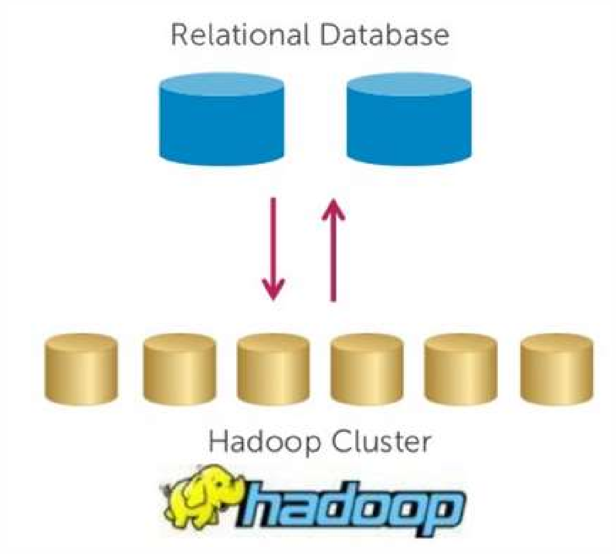
Hadoop Ecosystem



University of East London

What is Sqoop

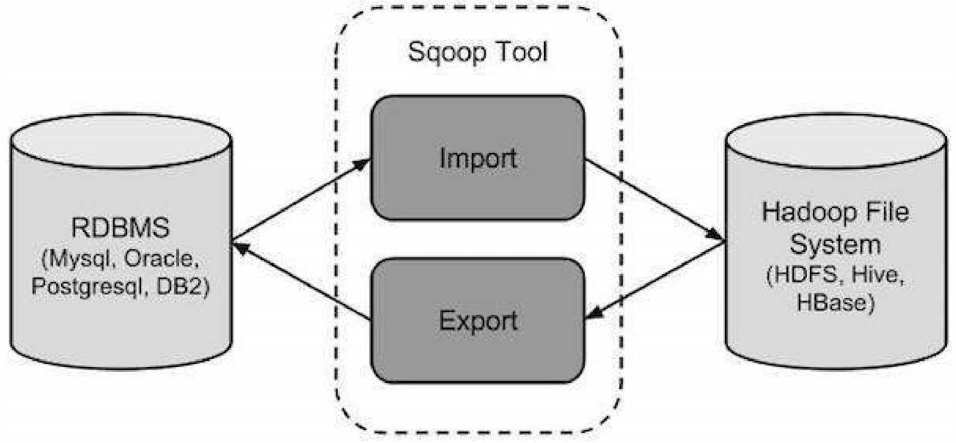
- Sqoop (**SQ**L-to-Had**oop**) is a **MapReduce** based tool designed to transfer data between Hadoop and relational databases such as MySQL, Oracle.
- Easy, parallel database import/export



Sqoop user guide:
https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

Why Sqoop?

- The traditional application management system, that is, the interaction of applications with relational database) is one of the sources that generate Big Data.
- Such Big Data, generated by RDBMS, is stored in Relational Databases
- We need a tool to interact with the relational databases for importing and exporting the Big Data residing in them.
- Sqoop provide feasible interaction between relational databases and Hadoop's HDFS.



How Sqoop Works?

Sqoop Import

The import tool imports individual tables from RDBMS to HDFS.
Each row in a table is treated as a record in HDFS.
All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS.
The files given as input to Sqoop contain records, which are called as rows in table.
Those are read and parsed into a set of records and delimited with user-specified delimiter.

Sqoop Import

- With the generic arguments, you point to your MySQL database and provide the necessary login information.

```
$ sqoop import (generic-args) (import-args)
$ sqoop-import (generic-args) (import-args)
```

- In the import arguments, you have the ability to specify what you want to import and how you want the import to be performed.

The Common Arguments

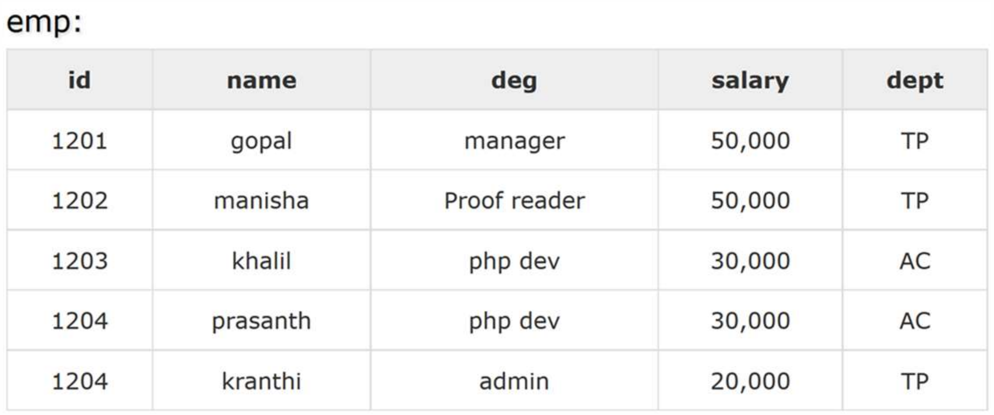
Argument	Description
`--connect <jdbc-uri>`	Specify JDBC connect string
`--connection-manager <class-name>`	Specify connection manager class to use
`--driver <class-name>`	Manually specify JDBC driver class to use
`--hadoop-home <dir>`	Override $HADOOP_HOME
`--help`	Print usage instructions
`-P`	Read password from console
`--password <password>`	Set authentication password
`--username <username>`	Set authentication username
`--verbose`	Print more information while working
`--connection-param-file <filename>`	Optional properties file that provides connection parameters

Argument	Description
`--append`	Append data to an existing dataset in HDFS
`--as-avrodatafile`	Imports data to Avro Data Files
`--as-sequencefile`	Imports data to SequenceFiles
`--as-textfile`	Imports data as plain text (default)
`--boundary-query <statement>`	Boundary query to use for creating splits
`--columns <col,col,col…>`	Columns to import from table
`--direct`	Use direct import fast path
`--direct-split-size <n>`	Split the input stream every *n* bytes when importing in direct mode
`--inline-lob-limit <n>`	Set the maximum size for an inline LOB
`-m,--num-mappers <n>`	Use *n* map tasks to import in parallel
`-e,--query <statement>`	Import the results of *`statement`*.
`--split-by <column-name>`	Column of the table used to split work units
`--table <table-name>`	Table to read
`--target-dir <dir>`	HDFS destination dir
`--warehouse-dir <dir>`	HDFS parent for table destination
`--where <where clause>`	WHERE clause to use during import
`-z,--compress`	Enable compression
`--compression-codec <c>`	Use Hadoop codec (default gzip)
`–null-string <null-string>`	The string to be written for a null value for string columns
`–null-non-string <null-string>`	The string to be written for a null value for non-string columns

University of East London

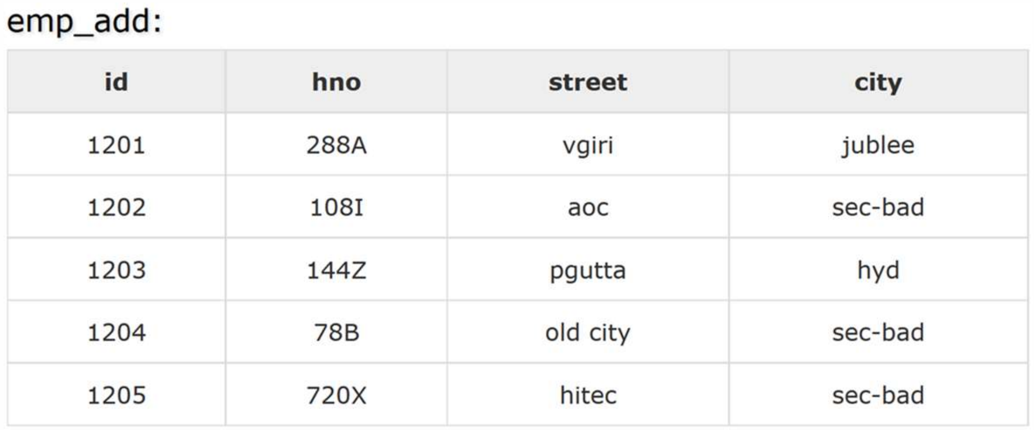
Sqoop Import - Example

Let us take an example of three tables named as ***emp***, ***emp_add***, and ***emp_contact***, which are in a database called ***userdb*** in a MySQL database server.



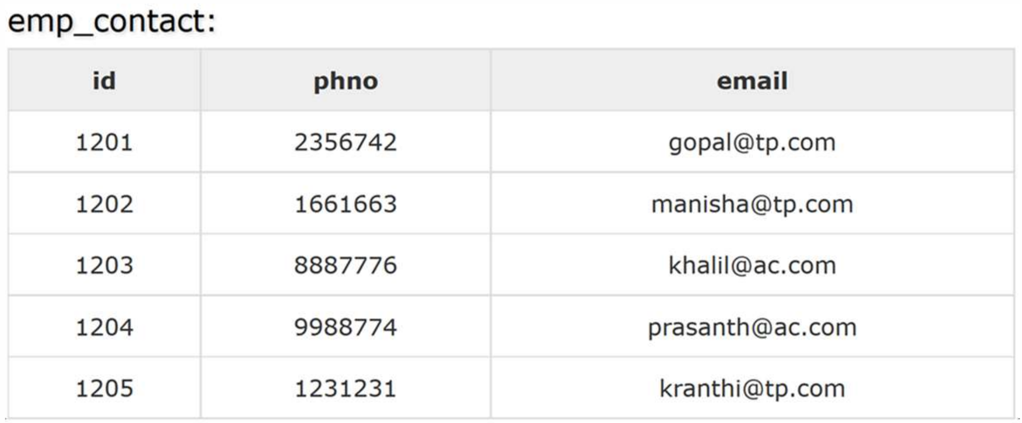
emp:

id	name	deg	salary	dept
1201	gopal	manager	50,000	TP
1202	manisha	Proof reader	50,000	TP
1203	khalil	php dev	30,000	AC
1204	prasanth	php dev	30,000	AC
1204	kranthi	admin	20,000	TP



emp_add:

id	hno	street	city
1201	288A	vgiri	jublee
1202	108I	aoc	sec-bad
1203	144Z	pgutta	hyd
1204	78B	old city	sec-bad
1205	720X	hitec	sec-bad



emp_contact:

id	phno	email
1201	2356742	gopal@tp.com
1202	1661663	manisha@tp.com
1203	8887776	khalil@ac.com
1204	9988774	prasanth@ac.com
1205	1231231	kranthi@tp.com

Source: https://www.tutorialspoint.com/sqoop/sqoop_import.htm

Importing a Table from MySQL to HDFS

```
sqoop import \
    --table emp -m 2 \
    --connect jdbc:mysql://quickstart.cloudera:3306/userdb \
    --username=root \
    --password=cloudera \
    --compression-codec=snappy \
    --as-parquetfile \
    --warehouse-dir=/user/hive/warehouse \
    --hive-import
```

- By default, Sqoop would use 4 map tasks.
- `--table emp -m 2` line specifies the **emp** table and request that two map tasks be used for the import.
- If you have a Hive metastore associated with your HDFS cluster, you can add `--hive-import` at the end.

Importing a Table from MySQL to HDFS

- The `–warehouse-dir` is the location in HDFS where you want the imported table to be placed. We use this directory as we want to access the tables through `Hue`.

- `Parquet` is an open source file format for Hadoop (Hive, Hbase, MapReduce, Pig, Spark). Parquet stores nested data structures in a flat columnar format. Compared to a traditional approach where data is stored in row-oriented approach, parquet is more efficient in terms of storage and performance.

Import Subset of Table Data

We can import a subset of a table using the `where` clause in Sqoop import tool. It executes the corresponding SQL query in the respective database server and stores the result in a target directory in HDFS. The syntax for where clause is as follows.

```
--where <condition>
```

Import Subset of Table Data

```
sqoop import \
    --table emp_add -m 2 \
    --where 'city="sec-bad"' \
    --connect jdbc:mysql://quickstart.cloudera:3306/userdb \
    --username=root \
    --password=cloudera \
    --compression-codec=snappy \
    --as-parquetfile \
    --warehouse-dir=/user/hive/warehouse \
    --hive-import
```

output

```
1202, 108I, aoc,     sec-bad
1204, 78B,  oldcity, sec-bad
1205, 720C, hitech,  sec-bad
```

Import All Tables

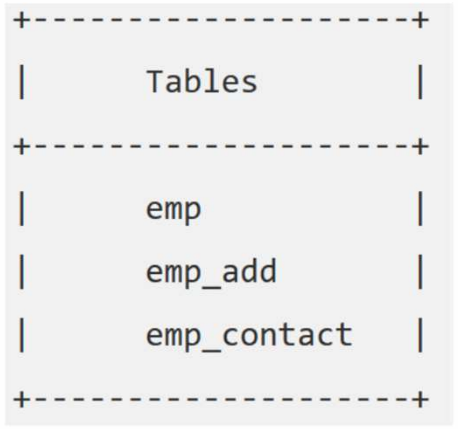
Import all the tables from the RDBMS database server to the HDFS. Each table data is stored in a separate directory and the directory name is same as the table name.

```
$ sqoop import-all-tables (generic-args) (import-args)
$ sqoop-import-all-tables (generic-args) (import-args)
```

Import All Tables - Example

```
sqoop import-all-tables \
    -m 2 \
    --connect jdbc:mysql://quickstart.cloudera:3306/userdb \
    --username=root \
    --password=cloudera \
    --compression-codec=snappy \
    --as-parquetfile \
    --warehouse-dir=/user/hive/warehouse \
    --hive-import
```

Note: If you are using the `import-all-tables`, it is mandatory that every table in that database must have a primary key field.



```
+--------------------+
|      Tables        |
+--------------------+
|      emp           |
|      emp_add       |
|      emp_contact   |
+--------------------+
```

Incremental Import

- Incremental import is a technique that imports only the newly added rows in a table. It is required to add ‘incremental’, ‘check-column’, and ‘last-value’ options to perform the incremental import.

```
--incremental <mode>
--check-column <column name>
--last value <last check column value>
```

Example: Let us assume the newly added data into *emp* table is as follows:

```
1206, Mike T., admin, 20000, GR
```

Incremental Import

- The incremental import command would be:

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp \
--m 1 \
--incremental append \
--check-column id \
--last value 1205
```

Verify Import Data in HDFS

- The following command is used to verify the imported data from *emp* table to HDFS *emp/* directory.

```
$ hdfs dfs -cat /emp/part-m-*
```

- The following command is used to see the modified or newly added rows from the *emp* table.

```
$ hdfs dfs -cat /emp/part-m-*1
```

- Or, easily see all the imported data in DataNode using web UI.

Sqoop Export

The aim is to export data back from the HDFS to the RDBMS database.

```
$ sqoop export (generic-args) (export-args)
$ sqoop-export (generic-args) (export-args)
```

- The target table must exist in the target database.
- The files which are given as input to the Sqoop contain records, which are called rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.
- The default operation is to insert all the record from the input files to the database table using the INSERT statement.
- In update mode, Sqoop generates the UPDATE statement that replaces the existing record into the database.
- It only understands HDFS directories not Hive or Hcatalog.

Sqoop Export - Example

Let us take an example of the employee data in file, in HDFS. The employee data is available in `emp_data` file in ‘`emp/`’ directory in HDFS.

It is mandatory that the table to be exported is created manually and is present in the database from where it has to be exported.

So, we need to first create the table ‘`employee`’ in mysql and then, export the table data (which is in `emp_data` file on HDFS) to the employee table in db database of Mysql database server.

Sqoop Export - Example

```
sqoop export \
    --table employee \
    --connect jdbc:mysql://quickstart.cloudera:3306/userdb \
    --username=root \
    --password=cloudera \
    --export-dir /emp/emp_data
```

The following command is used to verify the table in mysql command line.

```
mysql>select * from employee;
```

Sqoop MetaStore tool

- A *Sqoop metastore* is used to store Sqoop job information in a central place. This helps the **collaboration between Sqoop users and developers**.
- It is a shared metadata repository, in which remote users and/or multiple users can execute and define saved jobs in the metastore.
- For example, a user can create a job to load some specific data. Then any other user can access from any node in the cluster the same job and just run it again. This is very convenient when using Sqoop in Oozie workflows.

```
$ sqoop metastore (generic-args) (metastore-args)
$ sqoop-metastore (generic-args) (metastore-args)
```

How to Execute Sqoop MetaStore?

- Running `sqoop-metastore` launches a shared HSQLDB database instance on the current machine. Clients can connect to this metastore and create jobs which can be shared between users for execution.
- The location of the metastore's files on disk is controlled by the `sqoop.metastore.server.location` property in `conf/sqoop-site.xml`. This should point to a directory on the local filesystem.
- The metastore is available over TCP/IP. The port is controlled by the `sqoop.metastore.server.port` configuration parameter, and defaults to 16000.
- Clients should connect to the metastore by specifying `sqoop.metastore.client.autoconnect.url` or `--meta-connect` with the value `jdbc:hsqldb:hsql://<server-name>:<port>/sqoop`.
- This metastore may be hosted on a machine within the Hadoop cluster, or elsewhere on the network.

Sqoop Merge

- *Sqoop merge* tool enables you to combine two data sets (flatten two data sets into one), whereby entries in one data set overwrite entries in an older data set. It is a good tool for incremental import, where we have multiple mappers (blocks) in HDFS.

```
$ sqoop merge (generic-args) (merge-args)
$ sqoop-merge (generic-args) (merge-args)
```

Argument	Description
`--class-name <class>`	Specify the name of the record-specific class to use during the merge job.
`--jar-file <file>`	Specify the name of the jar to load the record class from.
`--merge-key <col>`	Specify the name of a column to use as the merge key.
`--new-data <path>`	Specify the path of the newer dataset.
`--onto <path>`	Specify the path of the older dataset.
`--target-dir <path>`	Specify the target path for the output of the merge job.

Example:

```
$ sqoop merge -new-data newer -onto older -target-dir HDFS path
```

Other useful Sqoop tools

- *sqoop-list-databases:* List databases present on a server.
- *sqoop-list-tables:* List tables present in a database.
- *sqoop-job:* The job tool allows you to create and work with saved jobs. Saved jobs remember the parameters used to specify a job, so they can be re-executed by invoking the job by its handle.
- *sqoop-create-hive-table:* To create a hive warehouse.

Source: https://sqoop.apache.org/docs/1.4.2/

Sqoop Performance Optimisation - Best Practice

1. **Adjusting the Number of Map Tasks:**
    1. Sqoop uses MapReduce to import/export data, with each map task handling a portion of the data.
    2. Increasing the number of map tasks (-m option) can parallelise the process and improve performance.
    3. Choose the number of map tasks based on the size of the data and the number of database partitions to avoid bottlenecks.
2. **Optimizing Data Splits:**
    - Properly configure the `--split-by` option to specify the column used for dividing data among map tasks.
    - Select columns with high cardinality (many distinct values) to achieve balanced data splits.
3. **Incremental Imports:**
    - Use incremental import options (`--incremental`, `--check-column`, `--last-value`) to fetch only the newly added or updated rows.
    - Reduces the amount of data transferred, saving time and resources

File Formats and Compression Techniques

1. **Choosing Appropriate File Formats:**
   - Parquet: A columnar storage format that provides efficient compression and encoding for large datasets.
   - Avro: A row-oriented format suitable for data serialization and data interchange.
   - Text or Sequence Files: Use these formats if compatibility with other tools is required, but they may offer less efficient compression.

2. **Using Compression Techniques:**
   - Enable compression (`--compression-codec`) to reduce the size of the imported/exported data.
   - Common codecs include `Snappy`, `Gzip`, and `Bzip2`.
   - Snappy is recommended for faster compression/decompression with moderate compression ratios.
   - Gzip offers higher compression but is slower.

3. **Tuning JDBC Fetch Size:**
   - Configure the JDBC fetch size (`--fetch-size`) to optimize the number of rows fetched from the database in each round trip.
   - A higher fetch size can improve import performance, especially for large tables.

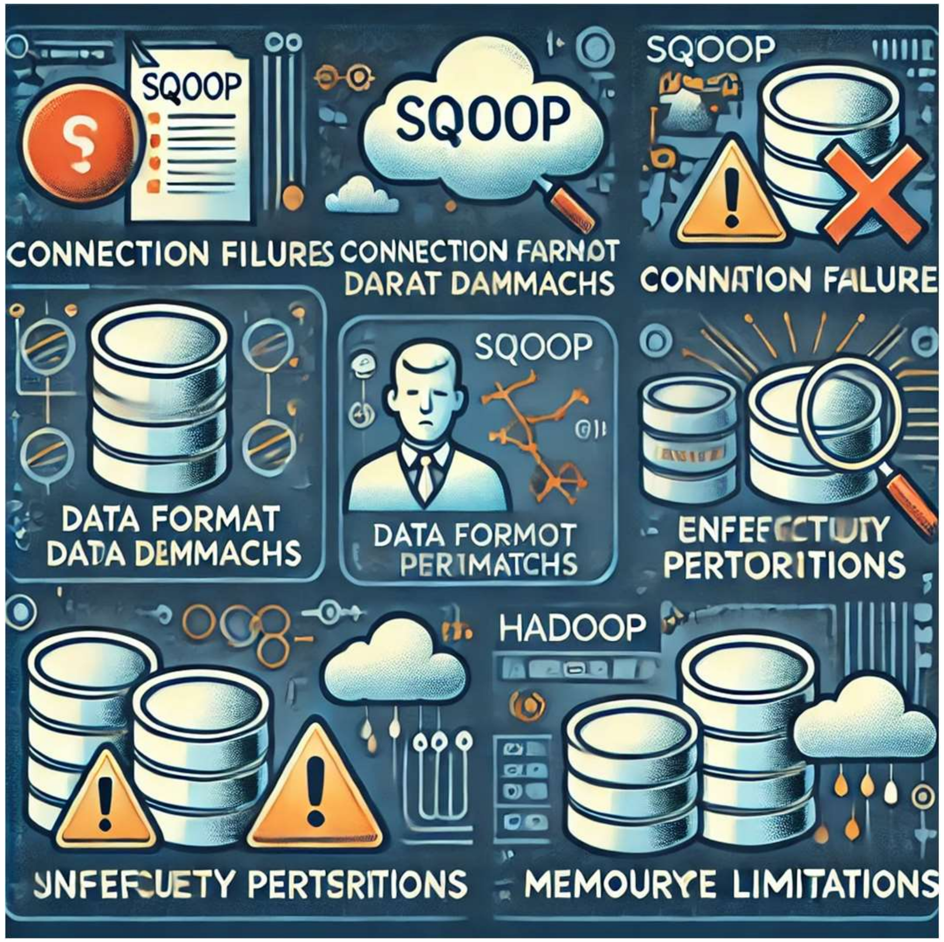
Common Issues in Sqoop Operations

1. **Connection Failures:**

- Commonly due to incorrect database connection parameters (hostname, port, username, password).
- Verify the JDBC URL and ensure that the database server is accessible.
- Check network configurations and firewall settings.

2. **Data Format Mismatches:**

- Occurs when the data types in the source database do not match the expected formats in Hadoop.
- Resolve by specifying appropriate data types using the `--map-column-java` option.
- Ensure that the imported data conforms to the Hadoop ecosystem's supported formats.



University of East London

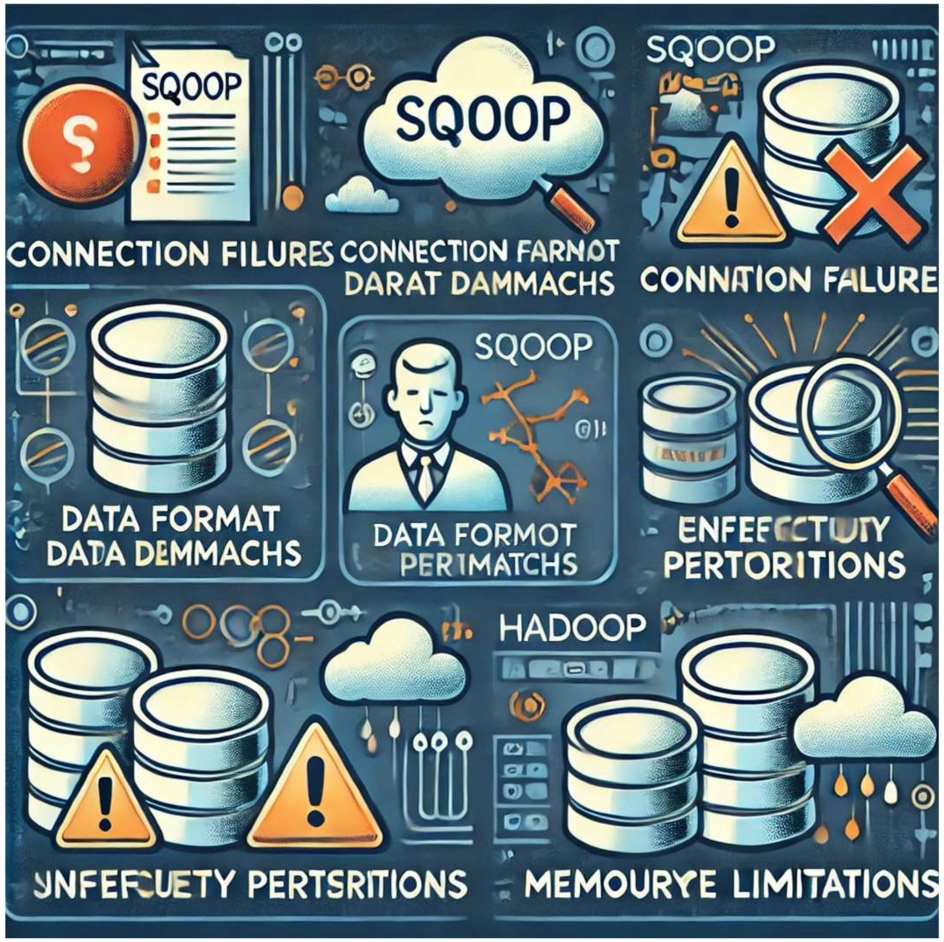
Common Issues in Sqoop Operations

3. **Insufficient Permissions:**

- Database user may not have sufficient permissions to read data or execute required queries.
- Ensure the database user has necessary read/write access and appropriate roles.

4. **Memory and Resource Limitations:**

- Import/export tasks may fail if there is insufficient memory or processing power.
- Tune Hadoop and Sqoop configurations to allocate more resources for large jobs.



University of East London

Logs, Error Codes, and Retry Mechanism

1. **Using Sqoop Logs for Troubleshooting:**

- Enable verbose logging (`--verbose`) to get detailed output about the operation.
- Check Hadoop job tracker logs for errors related to MapReduce tasks.
- Review Sqoop log files to identify specific issues and their causes.

2. **Understanding Common Error Codes:**

- *Error Code 101*: Connection failure. Verify the JDBC URL, network, and database settings.
- *Error Code 110*: Authentication failure. Check username and password.

University of East London

Logs, Error Codes, and Retry Mechanism

- *Error Code 120:* Insufficient memory. Increase the `mapreduce.job.memory.mb` parameter.

3. **Implementing Retry Mechanisms:**

- Use `--num-mappers` option to adjust the number of map tasks, as failed map tasks can be retried automatically.
- Configure Hadoop's `mapreduce.map.maxattempts` parameter to specify the number of retry attempts for failed tasks.
- Set up Sqoop jobs to rerun in case of failures, using tools like Oozie for job scheduling.

University of East London

Sqoop Alternatives: Comparison

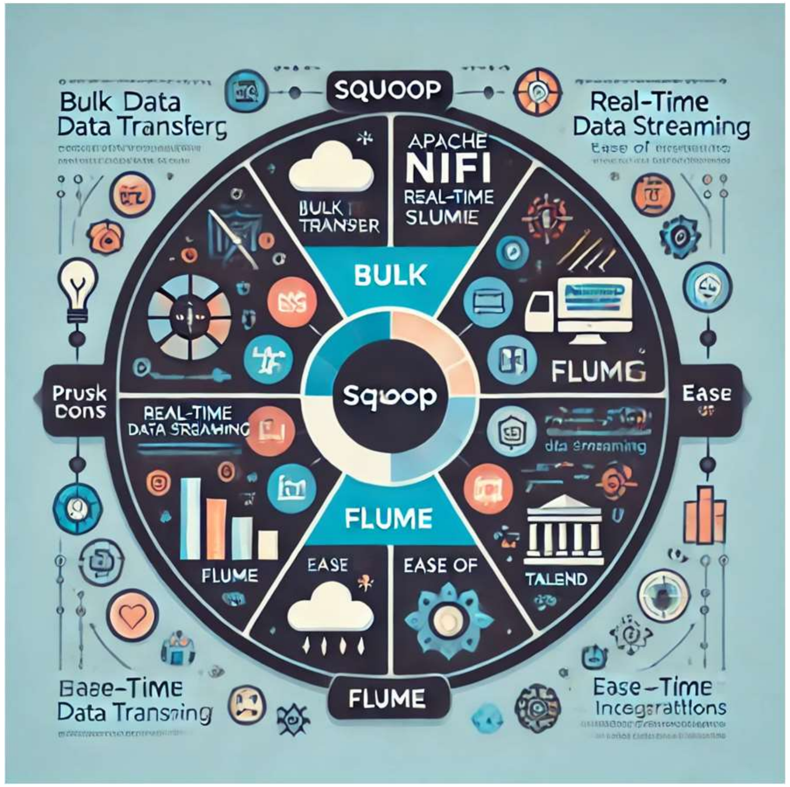
Several tools can transfer data between databases and Hadoop, each with unique features:

Apache Nifi:

- A data integration tool with real-time data flow management capabilities.
- Ideal for complex workflows requiring data transformation, routing, and tracking. Supports data ingestion from various sources in different formats.

Apache Flume:

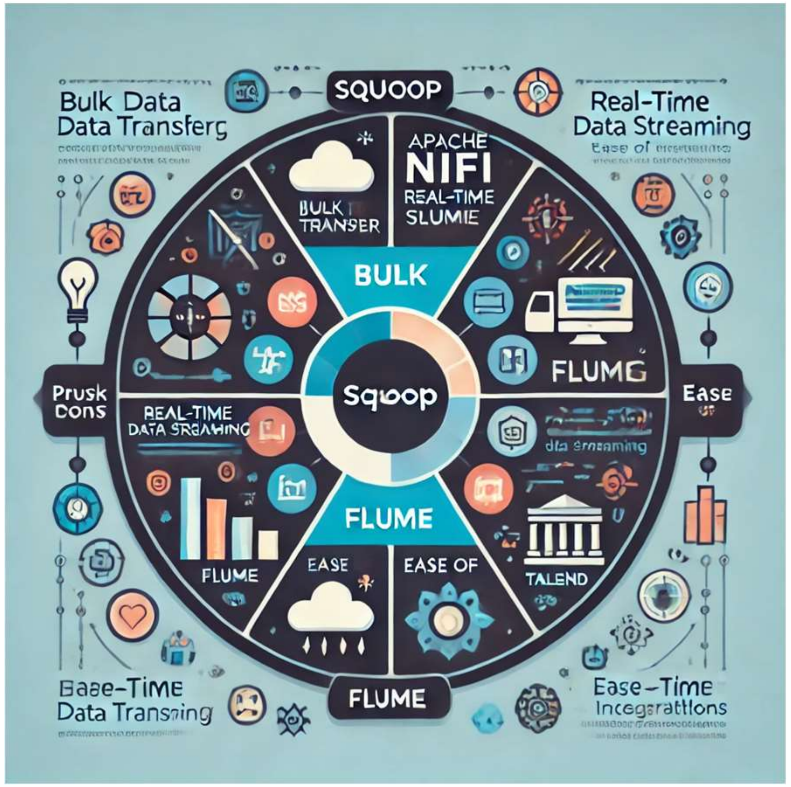
- Specialised in collecting and transporting large volumes of log data.
- Suitable for streaming data sources like web server logs, social media feeds, or event logs.
- Works well for continuous data ingestion into Hadoop.



Sqoop Alternatives: Comparison

Talend:

- An ETL (Extract, Transform, Load) tool with a graphical interface for data integration.
- Provides many connectors for databases, cloud services, and big data platforms.
- Suitable for complex ETL processes and data transformations.



Sqoop: Pros and Cons vs Alternatives

Pros of Using Sqoop:
- Designed for efficient bulk data transfers between RDBMS and Hadoop.
- Supports parallel data import/export, speeding up the process.
- Integrates tightly with Hadoop components like HDFS, Hive, and HBase.

Cons of Using Sqoop:
- Not suitable for real-time or streaming data transfers.
- Limited data transformation capabilities compared to ETL tools.
- Less flexibility in complex workflows or non-tabular data handling.

Comparison Highlights:
- Apache Nifi and Flume are better suited for real-time streaming and complex data flows.
- Talend offers a more user-friendly interface for data transformation and supports various data sources.
- Sqoop is ideal for simple, bulk data migration tasks but may need to be combined with other tools for more advanced requirements.

Summary

- Introduced Apache Sqoop
- Explained why it is needed
- Learnt Sqoop import and export commands
- Discussed advanced Sqoop components: MetaStore and Merge

University of East London