

**Processing Unstructured Data**

One of the most common problems encountered while dealing with big data or the Hadoop ecosystem is processing large volumes of unstructured logs.

In spite of having a tremendous volume and being unstructured in nature, where conventional RDBMSs and SQL fail, Hive is capable of processing unstructured (or rather, semi-structured) logs.

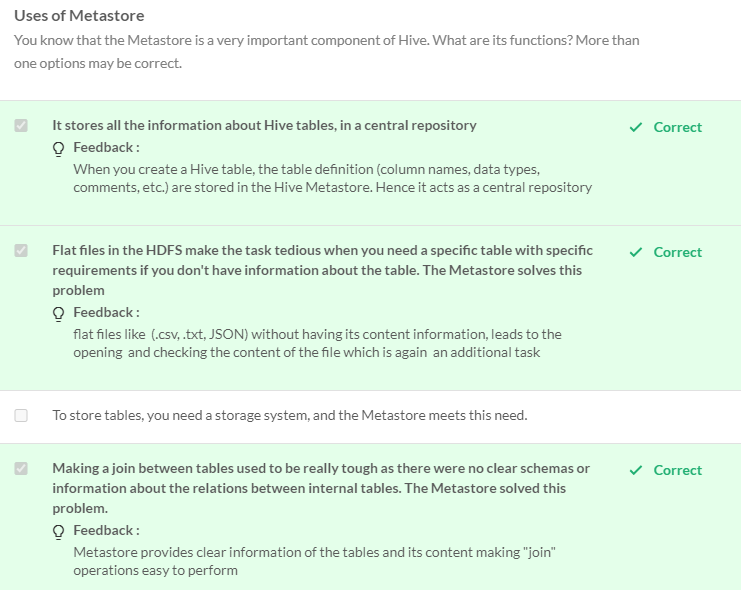
Besides, it is also used to parse unstructured data and store it in a query-able structured form.

To summarise, the main features of Hive are —

* An SQL-like interface to write queries such as SELECT, UPDATE, INSERT, DELETE, etc.
* A variety of built-in functions for working with dates, strings, etc.
* Easy ETL (extraction, transformation, and loading) of data
* Storage of both simple and complex data types. e.g. integers, arrays, etc.

Use cases of Hive



The Hive Metastore is the central repository of the software. It stores all the metadata about the data stored in the HDFS, such as the names of the tables, the columns, the data types, the dates of creation of the respective tables, etc., but **not the data itself**. 

**Hive Data Models**

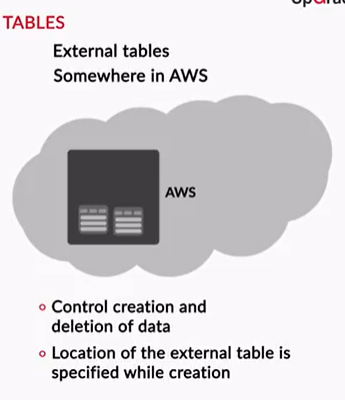
Hive stores and queries data using its data models. The purpose of using data models is to make querying convenient and fast. There are four main components in Hive data models, which are similar to how an RDBMS stores data:

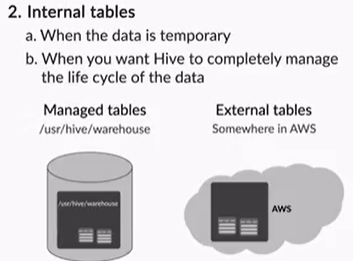
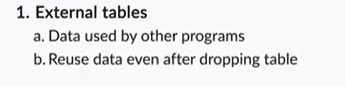
* Databases
* Tables
* Partitions
* Buckets

Hive has two types of tables:

* Managed (or internal) table
* External table

Note that**'managed table'**and**'internal table'**are synonymousterms.





To summarise, there are two types of tables in Hive:

* Managed/internal table
* External table

The benefit of using an external table is that when you drop a certain external table, Hive deletes only the metadata of the table; it does not touch the data itself (which is stored somewhere on the HDFS/S3).

A crucial advantage of external tables is that even when you drop a table, the data is still available for use to other programs running on the same cluster, such as Spark, Pig, etc.

On the other hand, when you drop (or delete) a managed table, both the metadata and the data are deleted.

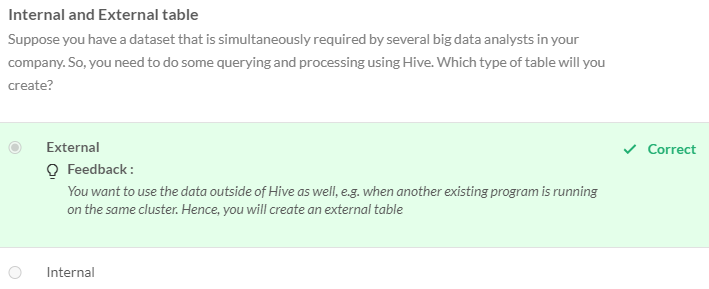
You should use external tables when —

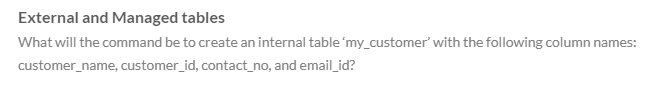
* You want to use the data outside of Hive as well. For example, when another existing program is running on the same cluster
* You want the data to remain stored on the HDFS even after dropping tables because Hive does not delete the data stored outside (of the Hive database).
* You do not want Hive to control the storage of your data (location/directories of storage/etc.).

On the other hand, you use managed tables when —

* The data is temporary. So, when the Hive table is dropped, the data stored in the internal table is deleted too.
* You want Hive to manage the life cycle of the data completely, i.e. both store and process it.

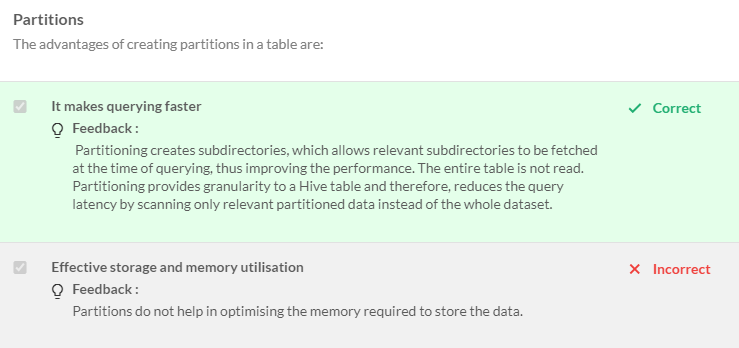
The **deserialiser** is used to **read files in a given format into Hive**, while the **serialiser** is used to **write files back to S3 or the HDFS** (in a specified format).





Apache Hive was created to help analysts and engineers query the big data stored in Hadoop (HDFS) using a SQL-like language. Apart from the ease of querying big data, Hive also provides various other features such as —

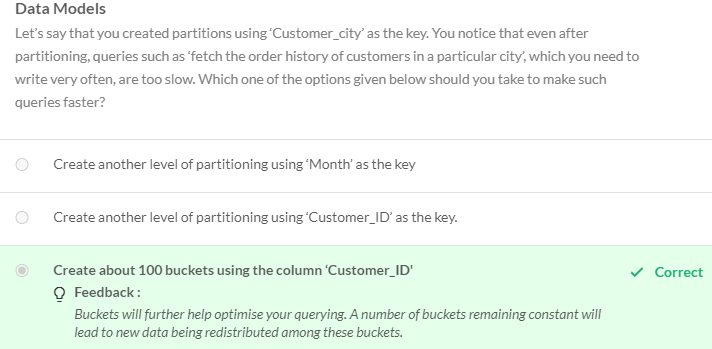
* The processing of semi-structured and unstructured data
* Various built-in functions to manipulate dates, strings, etc.
* Partitioning and bucketing techniques to improve query performance

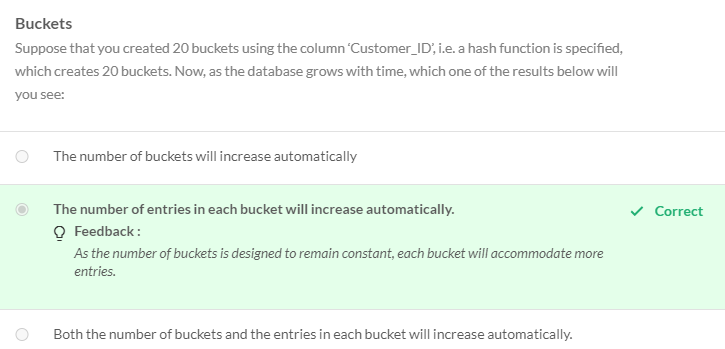


In this session, you studied the following:

* Partitions
* Creating and Querying Partitions
* Buckets
* Creating and Querying using buckets
* Advantages of these Data Models

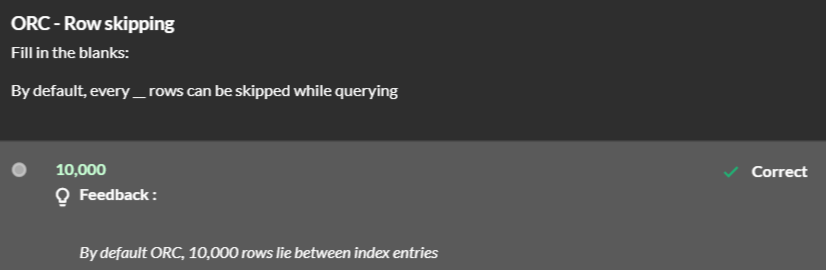
You further learnt its a good practice implementing these data models during Big data processing and analysis. This is an example of **subsetting the data**to make it more **manageable and faster** to work with. They improvise the performance and analysis speed significantly.





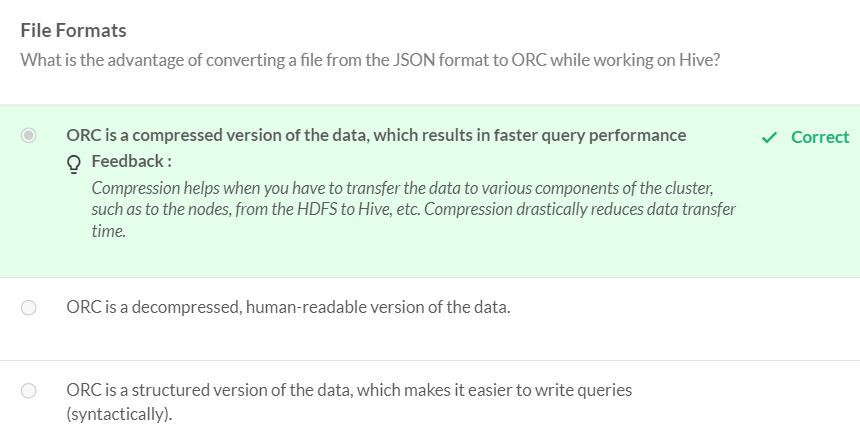
File Types in Hive:

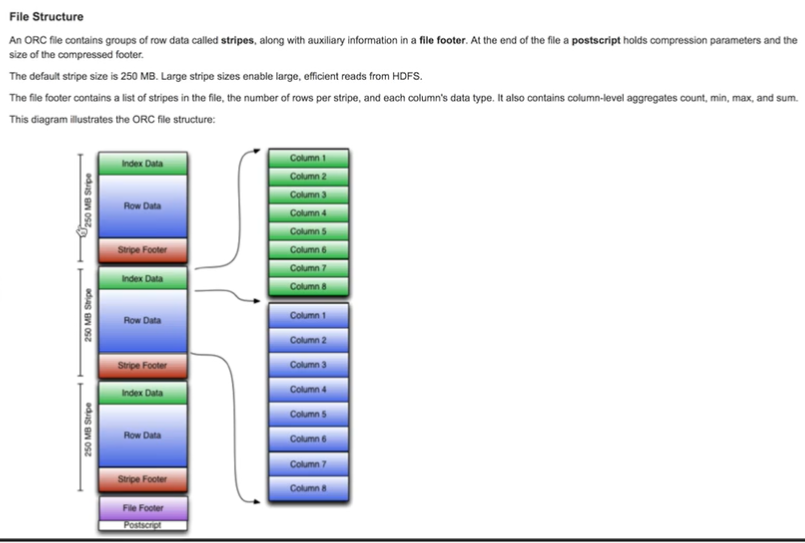
ORC – Optimized Row Columnar with partitioned rowas (10000 by default) called Stripes

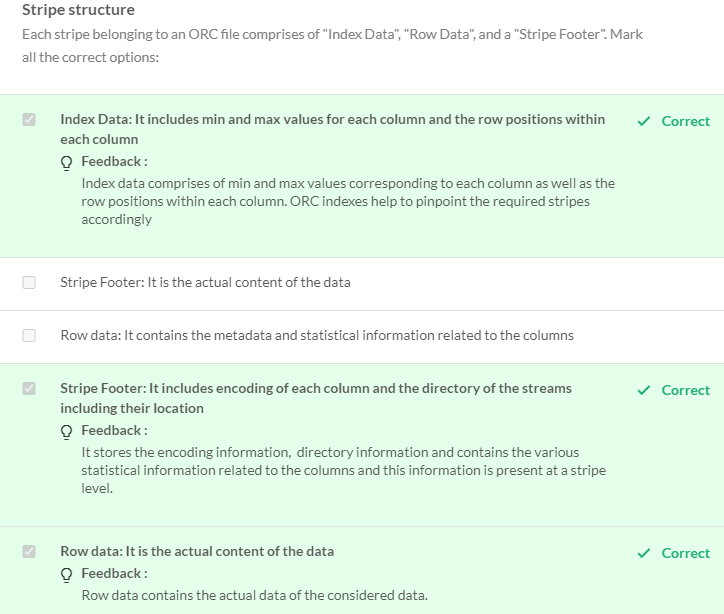


To summarise, Hive uses various (compressed) file formats to speed up query performance. By using compressed file formats, Hive is able to improve performance while reading, writing, and processing data.

Note that here SerDes (serialisers and deserialisers) play an important role in that they basically allow Hive to work with files in various compressed formats such as JSON, ORC, Parquet, etc.





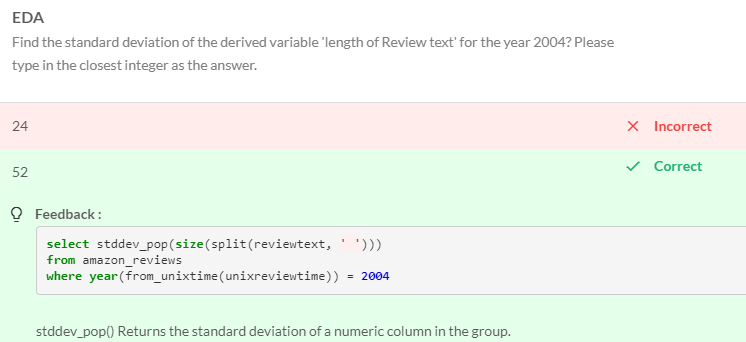
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC>

ORC (Optimized Row Columnar) is one of the most popular file formats, and in some cases, it literally enhances the querying speed by an order of magnitude.

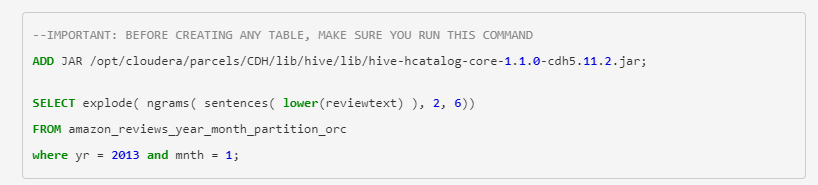
It does that by compressing the data into a (human non-readable) binary format and storing it within 'stripes of rows', each stripe containing the information about the min-max values of the stripe (index data), other metadata and the data itself.  While querying, Hive makes use of the index data to skip processing unnecessary stripes, thereby saving computational cost.

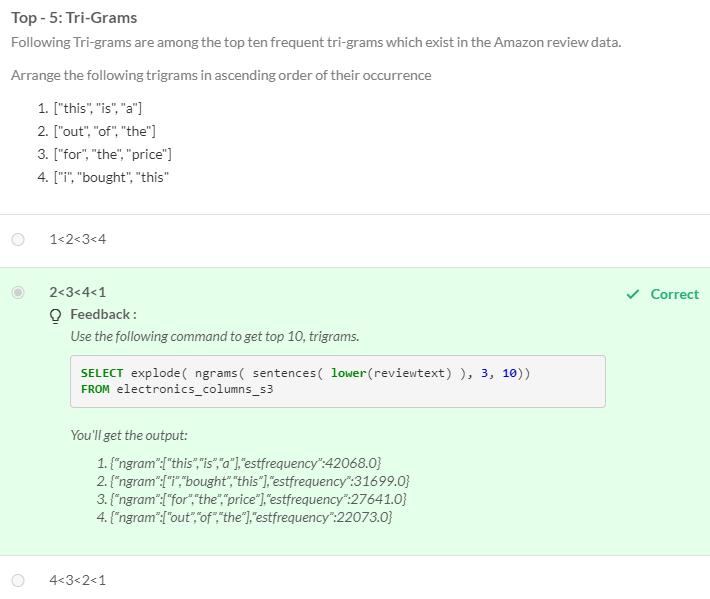
Other popular file formats are Parquets, Avro, RC etc.

<https://martin.atlassian.net/wiki/spaces/lestermartin/pages/19431488/Hive+Cheat+Sheet>



N Grams (2-Gram, Top 6 N-Grams)



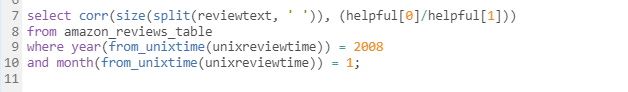


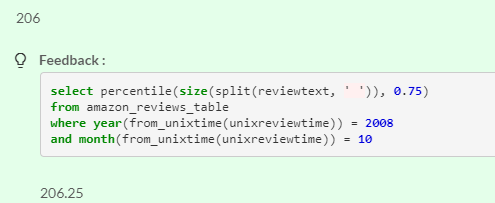
**Primitive data types** are simple data types and they are the basic building blocks of any programming language. For example, integer, string, etc.

**Complex data types** are used to store complex numbers and complex arithmetic structures such as struct, matrix, and arrays.

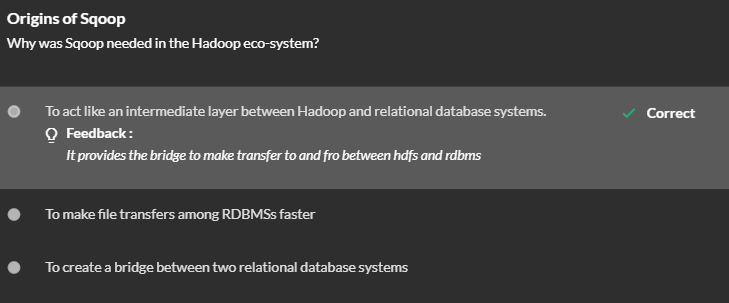
Complicated data types:

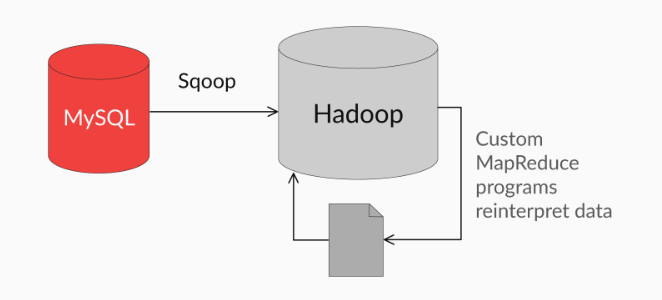
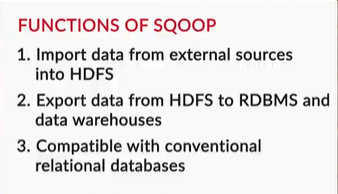






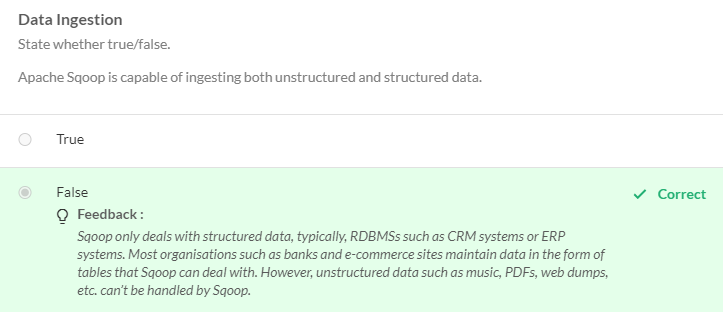
SQOOP

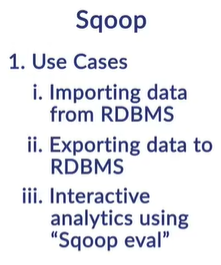




 Note that Sqoop is used to ingest structured data, while Flume is a tool used for semi-structured or unstructured data.

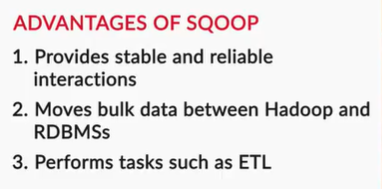
Since most of the data stored in RDBMSs is structured, Sqoop is more commonly used than Flume, and we will focus only on Sqoop in this course.



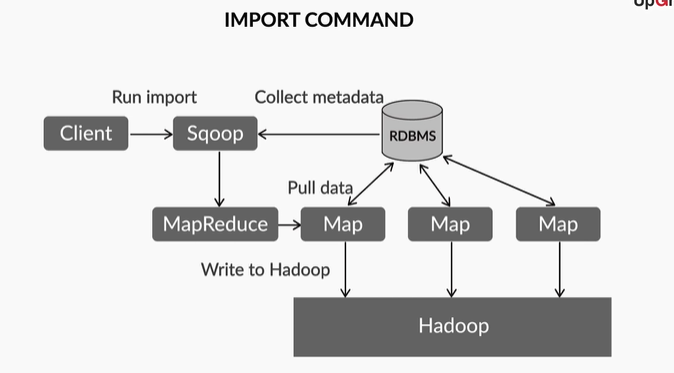


the three industry use cases of Sqoop:

1. Importing data from RDBMSs
2. Exporting data to RDBMSs
3. Interactive analytics on RDBMSs



Data Ingestion via SQOOP



To recap, the steps involved in an import command are:

1. User runs sqoop command through the shell
2. Sqoop collects metadata from the RDBMS (e.g. MySQL)
3. Sqoop launches MapReduce job (the Sqoop command will automatically be converted into the Java nature of a MapReduce job, using the metadata information)
4. Mappers are created against primary key ranges of the RDBMS, and the data is written into the Hadoop ecosystem

The basic idea of importing large datasets is to divide them into multiple parts and import these parts parallelly using a MapReduce-like paradigm.

As is the standard case, Sqoop runs the map phase first and divides the data into multiple smaller parts, each of which is assigned to a mapper. However, there is**no need for a reduce phase** since there is no aggregation required - we are writing into an HDFS type destination!

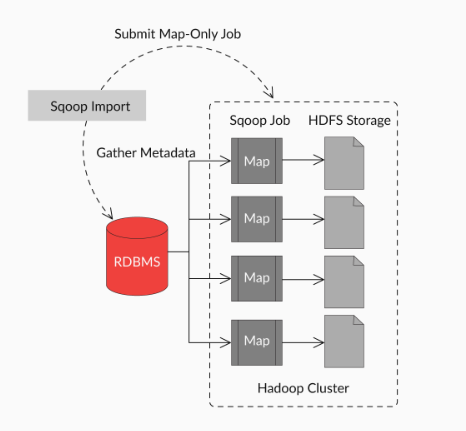
# Comprehension: How Sqoop Import Works

Sqoop uses the MapReduce programming model to execute parallel imports. The idea is to parallelise the import of large tables by dividing them into smaller subsets.

There are three main stages in performing an import. Let’s understand each of these steps individually:

1. **Making a connection with the database/RDBMS:** In the first step, Sqoop makes a connection with the RDBMS using what is called a ‘connector’. Connectors allow Sqoop to ignore the differences in the ‘SQL dialects’ of different databases (e.g. Teradata, Netezza, Oracle, MySQL, PostgreSQL) and allow it to treat all of them simply as RDBMSs.
2. **Reading the table to be imported and fetching the metadata:** After making a connection, Sqoop inspects the table to be imported and collects metadata such as the number of rows, number of columns, their names, the data types, etc.
3. **Launching a MapReduce program to perform parallel imports:** In this step, Sqoop divides the data into multiple parts and assigns a mapper to each part. Each of these mappers transfers a part of the table to Hadoop in a parallelised manner.

Sqoop Performs Import – Fig



While performing parallel imports, Sqoop uses a splitting column to split the workload. By default, it identifies the primary key column (if present) in a table and uses it as the splitting column. The highest and lowest values of the splitting column are retrieved from the database, and the map tasks operate on the individual subsets of the table.

For example, if the table has a primary key column ‘user\_ID’ (as integers), whose minimum value is '0' and maximum value is 'n', and Sqoop is directed to use, say, four tasks (the default number of mappers is four), it would run four processes (one for each task) using SQL statements of the form

SELECT \* FROM sometab WHERE id ≥ lo AND id < hi

with (lo, hi) set to (0, n/4), (n/4, n/2), (n/2, 3n/4), and (3n/4, n+1) in the four tasks respectively.

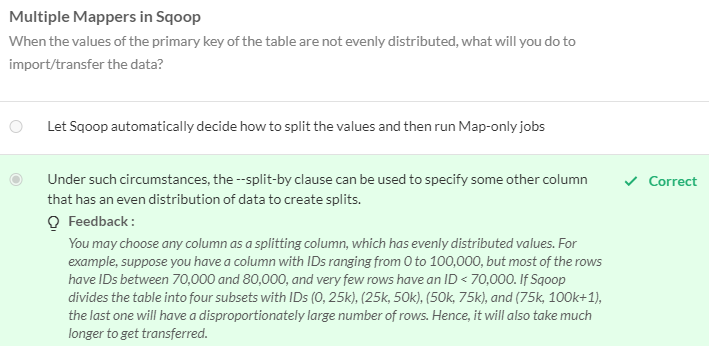
To summarise, Sqoop follows the following procedure for data ingestion:

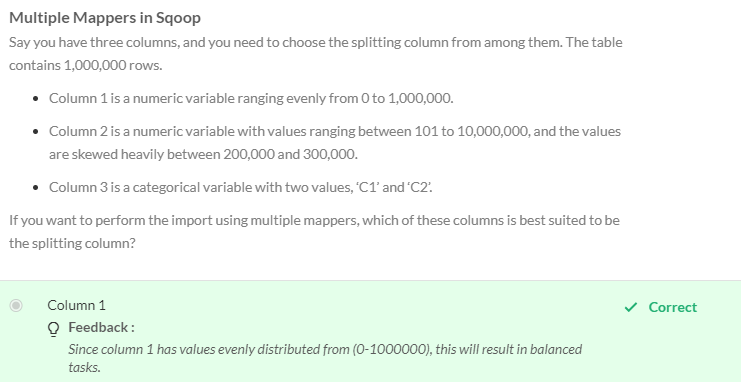
1. It looks at the range of the primary key (from the splitting/primary key column).
2. It sets the lower value of the primary key to some variable.
3. It sets the higher value of the primary key to another variable.
4. It generates SQL queries to fetch the data parallelly.

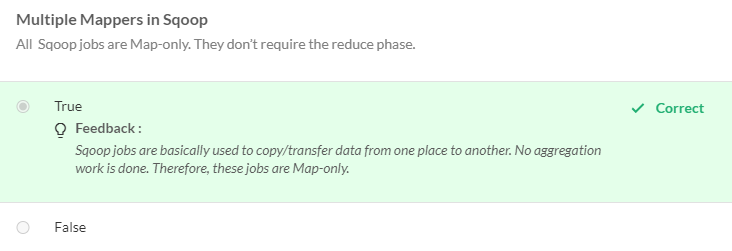
If the values of the primary key column/splitting column are not uniformly distributed across its range, this can result in unbalanced tasks. In such cases, you are advised to choose a different column using the '--split-by' clause explicitly.

For example, to specify that Sqoop should use the ‘customer\_ID’ column for splitting, you can write "--split-by customer\_ID".

The export operation is similar to the import operation, except that it works in the reverse direction. In the export operation too, the number of maps is specified, and then, the respective data blocks are assigned to each mapper program for transfer.





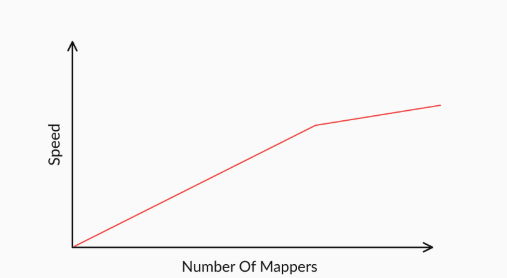


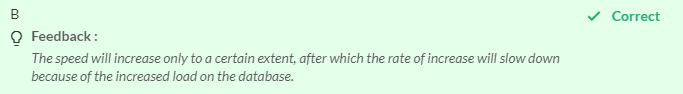
## Choosing the Optimal Number of Mappers

You can also specify the number of mappers manually, in which case, you need to choose an appropriate number of them. In general, using more mappers will lead to a higher degree of parallelisation, leading to a higher speed. However, it will also increase the load on the database, which may affect the other requests your database needs to handle.

Also, most databases support only a limited number of parallel connections, and this number is the upper bound you have for the number of mappers.

Thus, you need to use a ‘hit-and-trial’ approach to find the optimal number of mappers, which will be specific to your hardware, database, types of tables, etc.





If you look at Sqoop's [documentation](https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_purpose_11), eval is only used for **simple queries and testing connections.** As an analyst, though, it also gives you the ability to query easily with immediately visible results. As an example, you may use it if you quickly need to refer to some aggregated numbers such as "total sales", "number of purchases from 'Electronics' category", depending on what your industry is. Instead of making a separate connection to your relational database, you can use Sqoop to find these quick results.

## Additional Reading

* [Take a look at the syntax of Sqoop's eval command](https://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html#id1772462)
* There are times when you need to perform certain operations repeatedly. In such cases, the 'job' command is typically used. You store the frequently used or repeatedly used command as a Sqoop job. You can then execute that job as and when needed. Also, you can store ANY Sqoop command inside a job.  [Read more about Sqoop jobs here.](https://www.hdfstutorial.com/sqoop-jobs/)

Using big data for analytics and data processing requires the ingestion of the data into clusters for further processing. This is where Apache Sqoop fits in.

Sqoop allows easy and secure transfers via the import and export of data from structured data stores such as RDBMSs, EDW, etc.

Using Sqoop, you can provide the data from the external system onto the HDFS and populate tables in Hive.

You saw how easy it is to transfer large datasets from Hadoop to external data stores such as relational databases, using Sqoop. Beyond this, Sqoop offers many advanced features such as different data formats, compression, working with queries instead of tables, etc.

In this session, you also learnt the Sqoop commands and performed various operations on the RDS. The hands-on session started with the creation of RDS instances/databases. Then, you migrated the local database from the MySQL workbench to the RDS, and you configured Sqoop for a MySQL connector.

The session covered the following commands:

* list-databases: This command is used to list the databases available on the RDS.
* list-tables: This command lists all the tables of a database mentioned in the command.
* import: This command will transfer a table from the RDBMS (the RDS, in our case) to the HDFS.
* import-all-tables: This command will transfer all the tables from a database to the HDFS.
* job: This command allows you to save and execute repeatedly used commands.
* eval: This command will help you execute SQL queries.

Remember the options you used during the lab session. The important ones are —

* --connect: This will help you connect to the intended RDS/RDBMS.  Remember the port in your case is 3306 as you're using a MySQL database.
* --username: This will specify the username that you want to use to connect to the database.
* -P: This is always recommended as a prompt for the password.
* --warehouse-directory: This will directly place the imported tables inside the warehouse directory, which is easily accessible to Hive.

Graded Questions