* **Sequence File Format:**

SequenceFile is a flat file consisting of binary key/value pairs. It is extensively used in MapReduce as input/output formats. It is also worth noting that, internally, the temporary outputs of maps are stored using SequenceFile.

The SequenceFile provides a Writer, Reader and Sorter classes for writing, reading and sorting respectively.

There are 3 different SequenceFile formats:

1. Uncompressed key/value records.

2. Record compressed key/value records - only 'values' are compressed here.

3. Block compressed key/value records - both keys and values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.

The recommended way is to use the SequenceFile.createWriter methods to construct the 'preferred' writer implementation.

The SequenceFile.Reader acts as a bridge and can read any of the above SequenceFile formats.

* **NLine Input Format:**

NLineInputFormat which splits N lines of input as one split.

In many "pleasantly" parallel applications, each process/mapper processes the same input file (s), but with computations are controlled by different parameters.(Referred to as "parameter sweeps"). One way to achieve this, is to specify a set of parameters (one set per line) as input in a control file (which is the input path to the map-reduce application, where as the input dataset is specified via a config variable in JobConf.).

Performing a large number of these from several reduce tasks concurrently can swamp a database. If you want to export a very large volume of data, you may be better off generating the INSERT statements into a text file, and then using a bulk data import tool provided by your database to do the database import.

The NLineInputFormat can be used in such applications, that splits the input file such that by default, one line is fed as a value to one map task, and key is the offset. i.e. (k,v) is (LongWritable, Text). The location hints will span the whole mapred cluster.

* **DB Input Format**

Apache Hadoop’s strength is that it enables ad-hoc analysis of unstructured or semi-structured data. Relational databases, by contrast, allow for fast queries of very structured data sources. A point of frustration has been the inability to easily query both of these sources at the same time. The DBInputFormat component provided in Hadoop 0.19 finally allows easy import and export of data between Hadoop and many relational databases, allowing relational data to be more easily incorporated into your data processing pipeline.

DBInputFormat uses JDBC to connect to data sources. Because JDBC is widely implemented, DBInputFormat can work with MySQL, PostgreSQL, and several other database systems. Individual database vendors provide JDBC drivers to allow third-party applications (like Hadoop) to connect to their databases.To start using DBInputFormat to connect to your database, you’ll need to download the appropriate database driver.

Reading Tables with DBInputFormat:

The DBInputFormat is an InputFormat class that allows you to read data from a database. An InputFormat is Hadoop’s formalization of a data source; it can mean files formatted in a particular way, data read from a database, etc. DBInputFormat provides a simple method of scanning entire tables from a database, as well as the means to read from arbitrary SQL queries performed against the database. Most queries are supported, subject to a few limitations

* **DB Output Format:**

DBOutputFormat accepts <key,value> pairs, where key has a type extending DBWritable.

Returned RecordWriter writes only the key to the database with a batch SQL query.

The DBOutputFormat.setOutput() method then defines how the results will be written back to the database. Its three arguments are the JobConf object for the job, a string defining the name of the table to write to, and an array of strings defining the fields of the table to populate. e.g., DBOutputFormat.setOutput(job, "employees", "employee\_id", "name");.

The DBOutputFormat writes to the database by generating a set of INSERT statements in each reducer. The reducer’s close() method then executes them in a bulk transaction. Performing a large number of these from several reduce tasks concurrently can swamp a database. If you want to export a very large volume of data, you may be better off generating the INSERT statements into a text file, and then using a bulk data import tool provided by your database to do the database import.