

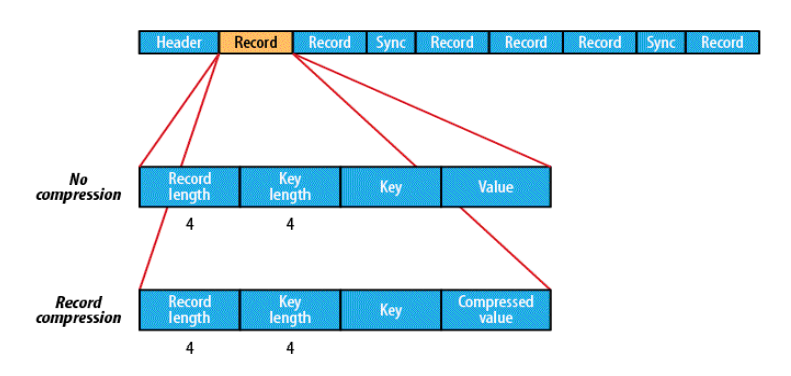
**Solution-**

**Input File Format-**

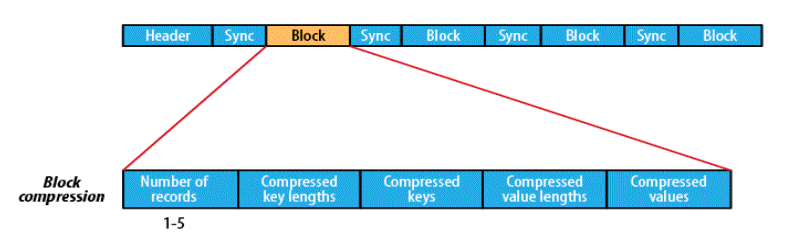
* A file format is a way in which information is stored or encoded in a computer file.
* In Hive it refers to how records are stored inside the file.
* Input Format reads key-value pairs from files.
* As we are dealing with structured data, each record has to be its own structure.
* How records are encoded in a file defines a file format.
* These file formats mainly vary between data encoding, compression rate, usage of space and disk I/O.
* Hive does not verify whether the data that you are loading matches the schema for the table or not.
* However, it verifies if the file format matches the table definition or not.
* There are various file formats supported by Hive.

1. **Sequence Files in Hive-**

* Traditionally, Hadoop saves its data internally in flat sequence files, which is a binary storage format for key value pairs.

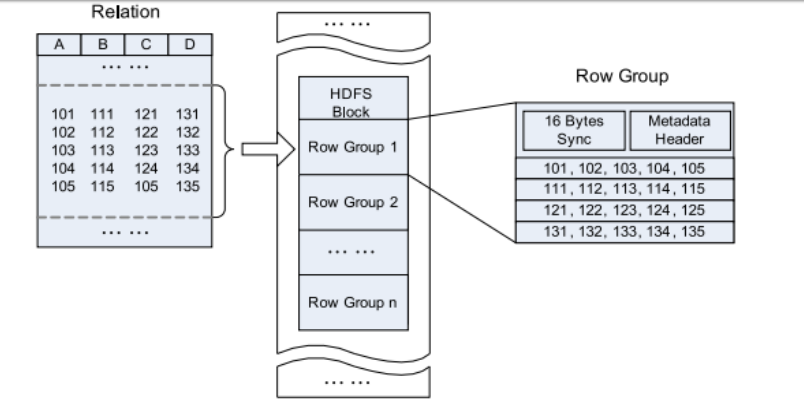
****

* It has the benefit of being more compact than text and fits well the map-reduce output format.
* Sequence files can be compressed on value, or block level, to improve its IO profile further.
* Unfortunately, sequence files are not an optimal solution for Hive since it saves a complete row as a single binary value.
* Consequently, Hive has to read a full row and decompress it even if only one column is being requested.

****

1. **RC File Format-**

* The RCFile splits data horizontally into row groups.
* For example, rows 1 to 100 are stored in one group and rows 101 to 200 in the next and so on. One or several groups are stored in a HDFS file. The RCFile saves the row group data in a columnar format.
* So instead of storing row one then row two, it stores column one across all rows then column two across all rows and so on.
* The benefit of this data organization is that Hadoop’s parallelism still applies since the row groups in different files are distributed redundantly across the cluster and process able at the same time.
* Subsequently, each processing node reads only the columns relevant to a query from a file and skips irrelevant ones.
* Additionally, compression on a column base is more efficient.
* It can take advantage of similarity of the data in a column.



1. **ORC File Format-**

* ORCFile was introduced in Hive 0.11 and offered excellent compression, delivered through a number of techniques including run-length encoding, dictionary encoding for strings and bitmap encoding.
* This focus on efficiency leads to some impressive compression ratios.

INLINE INDEX AND PREDICATE PUSHDOWN

* SQL queries will generally have some number of WHERE conditions which can be used to easily eliminate rows from consideration.
* In older versions of Hive, rows are read out of the storage layer before being later eliminated by SQL processing.
* There’s a lot of wasteful overhead and Hive 12 optimizes this by allowing predicates to be pushed down and evaluated in the storage layer itself.
* It’s controlled by the setting hive.optimize.ppd=true.
* This requires a reader that is smart enough to understand the predicates. Fortunately ORC has had the corresponding improvements to allow predicates to be pushed into it, and takes advantages of its inline indexes to deliver performance benefits.

For example if you have a SQL query like:

SELECT COUNT(\*) FROM CUSTOMER WHERE CUSTOMER.state = ‘CA’;

* The ORCFile reader will now only return rows that actually match the WHERE predicates and skip customers residing in any other state.
* ORC’s Predicate Pushdown will consult the Inline Indexes to try to identify when entire blocks can be skipped all at once.
* If a column is sorted, relevant records will get confined to one area on disk and the other pieces will be skipped very quickly.
* You can force Hive to sort on a column by using the SORT BY keyword when creating the table and setting hive.enforce.sorting to true before inserting into the table.
* ORC stores collections of rows in one file and within the collection the row data is stored in a columnar format.
* This allows parallel processing of row collections across a cluster. Each file with the columnar layout is optimised for compression and skipping of data/columns to reduce read and decompression load.
* ORC goes beyond RCFile and uses specific encoders for different column data types to improve compression further, e.g. variable length compression on integers.
* ORC introduces a lightweight indexing that enables skipping of complete blocks of rows that do not match a query.
* It comes with basic statistics — min, max, sum, and count — on columns.
* Lastly, a larger block size of 256 MB by default optimizes for large sequential reads on HDFS for more throughput and fewer files to reduce load on the namenode.

**Comparison of all the file formats-**

