**Qn 1: What are the uses of counters:**

Hadoop MapReduce Counter provides a way to measure the progress or the number of operations that occur within MapReduce programs. Basically, MapReduce framework provides a number of built-in counters to measure basic I/O operations, such as FILE\_BYTES\_READ/WRITTEN and Map/Combine/Reduce input/output records.

Hadoop inbuilt counters:

1) Task Counters

- Map input records (MAP\_INPUT\_RECORDS):- The number of input records consumed by all the maps in the job. Incremented every time a record is read from a RecordReader and passed to the maps map ( ) method by the framework.

-Split raw bytes (SPLIT\_RAW\_BYTES):- The number of bytes of input-split objects read by maps. These objects represent the split metadata (that is, the offset and length within a file) rather than the split data itself, so the total size should be small.

2) Job Counters

- Launched map tasks (TOTAL\_LAUNCHED\_MAPS):-The number of map tasks that were launched. Includes tasks that were started speculatively

-Launched reduce tasks (TOTAL\_LAUNCHED\_REDUCES) The number of reduce tasks that were launched. Includes tasks that were started speculatively.

**Qn 2: MR Unit testing is based on**

With MRUnit, you can craft test input, push it through your mapper and/or reducer, and verify it’s output all in a JUnit test.  As do other JUnit tests, this allows you to debug your code using the JUnit test as a driver.  A map/reduce pair can be tested using MRUnit’s MapReduceDriver.  A combiner can be tested using MapReduceDriver as well.  A PipelineMapReduceDriver allows you to test a workflow of map/reduce jobs.  Currently, partitioners do not have a test driver under MRUnit.  MRUnit allows you to do TDD and write light-weight unit tests which accommodate Hadoop’s specific architecture and constructs.

**Qn 3: How testing is useful in industry**

Testing is the process of evaluating a system or its component(s). It is basically used to check whether the developed code meets the requirements or not.

A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Testing can be conducted by the developer which is called as the unit testing or as in many it companies the testing is done by separate section which does the intense testing on the developed code so as it meets the clients expectations.

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. But it depends on the product being developed and the expectations of the client based on the above criteria different SDLC are mentioned.

Different types of SDLC are

V-Shaped Model

Iterative Model.

Spiral Model.

Big Bang Model.

Agile Model.

**Qn 4: Mapreduce Task Counters, File system counters, Job Counter**

**Hadoop Built-In counters:**There are some built-in counters which exist per job. Below are built-in counter groups-

* **MapReduce Task Counters** - Collects task specific information (e.g., number of input records) during its execution time.
* **FileSystem Counters** - Collects information like number of bytes read or written by a task
* **FileInputFormat Counters** - Collects information of number of bytes read through FileInputFormat
* **FileOutputFormat Counters** - Collects information of number of bytes written through FileOutputFormat
* **Job Counters -** These counters are used by JobTracker. Statistics collected by them include e.g., number of task launched for a job.

**Qn 5 : Raw comparator VS Writable Comparator**

**Raw Comparator:**

This interface permits implementers to compare records read from a stream without deserializing them into objects, thereby avoiding any overhead of object creation.

For example, the comparator for IntWritables implements the raw compare() method by reading an integer from each of the byte arrays b1 and b2 and comparing them directly from the given start positions (s1 and s2) and lengths (l1 and l2).

**Writable Comparator:**

WritableComparator is a general-purpose implementation of RawComparator for

WritableComparable classes.

It provides two main functions.

First, it provides a default implementation of the raw compare() method that deserializes the objects to be compared from the stream and invokes the object compare() method.

Second, it acts as a factory for RawComparator instances (that Writable implementations have registered).

**Qn 6: Partitioner, Sort comparator, Group comparator**

**Group Comparator** – It decides which map output keys will be united (grouped) into one key, and of course all collections of values will be grouped too. Usually it takes a first key as the only one for summary collection.

**Partitioner** – It is used to decide the which key should go to which reducer, by default it uses the hash code of the object to decide the reducer but one can override the partitioner to send particular to particular reducer. This is mostly used in case of composite key, secondary sort

**Sort Comparator-** Used to define how map output keys are sorted,SortComparator decides how map output keys are sorted. If the property mapred.output.key.comparator.class is set, either explicitly or by calling setSortComparatorClass() on Job, then an instance of that class is used. (In the old API the equivalent method is setOutputKeyComparatorClass() on JobConf.)