**Q1. Explain about the different complex data types in pig**

There are 4 complex Data Types. They are

1.Atom

2.Tuple

3.Bag

4.Map

5.Relation

**Atom**

1. Any single value in Pig Latin, irrespective of their data or type is known as an Atom.
2. It is stored as bytearray by default and can be used as string or number like int, long, float, double, chararray, and byte array are the atomic values of Pig.
3. A piece of data or a simple atomic value is known as a field. Example − ‘suresh or ‘20’

**Tuple**

1. A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type.
2. A tuple is similar to a row in a table of RDBMS.
3. A tuple is represented by ‘()’.

Example − (suresh, 20)

**Bag**

1. A bag is an unordered set of tuples.
2. In other words, a collection of tuples (non-unique) is known as a bag.
3. Each tuple can have any number of fields (flexible schema).
4. A bag is represented by ‘{}’.
5. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example − {(suresh, 20), (arvind, 35)}

1. A bag can be a field in a relation; in that context, it is known as inner bag.

Example − {suresh, 20, {9998887776, [suresh@gmail.com,}](mailto:suresh@gmail.com,%7d)}

**Map**

1. A map (or data map) is a set of key-value pairs.
2. The key needs to be of type chararray and should be unique.
3. The value might be of any type. It is represented by ‘[]’ Example: [name#suresh, age#20]

**Relation**

1. A relation is an outer bag of tuples.
2. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

**Q2. How can you interact with the shell in Apache pig**

The shell in apache pig is called grunt shell. We can launch it by using simple command “pig” or “pig -x local”.

If we are interacting with linux terminal from pig’s grunt shell then use “sh” prefix followed by linux terminal commands in pig grunt shell.

For example we can list files in current directory from grunt shell as

grunt> sh ls -l

However, the latest version of pig as of now 0.16 supports a few commands even without using this prefix “sh”

**Q3. Explain how pig differs from Map reduce**

|  |  |
| --- | --- |
| **PIG** | **MAP REDUCE** |
| It can be used for Structured and semi structured data | It can be used for Structured ,semi structured and un structured data |
| It can be useful for effectively performing joins, querying etc | It can be used for only map reduce |
| Development time is less (code development) | Development time is high (code development) |
| Processing Time is high since it runs on top of map reduce | Processing Time is comparatively less |
| Useful for performing analysis |  |

**Q4. Explain how pig differs from sql**

Pig Latin, looks similar to a Declarative language like SQL. One would find one-to-one corresponding operators. For Ex - Filter for where, Generate for select, Group for group by etc. But in reality, Pig Latin is far different from SQL. Pig Latin is a data flow procedural language, meaning we supply all the steps of "How" a task is to be accomplished, whereas in SQL we specify "What" is to be accomplished.

Another difference is that, SQL is query based and complex queries in SQL are written inside out - ie; queries required to be performed earlier are written as sub-queries of the ones required to be performed later. This can be confusing in terms of understanding. In Pig Latin, the script conveys the data flow in program execution order.

Pig Latin executes on massive data on a parallel distributed framework, originally Hadoop Map Reduce / HDFS. This is a batch mode execution good for streaming data access. SQL/RDBMS is not scalable like Hadoop, but is very good for individual record read / write.

**Q5. Explain the scalar data types in pig**

The scalar Data Types in pig are

1.Int

2.Long

3.Float

4.Doublr

5.Chararray

6.Bytearray

**1.Int**

It is like integer datatype in java. Int are represented in interfaces by java.lang.Integer. They store a four-byte signed integer. Constant integers are expressed as integer numbers, for example, 42.

**2.long**

It is like Long datatype in java. Longs are represented in interfaces by java.lang.Long. They store an eight-byte signed integer. Constant longs are expressed as integer numbers with an L appended, for example, 5000000000L.

**3.float**

It is like integer datatype in java A floating-point number. Floats are represented in interfaces by java.lang.Float and use four bytes to store their value. Since this is a floating-point number, in some calculations it will lose precision. For calculations that require no loss of precision, you should use an int or long instead. Constant floats are expressed as a floating-point number with an f appended. Floating-point numbers can be expressed in simple format, 3.14f, or in exponent format, 6.022e23f.

**4.double**

A double-precision floating-point number. Doubles are represented in interfaces by java.lang.Double and use eight bytes to store their value. You can find the range of values representable by Java’s Double type. Note that because this is a floating-point number, in some calculations it will lose precision. For calculations that require no loss of precision, you should use an int or long instead. Constant doubles are expressed as a floating-point number in either simple format, 2.71828, or in exponent format, 6.626e-34.

**5.chararray**

A string or character array. Chararrays are represented in interfaces by java.lang.String. Constant chararrays are expressed as string literals with single quotes, for example, 'fred'. In addition to standard alphanumeric and symbolic characters, you can express certain characters in chararrays by using backslash codes, such as \t for Tab and \n for Return. Unicode characters can be expressed as \u followed by their four-digit hexadecimal Unicode value. For example, the value for Ctrl-A is expressed as \u0001.

**6.bytearray**

A blob or array of bytes. Bytearrays are represented in interfaces by a Java class DataByteArraythat wraps a Java byte[]. There is no way to specify a constant bytearray.