1.       Explain about the different complex data types in pig

**Complex Types**: Pig supports three complex data types. They are listed below:

* **Tuple**: An ordered set of fields. Tuple is represented by braces. Example: (1,2)
* **Bag** : A set of tuples is called a bag. Bag is represented by curly braces. Example: {(1,2),(3,4)}
* **Map** : A set of key value pairs. Map is represented in a square brackets. Example: [key#value] . The # is used to separate key and value.

Pig allows nesting of complex data structures. Example: You can nest a tuple inside a tuple, bag and a Map.

2.       How can you interact with the shell in Apache pig.

The shell in apache pig is called grunt shell, we  can launch grunt shell by using simple command “pig” or “pig -x local”. And to interact with the linux terminal we can use “sh” command from the grunt shell.

3. Explain how pig differs from Map reduce

* Pig is application that runs on top of MapReduce and abstracts Java MapReduce jobs away from developers.
* Pig Latin uses a lot fewer lines of code than the Java MapReduce script.
* The Pig Latin script is easier to read for someone without a Java background.
* MapReduce jobs can written in Pig Latin.
* Java is a great and powerful language, but it has a higher learning curve than Pig Latin. Therefore, using a higher-level language, like Pig Latin, enables many more developers to write MapReduce jobs

4. Explain how pig differs from sql

SQL is declarative and **Pig**is procedural to a large extent.

SQL is a general purpose database language that has extensively been used for both transactional and analytical queries.

Pig can process both structured and unstructured data. It is a data flow language and environment for exploring very large datasets.

5. Explain the scalar data types in pig

**Scalar Types**

Pig’s scalar types are simple types that appear in most programming languages. With the exception of bytearray, they are all represented in Pig interfaces by java.lang classes, making them easy to work with in UDFs:

*int*

An integer. Ints 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.

*long*

A long integer. 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.

*float*

A floating-point number. Floats are represented in interfaces by java.lang.Float and use four bytes to store their value. Floating-point numbers can be expressed in simple format, 3.14f, or in exponent format, 6.022e23f.

*double*

A double-precision floating-point number. Doubles are represented in interfaces byjava.lang.Double and use eight bytes to store their valueConstant doubles are expressed as a floating-point number in either simple format, 2.71828, or in exponent format, 6.626e-34.

*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.

*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.