AVRO - SCHEMAS

Avro, being a schema-based serialization utility, accepts schemas as input. In spite of various schemas being available, Avro follows its own standards of defining schemas. These schemas describe the following details −

- type of file *recordbydefault*
- location of record
- name of the record
- fields in the record with their corresponding data types

Using these schemas, you can store serialized values in binary format using less space. These values are stored without any metadata.

Creating Avro Schemas

The Avro schema is created in JavaScript Object Notation *JSON* document format, which is a lightweight text-based data interchange format. It is created in one of the following ways −

- A JSON string
- A JSON object
- A JSON array

Example − The given schema defines a *recordtype* document within "Tutorialspoint" namespace. The name of document is "Employee" which contains two "Fields" → Name and Age.

```
{
   "type" : "record",
   "namespace" : "Tutorialspoint",
   "name" : "Employee",
   "fields" : [
      { "name" : "Name" , "type" : "string" },
      { "name" : "Age" , "type" : "int" }
   ]
}
```

We observed that schema contains four attributes, they are briefly described below −

- **type** − Describes document type, in this case a "record".
- **namespace** − Describes the name of the namespace in which the object resides.
- **name** − Describes the schema name.
- **fields** − This is an attribute array which contains the following −
    - **name** − Describes the name of field
    - **type** − Describes data type of field

Primitive Data Types of Avro

Avro schema is having primitive data types as well as complex data types. The following table describes the **primitive data types** of Avro −

Data type	Description
null	Null is a type having no value.

int	32-bit signed integer.
long	64-bit signed integer.
float	single precision $32 - bit$ IEEE 754 floating-point number.
double	double precision $64 - bit$ IEEE 754 floating-point number.
bytes	sequence of 8-bit unsigned bytes.
string	Unicode character sequence.

Complex Data Types of Avro

Along with primitive data types, Avro provides six complex data types namely Records, Enums, Arrays, Maps, Unions, and Fixed.

Record

As we know already by now, a record data type in Avro is a collection of multiple attributes. It supports the following attributes −

- **name**
- **namespace**
- **type**
- **fields**

Enum

An enumeration is a list of items in a collection, Avro enumeration supports the following attributes −

- **name** − The value of this field holds the name of the enumeration.
- **namespace** − The value of this field contains the string that qualifies the name of the Enumeration.
- **symbols** − The value of this field holds the enum's symbols as an array of names.

Example

Given below is the example of an enumeration.

```
{
   "type" : "enum",
   "name" : "Numbers", "namespace": "data", "symbols" : [ "ONE", "TWO", "THREE", "FOUR"
]
}
```

Arrays

This data type defines an array field having a single attribute items. This items attribute specifies the type of items in the array.

Example

```
{ " type " : " array ", " items " : " int " }
```

Maps

The map data type is an array of key-value pairs. The **values** attribute holds the data type of the content of map. Avro map values are implicitly taken as strings. The below example shows map from string to int.

Example

```
{"type" : "map", "values" : "int"}
```

Unions

A union datatype is used whenever the field has one or more datatypes. They are represented as JSON arrays. For example, if a field that could be either an int or null, then the union is represented as ["int", "null"].

Example

Given below is an example document using unions −

```
{
   "type" : "record",
   "namespace" : "tutorialspoint",
   "name" : "empdetails ",
   "fields" :
   [
      { "name" : "experience", "type": ["int", "null"] }, { "name" : "age", "type":
"int" }
   ]
}
```

Fixed

This data type is used to declare a fixed-sized field that can be used for storing binary data. It has field name and data as attributes. Name holds the name of the field, and size holds the size of the field.

Example

```
{ "type" : "fixed" , "name" : "bdata", "size" : 1048576}
```