Zebra Reference Guide

Table of contents

1. Zebra Types

Zebra supports simple types (int, long, float, double, string, bytes), complex types (record, collection, map) and Booleans.

Zebra Type	Description
int	signed 32-bit integer
long	signed 64-bit integer
float	32-bit floating point
double	64-bit floating point
string	character array (string) in Unicode UTF-8 format
bytes	byte array (blob)
record	An ordered set of fields. A field can be any Zebra type.
collection	A set of records.
map	A set of key/value pairs. The key is type string; the value can be any Zebra type.
bool	Boolean {0,1} false/true

Zebra type names are chosen to be as “technology neutral” as possible and are in resemblance to native types in modern programming languages.

Zebra	Pig	Avro	SQL
int	int	int	integer
long	long	long	long
float	float	float	float,real
double	double	double	double precision
string	chararray	string	varchar
bytes	bytearray	bytes	raw
record	tuple	record	hash
collection	bag	array	list

map	map	map	hasp
bool	boolean	boolean	bool

2. Store Schema

Use the Zebra store schema to write or store Zebra columns and to specify column types. The schema supports data type compatibility and conversion between Zebra/Pig, Zebra/MapReduce, and Zebra/Streaming. (*In a future release, the schema will also support type compatibility between Zebra/Pig-SQL and will guide the underlying serialization formats provided by Avro for projection, filtering, and so on.*)

The basic format for the store schema is shown here. The type name is optional; if not specified, the column defaults to type bytes.

```
column_name[:type_name] [, column_name[:type_name] ... ]
```

2.1. Schemas for Simple Data Types

Simple data types include int, long, float, double, string, and bytes. The following syntax also applies to Booleans.

2.1.1. Syntax

field_alias[:type] [, field_alias[:type] …]

2.1.2. Terms

field_alias	The name assigned to the field column.
:type	(Optional) The simple data type assigned to the field. The alias and type are separated by a colon (:). If the type is omitted, the field defaults to type bytes.
,	Multiple fields are separated by commas.

2.1.3. Examples

In this example the schema specifies names and types for 3 columns.

```
ZebraSchema.createZebraSchema(jobconf, “s1:string, f1:float, i1:int”);
```

In this example the schema specifies names for 3 columns; all 3 columns default to type

bytes.

```
ZebraSchema.createZebraSchema(jobconf, “f1, f2, f3”);
```

2.2. Schemas for Records

A record is an ordered set of fields. A field can be any Zebra type.

2.2.1. Syntax

record_alias:record (field_alias[:type]) [,(field_alias[:type]) …])

2.2.2. Terms

record_alias	The name assigned to the record column.
:record	The record designator.
( )	The record notation, a set of parentheses.
field_alias	The name assigned to the field.
:type	(Optional) The type assigned to a field (can be any Zebra type).
,	Multiple fields are separated by commas.

2.2.3. Examples

In this example the schema specifies a record with two fields.

```
ZebraSchema.createZebraSchema(jobconf, “r1:record(f1:int,f2:long)”);
```

In this example the schema specifies a record with two fields. Note that f2 will default to type bytes.

```
ZebraSchema.createZebraSchema(jobconf, “r1:record(r2:record(f1:int,f2)”);
```

2.3. Schemas for Collections

A collection is a set of records.

2.3.1. Syntax

collection_alias:collection (record)

2.3.2. Terms

collection_alias	The name assigned to the collection column.
:collection	The collection designator.
( )	The collection notation, a set of parentheses.
record	A record, specified one of two ways: • Explicit (see Record) c1:collection(r1:record(f1:int,f2:long))”); • Implicit c1:collection(f1:int,f2:long)”);

2.3.3. Examples

In this example the schema specifies a collection with a record consisting of two fields (explicit record notation).

```
ZebraSchema.createZebraSchema(jobconf,
“c1:collection(r1:record(f1:int,f2:long))”);
```

In this example the schema specifies a collection with a record consisting of two fields (implicit record notation).

```
ZebraSchema.createZebraSchema(jobconf, “c1:collection(f1:int,f2:long)”);
```

2.4. Schemas for Maps

A map is a set of key/value pairs.

2.4.1. Syntax

map_alias:map (type)

2.4.2. Terms

map_alias	The name assigned to the map column.
:map	The map designator.
( )	The map notation, a set of parentheses.
type	The type assigned to the map’s value (can be any Zebra type). Note that the map’s key is always type string and is not specified.

2.4.3. Examples

In this example the schema specifies a map with value of type string.

```
ZebraSchema.createZebraSchema(jobconf, “m1:map(string)”);
```

In this example the schema specifies a map with value of type map (with a value of type int).

```
ZebraSchema.createZebraSchema(jobconf, “m2:map(map(int))”);
```

3. Storage Specification

Use the Zebra storage specification to define Zebra column groups. The storage specification, when combined with a STORE statement, describes the physical structure of a Zebra table. Suppose we have the following statement:

```
STORE A INTO '$PATH/mytable' USING
org.apache.hadoop.zebra.pig.TableStorer('[a1, a2] AS cg1; [a3, a4, a5] AS
cg2');
```

The statement describes a table that has two column groups; the first column group has two columns, the second column group has three columns. The statement can be interpreted as follows:

- $PATH/mytable - the table, a file path to a directory named mytable
- $PATH/mytable/cg1 - the first column group, a subdirectory named cg1 under directory mytable
- $PATH/mytable/cg1/part00001 - a file consisting, conceptually, of columns a1 and a2
- $PATH/mytable/cg2 - the second column group, a subdirectory named cg2 under directory mytable
- $PATH/mytable/cg2/part00001 - a file consisting, conceptually, of columns a3, a4, and a5

3.1. Specification

The basic format for the Zebra storage specification is shown here. For this specification, note that the straight brackets **[ ]** designate a column group and the curly brackets { } designate an optional syntax component.

3.1.1. Syntax

[column_name {, column_name ...} **]** {AS column_group_name} {COMPRESS BY compressor_name} {SERIALIZE BY serializer_name}
{; **[**column_name {, column_name ...} **]** {AS column_group_name} {COMPRESS BY

compressor_name} {SERIALIZE BY serializer_name} ... }

3.1.2. Terms

[ ]	Brackets designate a column group. Multiple column groups are separated by semi-colons.
AS column_group_name	Optional. The name of the column group. Column group names are unique within one table and are case sensitive: c1 and C1 are different. Column group names are used as the physical column group directory path names. If specified, the AS clause must immediately follow the column group [ ]. If not specified, Zebra will assign unique default names for the table: CG0, CG1, CG2 ... (If CGx is specified by the programmer, then it cannot be used by Zebra.)
COMPRESS BY compressor_name	Optional. Valid values for compressor_name include gz (default) and lzo. If not specified, gz is used.
SERIALIZE BY serializer_name	Optional. Valid values for serializer_name include pig (default). (*In a future release, Avro will be available.*) If not specified, pig is used.
column_name	The name of one or more columns that form the column group.

3.1.3. Examples

In this example, one column group is specified; the two statements are equivalent.

```
STORE A INTO '$PATH' USING org.apache.hadoop.zebra.pig.TableStorer('[c1]');

STORE A INTO '$PATH' USING org.apache.hadoop.zebra.pig.TableStorer('[c1] AS
CG0 COMPRESS BY gz SERIALIZE BY pig;');
```

In this example, two column groups are specified. The first column group, C1, has two columns. The second column group, C2, has three columns.

```
STORE A INTO '$PATH' USING org.apache.hadoop.zebra.pig.TableStorer('[a1,
a2] AS C1; [a3, a4, a5] AS C2');
```

4. Load Schema

Use the Zebra load schema to load or read table columns.

4.1. Schema

The basic format for the Zebra load (read) schema is shown here. The column name can be any valid Zebra type. If no columns are specified, the entire Zebra table is loaded.

```
column_name [, column_name ... ]
```

4.1.1. Terms

column_name	The column name. Multiple columns are separated by commas.

4.1.2. Example

Three Pig examples are shown here.

```
-- All columns are loaded
A = LOAD '$PATH/tbl1' USING org.apache.hadoop.pig.zebra.pig.TableLoader();

-- Two columns are projected
B = LOAD '$PATH/tbl2' USING
org.apache.hadoop.zebra.pig.TableLoader('c1,c2');

-- Three columns are projected: a simple field, a map, and a record
C = LOAD '$PATH/tbl3' USING
org.apache.hadoop.zebra.pig.TableLoader('c1,c2#{key1},col4.{f1}')
```