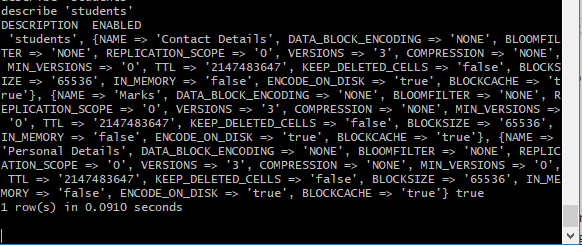
**DDL:**

A Data Definition Language (DDL) is used for defining data structures, particularly database schemas. Let us examine few of them.

**DESCRIBE:**

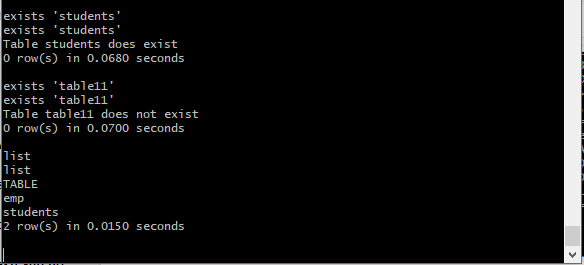
We have already seen how we can create a table using ‘create’ command. Now if we wish to check out the schema of the table we created, we can do so using ‘describe’ command. Let us use the ‘students’ table created earlier and describe it.



Here, we can see on the screen that the table describes the schema of the various column families of table ‘students’. We have specified three column families for the table namely ‘Contact Details’, ‘Personal Details’ and ‘Marks’. The command displays the comma separated key value pairs for each column family. The key value pairs contain the information about the specific column family like ‘REPLICATION\_SCOPE’, ‘VERSIONS’ etc. So basically the ‘describe’ command prints the schema of the table on the screen.

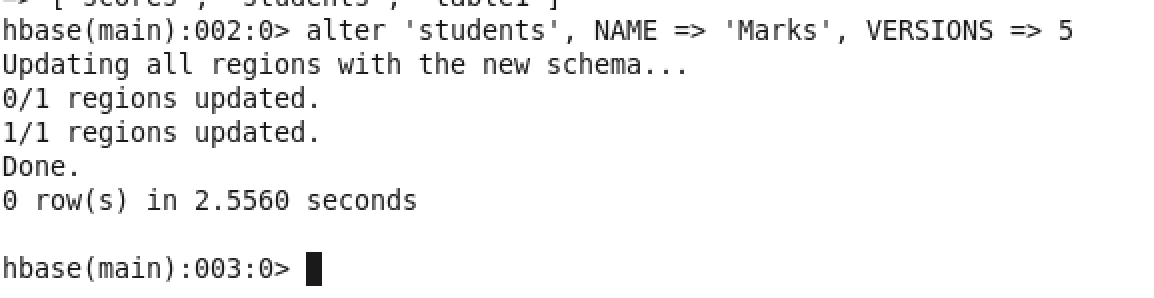
**EXISTS:**

The exists command shows whether or not the given table is present in the hbase storage or not. Here, we can see that because we have already created a table named ‘students’, ‘exists’ command on that table prints that the table is present. When we try to use this operator on a table that is not present, it will print that the table is not present.

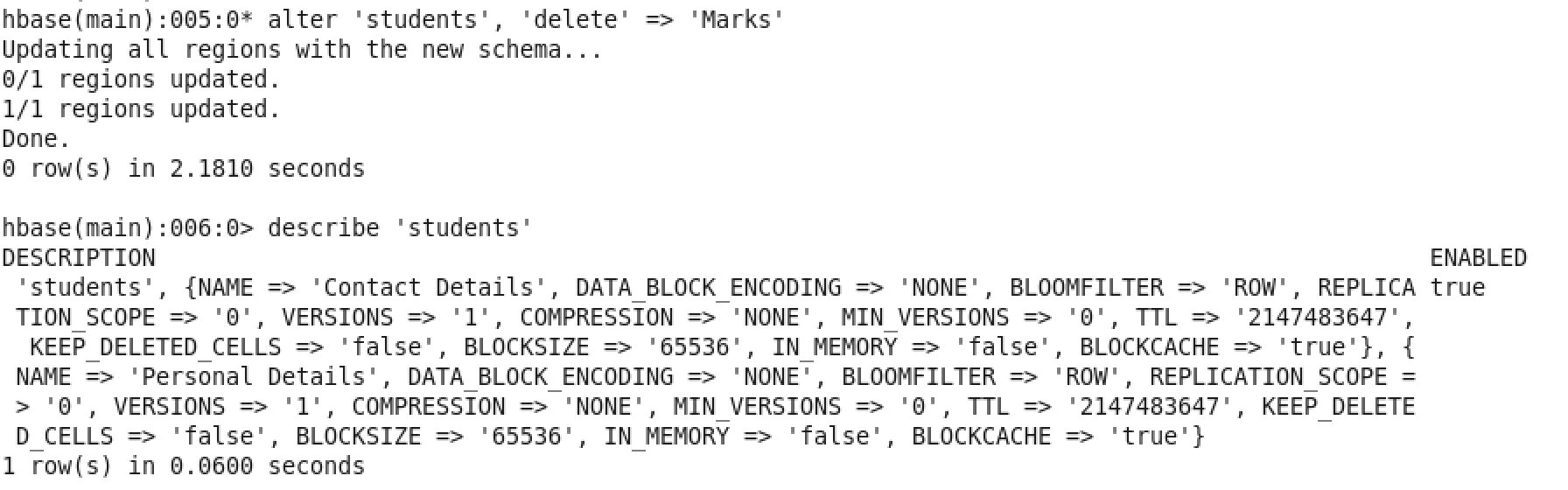


**ALTER:**

Alter command is basically used to modify the existing table. Like if we wanted to modify the number of versions of ‘Marks’ column family to 5, we could use this command.

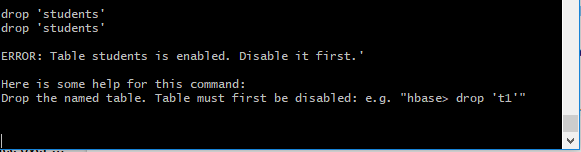


Other than VERSIONS, we can alter many other properties of the table. We can also delete the column family by applying delete method on it. Like as shown here:

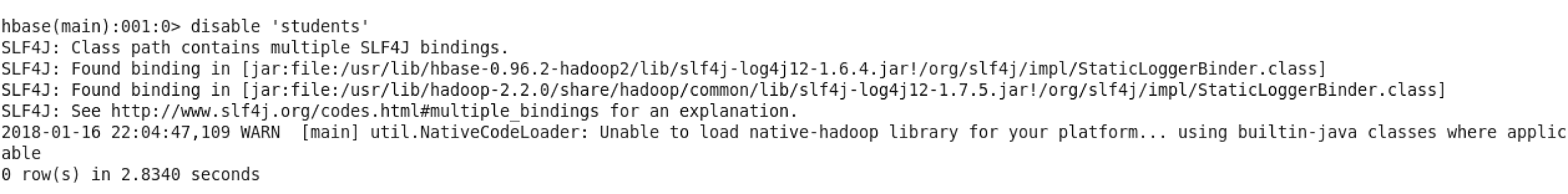


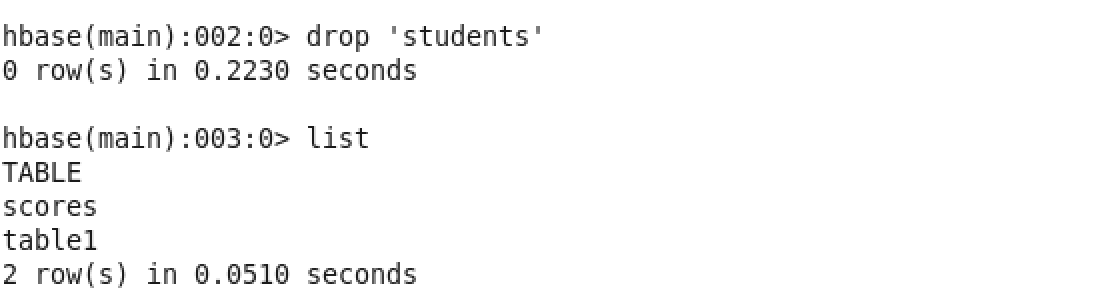
**DROP:**

Drop table is an operator that is used to delete the tables. But this operator cannot be applied directly to the table. Instead, the table is first disabled. And then it is dropped. Like this

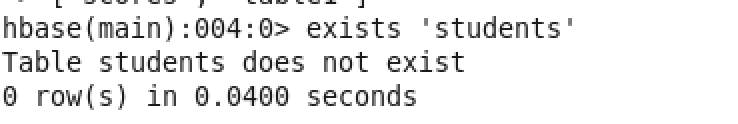
****

Now if we use disable first and then drop. The table is deleted.



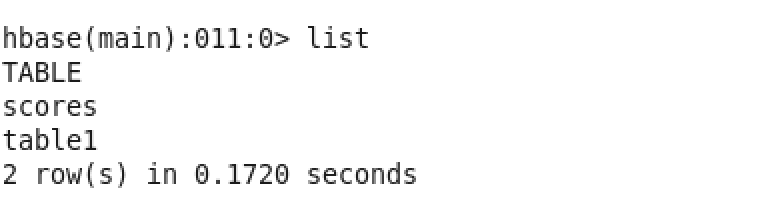


Now let us verify if the table is actually deleted or not using the exists command.

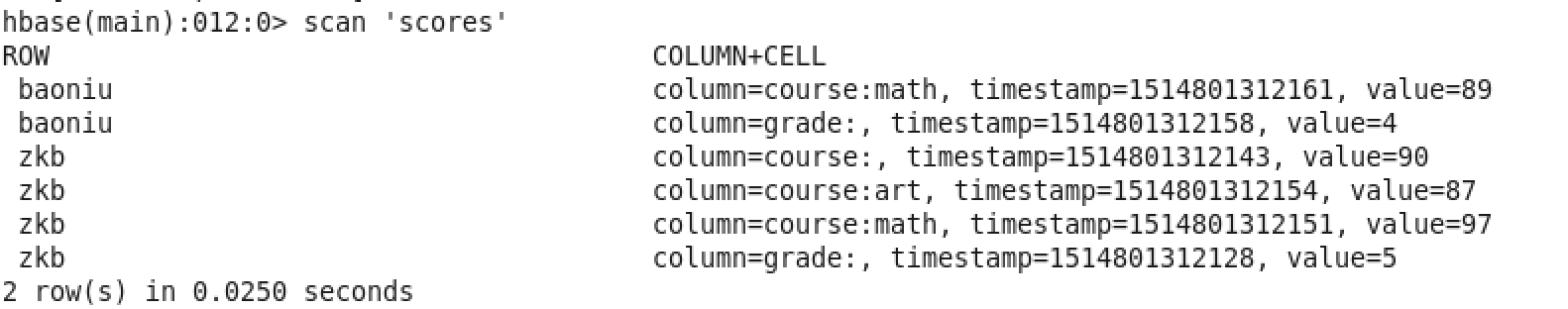
****

**Truncate:**

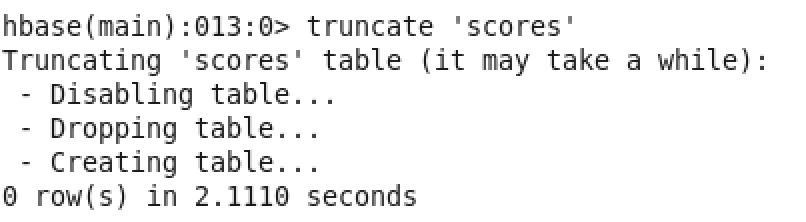
This command is basically used to remove all the data from the table. Note that we do not intend to delete the table, just the data that is stored in the table. Internally, this command disables the table, drops it and then again recreates it but for us, the end result is that the table’s data has been removed. Let us have a look. Let me fire a list command and check what all tables are present :



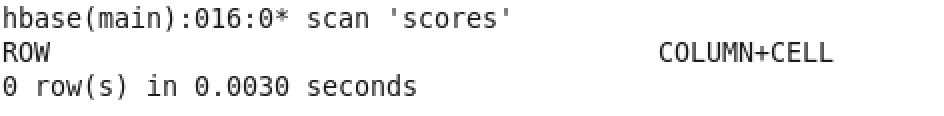
Here, we see that there are two tables scores and table1. Using the scan command let’s check if there are any records present in scores table. scan() command is used to display all the rows present in the table.



We see there are two rows. Let’s execute the truncate command on this table and check the outcome.



As discussed the truncate command first disabled the table, then dropped the table and recreated the same table for us. Now, ideally the table scores should be empty. Let’s find out by executing the scan command on this table:

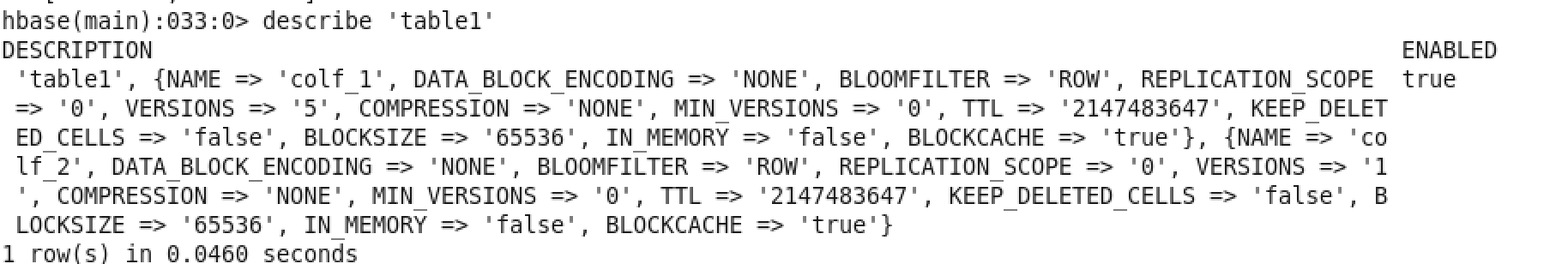


Indeed, now there are 0 rows present in the table.

**DML:**

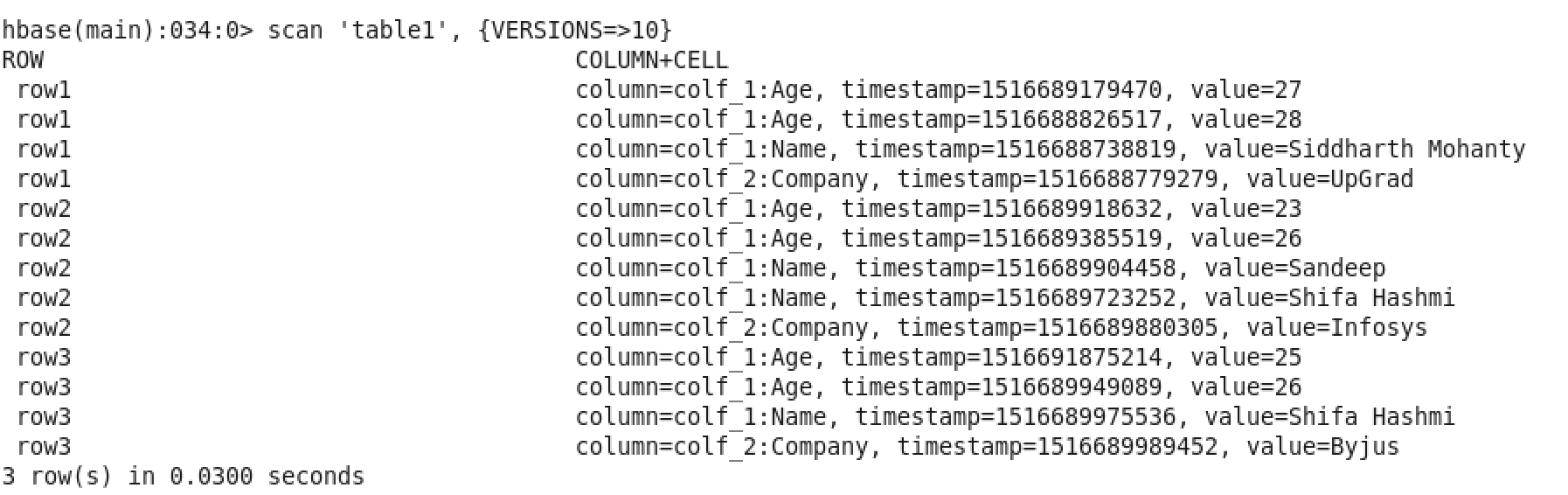
**A Data Manipulation Language (DML) is used for adding, deleting and modifying data in a database. Multiple put on the same cell. Demonstrating how hbase maintains multiple versions of data by timestamp.**

In the previous sessions, you learnt that HBase is capable of storing multiple versions of the data present in a single cell. Let's try and see how that is possible in HBase. By default, column families in an HBase table do not support storing multiple versions of data. We can confirm this by checking the table schema by firing a describe command in the console. We already have a table named 'table1' . Let's describe it and see its properties.



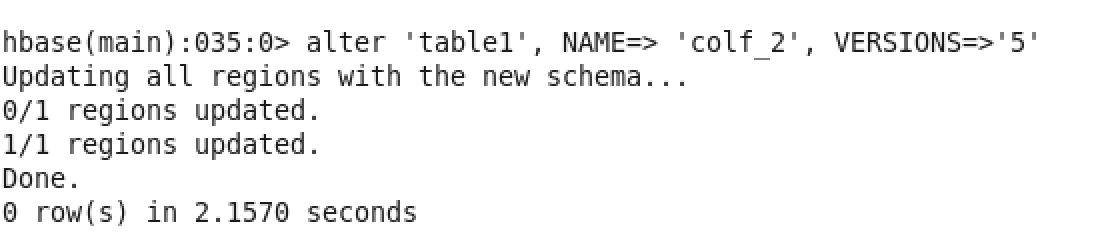
Here, we see that the version count for colf\_1 is 5 and version count for colf\_2 is 1 by default. The version count for colf\_1 is 5 because I had changed this manually before.

Previously we saw how scan command is used to retrieve all the rows present in the table. To see all the versions of a cell, we will execute a different variant of scan command i.e. scan 'table1',{VERSIONS => 10}. VERSIONS => 10 specifies the maximum number of versions which has to be displayed.

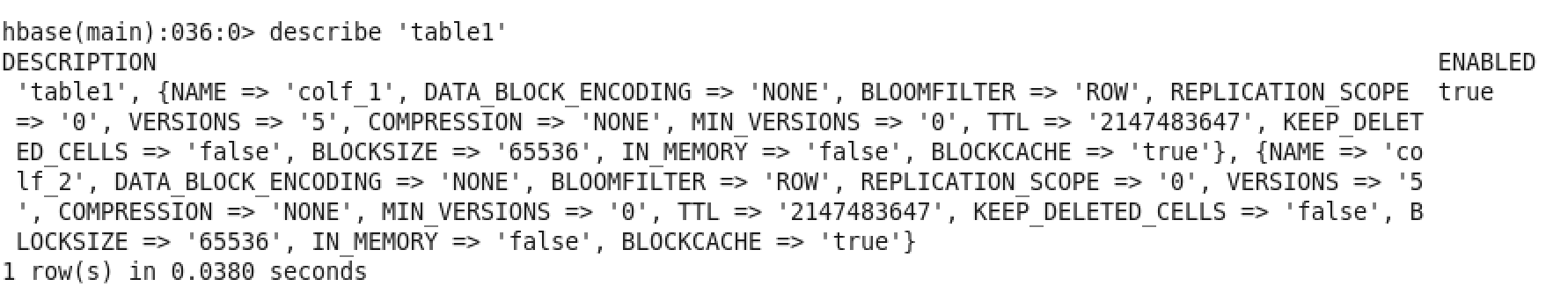


As checked before multiple versioning was allowed for colf\_1. Hence, for all the three rows we see multiple versions in age column. For row2, we even saw multiple versions in name column as well. As checked before multiple versioning was allowed for colf\_1.

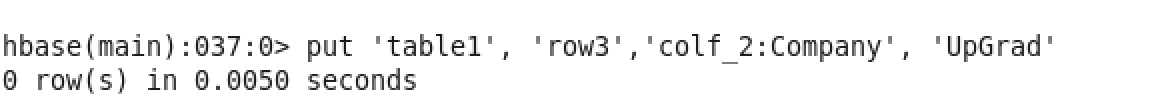
Now, let’s enable multiple version feature for colf\_2 as well. For that we need to fire the following alter command.



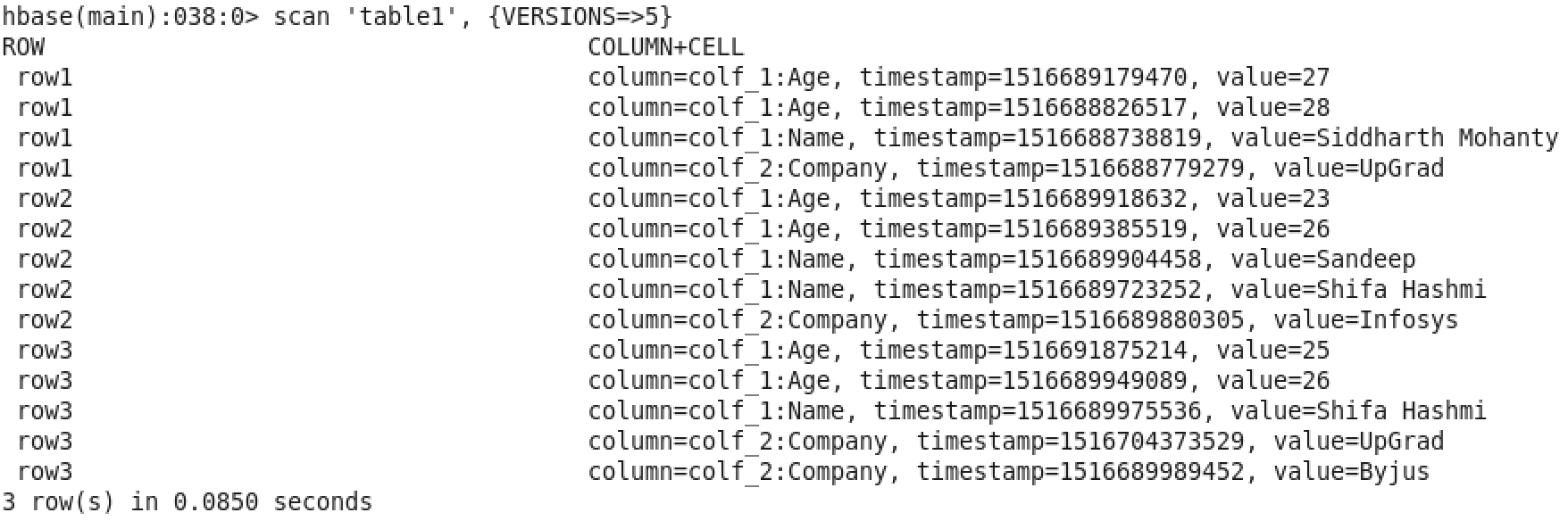
Let us verify if the version number has changed for colf\_2 or not by executing the describe command:



We see for colf\_2 the version count have changed to 5. Let’s try to update the company of row3 using put command.



Now let’s execute the scan ‘table1’, {VERSIONS=>5} command to check all the versions of the data present in the table.

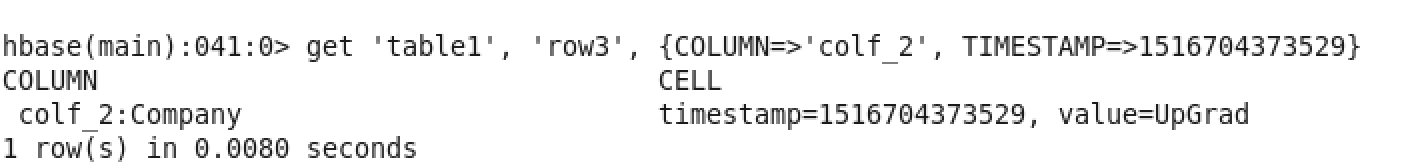


Indeed we can see for row3 and column company that there are two values UpGrad and Byjus.

**GET:**

* **Get data based on timestamp**

In HBase, if you fire a scan() command then only latest records are displayed. Past versions can be retrieved based on timestamp using the command get ‘table1’ which is the table name, ‘row3’ is the rowkey and in curly braces additional parameters such as column family name and the exact timestamp is specified.

****

* **Get data based on a filter condition**

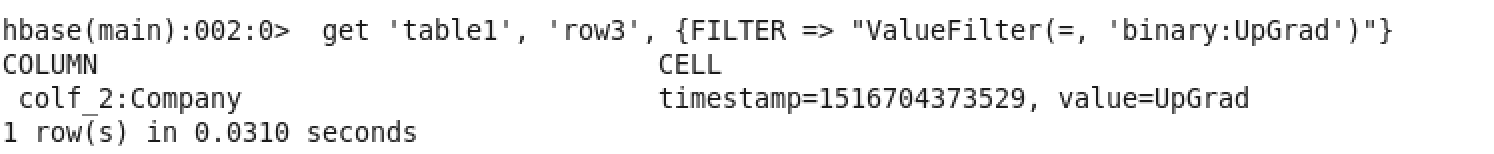
In HBase, fetching data based on a filtering condition is achieved by using Filters. In HBase, filters are like java methods which take two input parameters that are, a logical operator and a comparator. The logical operator specifies the type of the test i.e. equals, less than, etc. The type of filter and comparator together specify the left and right-hand side of the logical expression. Some commonly used filter functions are :

1. ValueFilter
2. QualifierFilter
3. FamilyFilter

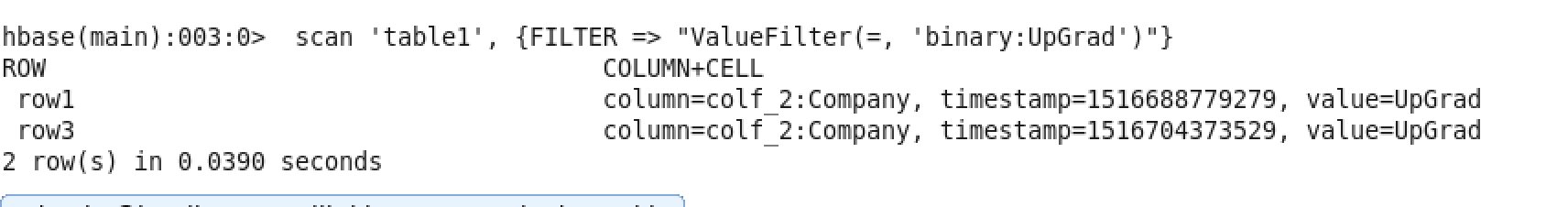
Let’s now check how a ValueFilter works.

A ValueFilter takes a comparison operator and comparator as the parameter. It compares each value with the comparator using the comparison operator. If the check is true then result is displayed in the console. To test this, let us fire this command

get 'table1', 'row3', {FILTER => "ValueFilter(=, 'binary:UpGrad')"}



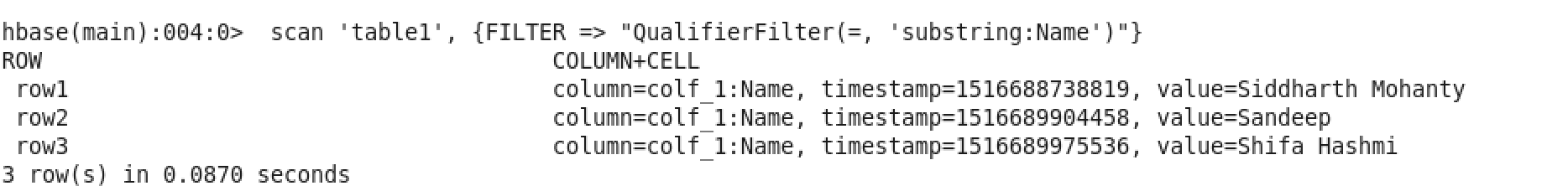
Let’s fire a scan command using ValueFilter



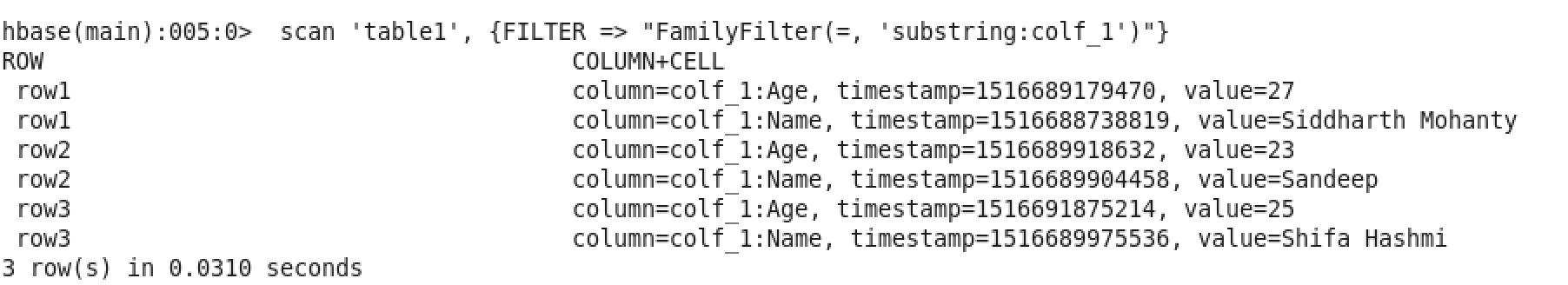
We can see that the company UpGrad exists for both row1 and row3.

Like ValueFilter, QualifierFilter also takes two parameters they are comparison operator and comparator. Each qualifier name is compared with the comparator using the compare operator and if the comparison is true, it returns the key-values in that column.

This column qualifier can be used to fetch an entire column’s data. Let’s see how we can fetch all the data present in column “Name”. We can do this by firing scan command with the QualifierFilter. The qualifier filter takes ‘=’ operator as the operator and “substring:Name” as comparator where “Name” is the qualifier name.



Similarly, familyfilter is used to fetch key-values for a specified column family. Let’s check that as well.



**COUNT:**

Count command is used to count the number of rows present in the table. It can be executed using count ‘table1’ as shown here:

