***Practical 6: Joins, Sorting, Subqueries using HiveQL***

**Steps involved how to perfume Joins, Sorting, Subqueries using HiveQL below in the following order-**

**JOINS**

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database.

There are different types of joins given as follows:

* JOIN
* LEFT OUTER JOIN
* RIGHT OUTER JOIN
* FULL OUTER JOIN
* **JOIN**

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

* **LEFT OUTER JOIN**

The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table. A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

* **RIGHT OUTER JOIN**

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate

* **FULL OUTER JOIN**

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

**SUB QUERIES:**

A Query present within a Query is known as a sub query. The main query will depend on the values returned by the subqueries.

Subqueries can be classified into two types

* + Subqueries in FROM clause
  + Subqueries in WHERE clause

**When to use:**

* + To get a particular value combined from two column values from different tables
  + Dependency of one table values on other tables
  + Comparative checking of one column values from other tables

**SORTING**

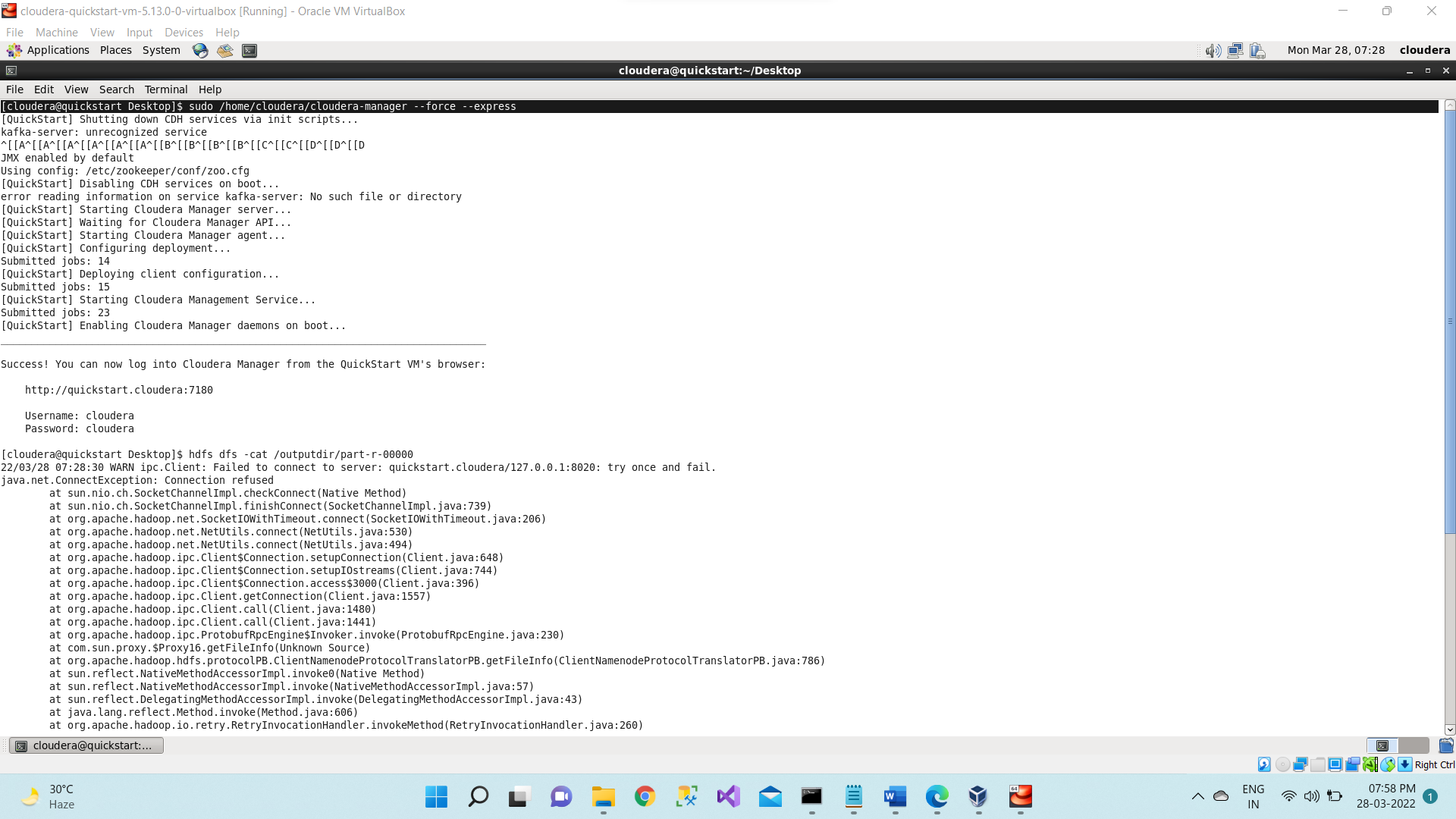
The SORT BY syntax is similar to the syntax of ORDER BY in SQL language.

Hive supports SORT BY which sorts the data per reducer. The difference between "order by” and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, "sort by” may give partially ordered final results.

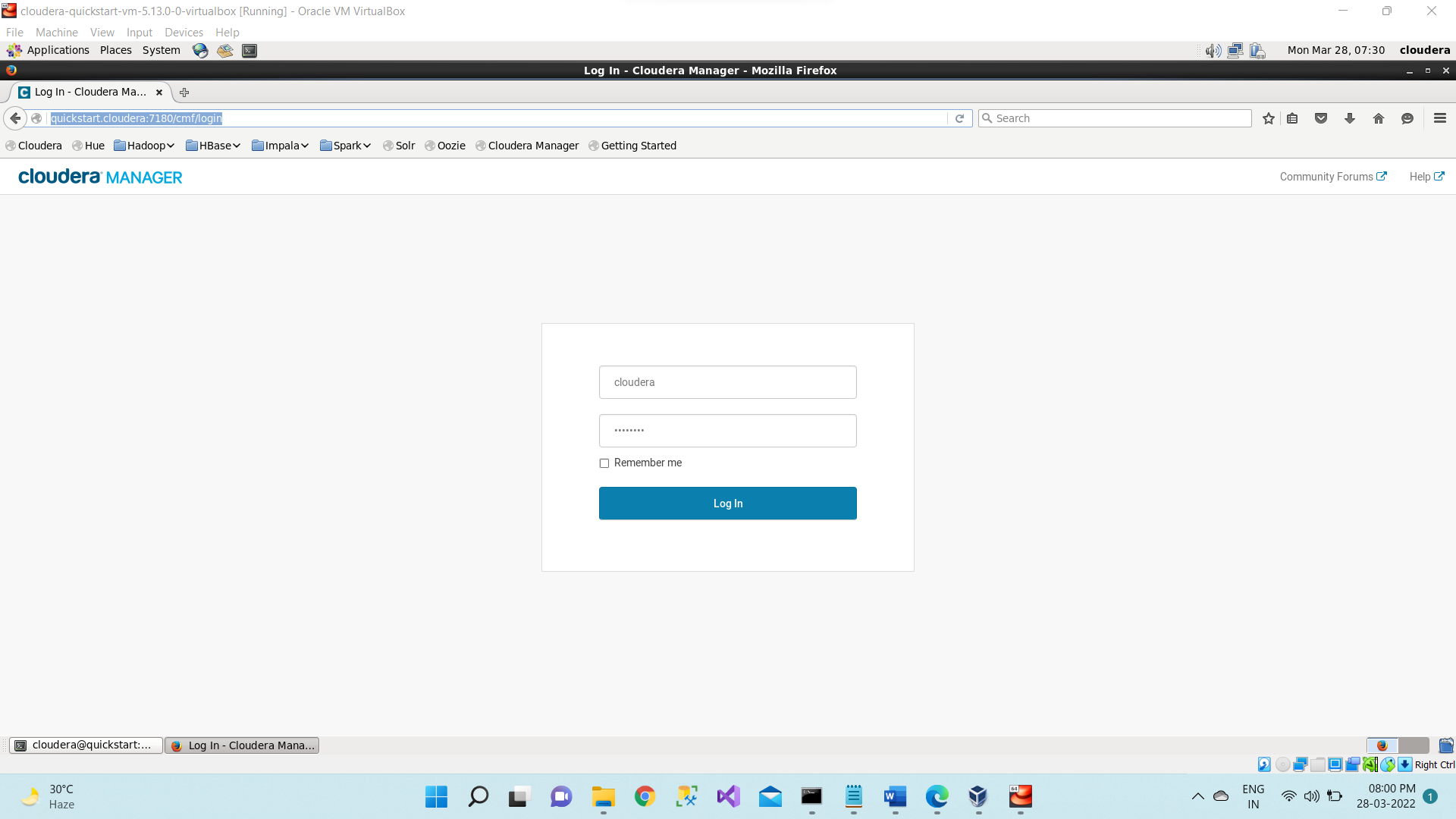
Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order

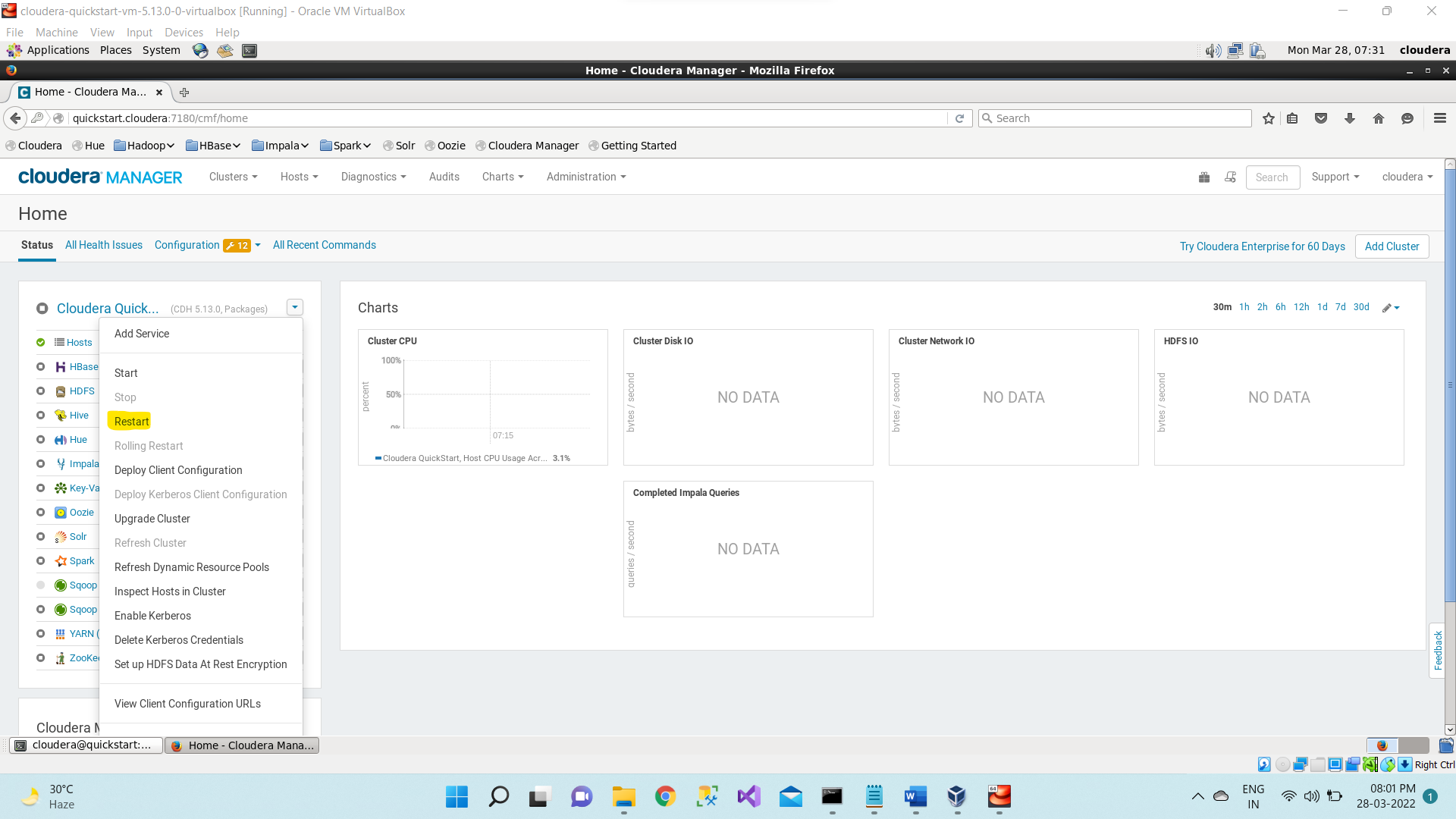
Before the perform the practical first perform 3 steps of the given following-

1. **sudo /home/cloudera/cloudera-manager --force –express**

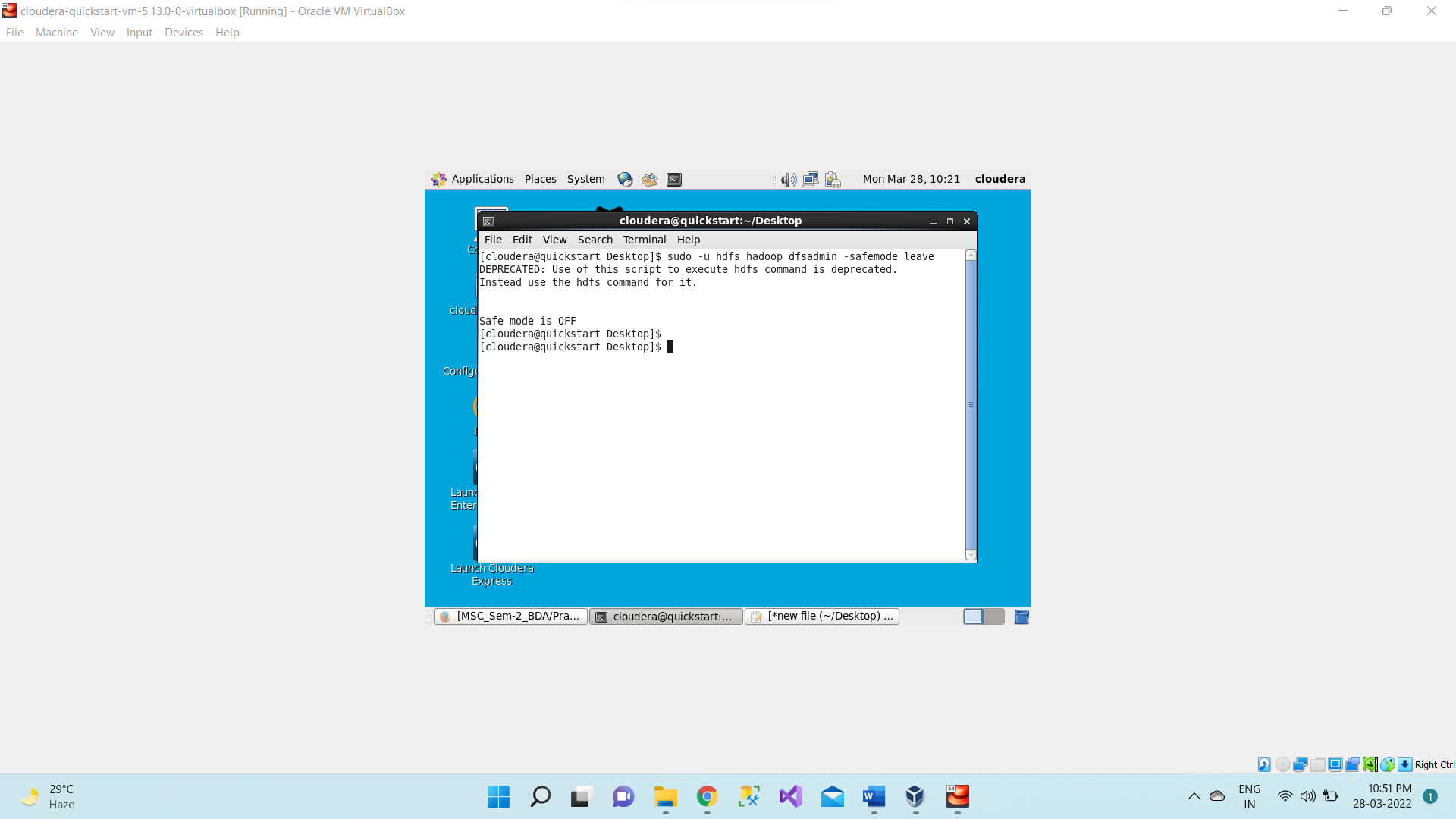


1. Start all Services.



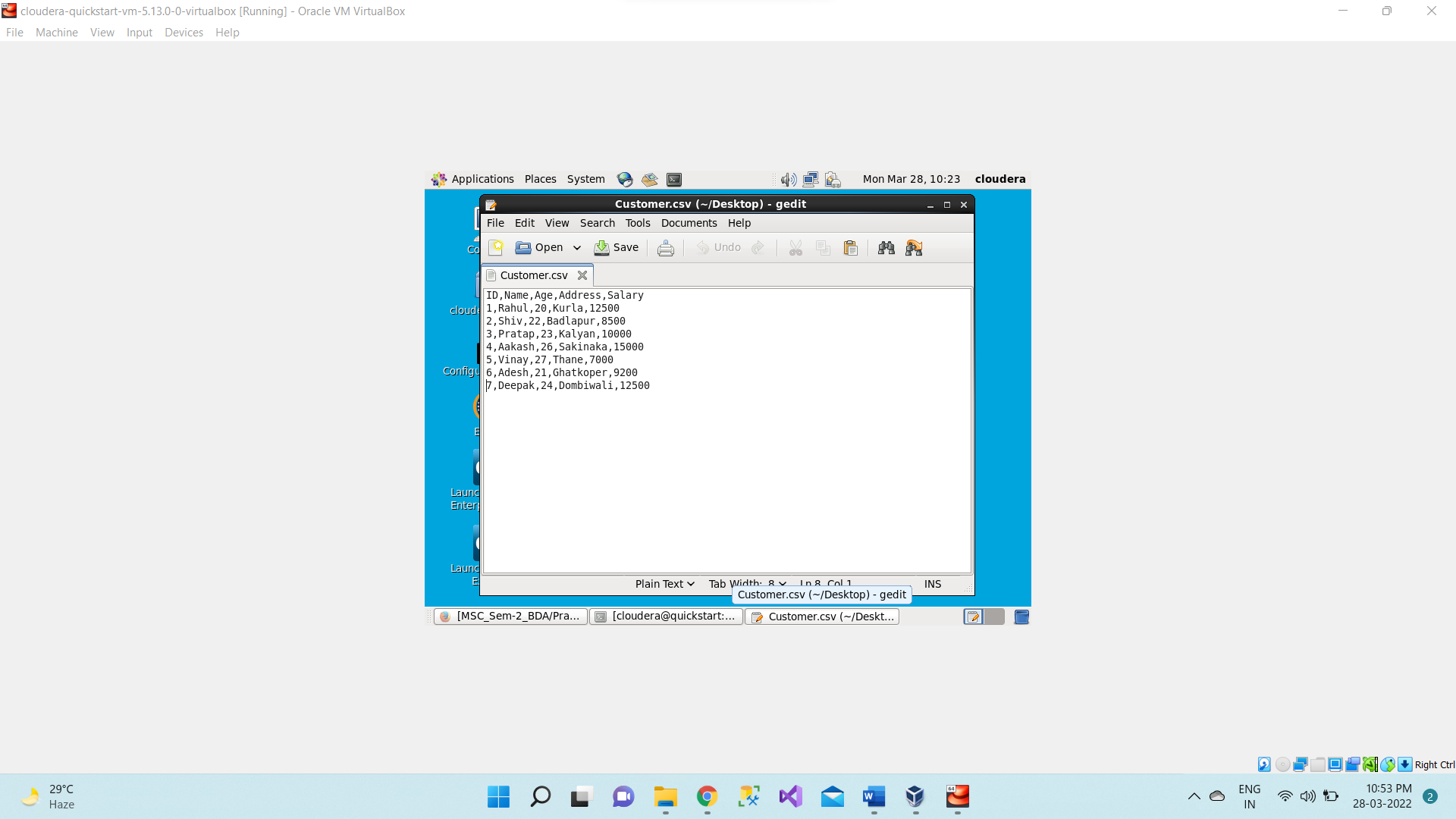


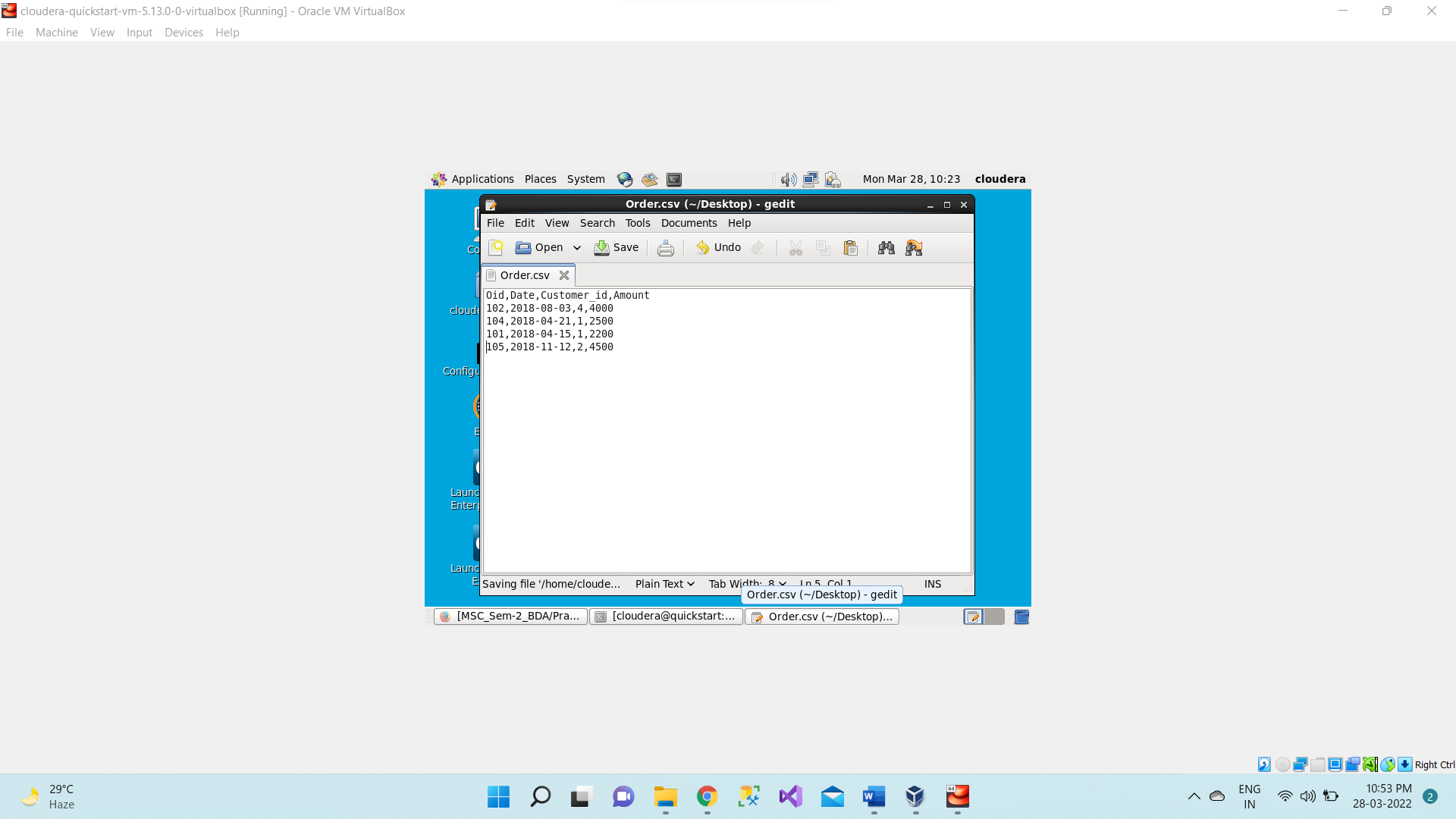
1. **sudo -u hdfs hadoop dfsadmin -safemode leave**



First, we will create the **Customer.csv** and **Order.csv** file.

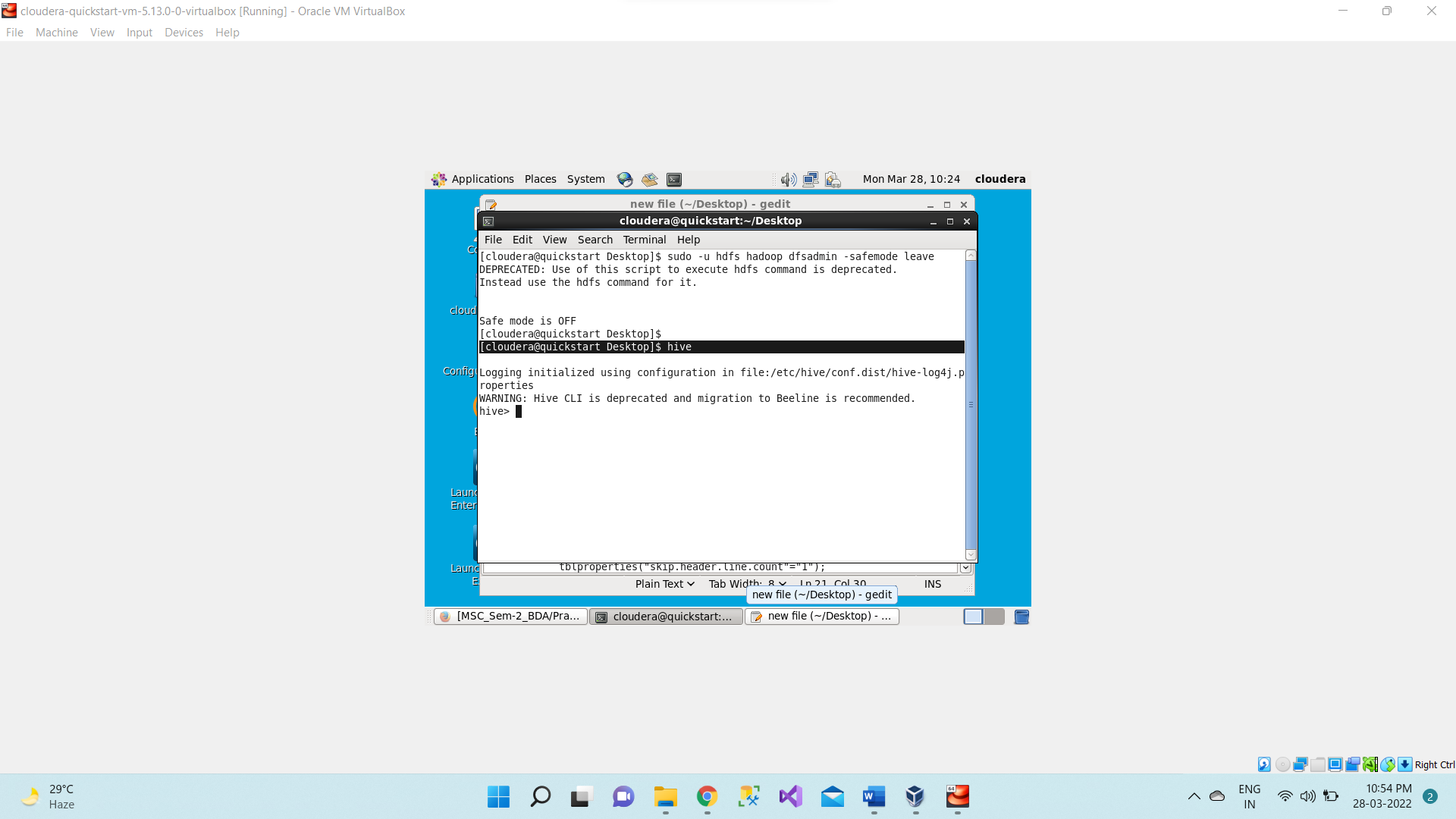
|  |  |
| --- | --- |
| **Customer.csv** | **Orders.csv** |
| **ID,Name,Age,Address,Salary**  1,Rahul,20,Kurla,12500  2,Shiv,22,Badlapur,8500  3,Pratap,23,Kalyan,10000  4,Aakash,26,Sakinaka,15000  5,Vinay,27,Thane,7000  6,Adesh,21,Ghatkoper,9200  7,Deepak,24,Dombiwali,12500 | **Oid,Date,Customer\_id,Amount**  102,2018-08-03,4,4000  104,2018-04-21,1,2500  101,2018-04-15,1,2200  105,2018-11-12,2,4500 |

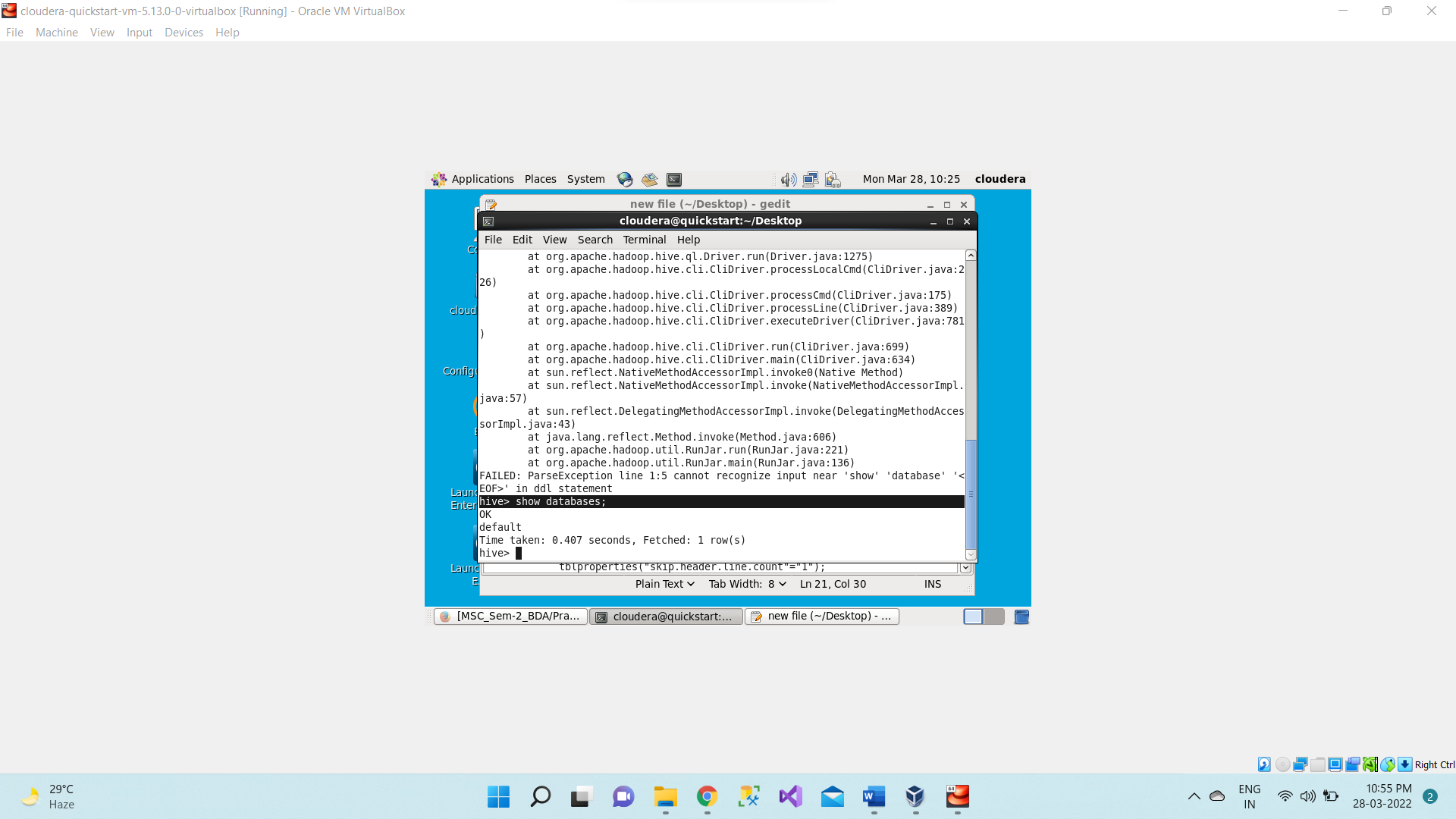




Open the terminal, now we use **hive** command to enter the **hive shell prompt** and in hive shell we could execute all of the hive commands.

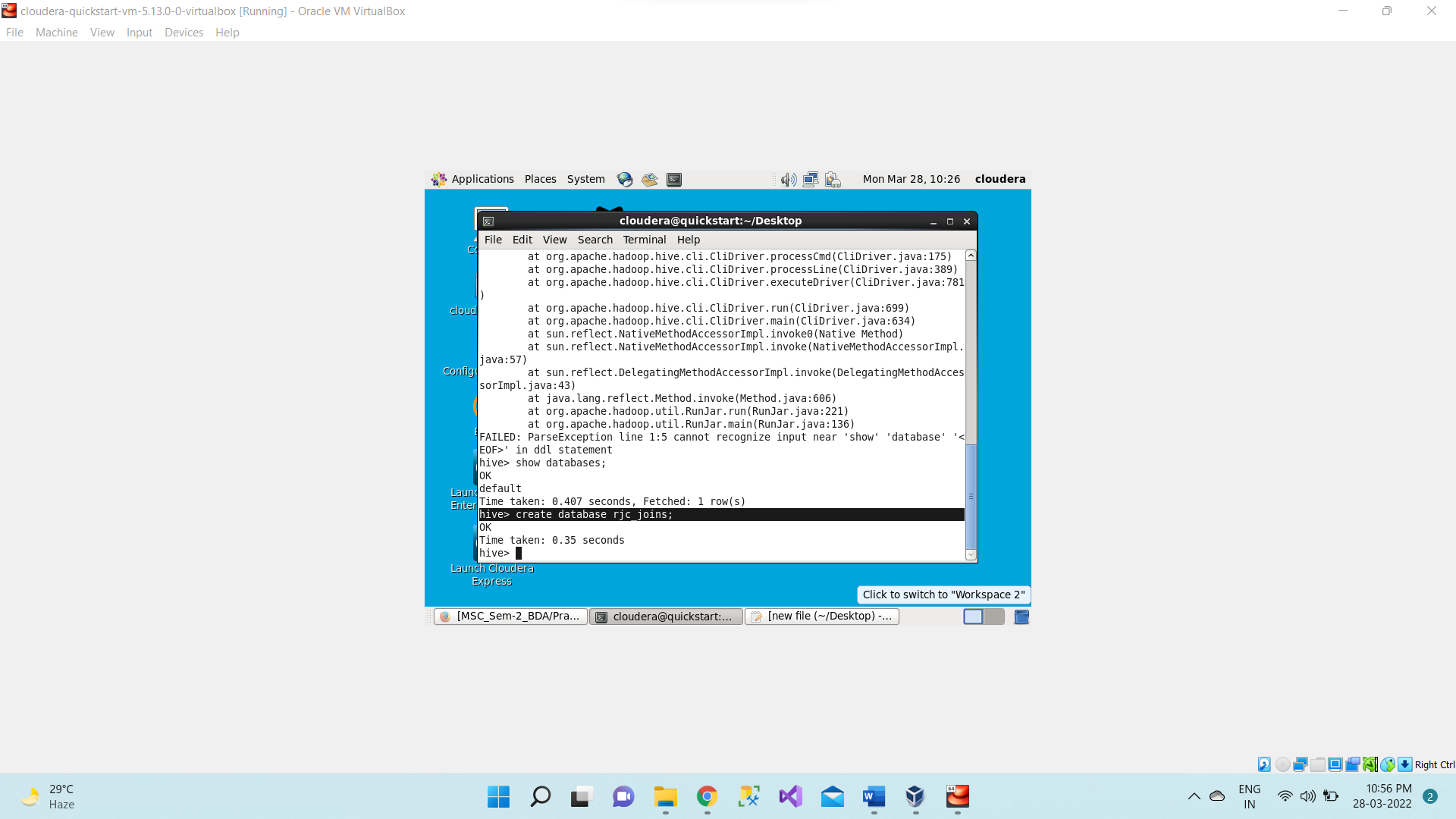
1. **hive**





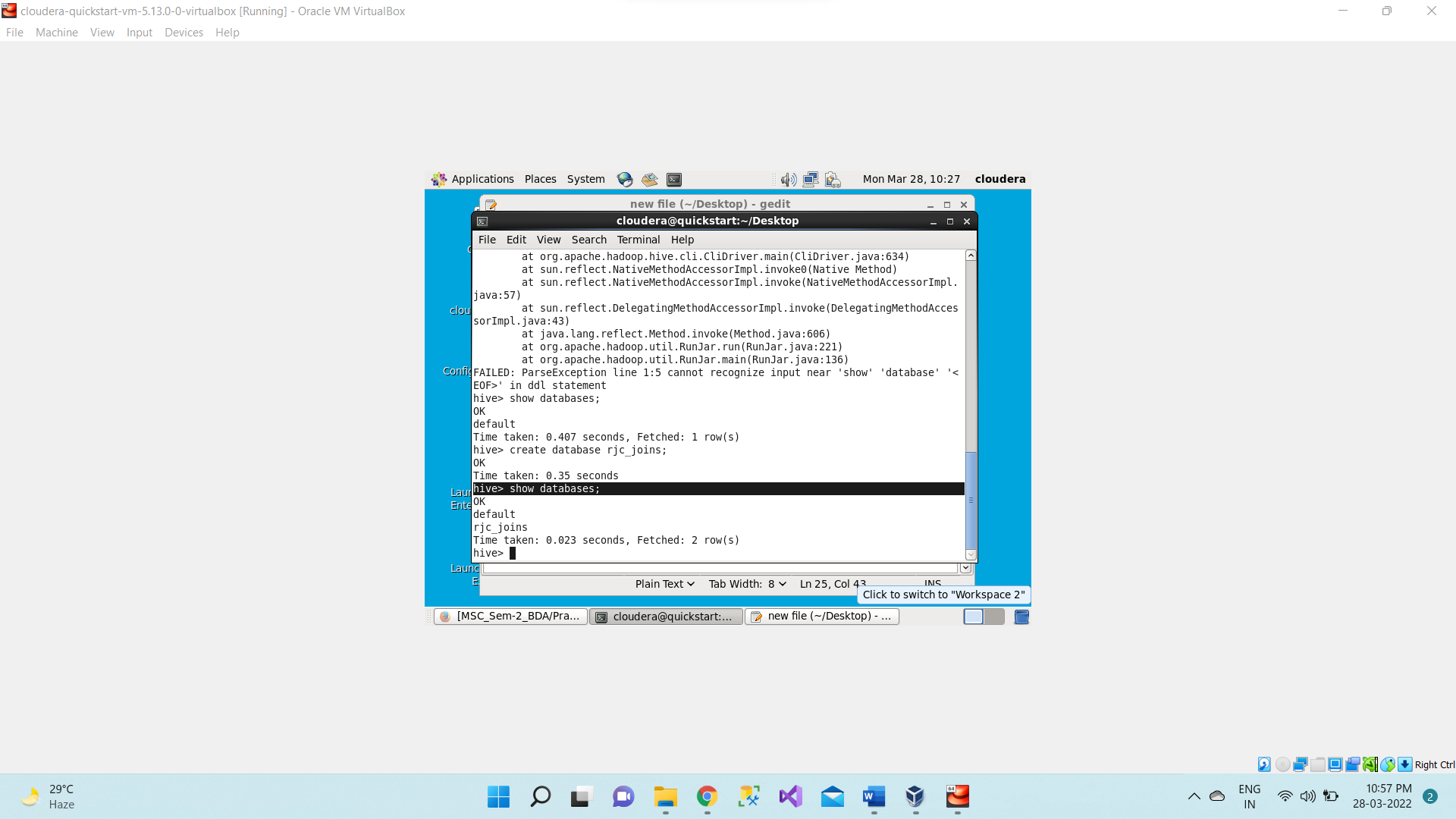
Now we will be creating a new database named as rjc\_joins using below command,

1. **create database rjc\_joins;**



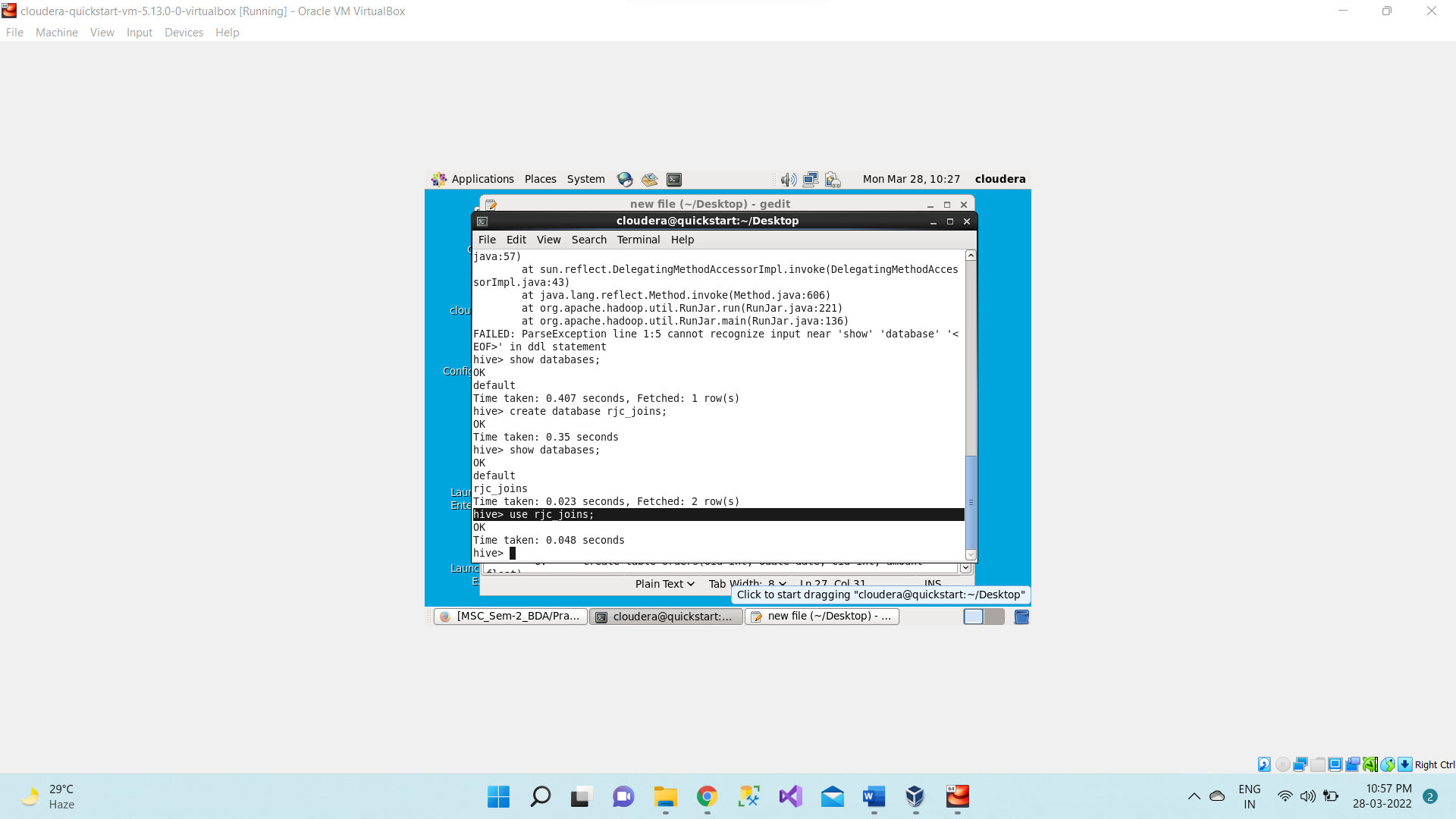
And then showing the databases.

**show databases;**



Now to work inside this database we use below command;

1. **use rjc\_joins;**



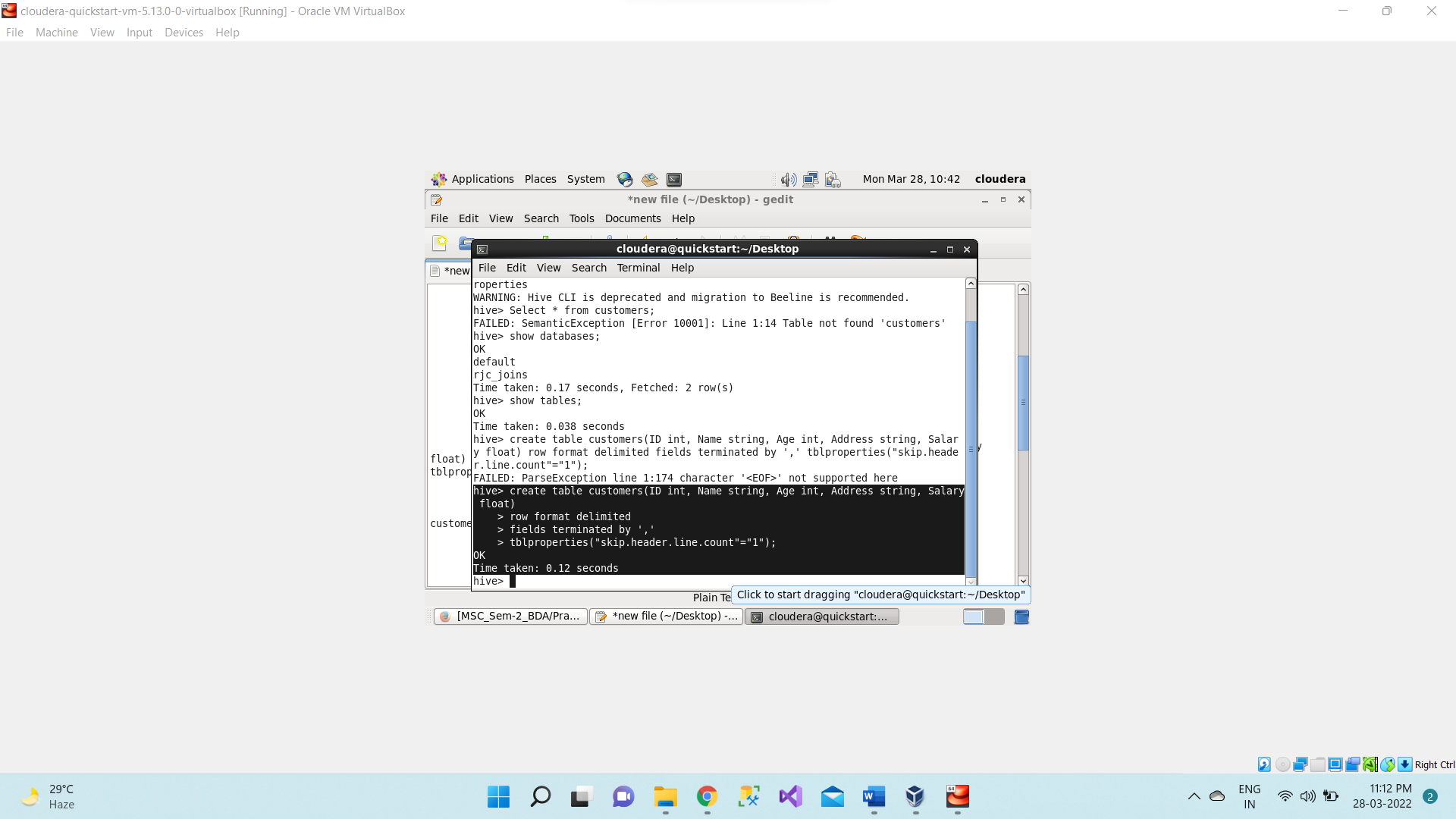
Now we will create two tables in one table we will load the **Customer.csv** file and in the other table we will load **Orders.csv** file.

1. **create table customers(ID int, Name string, Age int, Address string, Salary float)**

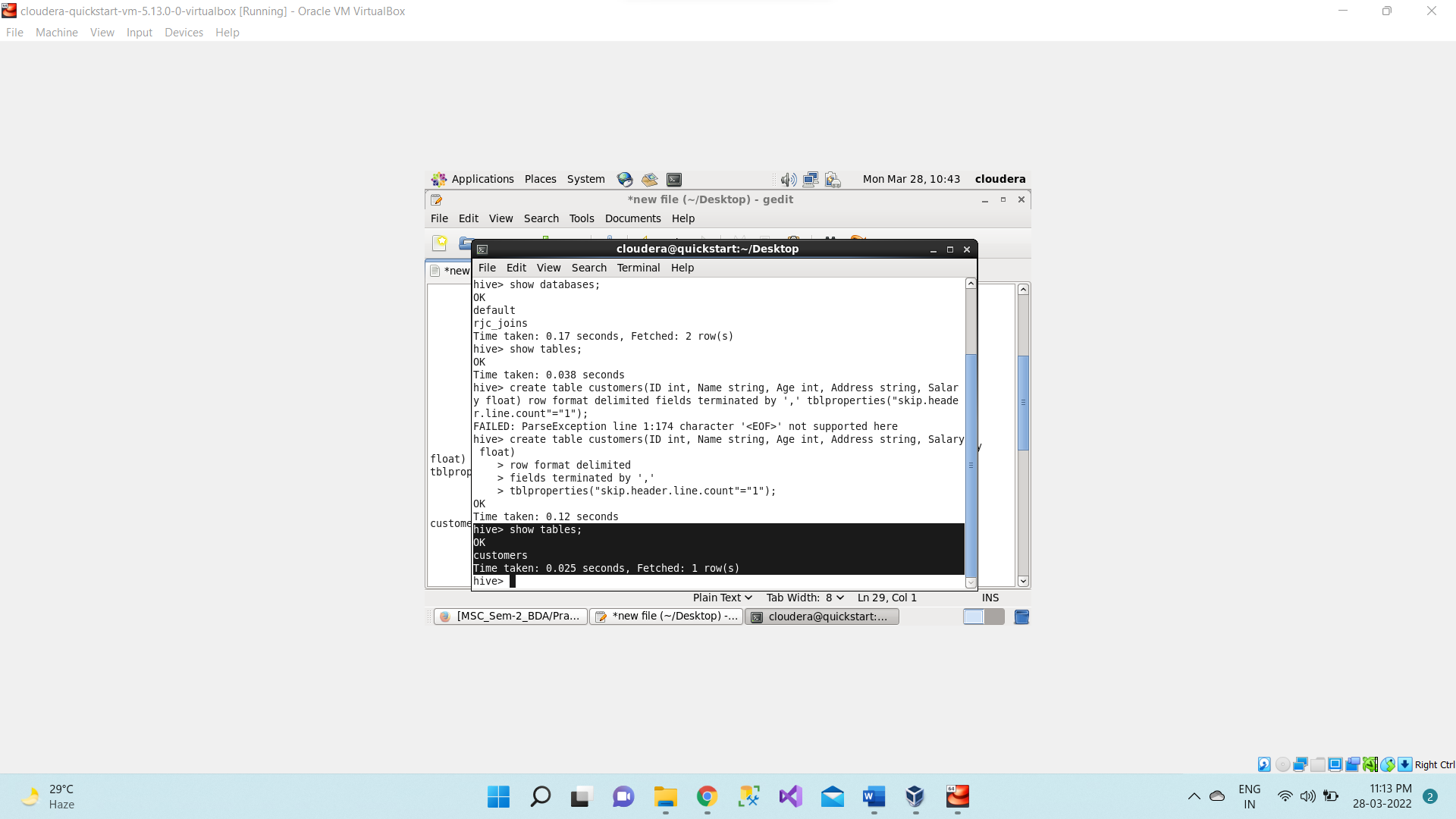
**row format delimited**

**fields terminated by ','**

**tblproperties("skip.header.line.count"="1");**

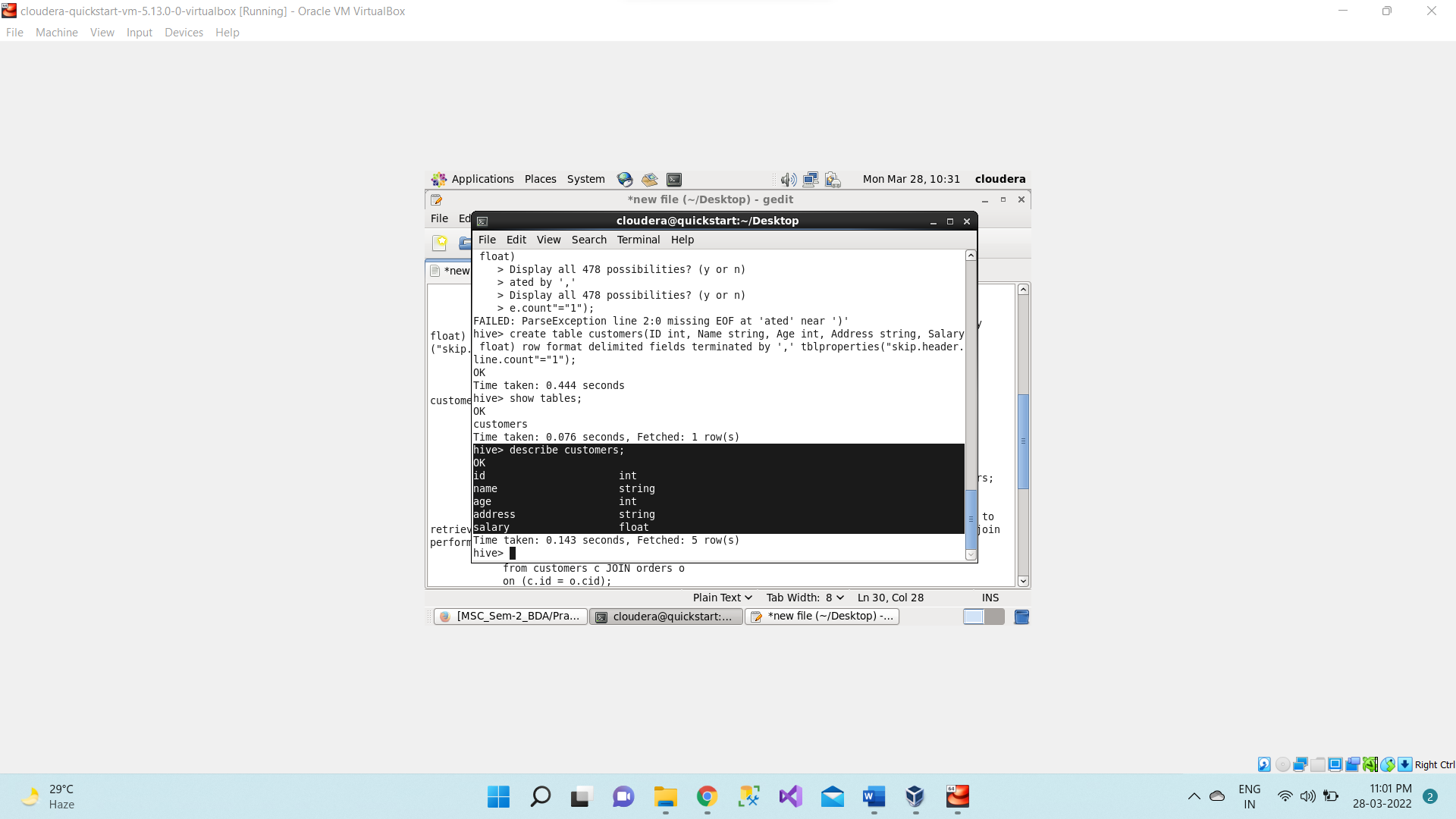


**show tables;**



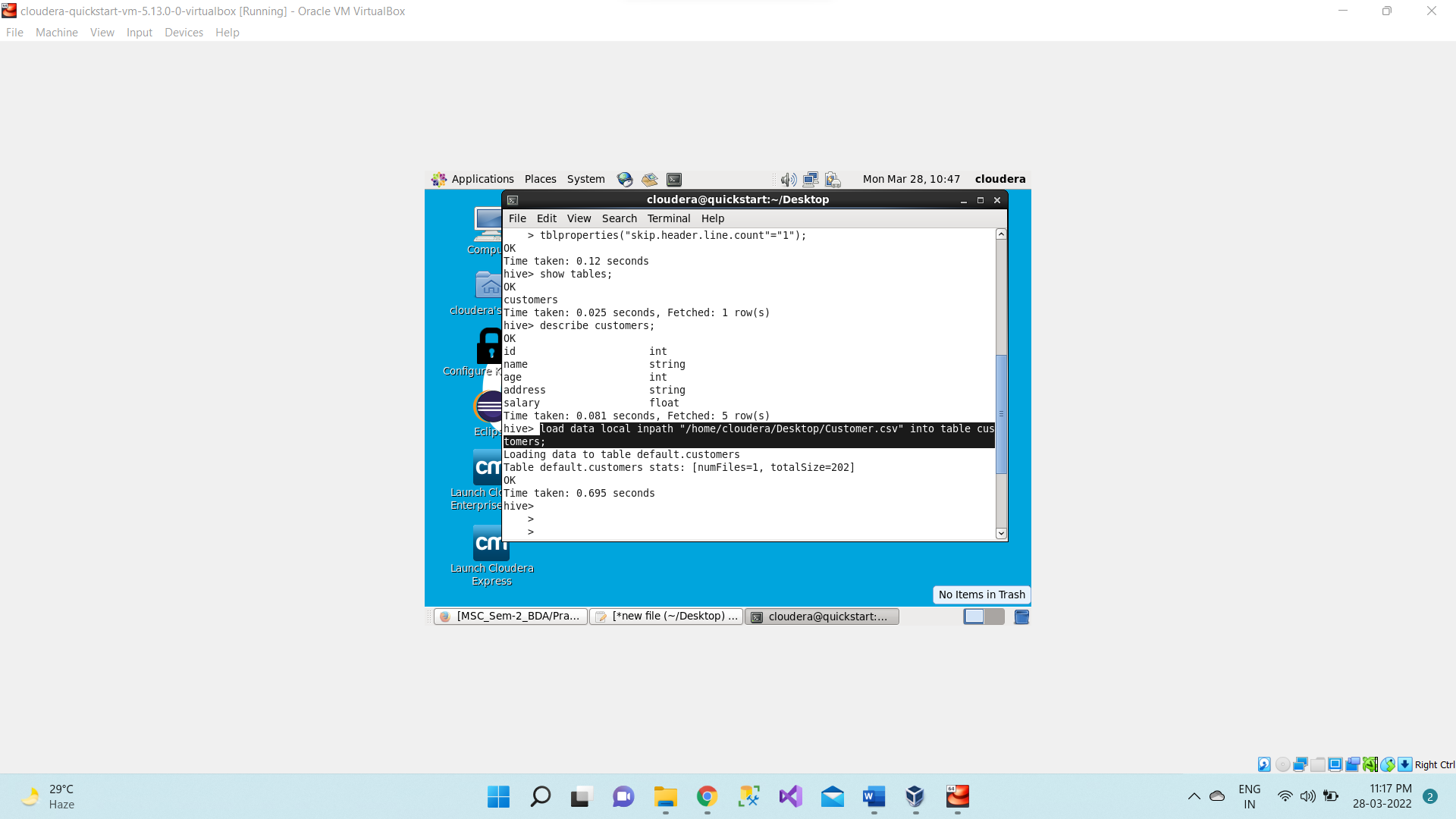
Now we will see the schema of the table using describe command, **describe customers;**

**describe customers;**

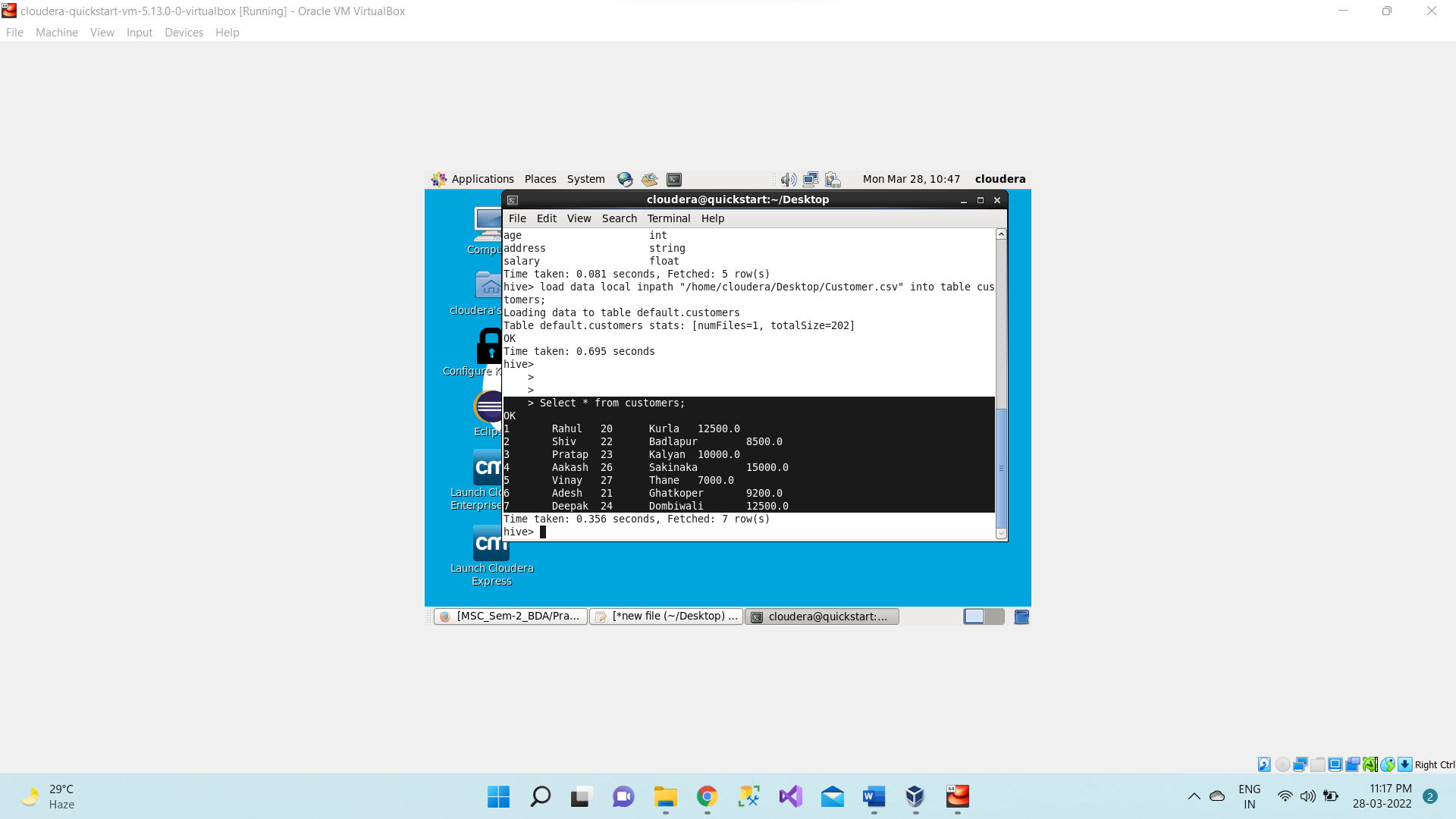


Now loading data in the **customers** table from **Customer.csv** file which present inside  /home/cloudera/**<dir\_name>** directory.

1. **load data local inpath "/home/cloudera/Desktop/Customer.csv" into table customers**



**select \* from customers;**



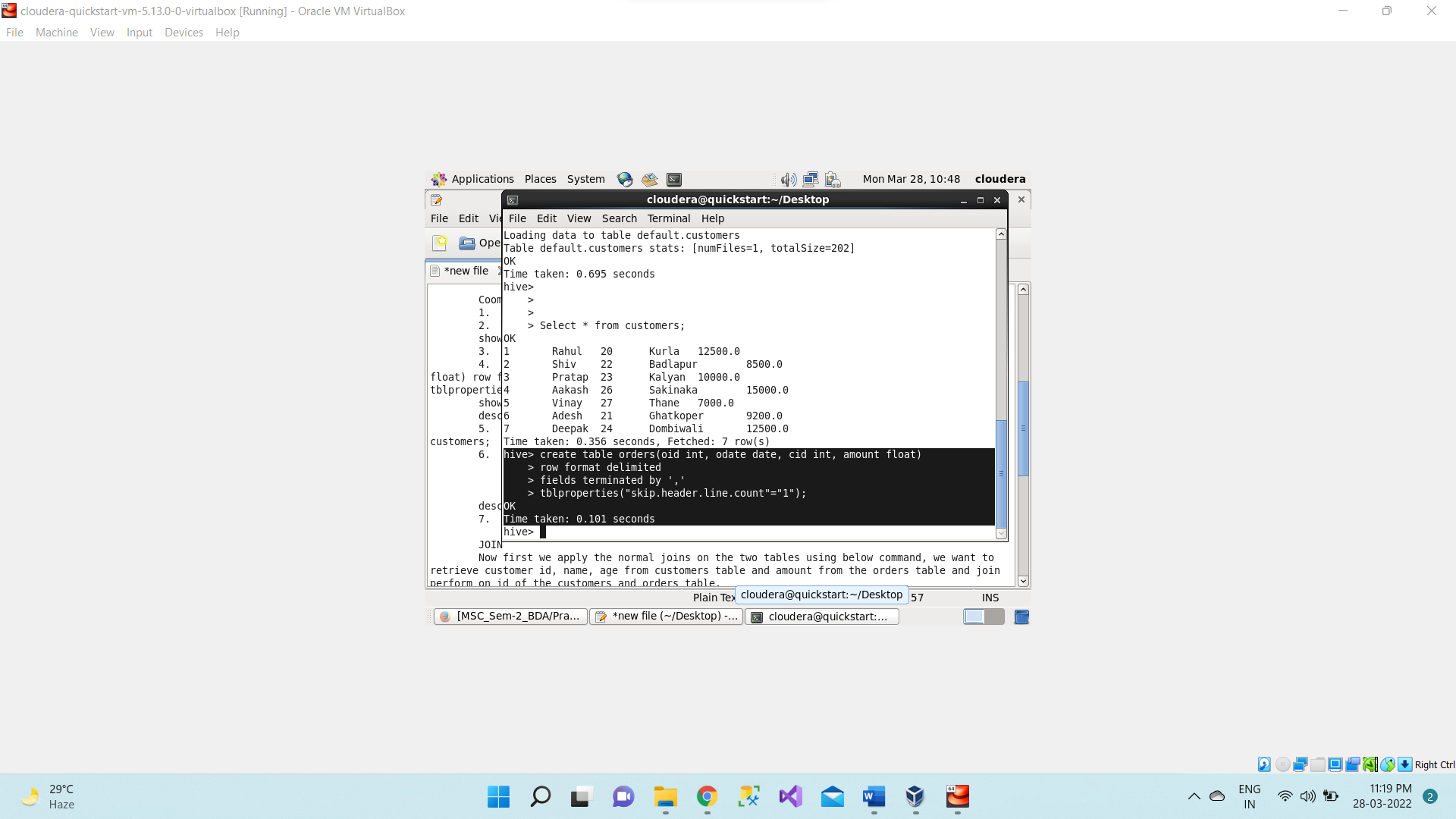
Creating a second table named as **orders** using below command,

1. **create table orders(oid int, odate date, cid int, amount float)**

**row format delimited**

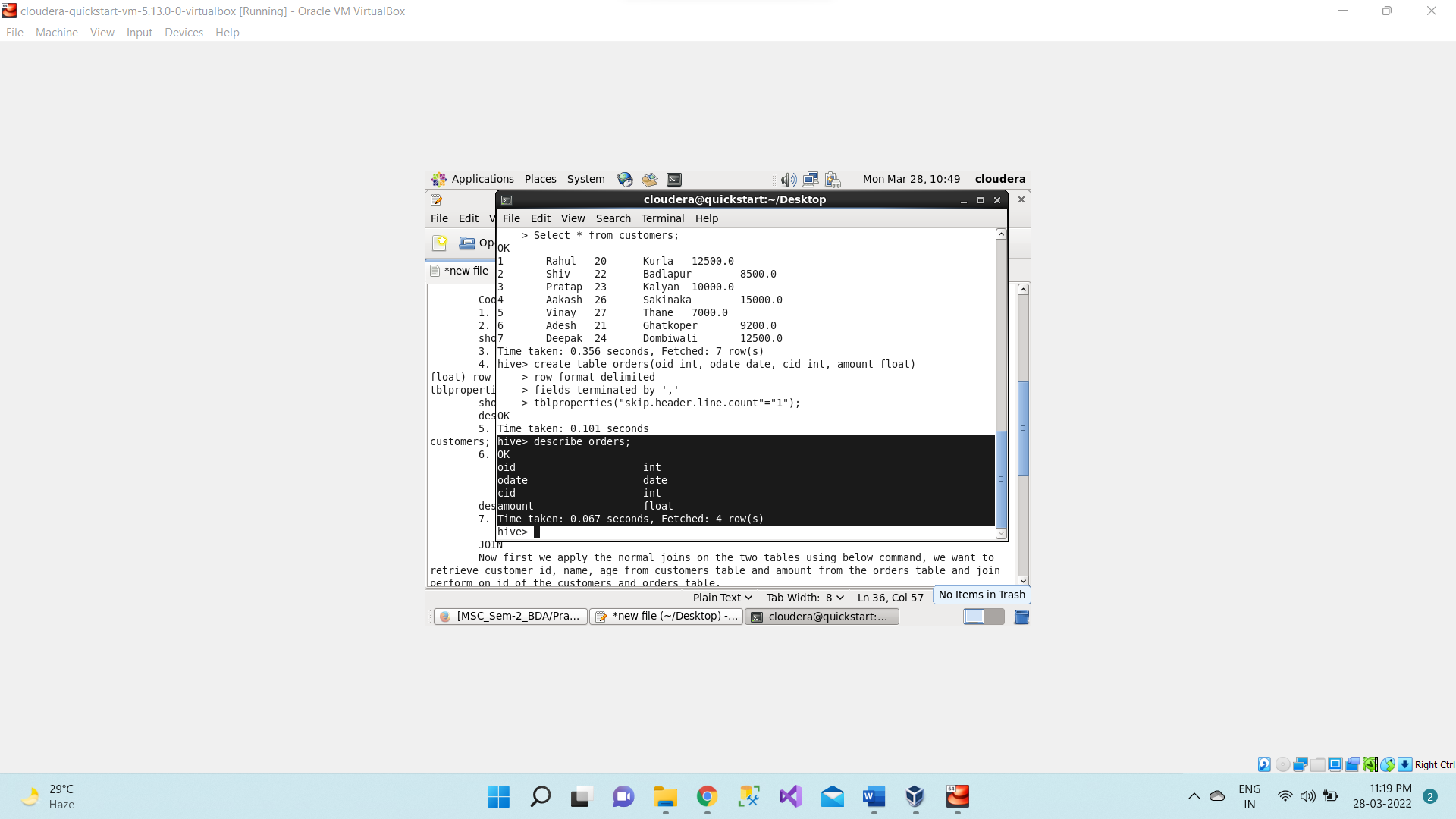
**fields terminated by ','**

**tblproperties("skip.header.line.count"="1");**



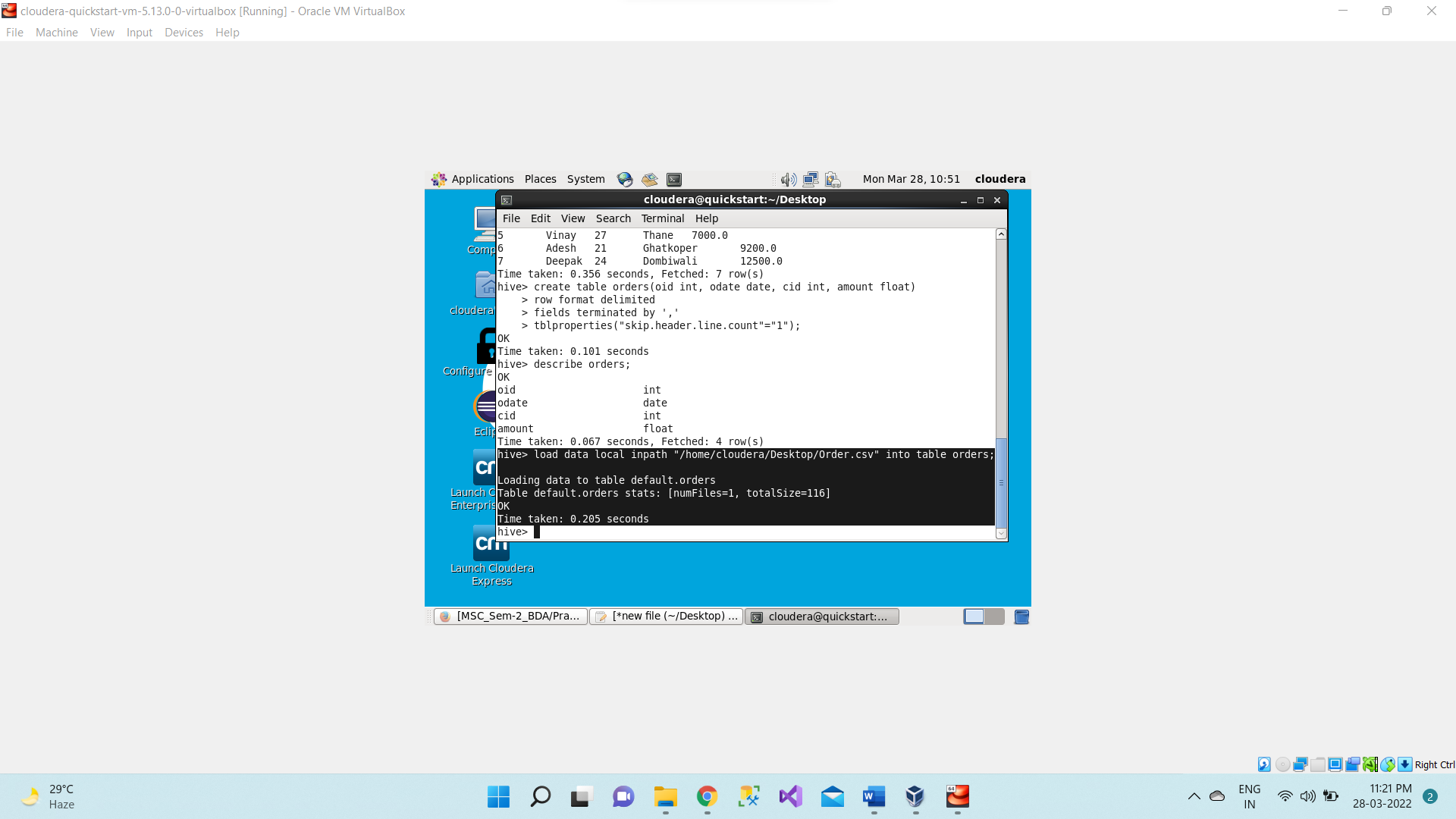
Now we will see the schema of the table using **describe** command,

**describe orders;**



Now loading data in the **orders** table from **Orders.csv** file which present inside  /home/cloudera/**<dir\_name>** directory.

1. **load data local inpath "/home/cloudera/Desktop/Orders.csv" into table orders;**



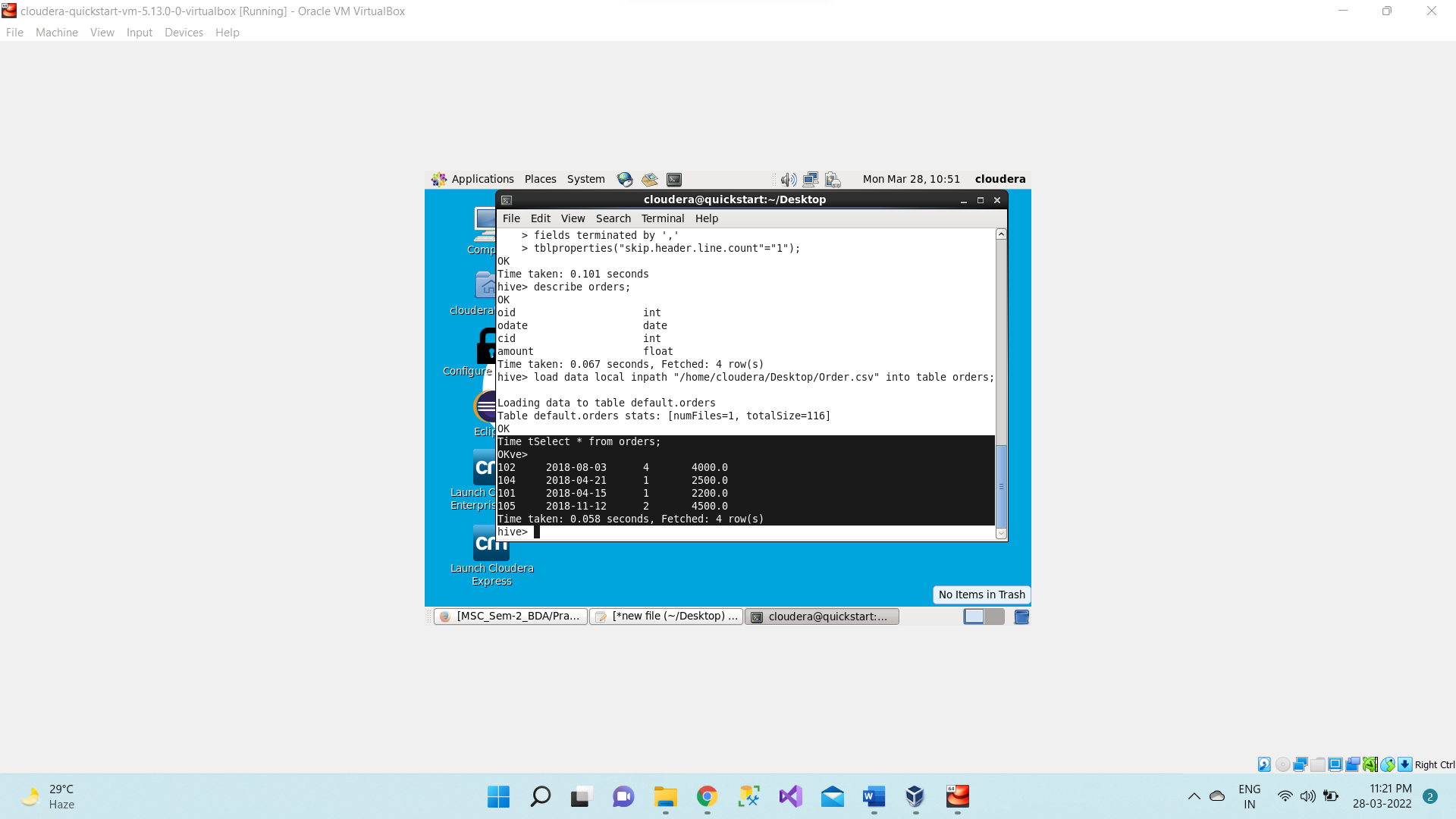
**JOIN**

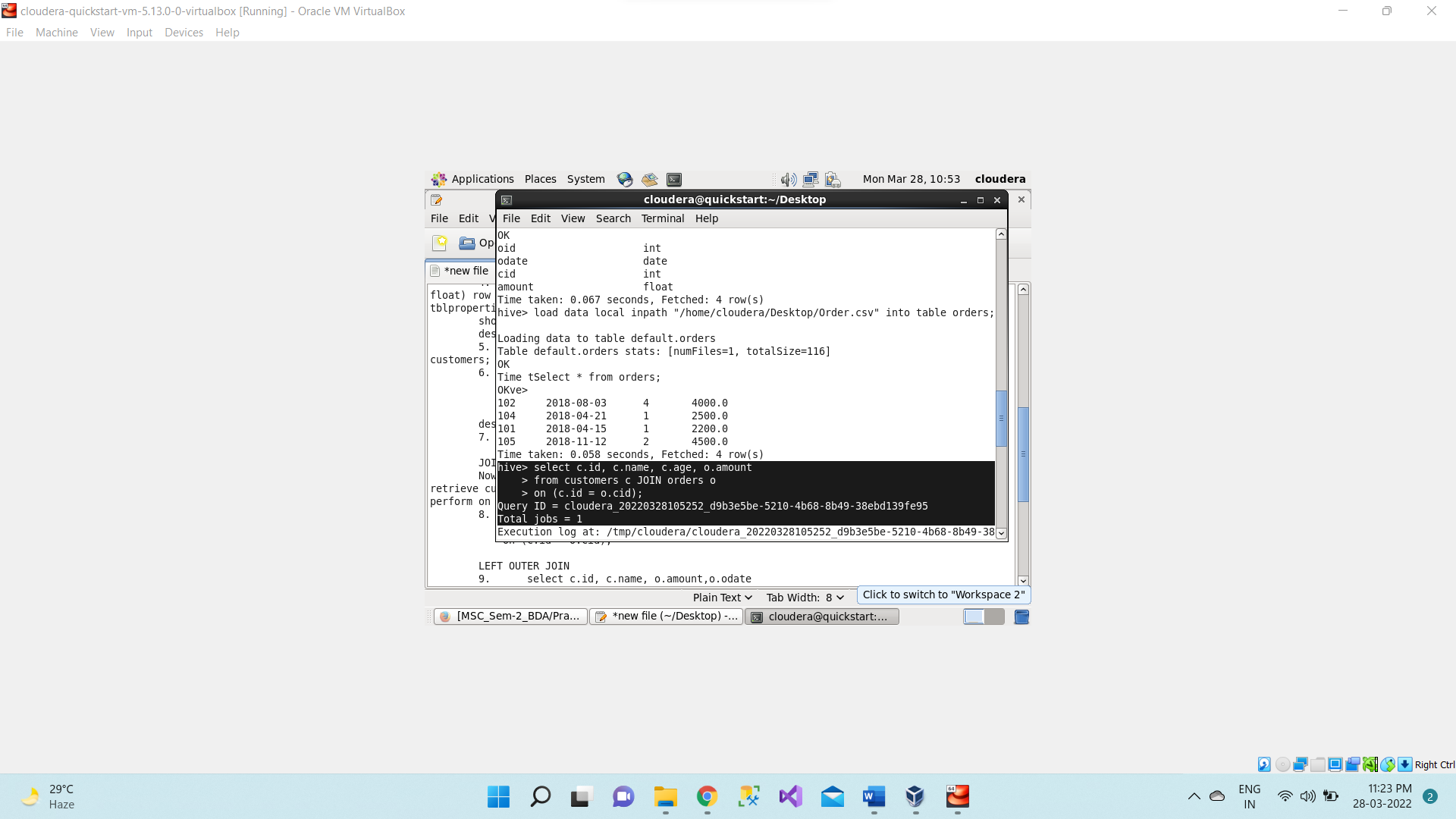
Now first we apply the normal joins on the two tables using below command, we want to retrieve customer id, name, age from customers table and amount from the orders table and join perform on id of the customers and orders table.

1. **select c.id, c.name, c.age, o.amount**

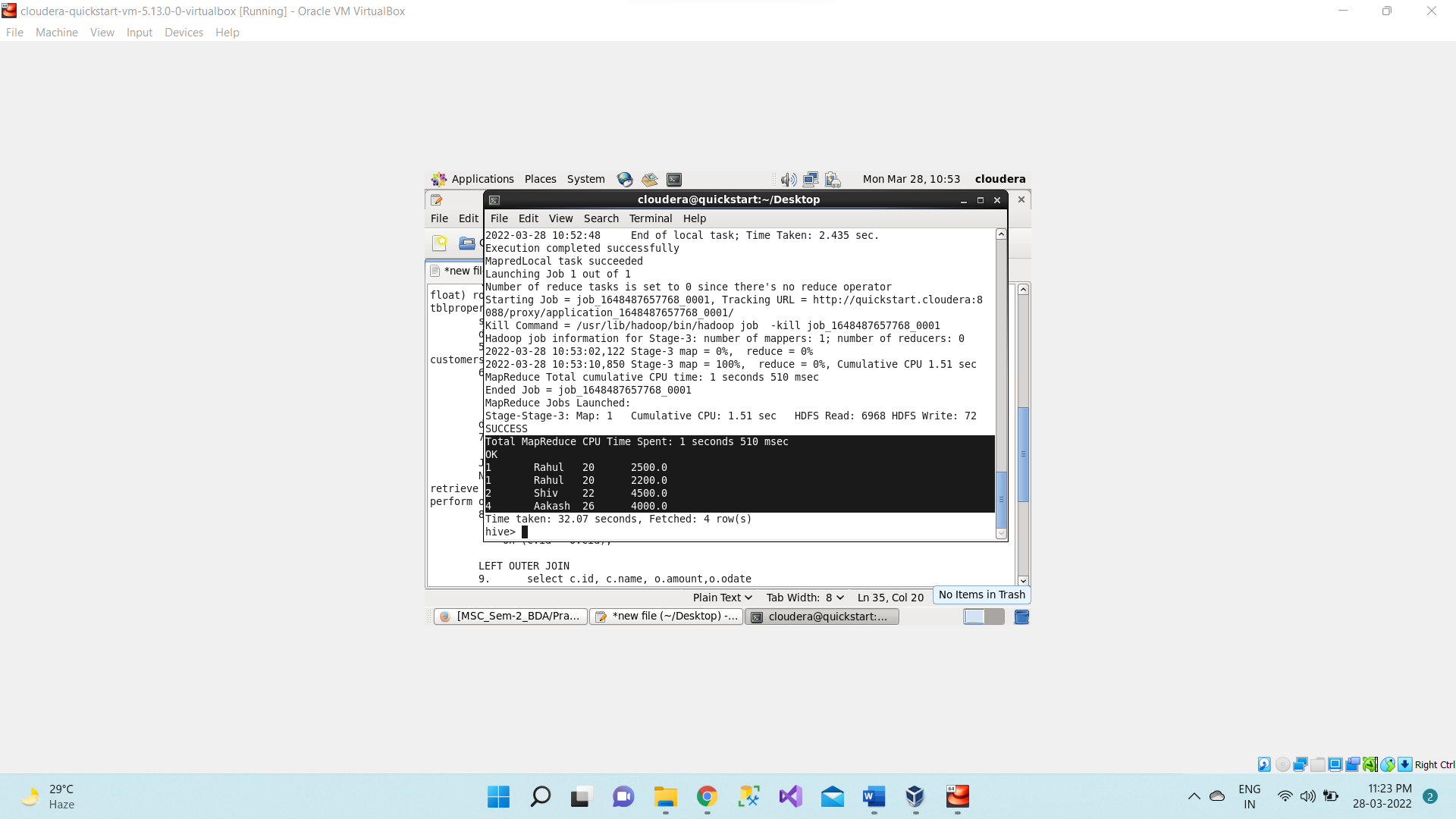
**from customers c JOIN orders o**

**on (c.id = o.cid);**





**Mapreduce task is performed.**



**LEFT OUTER JOIN**

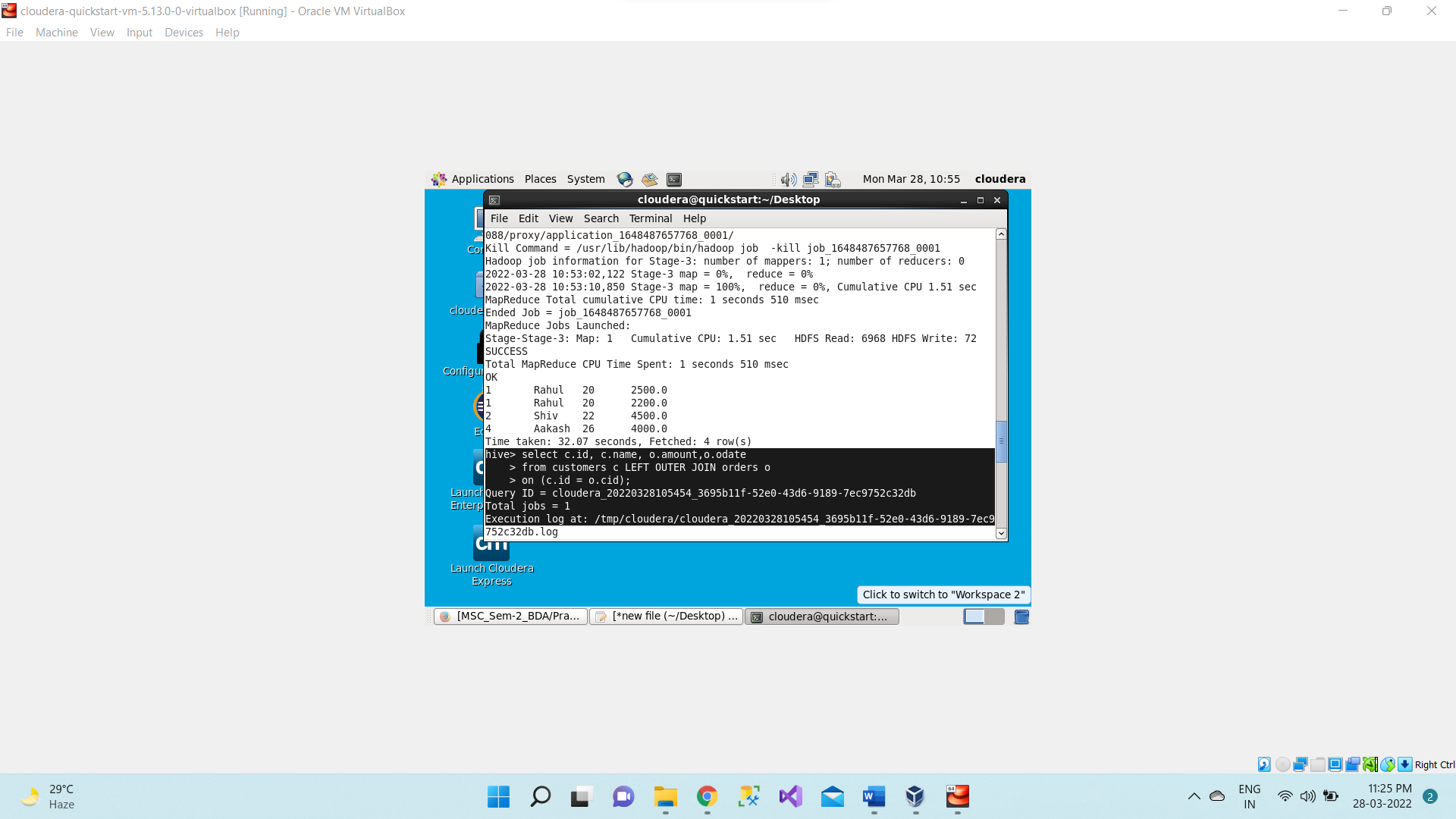
The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are  no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

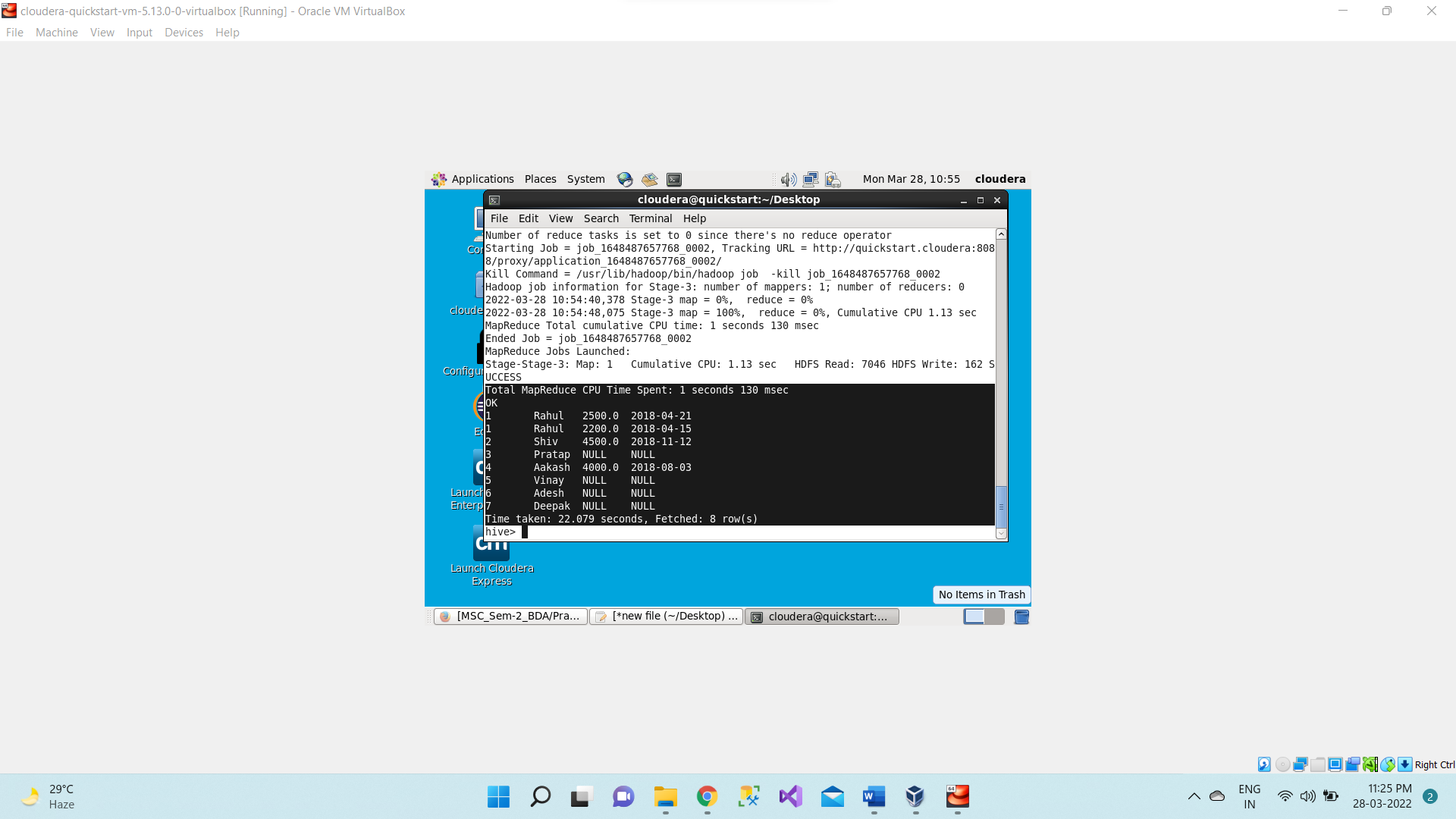
1. **select c.id, c.name, o.amount,o.odate**

**from customers c LEFT OUTER JOIN orders o**

**on (c.id = o.cid);**



**Mapreduce task is performed**



**RIGHT OUTER JOIN**

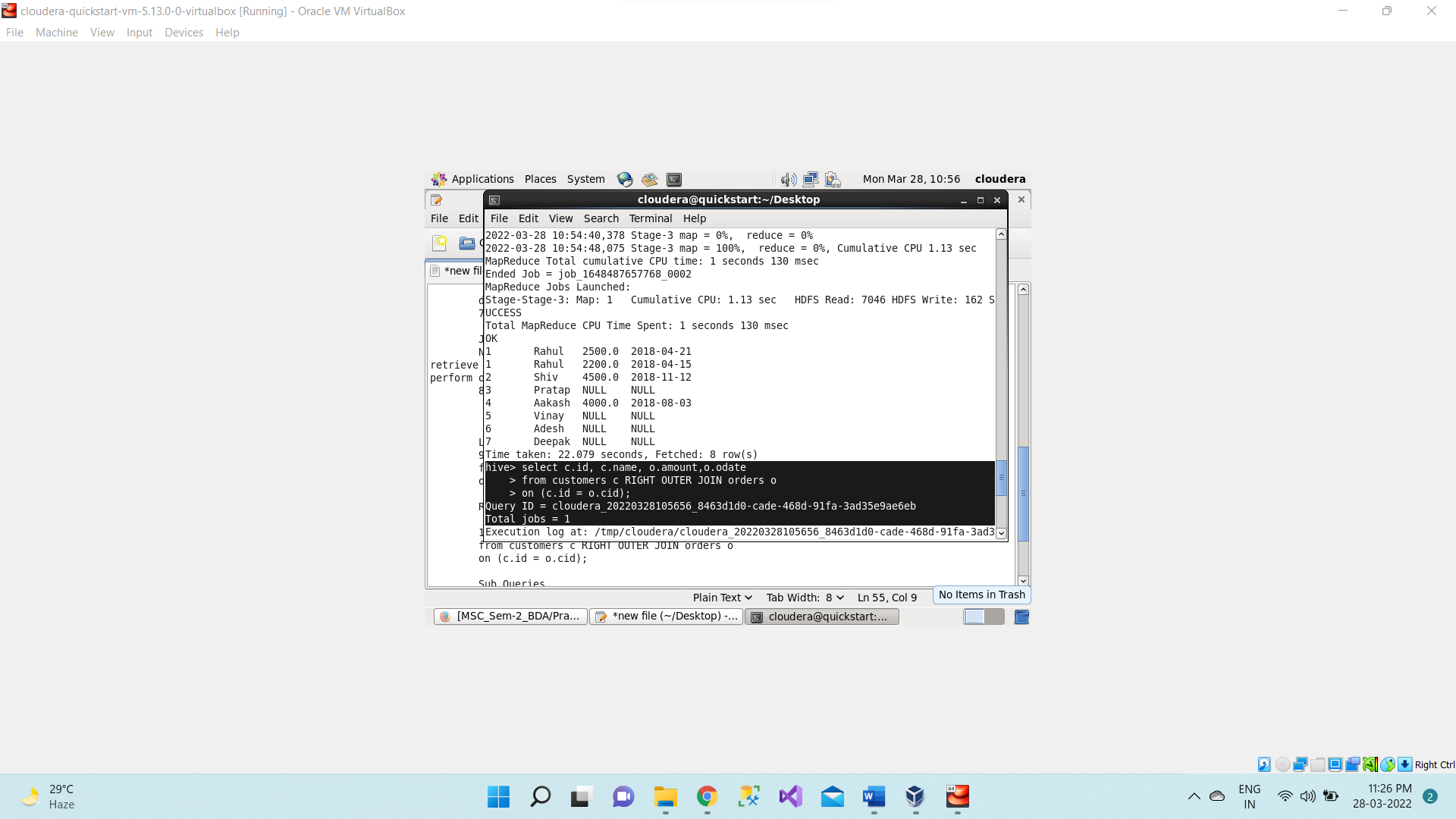
The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

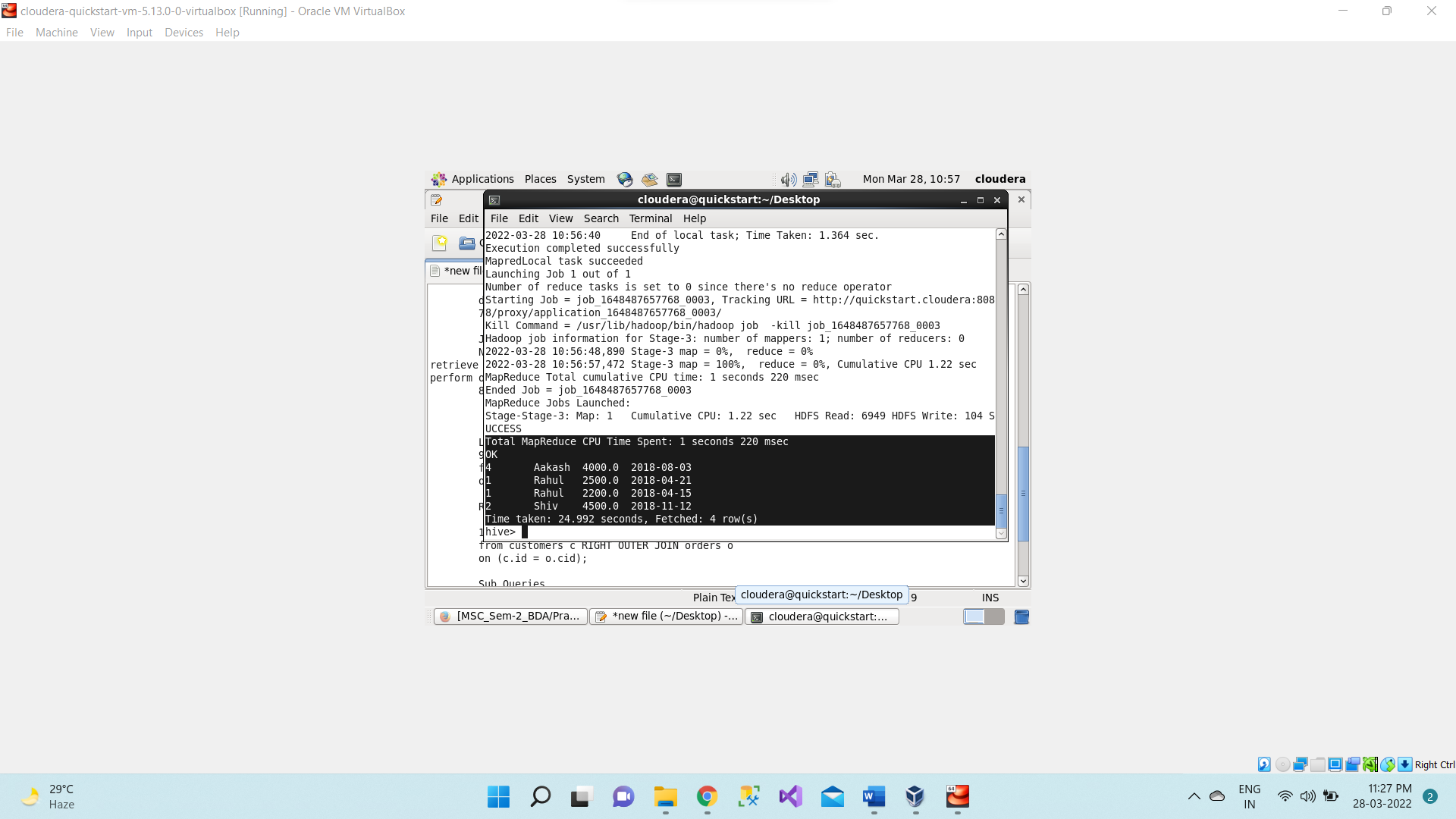
1. **select c.id, c.name, o.amount,o.odate**

**from customers c RIGHT OUTER JOIN orders o**

**on (c.id = o.cid);**



**Mapreduce task is performed**



Now we will be using the concept of **subqueries** for finding the second largest salary from the customers table.

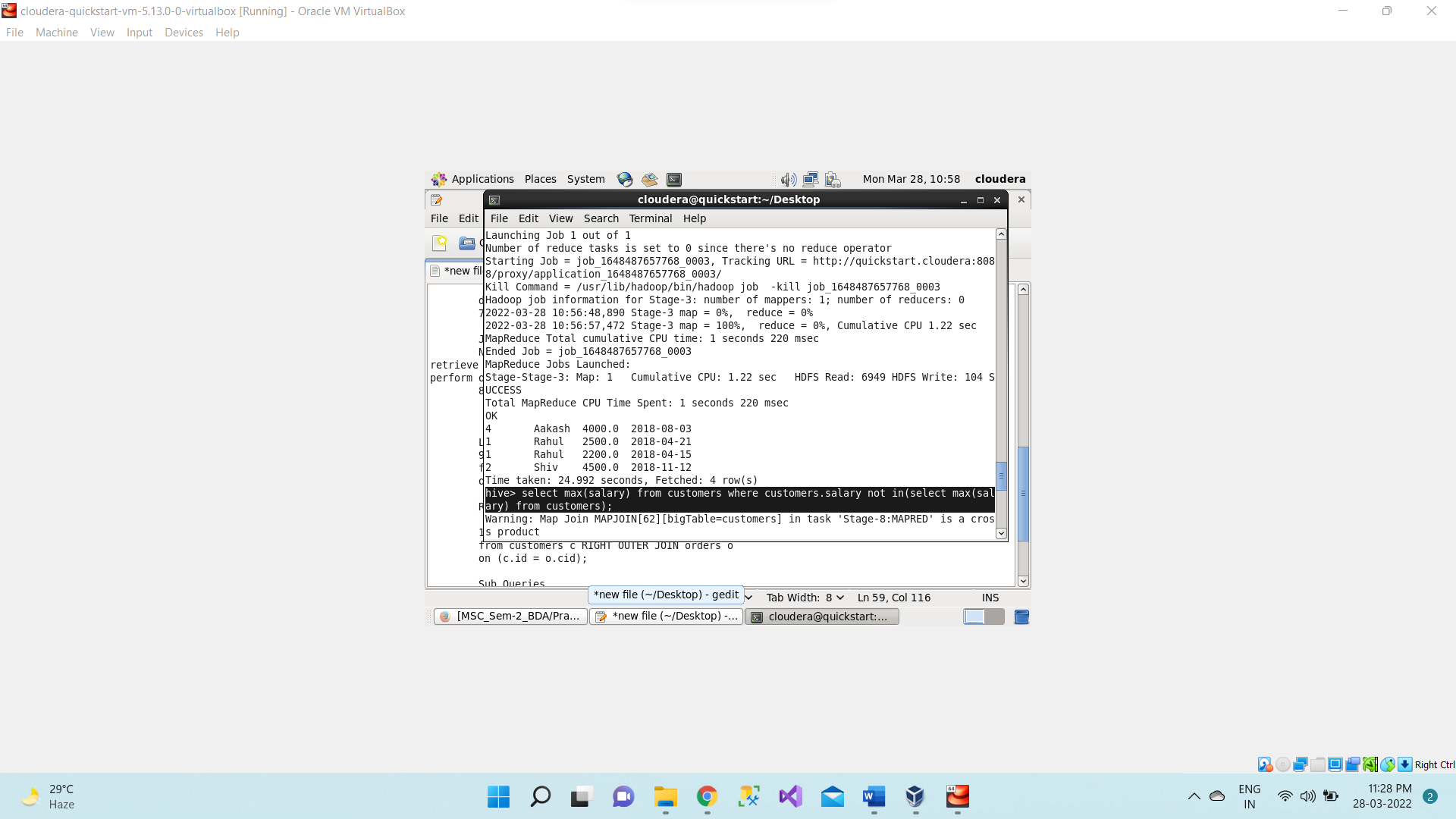
**Sub Queries**

A Query present within a Query is known as a sub query. The main query will depend on the values returned by the subqueries.

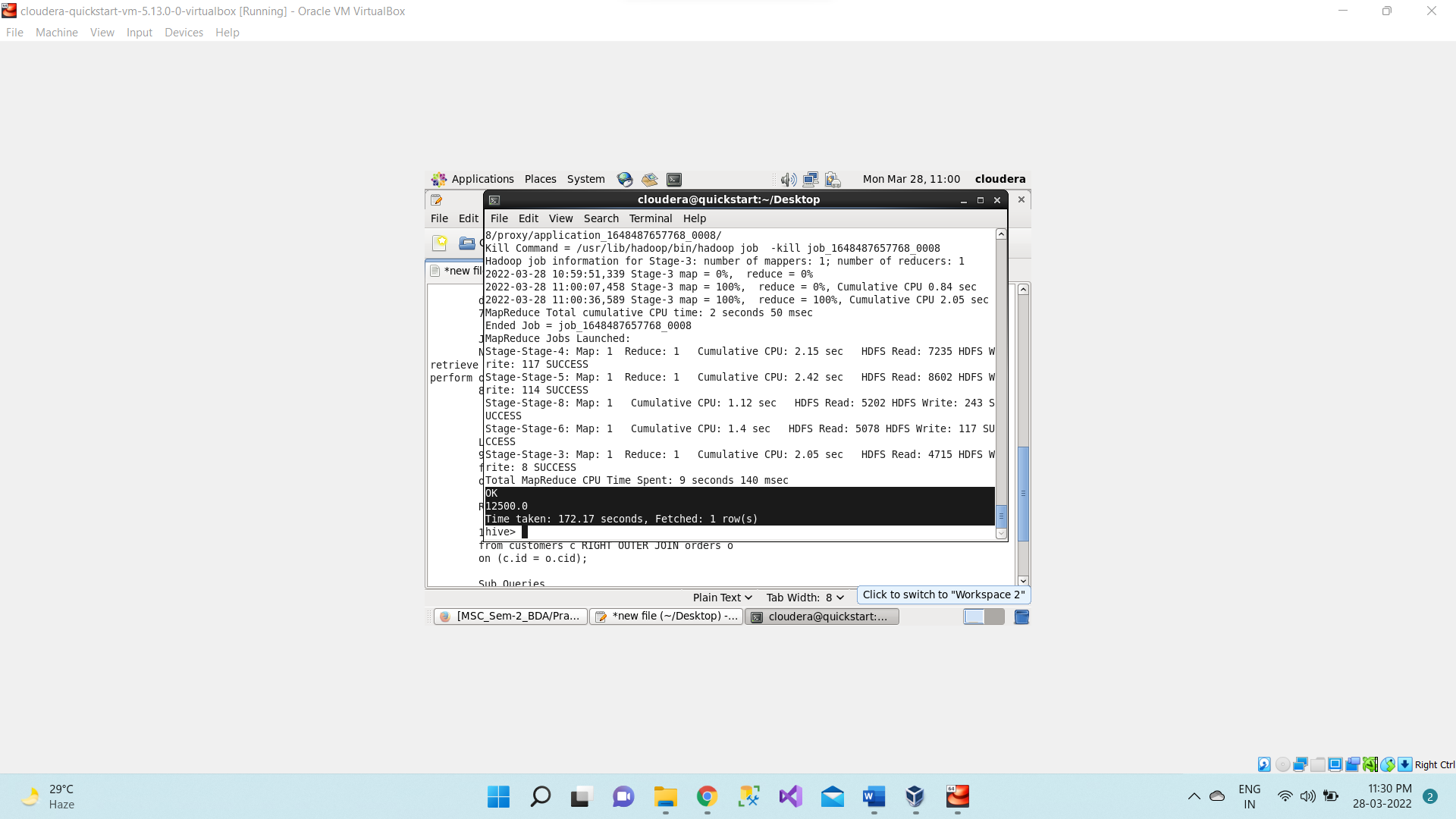
Subqueries can be classified into two types

* + Subqueries in FROM clause
  + Subqueries in WHERE clause

1. **select max(salary) from customers where customers.salary not in(select max(salary) from customers);**



**Mapreduce task is performed**



As we can see from the above output the **second largest salary** is **12500. 00.**

**Sorting**

The SORT BY syntax is similar to the syntax of ORDER BY in SQL language.

Hive supports SORT BY which sorts the data per reducer. The difference between "order  by" and "sort by" is that the former guarantees total order in the output while the latter  only guarantees ordering of the rows within a reducer. If there are more than one reducer, “sort by" may give partially ordered final results.

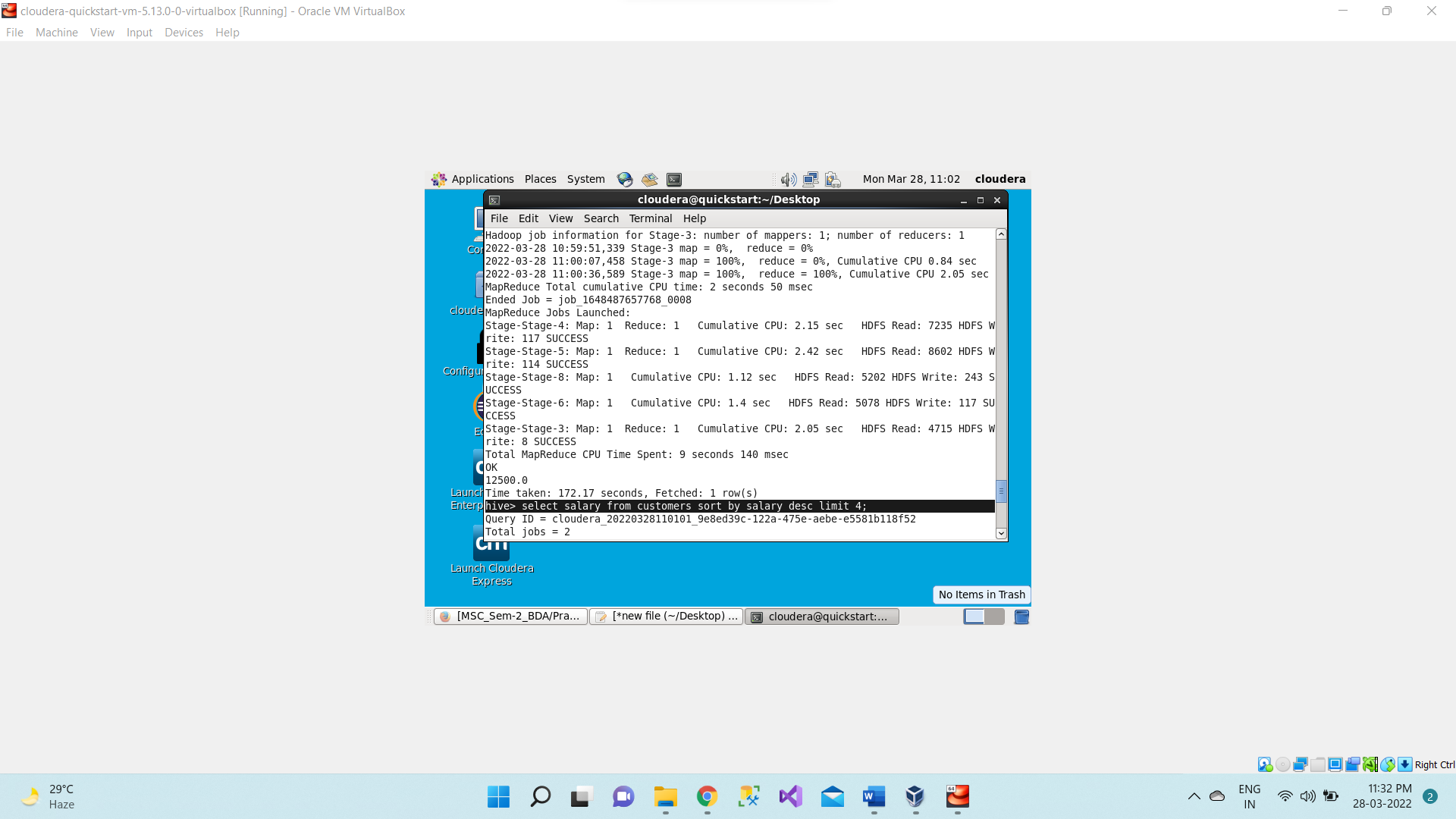
Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer.  The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order.

LIMIT can be used to minimize sort time.

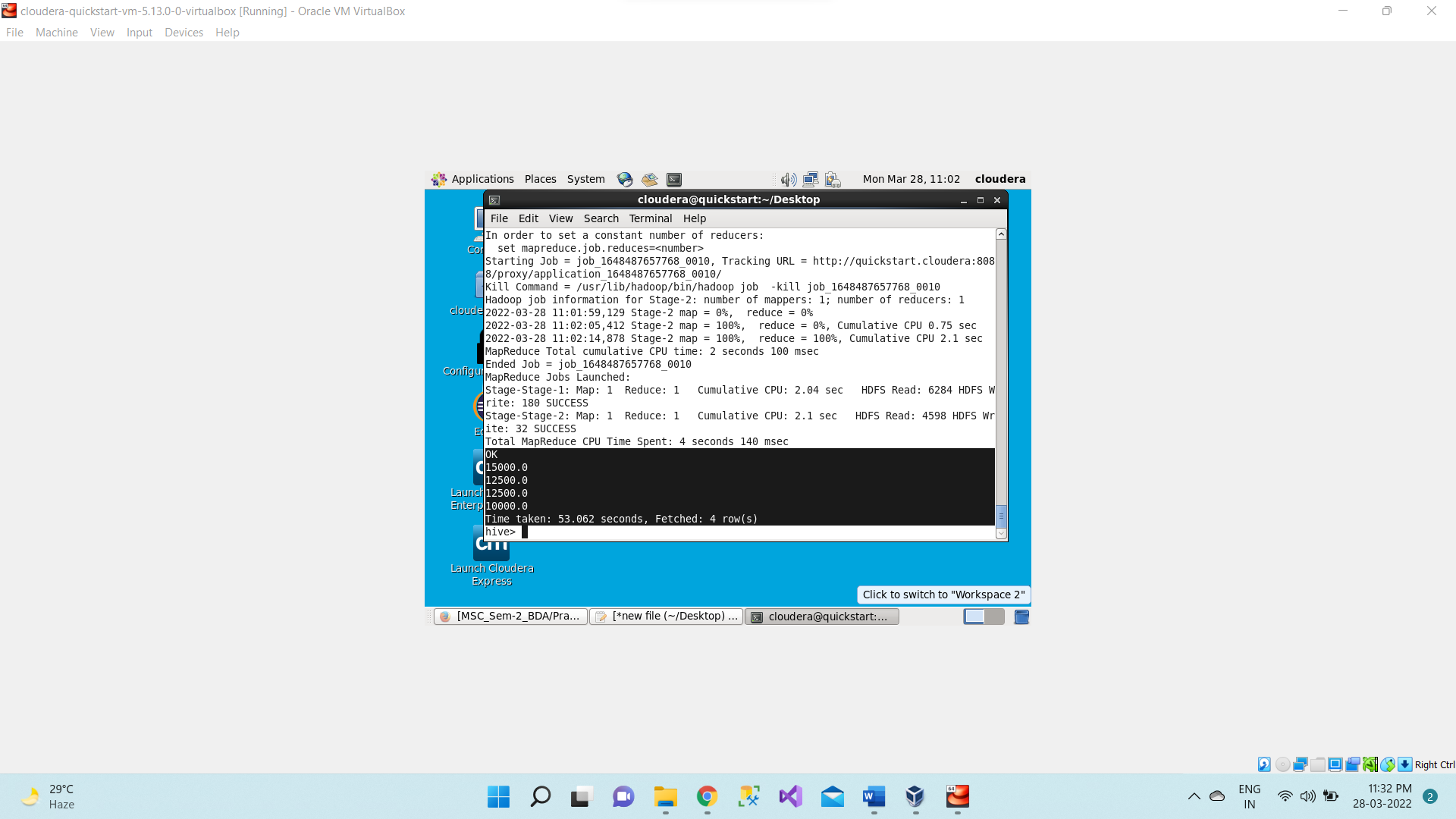
Now finding the fourth largest salary from the customers table using Sort by clause.

1. **select salary from customers sort by salary desc limit 4;**

It will give the only 4 records in the output after sorting them in descending order. This is not a complete syntax only we are showing what output it will give.



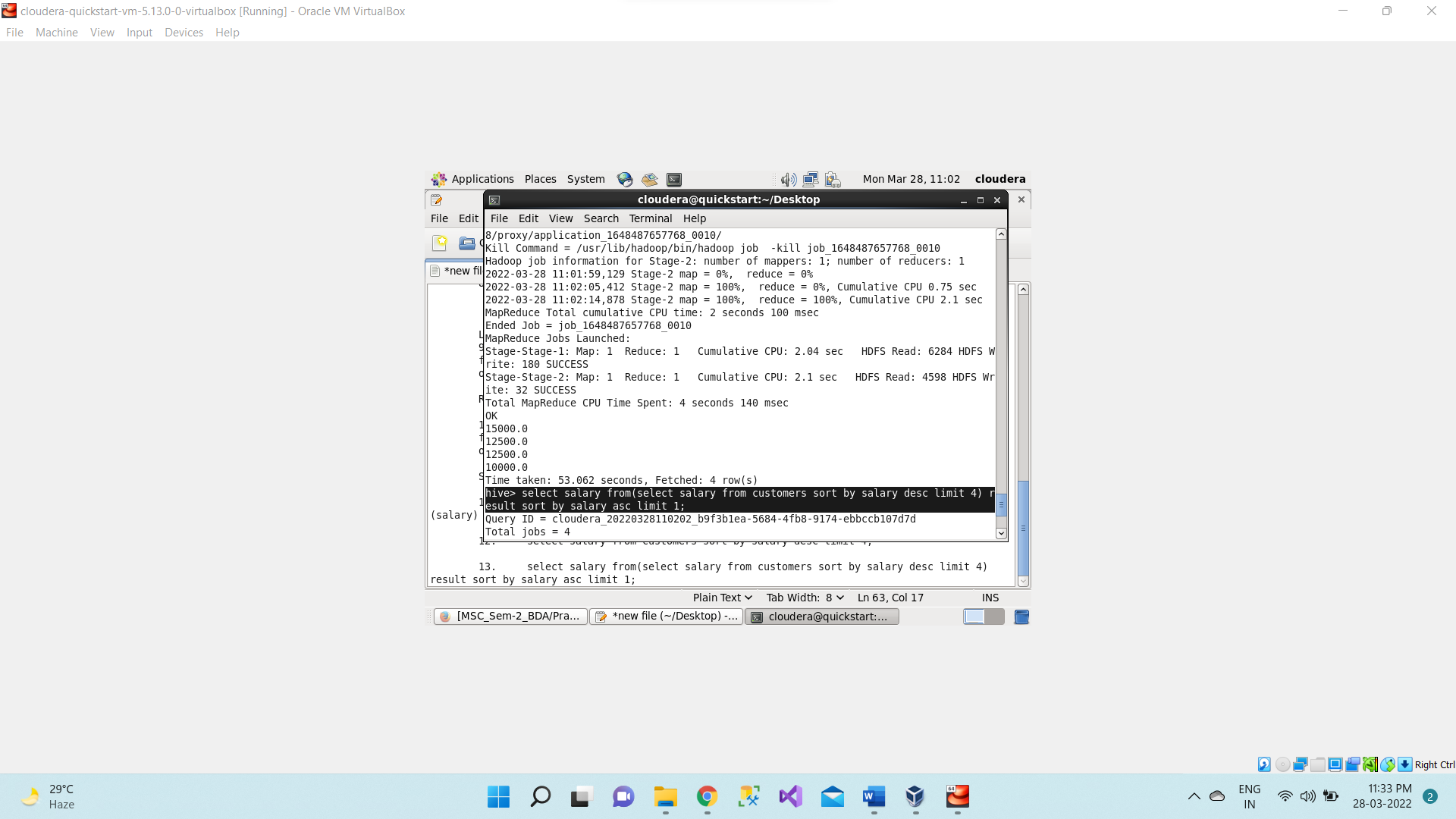
**Mapreduce task is performed**



Now what records which we have got by executing the above queries now we will use this query as subqueries and we will now sort them in ascending order to find fourth largest salary of customer table.

1. **select salary from(select salary from customers sort by salary desc limit 4) result sort by salary asc limit 1;**

Now whatever result we get from subquery we will store them in result table and then it will sort the result table in ascending order and as we want fourth largest salary so we are limiting it to 1.



Now we got the fourth largest salary i.e., 10000.0 as an output.

