# Part 1 : Theoretical Questions

Q1.1 Which of the following issues may be caused by lot of small files in HDFS:

1. **NameNode memory usage increases significantly**

To HDFS είναι πολύ καλό στο να κρατάει στην cache όλα τα ονόματα αρχείων και τις διευθύνσεις των block των αρχείων αυτών στην μνήμη. Αυτό συμβαίνει γιατί κάθε αρχείο, διαδρομή ή block διατηρείται σαν αντικείμενο στην μνήμη του namenode, όπου κάθε αντικείμενο χρειάζεται 150 bytes. Άρα ένα μεγάλο πλήθος αρχείων, δηλαδή περίπου 10 εκατομμύρια αρχεία, χρειάζονται περίπου 3GB μνήμη, κάτι το οποίο για ένα απλό σύστημα χρήστη θα ήταν μεγάλη σπατάλη.

Q1.2 How is uniform data distribution across the servers achieved in HDFS?

1. **By splitting files into blocks**

Όπως και σε ένα σύστημα αρχείων για ένα μόνο υπολογιστικό σύστημα, τα αρχεία στο HDFS χωρίζονται σε μικρότερα τμήματα σε μέγεθος block, τα οποία, στη συνέχεια, αποθηκεύονται ως ανεξάρτητες μονάδες. Σε αντίθεση με ένα σύστημα αρχείων για ένα μόνο υπολογιστικό σύστημα, ένα αρχείο στο HDFS, που έχει μέγεθος μικρότερο από ένα block, δεν καταλαμβάνει ένα ολόκληρο block στο αποθηκευτικό μέσο (ωστόσο, στο πλαίσιο του HDFS συνεχίζει να θεωρείται ότι καταλαμβάνει ένα ολόκληρο HDFS block).

Q1.3 If you have a very important file and you aim to store it into HDFS, what is the best way to minimize the risk from losing it?

Ο καλύτερος τρόπος είναι η δημιουργία αντιγράφων ασφαλείας των αρχείων (replication), τα οποία απαρτίζουν τα μεταδεδομένα των αρχείων που υπάρχουν στο HDFS. Το Hadoop μπορεί να ρυθμιστεί, έτσι ώστε το namenode να αντιγράφει τα αρχεία αυτά σε πολλά συστήματα αρχείων. Οι παραπάνω αντιγραφές είναι σύγχρονες και ατομικές. Για παράδειγμα, μια συχνή επιλογή σε μια συστάδα με HDFS είναι να αντιγράφονται τα αρχεία των μεταδεδομένων στον τοπικό σκληρό δίσκο του namenode, καθώς και σε μια απομακρυσμένη προσάρτηση NFS (remote NFS mount).

Q1.4 You were told that two servers in HDFS were down: Datanode and Namenode. Which would be your reaction:

1. **Restore Namenode first**

Το Datanode είναι υπέυθυνο για την αποθήκευση των δεδομένων στο HDFS. Παρόλα αυτά όταν ένα Datanode τεθεί εκτός λειτουργίας, δεν επηρεάζει την διαθεσιμότητα των δεδομένων ή των συστάδων. Το Namenode θα αναλάβει την δημιουργία αντιγράφων των block τα οποία είναι διαχειρίσιμα από το συγκεκριμένο Datanode. Αντίθετα το Namenode είναι κρίσιμο σημείο για το σύστημα Hadoop και όταν αυτό πέσει τότε η HDFS/Hadoop συστάδα είναι εκτός πρόσβασης και θεωρείται εκτός λειτουργίας.

Q1.5 You are writing a 10GB file into HDFS with a replication of 2 and block size of 64MB. How much total disk space will this file use? Explain your answer, shortly.

Ο τύπος για τον υπολογισμό του πραγματικού μεγέθους είναι σχετικά ο παρακάτω:

total storage=file storate \* no of replication

συνεπώς θα χρειαστούν περίπου 20GB συνολικού χώρου στον σκληρό δίσκο.

Q1.6 Your colleague mistakenly dropped the table 'access\_log' created with the following statement, in Hive:

CREATE EXTERNAL TABLE access\_log (

id STRING,

user\_id STRING,

request STRING,

response STRING,

status\_code INT

)

LOCATION '/data/access\_logs'

STORED AS TEXTFILE;

Which would be your reaction? **Explain** your answer**, shortly**.

Μιας και ο παραπάνω πίνακας “access\_log” ήταν εξωτερικός (external), το Hive δεν διαχειρίζεται την αποθήκευση των δεδομένων του. Κατά την διαγραφή του πίνακα, μόνο ο ορισμός του διαγράφεται, ενώ τα δεδομένα διατηρούνται. Συνεπώς κατά την δημιουργία του ίδιου πίνακα με το ίδιο όνομα, το λάθος θα διορθωνόταν πλήρως.

Q1.7 You have a bunch of data in your local filesystem (which is not part of the cluster) and you need to load it in Hive. Describe the steps that you are going to follow in order the data to be accessible through Hive queries. Code is not required, just a short description of the steps.

Θεωρώντας ότι τα δεδομένα που έχουμε στο τοπικό μας σύστημα σε Comma Seperated Values αρχεία (csv), μπορούμε να φορτώσουμε τα αρχεία σε αντίστοιχους πίνακες που έχουμε δημιουργήσει μέσα από το HIVE, όπου κάθε στήλη είναι και ένα γνώρισμα του πίνακα. Στην συνέχεια μπορούμε να φορτώσουμε το αρχείο students.csv για παράδειγμα το οποίο βρίσκεται στον φάκελο data, στον πίνακα students με την παρακάτω εντολή:

**Load data local inpath '/data/students.csv' into table students**

Q1.8 You have an external table in Hive and want to store this data in more compact and efficient format. Your decision is:

1. Create a table in a new format with the CREATE TABLE statement and fill it from the external table with the INSERT INTO TABLE statement.
2. CREATE VIEW over an external table.
3. Create and fill a new table with the CREATE TABLE ... AS SELECT statement.

Select one of the options ***and explain*** your answer, **shortly**.

**In addition**, provide at least one type of format that could improve the performance on query aggregations. **E*xplain*** your answer, **shortly**.

Η καλύτερη επιλογή από άποψη απόδοσης κατά την μεταφορά των δεδομένων, είναι η δημιουργία μιας όψης από τον πίνακα (CREATE VIEW). Η δημιουργία όψης είναι πολύ πιο γρήγορη από την δημιουργία ενός νέου πραγματικού πίνακα, καθώς μπορούν να εκτελεστούν ερωτήματα στην όψη όπως σε έναν πίνακα.

* RC (Row-Columnar) File Input Format: Ο τύπος αυτός προσφέρει μεγάλης απόδοσης συμπίεση στις εγγραφές του πίνακα, καθιστώντας τον τύπο αυτό μια πολύ καλή επιλογή για συναρτήσεις που χρησιμοποιούν πολλαπλές γραμμές, όπως ερωτήματα με συναρτήσεις τύπου aggregation.
* ORC (Optimized Row Columnar)Input Format: Ο τύπος αυτός είναι μια αναβαθμισμένη έκδοση του RC τύπου, αποθηκεύοντας τα δεδομένα με έναν βελτιστοποιημένο τρόπο, προσφέροντας μεγαλύτερη απόδοση από τους υπόλοιπους τύπους, μειώνοντας το συνολικό μέγεθος των δεδομένων μέχρι και 75%.

Q1.9 Why does partitioning optimize Hive queries? Please provide a short answer.

Χωρίς partition το Hive διαβάζει όλα τα δεδομένα στην διαδρομή και εφαρμόζει το ερώτημα σε αυτά. Αυτό οδηγεί στο διάβασμα όλων των δεδομένων και συνεπώς η διαδικασία μετατρέπεται σε αργή και επίπονη για το σύστημα. Για τον λόγο αυτό, οι χρήστες πρέπει να εφαρμόζουν φιλτράρισμα σε μια συγκεκριμένη στήλη του πίνακα, σπάζοντας ουσιαστικά τον πίνακα σε μικρότερα τμήματα.

Q1.10 You join two Hive tables: A(key INT, value STRING) and B(key INT, value STRING). Тable B is small enough to be stored in RAM of a single compute node in the cluster and A is much bigger than B (A exceeds the average RAM of a cluster node). Provide an efficient query optimizing the join (INNER JOIN over A.key=B.key) between A and B, in Hive. Assume that the query returns A.value and B.value.

Όταν η Hive εκτελεί ένα ερώτημα JOIN, θα πρέπει να επιλέξει ποιος πίνακας θα γίνει streamed και ποιος θα γίνει cached. Για να επιλέξει, η Hive χρησιμοποιεί τον τελευταίο πίνακα στο ερώτημα ως streamed. Αυτό σημαίνει πως ο πρώτος πίνακας σε ένα ερώτημα με 2 πίνακες, θα γίνει cached. Θα πρέπει λοιπόν να εγγυηθούμε πως ο δεύτερος πίνακας είναι ο μεγαλύτερος από τους 2. Έτσι λοιπόν θα έπρεπε να εκτελέσουμε:

**SELECT A.value, B.value FROM A INNER JOIN over A.key=B.key**

# Part 2: Practical questions

Q2.1

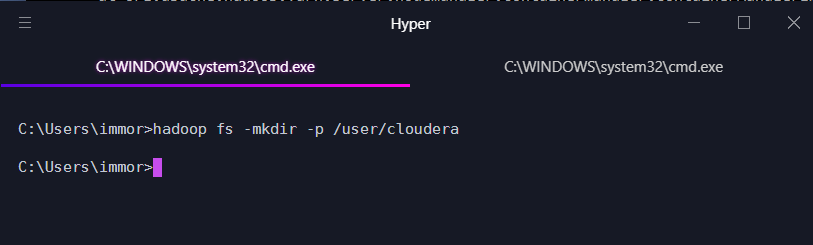
Α) Για την δημιουργία των διαδρομών που ζητούνται θα χρησιμοποιήσουμε την εντολή:

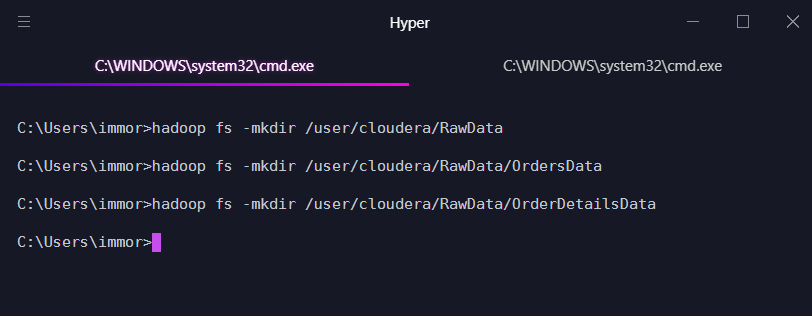
hadoop fs -mkidr <path>

ενώ εάν θέλουμε να γίνει και η δημιουργία των γονικών φακέλων του δοσμένου path θα πρέπει να κάνουμε χρήση και της επιλογής -p:

hadoop fs -mkidr -p <path>

Έτσι για την δημιουργία των ζητούμενων φακέλων, έχουμε :

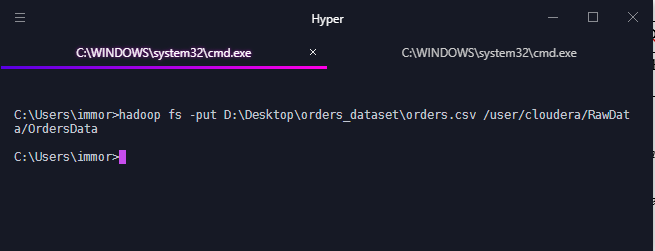




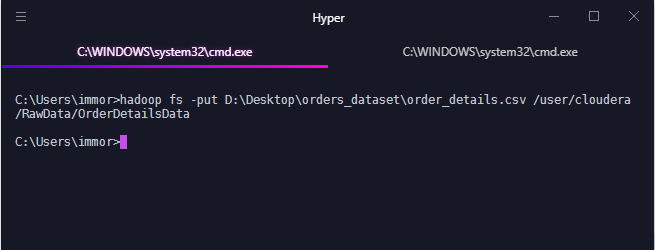
Β) Για το φόρτωμα αρχείων στους αντίστοιχους HDFS φακέλους θα χρησιμοποιήσουμε την εντολή put.

hadoop fs -put <localsrc> … <HDFS\_dest\_Path>

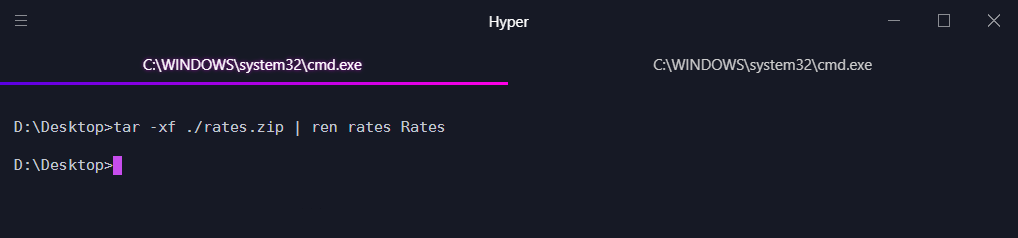
Φόρτωμα του αρχείου orders.csv στον φάκελο OrdersData.



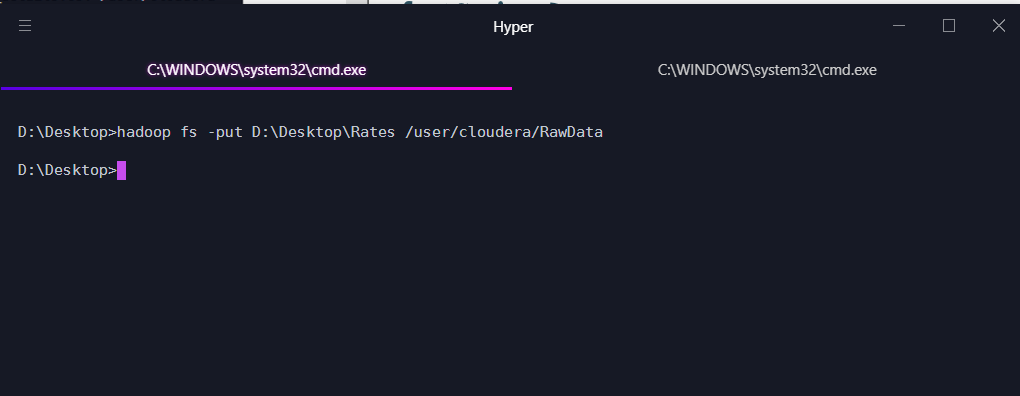
Φόρτωμα του αρχείου order\_details.csv στον φάκελο OrderDetailsData.



Στην συνέχεια κάνουμε αποσυμπίεση το zip αρχείο στον φάκελο rates και τον μετονομαζουμε σε Rates.

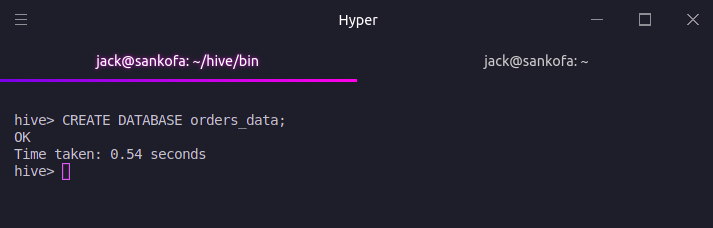


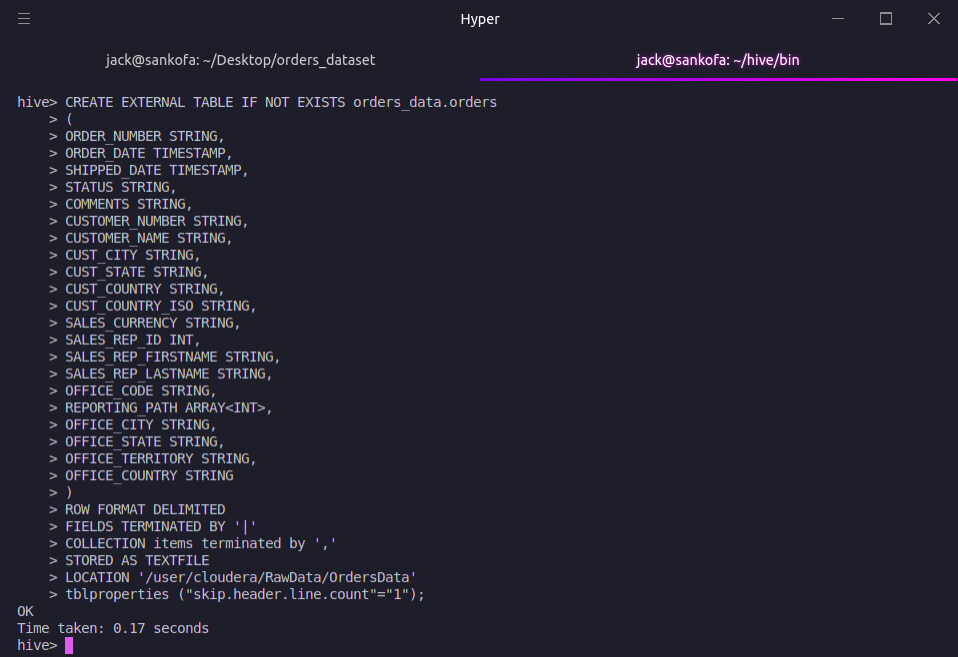
Ανεβάζουμε τον φάκελο Rates στον φάκελο RawData.



# Q2.2

Δημιουργία βάσης με όνομα orders\_data





CREATE EXTERNAL TABLE IF NOT EXISTS orders\_data.orders

(

ORDER\_NUMBER STRING,

ORDER\_DATE TIMESTAMP,

SHIPPED\_DATE TIMESTAMP,

STATUS STRING,

COMMENTS STRING,

CUSTOMER\_NUMBER STRING,

CUSTOMER\_NAME STRING,

CUST\_CITY STRING,

CUST\_STATE STRING,

CUST\_COUNTRY STRING,

CUST\_COUNTRY\_ISO STRING,

SALES\_CURRENCY STRING,

SALES\_REP\_ID INT,

SALES\_REP\_FIRSTNAME STRING,

SALES\_REP\_LASTNAME STRING,

OFFICE\_CODE STRING,

REPORTING\_PATH ARRAY<INT>,

OFFICE\_CITY STRING,

OFFICE\_STATE STRING,

OFFICE\_TERRITORY STRING,

OFFICE\_COUNTRY STRING

)

ROW FORMAT DELIMITED

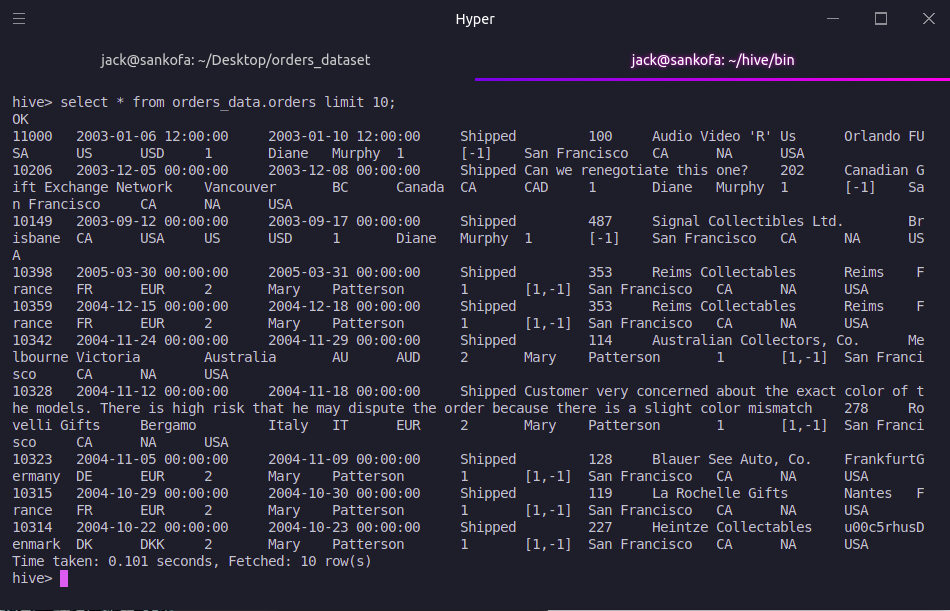
FIELDS TERMINATED BY '|'

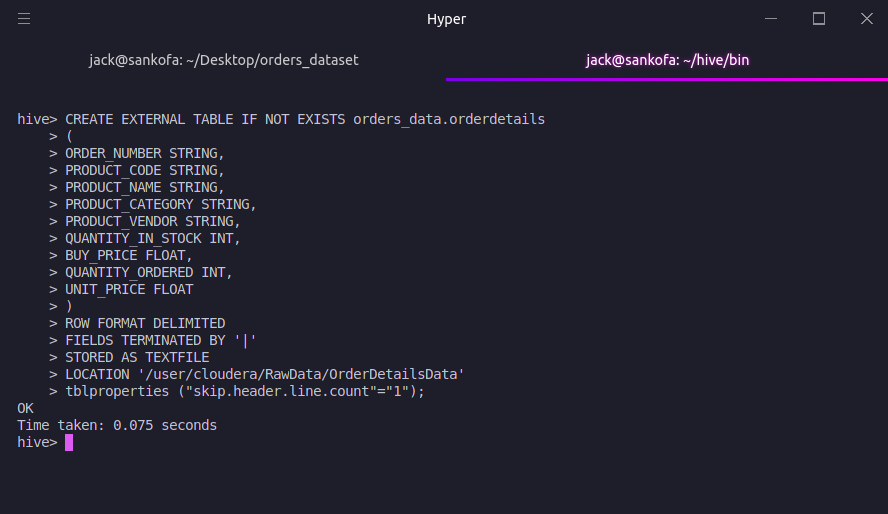
COLLECTION items terminated by ','

STORED AS TEXTFILE

LOCATION '/user/cloudera/RawData/OrdersData'

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





CREATE EXTERNAL TABLE IF NOT EXISTS orders\_data.orderdetails

(

ORDER\_NUMBER STRING,

PRODUCT\_CODE STRING,

PRODUCT\_NAME STRING,

PRODUCT\_CATEGORY STRING,

PRODUCT\_VENDOR STRING,

QUANTITY\_IN\_STOCK INT,

BUY\_PRICE FLOAT,

QUANTITY\_ORDERED INT,

UNIT\_PRICE FLOAT

)

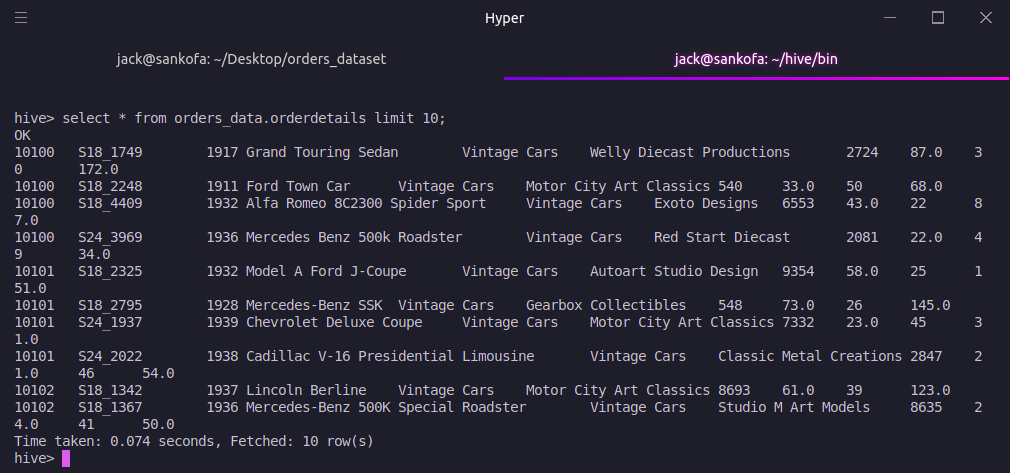
ROW FORMAT DELIMITED

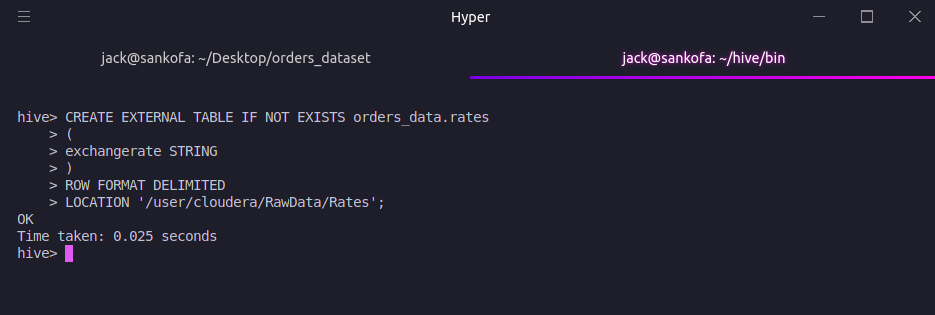
FIELDS TERMINATED BY '|'

STORED AS TEXTFILE

LOCATION '/user/cloudera/RawData/OrderDetailsData'

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





CREATE EXTERNAL TABLE IF NOT EXISTS orders\_data.rates

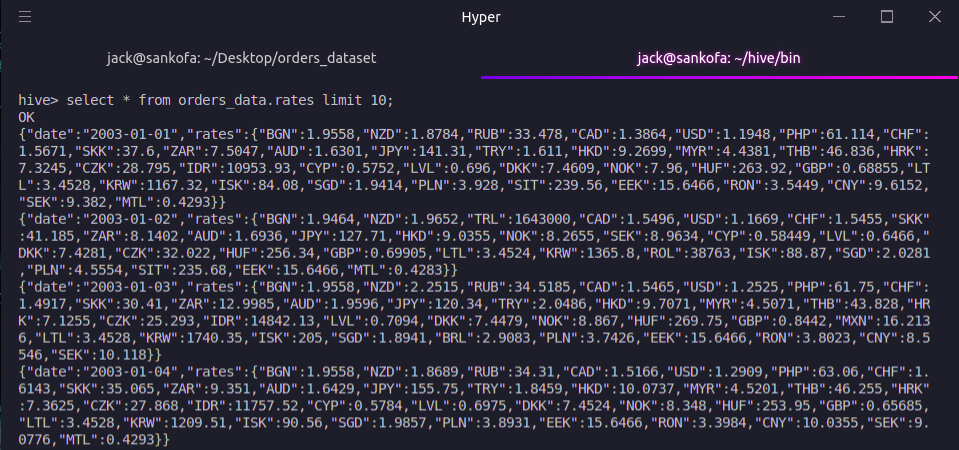
(

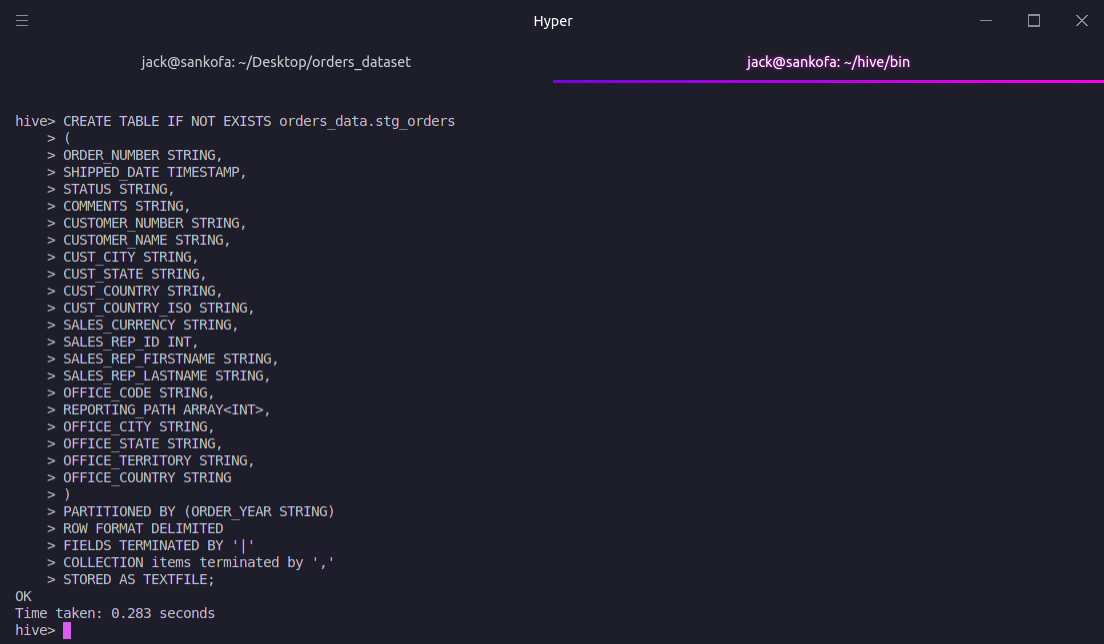
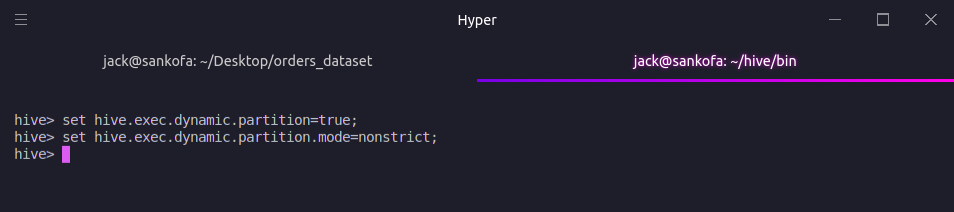
exchangerate STRING

)

ROW FORMAT DELIMITED

LOCATION '/user/cloudera/RawData/Rates';





CREATE EXTERNAL TABLE IF NOT EXISTS orders\_data.stg\_orders

(

ORDER\_NUMBER STRING,

SHIPPED\_DATE TIMESTAMP,

STATUS STRING,

COMMENTS STRING,

CUSTOMER\_NUMBER STRING,

CUSTOMER\_NAME STRING,

CUST\_CITY STRING,

CUST\_STATE STRING,

CUST\_COUNTRY STRING,

CUST\_COUNTRY\_ISO STRING,

SALES\_CURRENCY STRING,

SALES\_REP\_ID INT,

SALES\_REP\_FIRSTNAME STRING,

SALES\_REP\_LASTNAME STRING,

OFFICE\_CODE STRING,

REPORTING\_PATH ARRAY<INT>,

OFFICE\_CITY STRING,

OFFICE\_STATE STRING,

OFFICE\_TERRITORY STRING,

OFFICE\_COUNTRY STRING

)

PARTITIONED BY (order\_year STRING)

ROW FORMAT DELIMITED

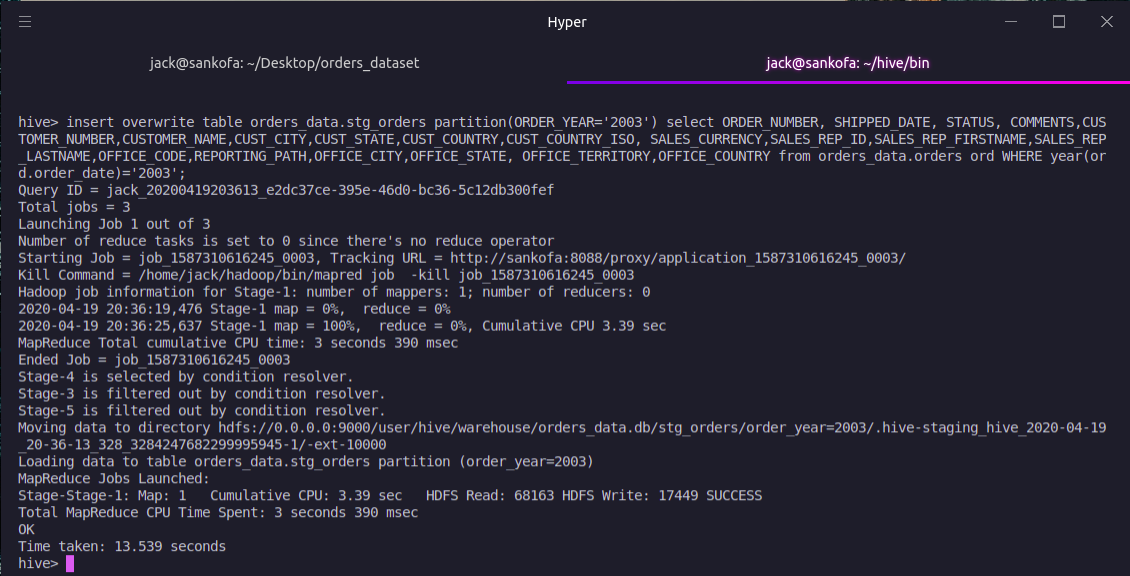
FIELDS TERMINATED BY '|'

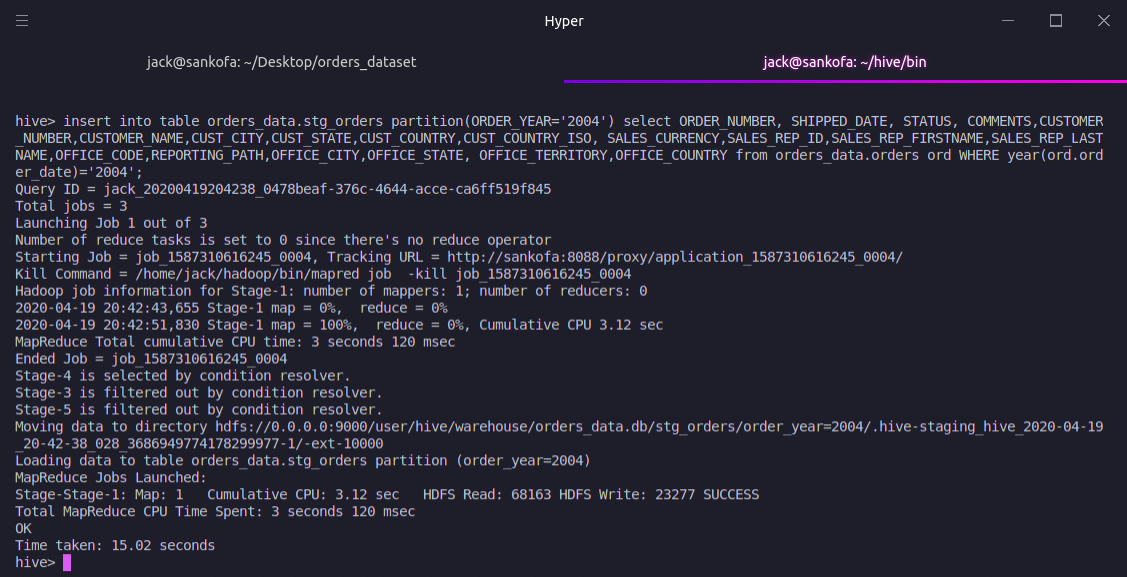
COLLECTION items terminated by ','

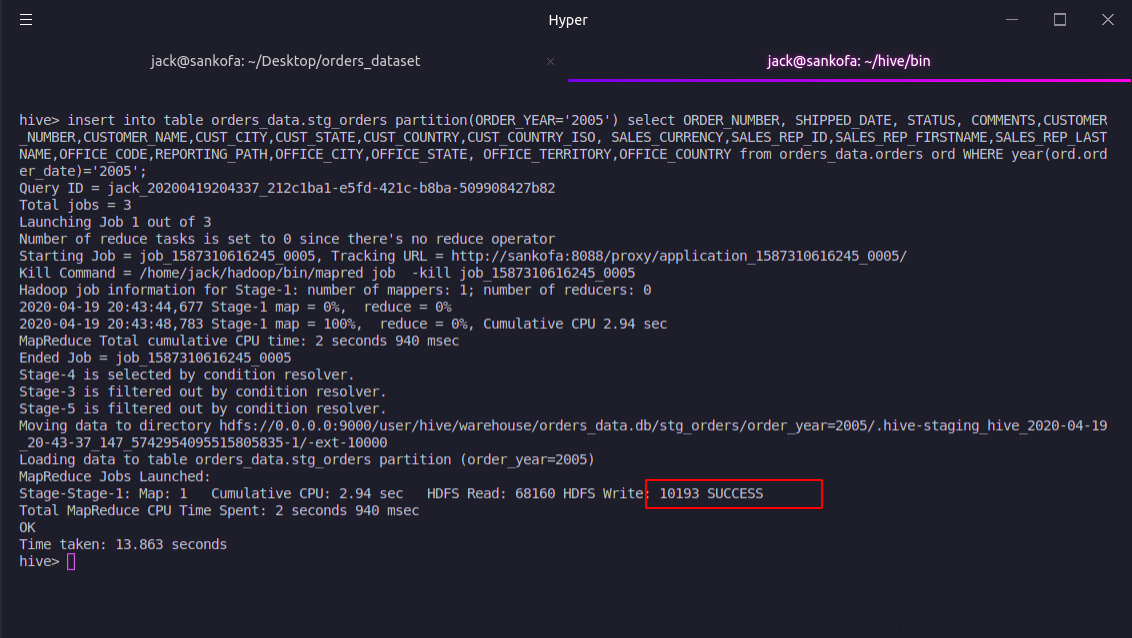
STORED AS TEXTFILE

# Q2.3

Δημιουργία του πινακα stg\_orders όπου με χρήση dynamic partitioning εισάγουμε τα στοιχεία του πίνακα με βάση την χρονιά 2003. Με την χρήση της συνάρτησης Year πήραμε την εκάστοτε χρονιά από το πεδίο order\_date.

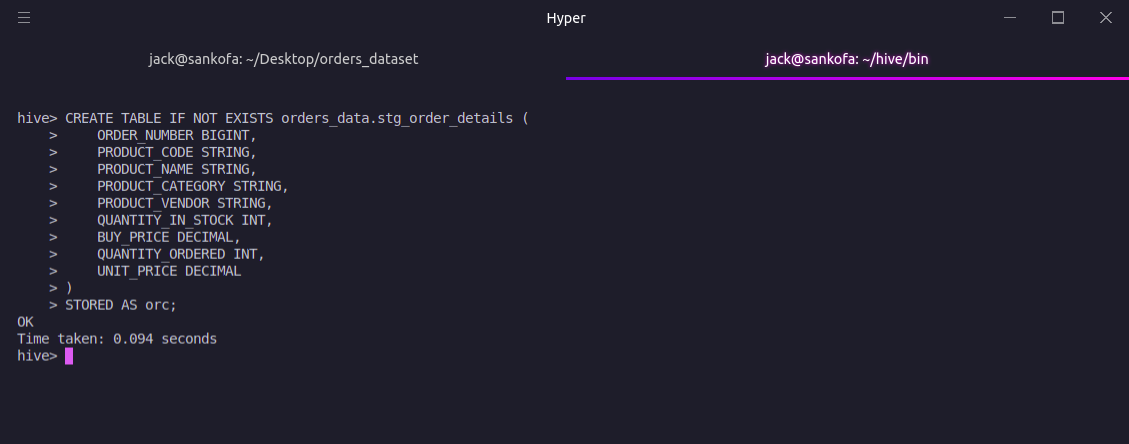


Αντίστοιχα δημιουργία των partition για τα έτη 2004 και 2005.

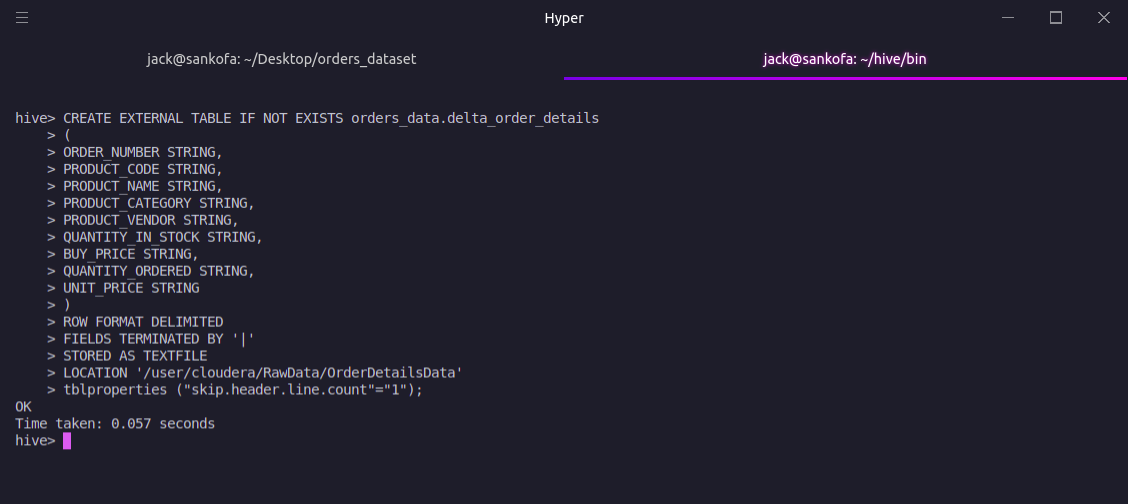


# Q2.4

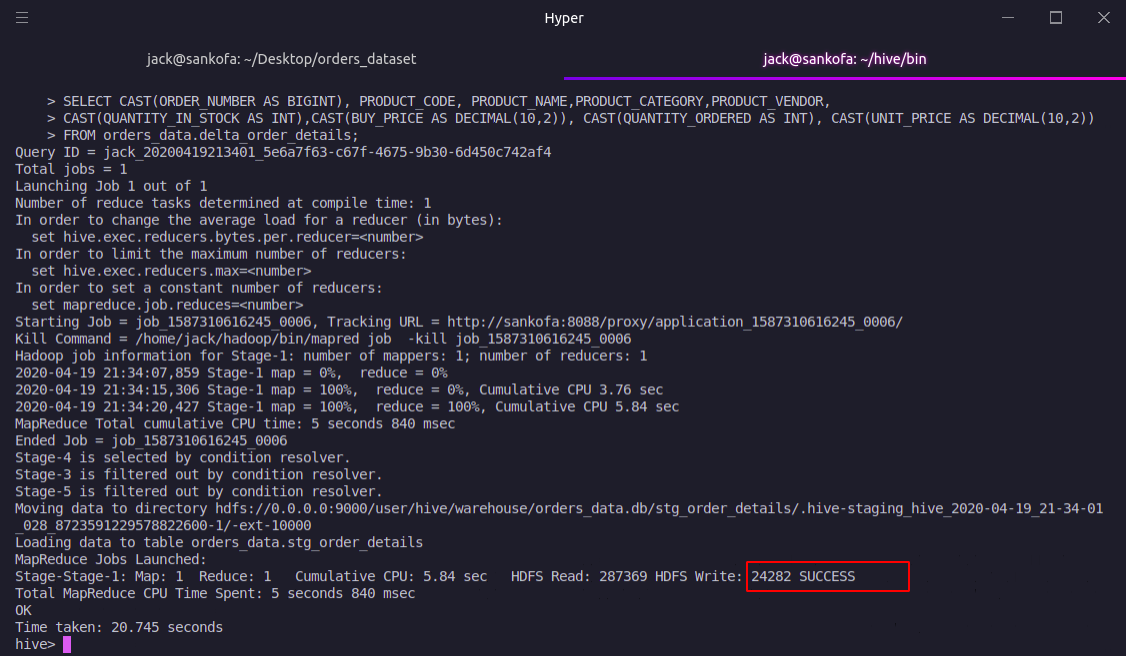
Δημιουργία του πίνακα stg\_order\_details.



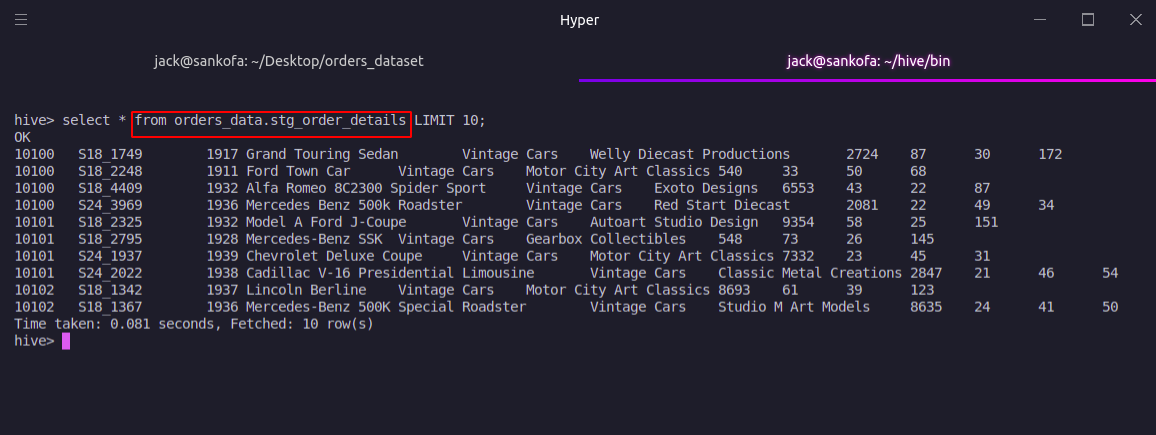
Δημιουργία ενός πίνακα delta\_order\_details με όλα τα στοιχεία να είναι string.



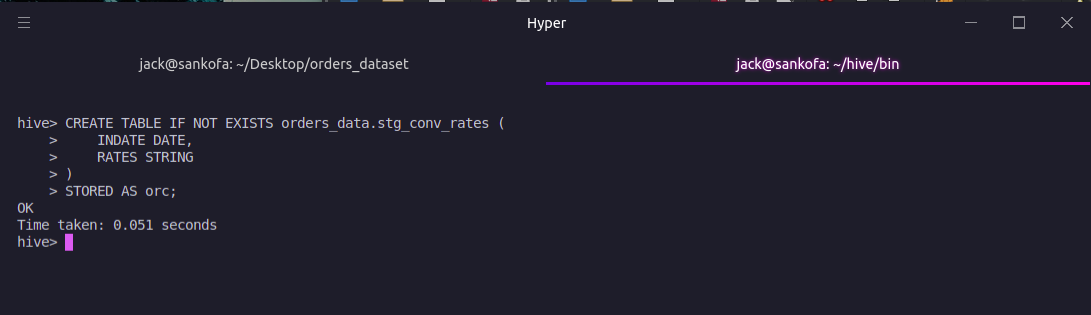
Εισαγωγή των στοιχείων στον πίνακα μετά από cast όλων τον απαραίτητων στοιχείων του πίνακα.



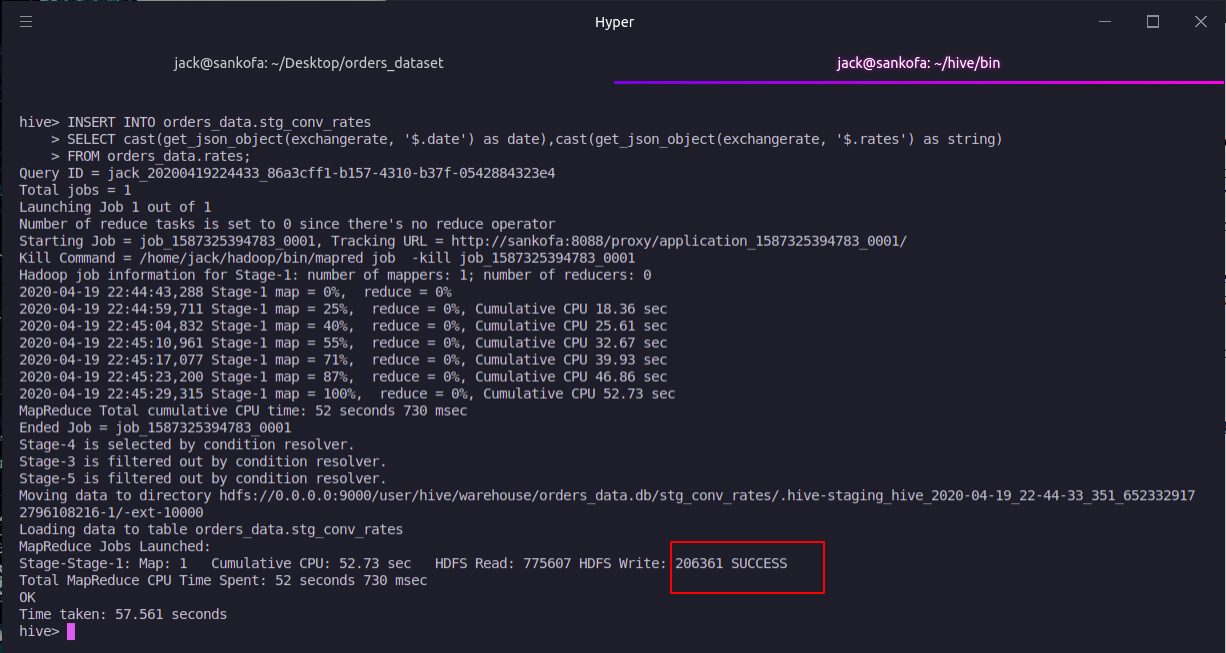
Εμφάνιση των αποτελεσμάτων από τον τελικό πίνακα.



Αντίστοιχα, δημιουργία του πίνακα stg\_conv\_rates.

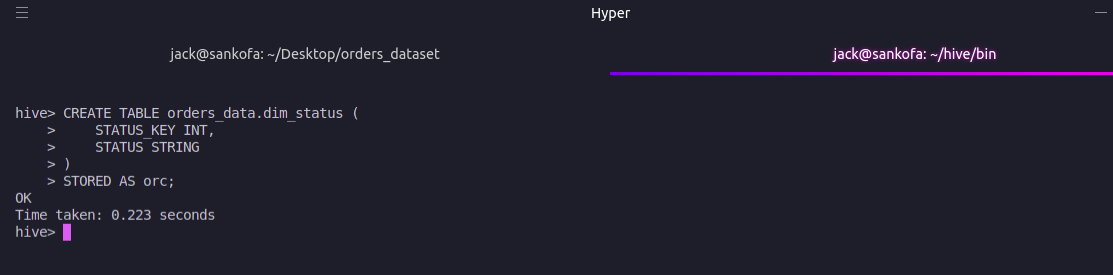


Εισαγωγή των στοιχείων στον πίνακα stg\_conv\_rates από τον πίνακα rates.

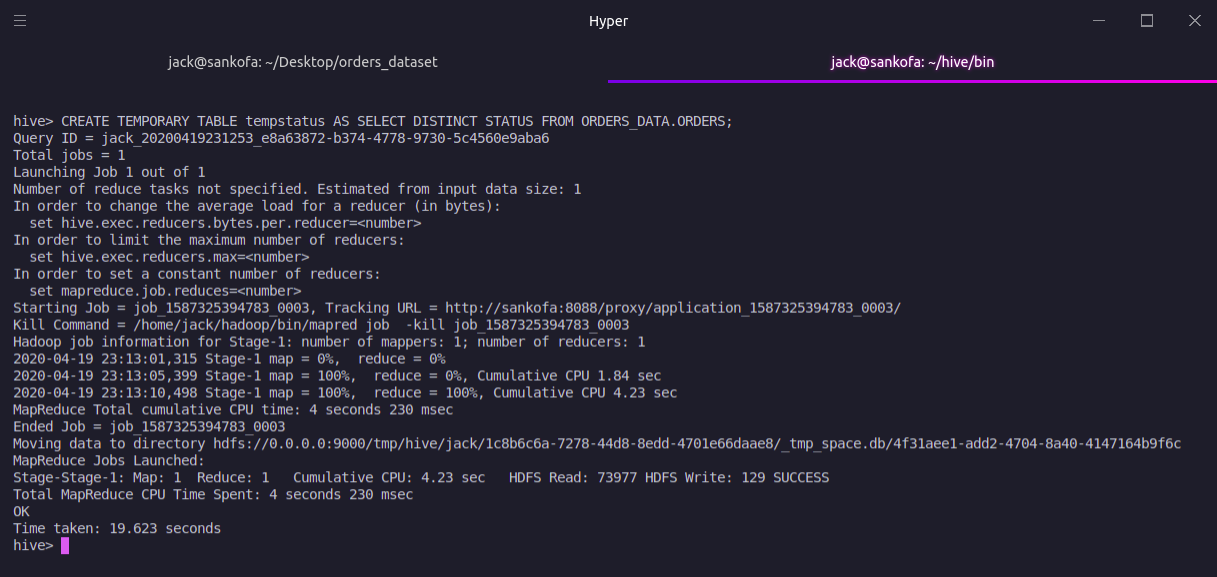


# Q2.5

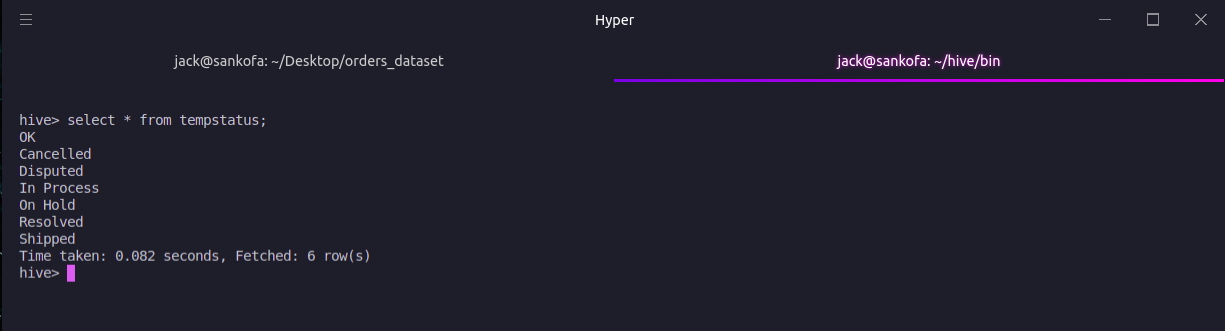
Δημιουργία πίνακα dim Status.



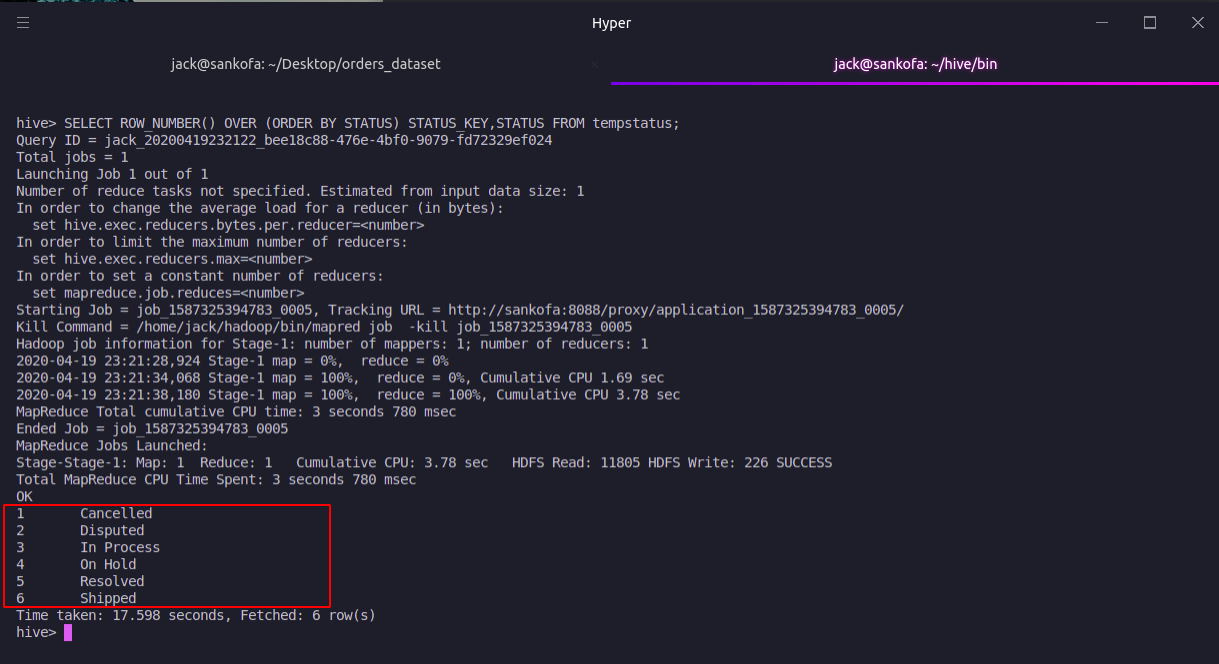
Δημιουργία προσωρινού πίνακα tempstatus και εισαγωγή των μοναδικών status από τον πίνακα orders με την χρήση του dinstinct.



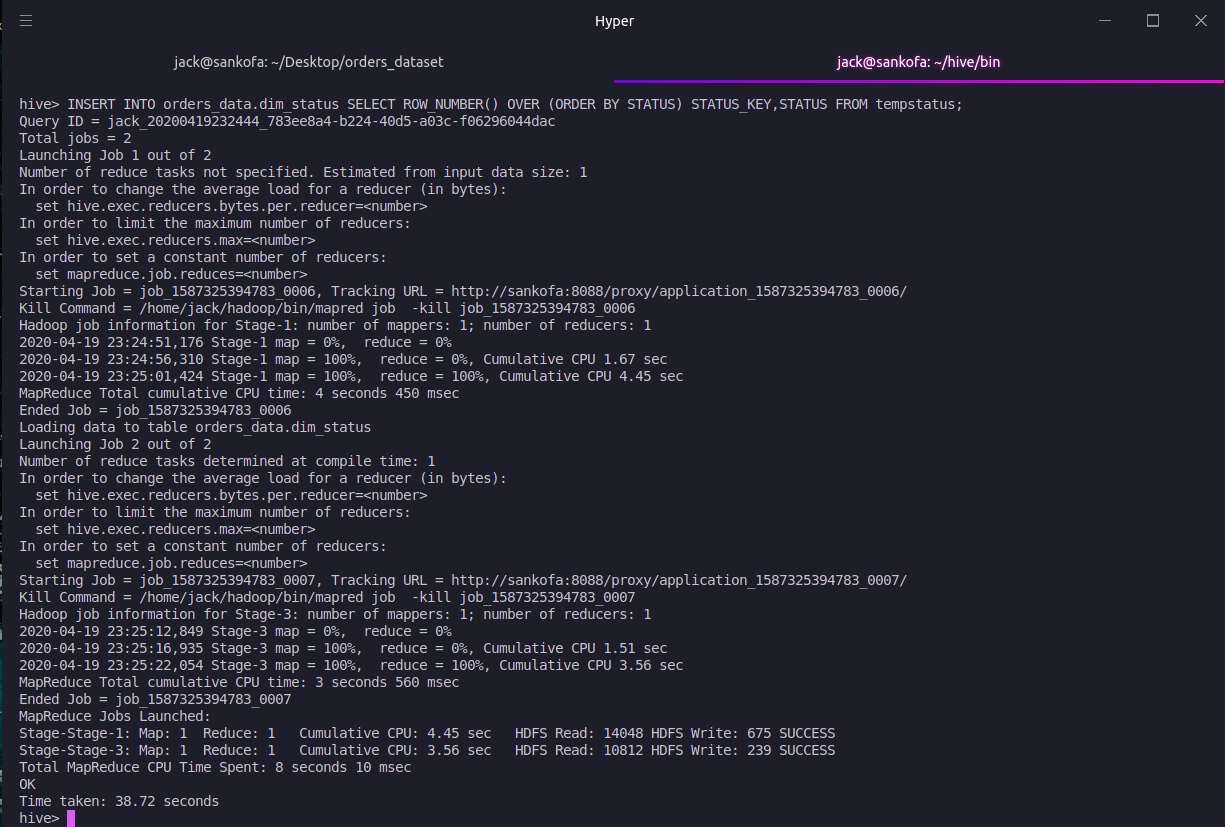
Eμφανίζουμε τα μοναδικά αποτελέσματα από το status του πίνακα Orders.



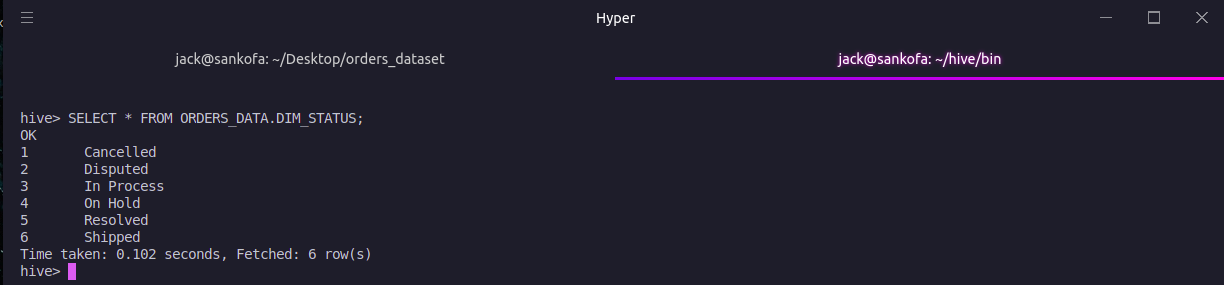
Εμφανίζουμε τα αποτελέσματα με την χρήση του ROW\_NUMBER για την εισαγωγή μοναδικών αριθμών σε κάθε μοναδικό status.



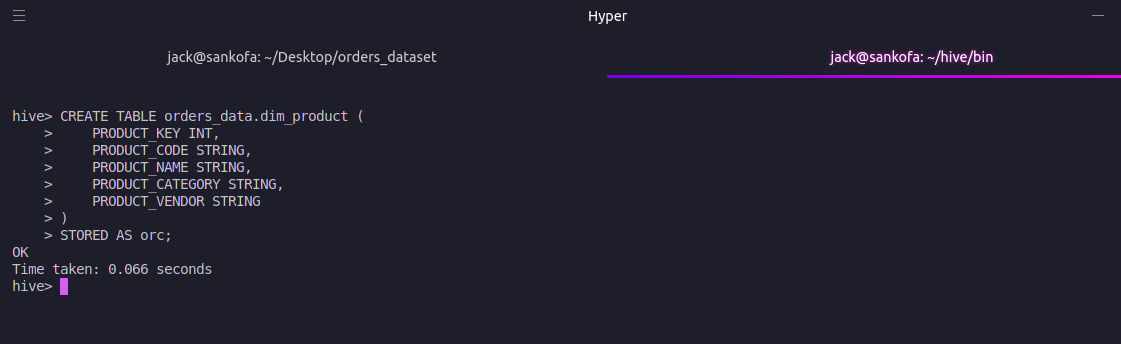
Εισαγωγή των παραπάνω στοιχείων στον πίνακα dimstatus.



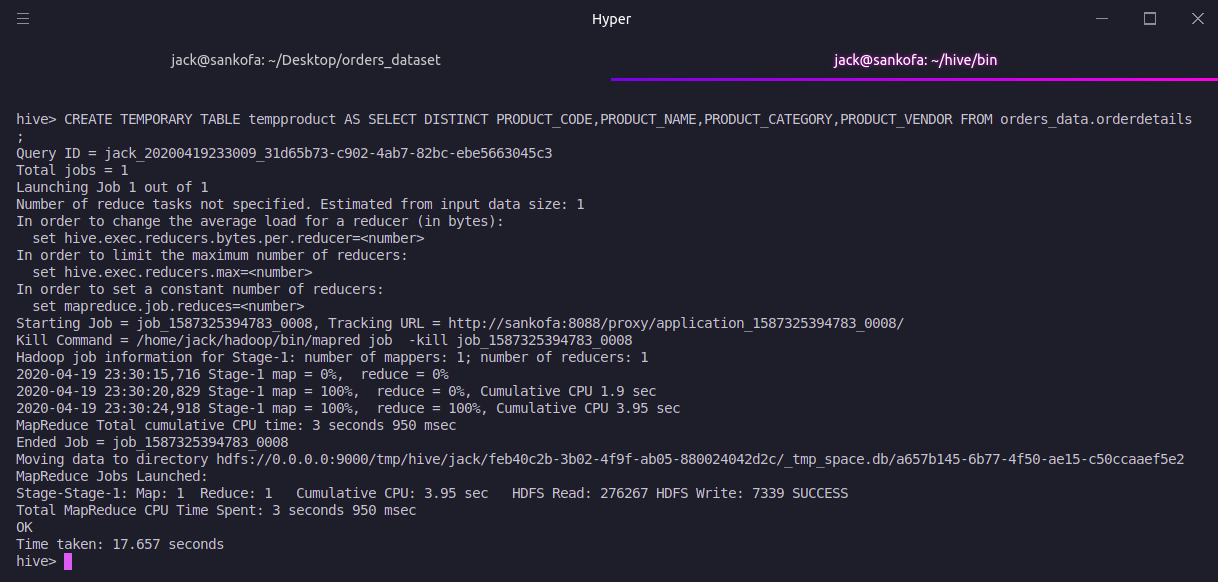
Εμφάνιση του τελικού αποτελέσματος.



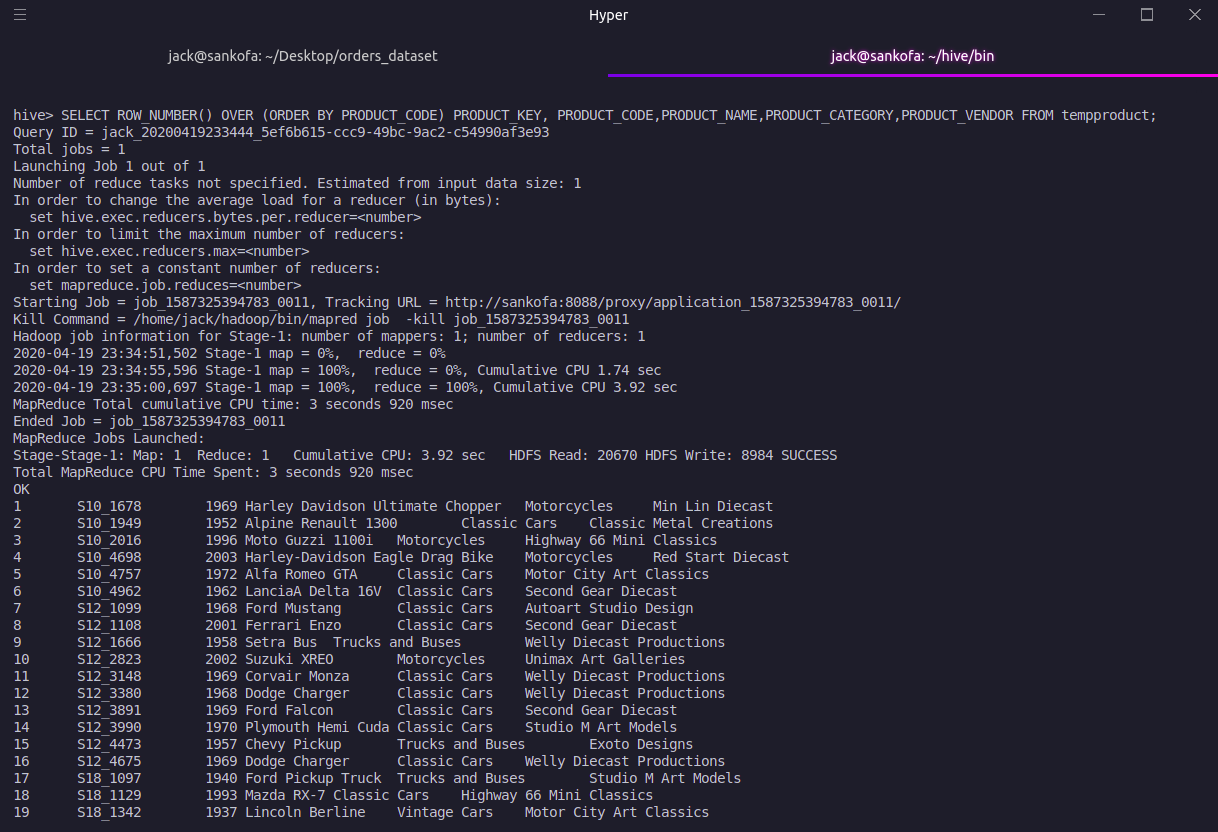
Αντίστοιχα, δημιουργία του πίνακα dim\_product.



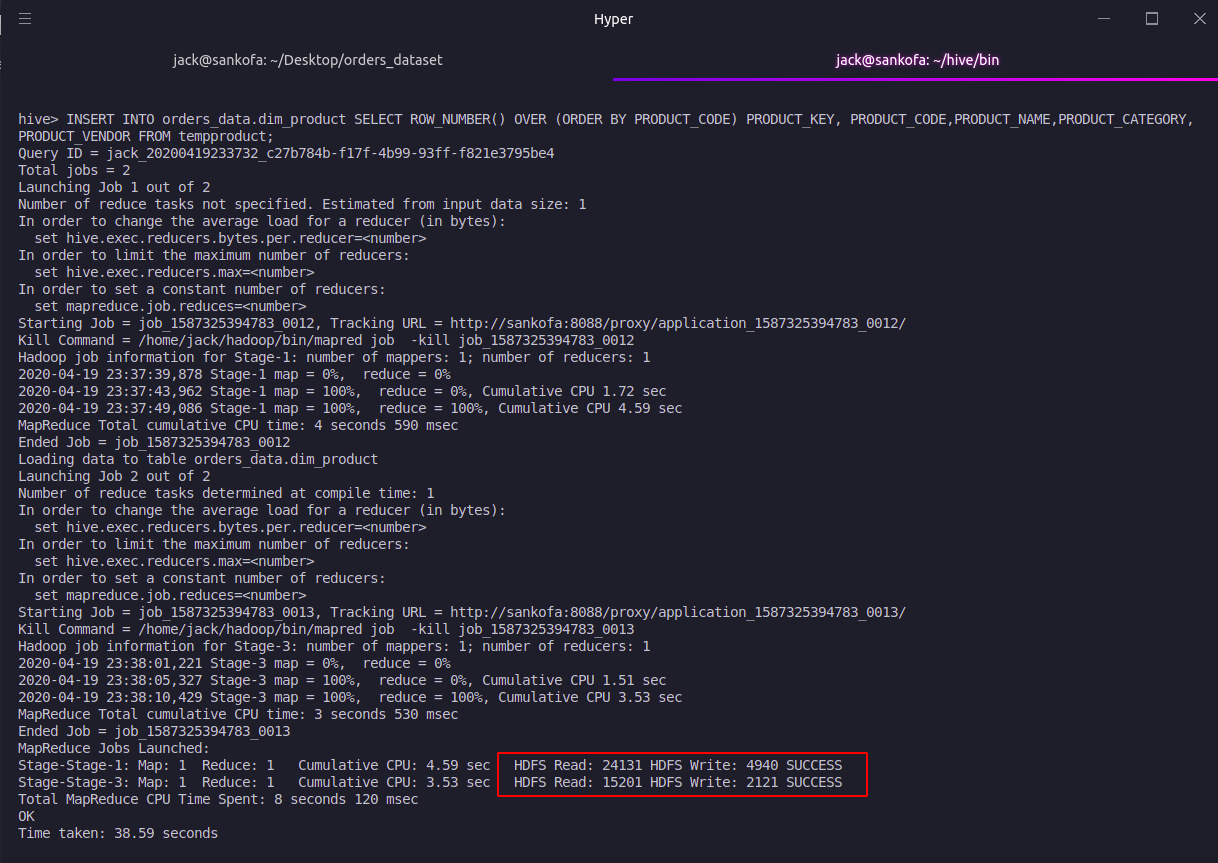
Δημιουργία ενός προσωρινού πίνακα tempproduct με την εισαγωγή των μοναδικών στοιχείων των προϊόντων από τον πίνακα orderdetails.



Εμφανίζουμε τα αποτελέσματα με την χρήση του ROW\_NUMBER για την εισαγωγή μοναδικών αριθμών σε κάθε μοναδικό προϊόν.



Εισαγωγή των στοιχείων στον πίνακα dim\_product.



Εμφάνιση των αποτελεσμάτων από τον πίνακα με την εισαγωγή πλεόν του κλειδιού για το κάθε προϊόν.

