Loading the Lookup Table

Commands to load the relevant data in the Lookup Table

Here we are going to prepare Card Lookup metrics data from Card Transactions and Member Score Hive tables using Spark SQL (PySpark) and create a staging table on top of the prepared data in hive. Later, from Hive, we transfer the lookup data from staging to final Hive-HBase table created (Refer table definition in CreateNoSQL.pdf document).

1. PySpark code for generating lookup data and store the data in Hive as a staging layer:

```
# Import necessary PySpark libraries
import pyspark
from pyspark.sql import SparkSession

# Create a Spark Session with Hive support
spark = SparkSession \
        .builder \
        .appName('Credit Card Lookup data preparation') \
        .enableHiveSupport() \
        .getOrCreate()

# Set log level to ERROR
spark.sparkContext.setLogLevel('ERROR')

# Prepare the card lookup data from Card Transactions and Member Score
Hive tables using Spark SQL and store the results in a temporary view
spark.sql('''
WITH transaction_details AS
(
    SELECT
        card_id,
        member_id,
        amount,
        postcode,
        transaction_dt,
        RANK() OVER(PARTITION BY card_id ORDER BY transaction_dt DESC)
AS txn_rank
    FROM
        cred_financials_data.card_transactions
)
SELECT
    card_id,
    ROUND(AVG(amount) + 3 *MAX(std_dev), 0) AS ucl,
    FIRST_VALUE(postcode) OVER(PARTITION BY card_id ORDER BY (SELECT
1)) AS postcode,
    MAX(transaction_dt) AS transaction_dt,
    credit_score
FROM
    (
        SELECT
            txn.card_id,
            txn.amount,
            FIRST_VALUE(txn.postcode) OVER(PARTITION BY card_id ORDER BY
txn.txn_rank) AS postcode,
            txn.transaction_dt,
            mem.score as credit_score,
            ROUND(STDDEV(txn.amount) OVER(PARTITION BY card_id ORDER BY
(SELECT 1)), 0) AS std_dev
        FROM
            transaction_details txn
            INNER JOIN cred_financials_data.member_score mem
                ON txn.member_id = mem.member_id
        WHERE
            txn.txn_rank <= 10
    ) a
GROUP BY
    card_id,
    postcode,
    credit_score
''').createOrReplaceTempView('card_lookup_tmp')


# Create a staging table in Hive and load the data from temporary view
spark.sql('CREATE TABLE cred_financials_data.card_lookup_stg AS SELECT
* FROM card_lookup_tmp')

# Drop temporary view to release the memory
spark.catalog.dropTempView('card_lookup_tmp')

spark.stop()
```

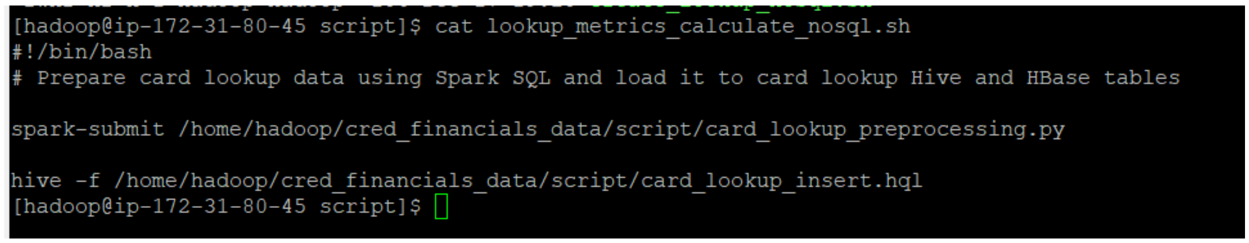
2. Insert the lookup data from staging to final Hive-HBase table and later drop staging table.

```
USE cred_financials_data;
```

-- Insert the card lookup data from staging (which got prepared using Spark SQL) to bucketing table
INSERT OVERWRITE TABLE card_lookup SELECT * FROM card_lookup_stg;

-- Drop staging table
DROP TABLE card_lookup_stg;

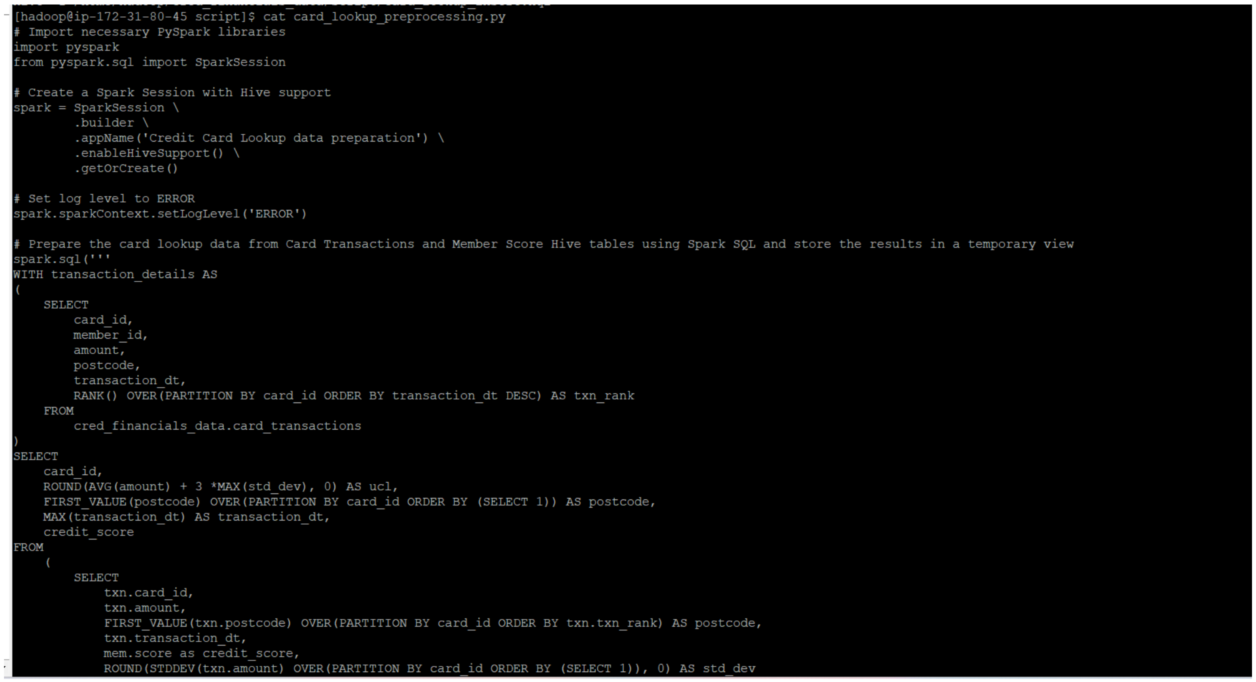
All the above steps are wrapped into a single shell script (trigger spark-submit command by passing PySpark code **– card_lookup_preprocessing.py** and call **card_lookup_insert.hql** script to insert the lookup data into final hive-hbase table) with name lookup_metrics_calculate_nosql.sh placed in the path:
/home/hadoop/cred_financials_data/script/lookup_metrics_calculate_nosql.sh



```
[hadoop@ip-172-31-80-45 script]$ cat lookup_metrics_calculate_nosql.sh
#!/bin/bash
# Prepare card lookup data using Spark SQL and load it to card lookup Hive and HBase tables

spark-submit /home/hadoop/cred_financials_data/script/card_lookup_preprocessing.py

hive -f /home/hadoop/cred_financials_data/script/card_lookup_insert.hql
[hadoop@ip-172-31-80-45 script]$ 
```

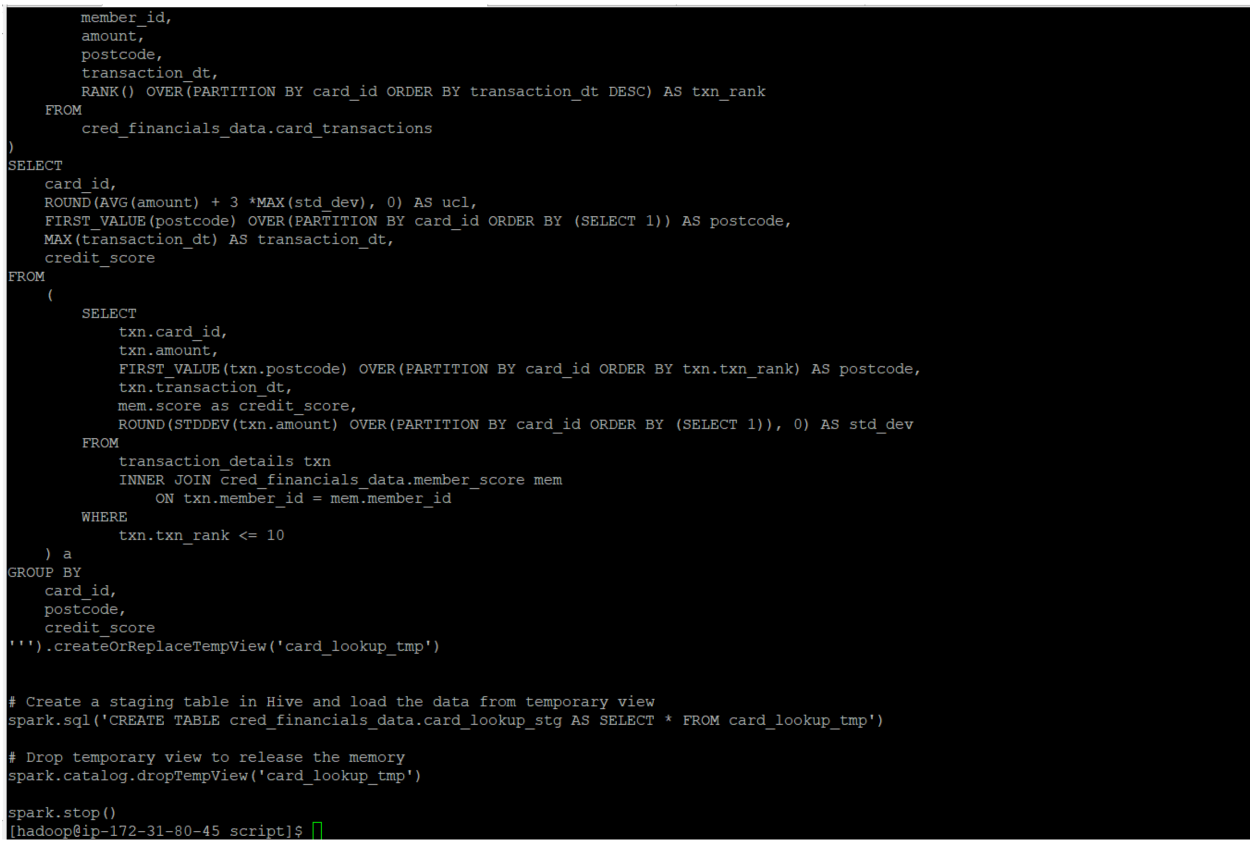


```
[hadoop@ip-172-31-80-45 script]$ cat card_lookup_preprocessing.py
# Import necessary PySpark libraries
import pyspark
from pyspark.sql import SparkSession

# Create a Spark Session with Hive support
spark = SparkSession \
        .builder \
        .appName('Credit Card Lookup data preparation') \
        .enableHiveSupport() \
        .getOrCreate()

# Set log level to ERROR
spark.sparkContext.setLogLevel('ERROR')

# Prepare the card lookup data from Card Transactions and Member Score Hive tables using Spark SQL and store the results in a temporary view
spark.sql('''
WITH transaction_details AS
(
    SELECT
        card_id,
        member_id,
        amount,
        postcode,
        transaction_dt,
        RANK() OVER(PARTITION BY card_id ORDER BY transaction_dt DESC) AS txn_rank
    FROM
        cred_financials_data.card_transactions
)
SELECT
    card_id,
    ROUND(AVG(amount) + 3 *MAX(std_dev), 0) AS ucl,
    FIRST_VALUE(postcode) OVER(PARTITION BY card_id ORDER BY (SELECT 1)) AS postcode,
    MAX(transaction_dt) AS transaction_dt,
    credit_score
FROM
    (
        SELECT
            txn.card_id,
            txn.amount,
            FIRST_VALUE(txn.postcode) OVER(PARTITION BY card_id ORDER BY txn.txn_rank) AS postcode,
            txn.transaction_dt,
            mem.score as credit_score,
            ROUND(STDDEV(txn.amount) OVER(PARTITION BY card_id ORDER BY (SELECT 1)), 0) AS std_dev
```

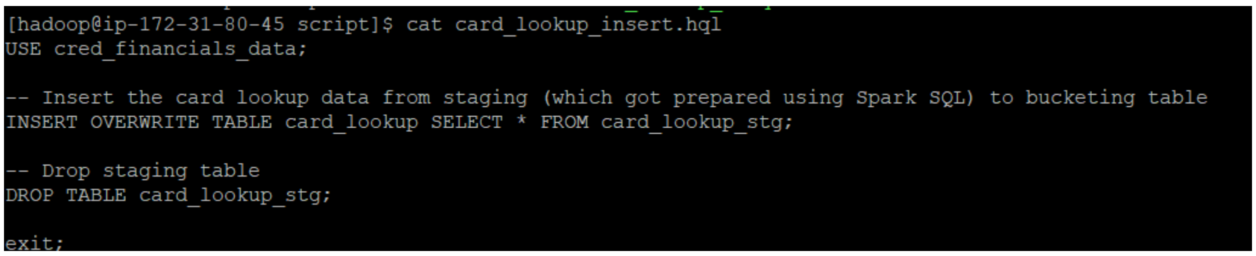


```
        member_id,
        amount,
        postcode,
        transaction_dt,
        RANK() OVER(PARTITION BY card_id ORDER BY transaction_dt DESC) AS txn_rank
    FROM
        cred_financials_data.card_transactions
)
SELECT
    card_id,
    ROUND(AVG(amount) + 3 *MAX(std_dev), 0) AS ucl,
    FIRST_VALUE(postcode) OVER(PARTITION BY card_id ORDER BY (SELECT 1)) AS postcode,
    MAX(transaction_dt) AS transaction_dt,
    credit_score
FROM
    (
        SELECT
            txn.card_id,
            txn.amount,
            FIRST_VALUE(txn.postcode) OVER(PARTITION BY card_id ORDER BY txn.txn_rank) AS postcode,
            txn.transaction_dt,
            mem.score as credit_score,
            ROUND(STDDEV(txn.amount) OVER(PARTITION BY card_id ORDER BY (SELECT 1)), 0) AS std_dev
        FROM
            transaction_details txn
            INNER JOIN cred_financials_data.member_score mem
                ON txn.member_id = mem.member_id
        WHERE
            txn.txn_rank <= 10
    ) a
GROUP BY
    card_id,
    postcode,
    credit_score
''').createOrReplaceTempView('card_lookup_tmp')


# Create a staging table in Hive and load the data from temporary view
spark.sql('CREATE TABLE cred_financials_data.card_lookup_stg AS SELECT * FROM card_lookup_tmp')

# Drop temporary view to release the memory
spark.catalog.dropTempView('card_lookup_tmp')

spark.stop()
[hadoop@ip-172-31-80-45 script]$
```



```
[hadoop@ip-172-31-80-45 script]$ cat card_lookup_insert.hql
USE cred_financials_data;

-- Insert the card lookup data from staging (which got prepared using Spark SQL) to bucketing table
INSERT OVERWRITE TABLE card_lookup SELECT * FROM card_lookup_stg;

-- Drop staging table
DROP TABLE card_lookup_stg;

exit;
```

Execute the shell script with the below command:

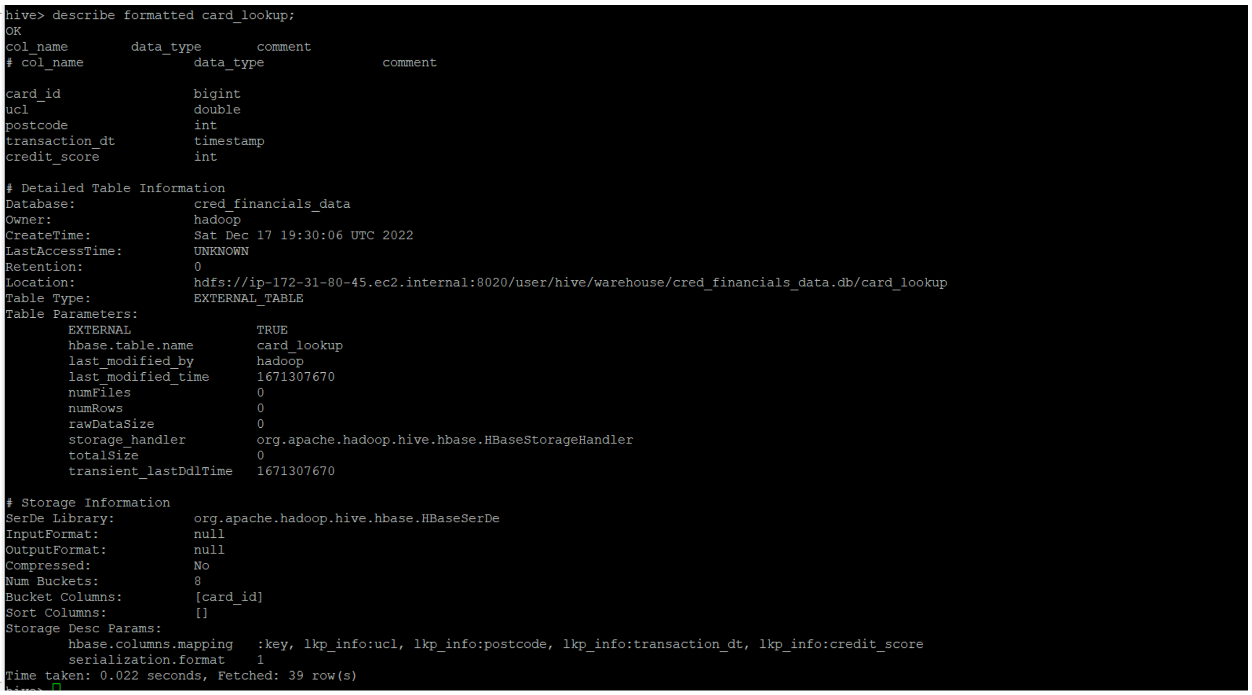
/home/hadoop/cred_financials_data/script/lookup_metrics_calculate_nosql.sh

Command to see the table created and it's content

Hive: -

use cred_financials_data;

describe formatted card_lookup;



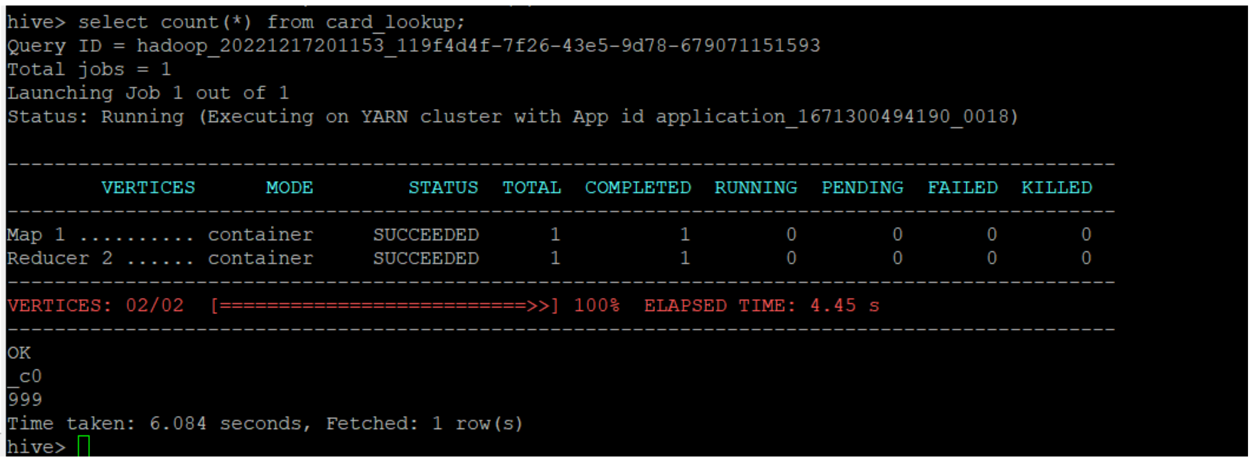
```
hive> describe formatted card_lookup;
OK
col_name        data_type       comment
# col_name              data_type               comment

card_id                 bigint
ucl                     double
postcode                int
transaction_dt          timestamp
credit_score            int

# Detailed Table Information
Database:               cred_financials_data
Owner:                  hadoop
CreateTime:             Sat Dec 17 19:30:06 UTC 2022
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://ip-172-31-80-45.ec2.internal:8020/user/hive/warehouse/cred_financials_data.db/card_lookup
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        hbase.table.name        card_lookup
        last_modified_by        hadoop
        last_modified_time      1671307670
        numFiles                0
        numRows                 0
        rawDataSize             0
        storage_handler         org.apache.hadoop.hive.hbase.HBaseStorageHandler
        totalSize               0
        transient_lastDdlTime   1671307670

# Storage Information
SerDe Library:          org.apache.hadoop.hive.hbase.HBaseSerDe
InputFormat:            null
OutputFormat:           null
Compressed:             No
Num Buckets:            8
Bucket Columns:         [card_id]
Sort Columns:           []
Storage Desc Params:
        hbase.columns.mapping   :key, lkp_info:ucl, lkp_info:postcode, lkp_info:transaction_dt, lkp_info:credit_score
        serialization.format    1
Time taken: 0.022 seconds, Fetched: 39 row(s)
```

select count(*) from card_lookup;



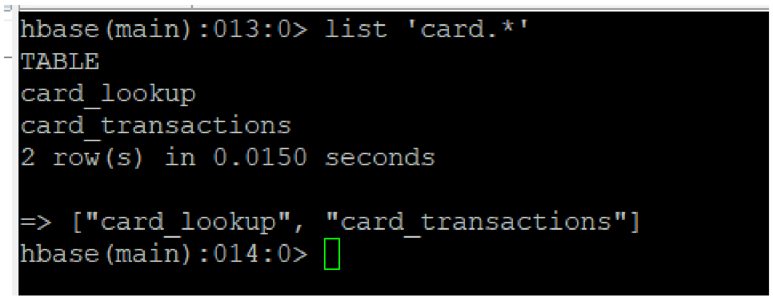
```
hive> select count(*) from card_lookup;
Query ID = hadoop_20221217201153_119f4d4f-7f26-43e5-9d78-679071151593
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1671300494190_0018)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 4.45 s
----------------------------------------------------------------------------------------------
OK
_c0
999
Time taken: 6.084 seconds, Fetched: 1 row(s)
hive>
```

Returned 999 records which matches with the source MySQL tables.

<u>**HBase: -**</u>

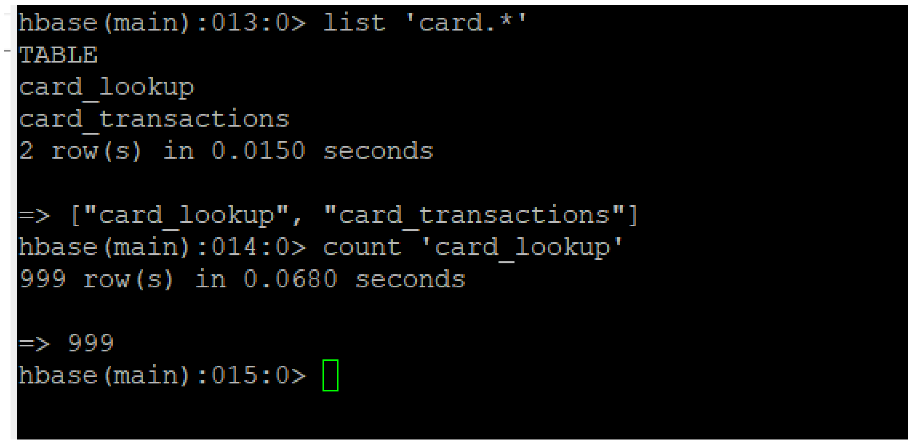
list 'card.*'



```
hbase(main):013:0> list 'card.*'
TABLE
card_lookup
card_transactions
2 row(s) in 0.0150 seconds

=> ["card_lookup", "card_transactions"]
hbase(main):014:0>
```

count ‘card_lookup’



```
hbase(main):013:0> list 'card.*'
TABLE
card_lookup
card_transactions
2 row(s) in 0.0150 seconds

=> ["card_lookup", "card_transactions"]
hbase(main):014:0> count 'card_lookup'
999 row(s) in 0.0680 seconds

=> 999
hbase(main):015:0>
```