AWS:

To load-

PIG SCRIPTS

REGISTER /usr/lib/pig/piggybank.jar;

category= LOAD '[s3://capstone.yelp/input/yelp\_academic\_dataset\_business.csv](https://s3-us-west-1.amazonaws.com/capstone.yelp/input/businesscategories.csv)' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'YES\_MULTILINE');

businessCols = FOREACH category GENERATE $15,$23,$36,$61,$38,$58,$66,$69,$20; [ordering is (business\_id,name,review\_count,stars,state,city,latitude,longitude,categories)]

STORE businessCols INTO 's3://capstone.yelp/output/businessCategoryRecords.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'YES\_MULTILINE');

reviews= LOAD '[s3://capstone.yelp/input/yelp\_academic\_dataset\_review.csv](https://s3-us-west-1.amazonaws.com/capstone.yelp/input/businesscategories.csv)' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'YES\_MULTILINE');

review = FOREACH reviews GENERATE $0, $4, $6, $7;

STORE review INTO 's3://capstone.yelp/output/reviewRecords.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'YES\_MULTILINE');

Hive Queries

CREATE EXTERNAL TABLE IF NOT EXISTS review(user\_id String, business\_id String, review\_stars Int, date String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION 's3://capstone.yelp/output/reviewRecords.csv/';

CREATE TABLE reviewSeason (user\_id String, business\_id String, review\_stars Int, date String, season String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

INSERT INTO TABLE reviewSeason SELECT user\_id, business\_id, review\_stars, date, "Spring" from review where date like '%-01-%' OR date like '%-02-%' OR date like '%-03-%';

INSERT INTO TABLE reviewSeason SELECT user\_id, business\_id, review\_stars, date, "Summer" from review where date like '%-06-%' OR date like '%-05-%' OR date like '%-04-%';

INSERT INTO TABLE reviewSeason SELECT user\_id, business\_id, review\_stars, date, "Fall" from review where date like '%-07-%' OR date like '%-08-%' OR date like '%-09-%';

INSERT INTO TABLE reviewSeason SELECT user\_id, business\_id, review\_stars, date, "Winter" from review where date like '%-12-%' OR date like '%-11-%' OR date like '%-10-%';

CREATE TABLE businessCategory(business\_id String,name String,review\_count String,stars String,state String,city String,latitude String,longitude String,categories String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION 's3://capstone.yelp/output/businessCategoryRecords.csv/';

CREATE TABLE businessCategorySeason(business\_id String, name String, review\_count String, business\_stars String, review\_stars String, state String, city String ,latitude String, longitude String, season String, categories String) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

INSERT INTO TABLE businessCategorySeason SELECT r.business\_id String, b.name String, review\_count String, b.stars String, r.review\_stars String, b.state String, b.city String, b.latitude String, b.longitude String, r.season String, b.categories String from reviewSeason r join businessCategory b on (r.business\_id = b.business\_id);

To store Hive Data into S3 bucket:

CREATE EXTERNAL TABLE businessCategorySeasonHive (business\_id String, name String, review\_count String, business\_stars String, review\_stars String, state String, city String , latitude String, longitude String,season String , categories String) row format delimited fields terminated by '\t' lines terminated by '\n' STORED AS TEXTFILE LOCATION 's3n://capstone.yelp/output/businessCategorySeasonRecords.csv/';

INSERT OVERWRITE DIRECTORY 's3n://capstone.yelp/output/businessCategorySeasonRecords.csv/' select \* from businessCategorySeason;

Spark:

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

// Import statement to implicitly convert an RDD to a DataFrame

import sqlContext.implicits.\_

// Create a custom class to represent the Customer

case class yelpcategory(business\_id: String, name: String, review\_count: String, business\_stars: String, review\_stars: String, state: String, city: String ,latitude: String, longitude: String, season: String, categories: String)

// Create a DataFrame of Customer objects from the dataset text file.

val yelpCategory = sc.textFile("s3n://capstone.yelp/output/businessCategorySeasonRecords1.csv/").map(\_.split(“,”)).map(p => yelpcategory(p(0), p(1), p(2), p(3), p(4), p(5), p(6), p(7), p(8), p(9), p(10))).toDF()

yelpCategory.registerTempTable("yelpcategory")

yelpCategory.show()

yelpCategory.printSchema()

QUERIES

Selecting business, count() group by season.

SELECT season, COUNT(business\_id) FROM businessCategorySeason.csv GROUP BY season;

yelpCategory.groupBy("season", "name").count().show()

Select count of business for group by on states.

SELECT states, COUNT(business\_id) FROM businessCategorySeason.csv GROUP BY states;

yelpCategory.groupBy("state", "name").count().show()

Select count of business group by categories and season.

SELECT season, categories, COUNT(business\_id) FROM businessCategorySeason.csv GROUP BY season, categories;

yelpCategory.groupBy("season", "categories").count().show()

10 best things to do.

SELECT category, count(categories) as 'Hot Choices' FROM businessCategorySeason.csv GROUP BY categories, Order BY count(categories) desc, limit 10;

yelpCategory.groupBy("categories").count().orderBy(desc("count")).show(10)

10 best places to be.

SELECT name, count(business\_id) as 'Hot Places' FROM businessCategorySeason.csv GROUP BY business\_id, Order BY count(business\_id)desc, limit 10;

yelpCategory.groupBy("name").count().orderBy(desc("count")).show(10)

Customer's Choice

SELECT name as business\_name, stars, category FROM businessCategorySeason.csv GROUP BY business\_name, stars, category ORDER by stars DESC, LIMIT 10;

yelpCategory.groupBy("name", "business\_stars", "categories").count().orderBy(desc("business\_stars")).show(10)

SCHEDULING JOBS:

1. Create steps in EMR.
2. Give path for the pig and hive scripts.

**Spark queries:**

val busRDD= sc.textFile("/user/cloudera/inputfiles/businessreview.csv")  
  
val busCategoriesRDD= sc.textFile("/user/cloudera/inputfiles/businessCategories3.csv")  
  
val busParsedRDD = busRDD.map(rec => (rec.split(",")(0), rec))  
  
val busCategoriesParsedRDD = busCategoriesRDD.map(rec => (rec.split(",")(0), rec))  
  
val busJoinRDD = busParsedRDD.join(busCategoriesParsedRDD)  
  
val reviewCounRDD = busJoinRDD.map(t => (t.\_2.\_1.split(",")(1), t.\_2.\_1.split(",")(7).toFloat))  
  
val busreviewCountDay = reviewCounRDD.reduceByKey((x, y) => x + y)  
  
busreviewCountDay.take(10).foreach(println)  
  
  
  
val cateRDD=busJoinRDD.map(t => ((t.\_2.\_1.split(",")(1),t.\_2.\_1.split(",")(2)),(t.\_2.\_2.split(",")(1),t.\_2.\_1.split(",")(7).toInt)))  
  
val maxReviewRDD = cateRDD.reduceByKey((x,y) => if(x.\_2 >= y.\_2) x else y)  
  
maxReviewRDD.take(100).foreach(println)  
  
  
  
val cateRDD=busJoinRDD.map(t => ((t.\_2.\_1.split(",")(1),t.\_2.\_1.split(",")(2),t.\_2.\_2.split(",")(1)),t.\_2.\_1.split(",")(6).toDouble)).reduceByKey((x,y)=>x+y)  
  
cateRDD.reduceByKey((x,y) => if(x >= y) x else y).take(10).foreach(println)  
  
val stateRDD = cateRDD.filter(l => l.\_1.\_2.contains("AZ"))  
  
stateRDD.take(10).foreach(println)