SparkSQL

Tooling Session

Outline

- Environment Setup
- SparkSQL
- Bonus Project 2

Environment Setup

1. Download the Latest version of Java: https://www.java.com/en/download/
2. Download and Setup Apache Spark:
   a. https://downloads.apache.org/spark/spark-2.4.7/spark-2.4.7-bin-hadoop2.7.tgz
   b. Setup your environment variables (Windows: https://www.ics.uci.edu/~shantas/Install_Spark_on_Windows10.pdf)
3. Download IntelliJ IDEA Community Edition:
   a. https://www.jetbrains.com/idea/download/#section=windows
   b. Setup IDEA to work with Spark and Scala:
      https://dev.to/bartoszgajda55/setting-up-intellij-idea-for-apache-spark-and-scala-development-4ne2

add plugin for scala

SparkSQL

Documentation: https://spark.apache.org/docs/latest/sql-programming-guide.html

SparkSQL - Loading CSV file

- StructType is used to define the columns of a csv file
- StructField defines each column by its name, its type, and whether it can take the value NULL or not

```
val schema = StructType(Array(
  StructField("VendorID", DataTypes.StringType,false),
  StructField("tpep_pickup_datetime", DataTypes.TimestampType,false),
  StructField("tpep_dropoff_datetime", DataTypes.TimestampType,false),
  StructField("passenger_count", DataTypes.IntegerType,false),
  StructField("trip_distance", DataTypes.DoubleType,false),
  StructField("pickup_longitude", DataTypes.DoubleType,false),
  StructField("pickup_latitude", DataTypes.DoubleType,false),
  StructField("RatecodeID", DataTypes.IntegerType,false),
  StructField("store_and_fwd_flag", DataTypes.StringType,false),
  StructField("dropoff_longitude", DataTypes.DoubleType,false),
  StructField("dropoff_latitude", DataTypes.DoubleType,false),
  StructField("payment_type", DataTypes.IntegerType,false),
  StructField("fare_amount", DataTypes.DoubleType,false),
  StructField("extra", DataTypes.DoubleType,false),
  StructField("mta_tax", DataTypes.DoubleType,false),
  StructField("tip_amount", DataTypes.DoubleType,false),
  StructField("tolls_amount", DataTypes.DoubleType,false),
  StructField("improvement_surcharge", DataTypes.DoubleType,false),
  StructField("total_amount", DataTypes.DoubleType, false)
))
```

SparkSQL - Loading CSV file

- spark.read allows to read a csv file following the previously defined schema
- We set the header to true becasue the first row defines the columns names
- Spark.select acts the same way as the SELECT keyword in SQL

```
val tripsDF = spark.read.schema(schema).option("header", true).csv(taxifile)
  .select( cols = $"pickup_longitude", $"dropoff_longitude",$"pickup_latitude", $"dropoff_latitude", $"tpep_pickup_datetime", $"tpep_dropoff_datetime")
```

SparkSQL - Conditions

- Spark.where acts the same way as the WHERE keyword in SQL
- It filters the results based on the conditions applied on specific columns

&& — logical and
=!= — not equal
always make the table smaller choosing columns that you need for computation

```
val trips = tripsDF.where( condition = $"pickup_longitude" =!= 0 && $"pickup_latitude" =!= 0 && $"dropoff_longitude" =!= 0 && $"dropoff_latitude" =!= 0)
```

SparkSQL - Transformations

- Spark.withColumn applies transformations on a specific column resulting in:
  - modifications to the existing column
  - creation of a new column

```
.withColumn("tpep_pickup_datetime", unix_timestamp($"tpep_pickup_datetime"))
```

SparkSQL - Aggregate

- Spark.agg can perform aggregate functions like min,max, avg, count ...

```
var result = time{ReturnTrips.compute(trips, dist, spark).agg(count("*")).first
```

SparkSQL - Sorting

- Spark.sort can take multiple columns and works in a very similar fashion to ORDER BY in SQL
- Remember: Pre-sorting a certain column can allow for computational speedups and is sometimes worth the trade-off

SparkSQL - Joining

- Spark.join can perform joins between two data frames
- You can specify any conditions for the join. Whether they be equalities or inequalities

```
trips.as( alias = "a").join(trips.as( alias = "b"),
   joinExprs = ($"a.lat" === $"b.lat"))
```

Bonus Project 2 - Task - Identify b, the return trip of a

```
select *
from tripsProvided a,
     tripsProvided b
where distance(a.dropofflocation, b.pickuplocation) < r and
      distance(b.dropofflocation, a.pickuplocation) < r and
      a.dropofftime < b.pickuptime and
      a.dropofftime + 8 hours > b.pickuptime
```

Bonus Project 2 - Haversine

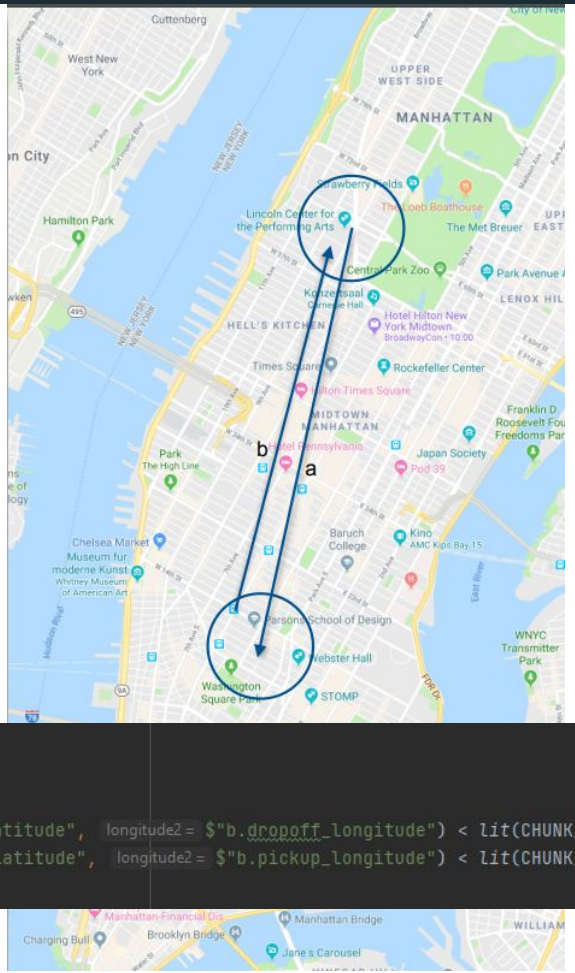
The **haversine formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the **law of haversines**, that relates the sides and angles of spherical triangles.

This formula is enough for distance computations, **be careful about degrees and radians**

Bonus Project 2 - Task

Yay, spark sql can take this as input!

Oh no, the optimiser gets very confused, resorts to using a cross product.




```
trips.as( alias = "a").join(trips.as( alias = "b"),
   joinExprs = ($"a.tpep_dropoff_datetime" < $"b.tpep_pickup_datetime") &&
    ($"a.tpep_dropoff_datetime" + 28800 > $"b.tpep_pickup_datetime") &&
    (haversine( latitude1 = $"a.pickup_latitude", longitude1 = $"a.pickup_longitude", latitude2 = $"b.dropoff_latitude", longitude2 = $"b.dropoff_longitude") < lit(CHUNK_SIZE)) &&
    (haversine( latitude1 = $"a.dropoff_latitude", longitude1 = $"a.dropoff_longitude", latitude2 = $"b.pickup_latitude", longitude2 = $"b.pickup_longitude") < lit(CHUNK_SIZE)))
)
```

Bonus Project 2 - How to solve this ?

Any ideas?

Bonus Project 2 - How to solve this ?

Bucketizing: Create buckets for columns that are used in the join (e.g. longitude) and perform an equi-join on these columns.

Bucketizing on multiple dimensions for the buckets to alleviate the cross product. Be careful that adding too many dimensions will increase the data set size. **Find the sweet spot**

```
explode(…-1,…,…+1) // per dimension
```

Case Example - Return People that have eaten 10 Apples within each other

Name	Apples Eaten
John	1
Kevin	45
Felix	55
Thomas	34
Erik	81

Case Example - Bucketize and Explode

Name	Apples Eaten	Bucket
John	1	0
Kevin	45	4
Felix	55	5
Thomas	34	3
Erik	81	8

Name	Apples Eaten	Bucket
John	1	-1,0,1
Kevin	45	3,4,5
Felix	55	4,5,6
Thomas	34	2,3,4
Erik	81	7,8,9

Case Example - Equi Join on buckets

Name	Apples Eaten	Bucket	Name
John	1	-1,0,1	-
Kevin	45	3,4,5	Felix, Thomas
Felix	55	4,5,6	Kevin, Thomas
Thomas	34	2,3,4	Felix, Kevin
Erik	81	7,8,9	-

- Kevin is only compared to Felix and Thomas
- Felix is compared to Kevin and Thomas
- Thomas is compared to Kevin and Felix
- John and Erik are not compared to anyone

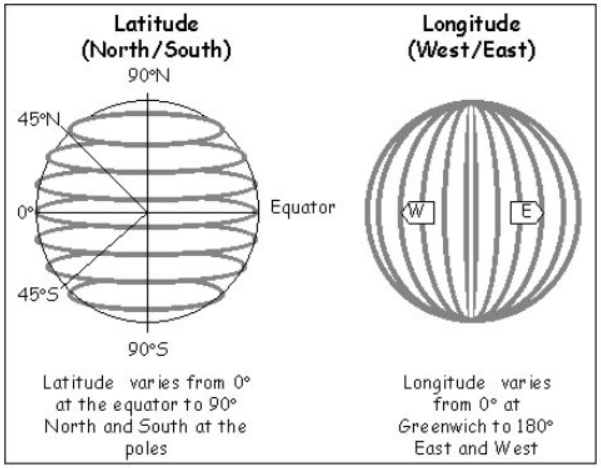
Bonus Project 2 - How to bucketize ?

Nice try, i already said too much.

Bonus Project 2 - How to bucketize ?

bucketize using the condition in the problem statement

Nice try, i already said too much. Okay look at this.



Bonus Project 2 - Further Optimizations

- Z-order curve
- Bloom Filter

y:	x: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	000000	000001	000100	000101	010000	010001	010100	010101
1 001	000010	000011	000110	000111	010010	010011	010110	010111
2 010	001000	001001	001100	001101	011000	011001	011100	011101
3 011	001010	001011	001110	001111	011010	011011	011110	011111
4 100	100000	100001	100100	100101	110000	110001	110100	110101
5 101	100010	100011	100110	100111	110010	110011	110110	110111
6 110	101000	101001	101100	101101	111000	111001	111100	111101
7 111	101010	101011	101110	101111	111010	111011	111110	111111