Querying data with MapReduce

Hadoop has a few things in common with

Databases

Hadoop vs Databases

Like databases, Hadoop is used for storing data

Hadoop vs Databases

There is one big difference though

Hadoop vs Databases

The data in Hadoop is
Unstructured

Hadoop vs Databases

Unlike databases, HDFS data doesn't have any schema

Hadoop vs Databases

It's basically in the form of files

Text files

Log files

Video/Audio files

Hadoop vs Databases

There's no concept of rows/columns

There are no tables

Hadoop vs Databases

The records in a text file could be seen as rows

Hey Diddle Diddle
The cat and the fiddle
The cow jumped over the moon

Hadoop vs Databases

Hey Diddle Diddle
The cat and the fiddle
The cow jumped over the moon

But Hadoop does not impose any constraints on those rows

Hadoop vs Databases

This is not to say that Hadoop can't be used to store structured data

Hadoop vs Databases

You could store your data in a structured format even in Hadoop

csv files

xml files

jsons

Hadoop vs Databases

Each file could represent a row, or in the case of csvs a table

csv files
xml files
jsons

Hadoop vs Databases

But unlike databases, Hadoop will not enforce any constraints on their structure

csv files
xml files
jsons

Hadoop vs Databases

Hadoop will also not guarantee the ACID constraints that a database provides

csv files
xml files
jsons

Hadoop vs Databases

So, why should we store structured data in Hadoop?

Hadoop vs Databases

Because Hadoop can parallelize any processing tasks on that data

Hadoop vs Databases

Since processing is parallelized, you can scale linearly

Hadoop vs Databases

This means that all it takes to double performance is to double the number of nodes

Hadoop vs Databases

This is not true of traditional databases

Hadoop vs Databases

In a database you need to do different things to improve read performance vs write performance

Hadoop vs Databases

These 2 are usually in counterbalance

Hadoop vs Databases

For example, you could improve read performance in a database by building an index

Hadoop vs Databases

But, keeping the index updated will affect the write performance

Hadoop vs Databases

Not just that, an index requires disk space and the cost of disk space increases in a non-linear fashion

Hadoop vs Databases

Basically, scaling performance is pretty complicated with databases

Hadoop vs Databases

The ease of scaling with Hadoop makes it a very attractive option for storing structured data

Hadoop vs Databases

On the flip side, defining data processing tasks in Hadoop can be pretty daunting

Hadoop vs Databases

When, you store data in a database, you query it using a Structured Query Language

Hadoop vs Databases

When, you store data in a Hadoop, you have to process it using MapReduce

Hadoop vs Databases

SQL is really much easier to use and understand :)

Hadoop vs Databases

However, it's possible to take each SQL construct and express it using MapReduce

Hadoop vs Databases

Select

Where

Group by

Having

Join

We could have a MapReduce strategy for each of these

Hadoop vs Databases

Select MR

Where MR

Group by MR

Having MR

Join MR

Then you can still have a user define SQL queries

Hadoop vs Databases

Select MR

Where MR

Group by MR

Having MR

Join MR

And in the background convert them into a combination of these MR jobs

Hadoop vs Databases

Select MR

Where MR

Group by MR

Having MR

Join MR

You can leverage the power of MapReduce but still keep the query interface very simple and SQL like

Hadoop vs Databases

Select MR

Where MR

Group by MR

Having MR

Join MR

Apache Hive is a data warehousing framework built using exactly this idea

Hadoop vs Databases

Select MR

Where MR

Group by MR

Having MR

Join MR

Let's go through the MapReduce strategies for each of these constructs

Let's say you've got a large amount of financial data

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538
3IINFOTECH	EQ	7.85	7.9	7.45	7.65	7.6	7.75	764766	5828471.15	01-APR-2014	946
3MINDIA	EQ	3551	3589	3551	3588.25	3581	3527.55	253	906391.15	01-APR-2014	37
8KMILES	EQ	100.15	111.5	100.15	103.3	101.6	104.95	4855	489279.15	01-APR-2014	114
A2ZMES	EQ	10.5	10.85	10.05	10.8	10.8	10.45	386385	4081822.25	01-APR-2014	848
AARTIDRUGS	EQ	265	267.95	261.1	264.25	265.05	262.65	3952	1047797.65	01-APR-2014	140
AARTIIND	EQ	124	124.5	123.05	123.45	123.65	122.85	12410	1533857.7	01-APR-2014	346
AARVEEDEN	EQ	35.2	35.45	35.2	35.4	35.4	35.3	611	21608.95	01-APR-2014	10

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538
3IINFOTECH	EQ	7.85	7.9	7.45	7.65	7.6	7.75	764766	5828471.15	01-APR-2014	946
3MINDIA	EQ	3551	3589	3551	3588.25	3581	3527.55	253	906391.15	01-APR-2014	37
8KMILES	EQ	100.15	111.5	100.15	103.3	101.6	104.95	4855	489279.15	01-APR-2014	114
A2ZMES	EQ	10.5	10.85	10.05	10.8	10.8	10.45	386385	4081822.25	01-APR-2014	848
AARTIDRUGS	EQ	265	267.95	261.1	264.25	265.05	262.65	3952	1047797.65	01-APR-2014	140
AARTIIND	EQ	124	124.5	123.05	123.45	123.65	122.85	12410	1533857.7	01-APR-2014	346
AARVEEDEN	EQ	35.2	35.45	35.2	35.4	35.4	35.3	611	21608.95	01-APR-2014	10

The data is in the form of files one for each day of trading

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538
3IINFOTECH	EQ	7.85	7.9	7.45	7.65	7.6	7.75	764766	5828471.15	01-APR-2014	946
3MINDIA	EQ	3551	3589	3551	3588.25	3581	3527.55	253	906391.15	01-APR-2014	37
8KMILES	EQ	100.15	111.5	100.15	103.3	101.6	104.95	4855	489279.15	01-APR-2014	114
A2ZMES	EQ	10.5	10.85	10.05	10.8	10.8	10.45	386385	4081822.25	01-APR-2014	848
AARTIDRUGS	EQ	265	267.95	261.1	264.25	265.05	262.65	3952	1047797.65	01-APR-2014	140
AARTIIND	EQ	124	124.5	123.05	123.45	123.65	122.85	12410	1533857.7	01-APR-2014	346
AARVEEDEN	EQ	35.2	35.45	35.2	35.4	35.4	35.3	611	21608.95	01-APR-2014	10

We'll take all the files and dump them into an HDFS directory

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTT

We could argue that the file's header row represents the column names of a table

Let's see if can build MapReduce strategies for queries on this table

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP
WHERE SYMBOL = "3MINDIA"
AND SERIES = "EQ"
```

Pick all the records where this condition is satisfied

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

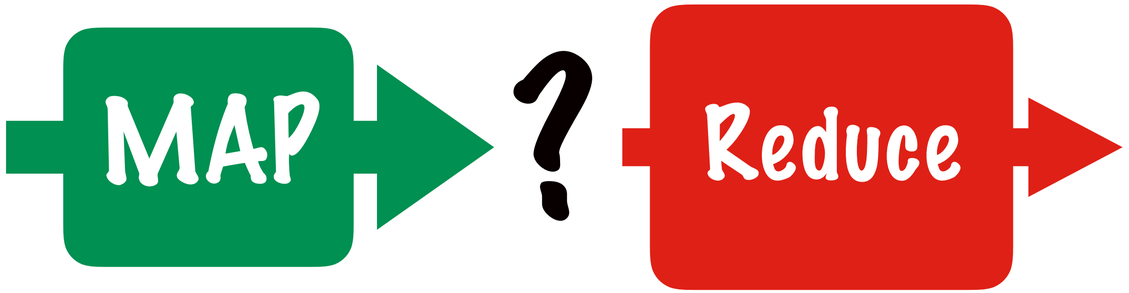
```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP
WHERE SYMBOL= "3MINDIA"
AND SERIES = "EQ"
```

Keep only these 4 columns from those records

```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP
WHERE SYMBOL= "3MINDIA"
AND SERIES = "EQ"
```

<Rownum, Input Row>

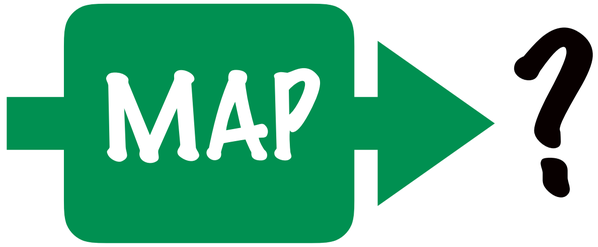
Input



<Null, Output row>

Output

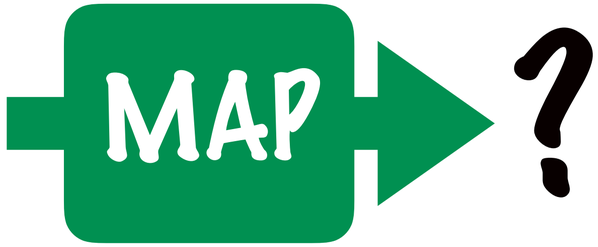
<Rownum, Input Row>



The input row is a line in the csv file

20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

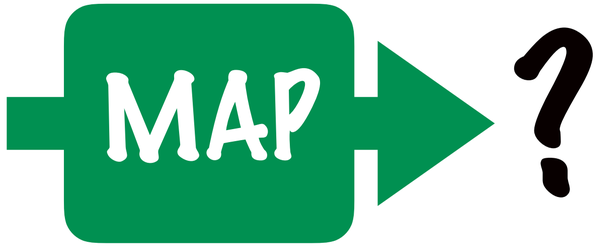
<Rownum, Input Row>



20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The line comes to the Mapper as a Text object

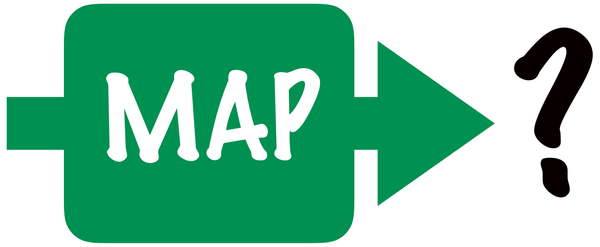
<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The Mapper should parse the Text to extract all the fields

<Rownum, Input Row>

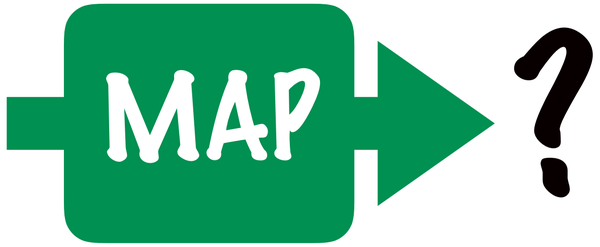


SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The Mapper will first check if the Where conditions are satisfied

WHERE SYMBOL = "3MINDIA" AND SERIES = "EQ"

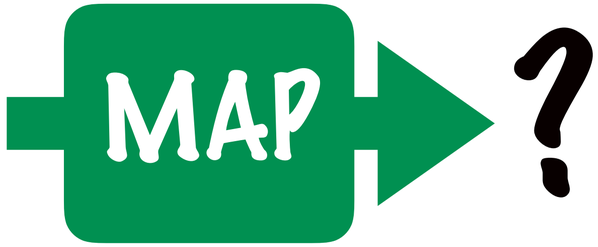
<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

When they are not, there is no output from the Mapper

<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
3MINDIA	EQ	3551	3589	3551	3588.25	3581	3527.55	253	906391.15	01-APR-2014	37

If they are satisfied, then a Text object is constructed using the columns in select

SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP

<Rownum, Input Row>

?

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
3MINDIA		3551			3588.25					01-APR-2014	

This row is written to the output

<Rownum, Input Row>

<?, Output Row>

3MINDIA	3551	3588.25	01-APR-2014

Each row the Map writes represents a row in the final output

<Rownum, Input Row>

<?, Output Row>

3MINDIA	3551	3588.25	01-APR-2014

We don't want these to be combined in any way

<Rownum, Input Row>

<?, Output Row>

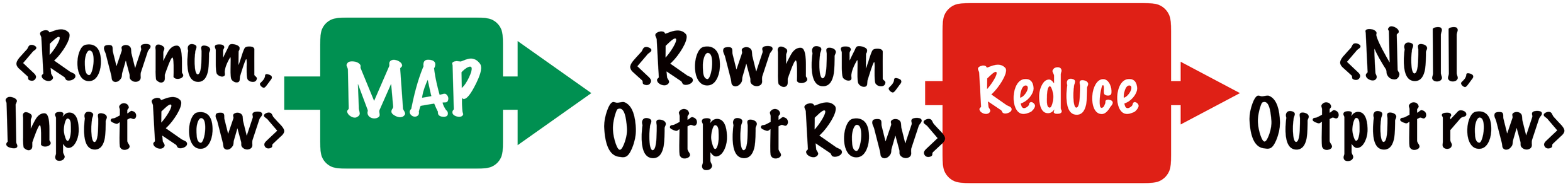
3MINDIA	3551	3588.25	01-APR-2014

So we can just have the Rownum itself as the key

<Rownum, Input Row>

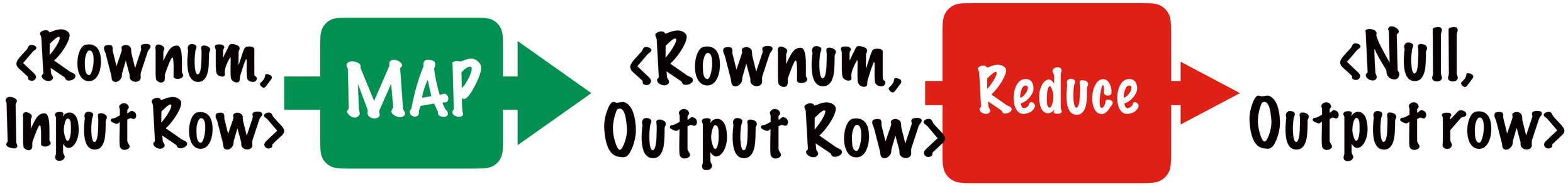
<Rownum, Output Row>

```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP
WHERE SYMBOL= "3MINDIA"
AND SERIES = "EQ"
```



The Reducer is almost like an identity function

```
SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP
WHERE SYMBOL= "3MINDIA"
AND SERIES = "EQ"
```



We discard the Rownum and just write out each row as it comes to the reducer

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }


        if (data[0].equals("NTPC") && data[1].equals("EQ")){

           String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[10];
           // symbol,     open,        close,          date

            context.write(key, new Text(output));
        }


            }
        }
```

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }


        if (data[0].equals("NTPC") && data[1].equals("EQ")){

           String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[10];
           // symbol,     open,        close,          date

            context.write(key, new Text(output));
        }


            }
        }
```

<Rownum, Input Row>

<Rownum, Output Row>

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }


        if (data[0].equals("NTPC") && data[1].equals("EQ")){

           String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[10];
           // symbol,     open,        close,         date

            context.write(key, new Text(output));
        }


            }
        }
```

We parse the line and get an array of Strings

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
    String[] data = line.split(",");

        if(data[0].equals("SYMBOL")){
                return;
            }

    if (data[0].equals("NTPC") && data[1].equals("EQ")){

        String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[0];
        // symbol,     open,         close,          date

         context.write(key, new Text(output));
    }


        }
    }
```

If the row we are reading is a Header row in the file, ignore it

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
     String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

         if (data[0].equals("3MINDIA") && data[1].equals("EQ")){


           String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[10];
           // symbol,     open,        close,         date

            context.write(key, new Text(output));
        }


            }
        }
```

WHERE SYMBOL = "3MINDIA"
AND SERIES = "EQ"

select.Map

```
public class Map extends Mapper<LongWritable,Text,LongWritable,Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
     String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }


        if (data[0].equals("3MINDIA") && data[1].equals("EQ")){

           String output = data[0]+" , "+data[2]+" , "+data[5]+" , "+data[10];
                           // symbol,      open,          close,          date


                context.write(key, new Text(output));
        }


            }
        }
```

SELECT SYMBOL, OPEN, CLOSE, TIMESTAMP

select.Reduce

```
public class Reduce extends Reducer<LongWritable,Text,NullWritable,Text> {
    @Override
    public void reduce(final LongWritable key, final Iterable<Text> values,
                       final Context context) throws IOException, Interrupte


        for (Text value : values) {
            context.write( NullWritable.get(), value);
        }


    }
}
```

select.Reduce

```
ic class Reduce extends Reducer<LongWritable,Text,NullWritable,Text> {
@Override
ublic void reduce(final LongWritable key, final Iterable<Text> values,
                  final Context context) throws IOException, InterruptedExceptio
```

```
    for (Text value : values) {
            context.write( NullWritable.get(), value);
        }
```

```
}
```

Select MR

Where MR

Group by MR

Having MR

Join MR

We would implement the Select and Where of a query in the Mapper, and not do anything in the reducer

Select MR

Where MR

Group by MR

Having MR

Join MR

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

Pick all the records where this condition is satisfied

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

```
SELECT SYMBOL, Max(OPEN)
WHERE SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

For each SYMBOL

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

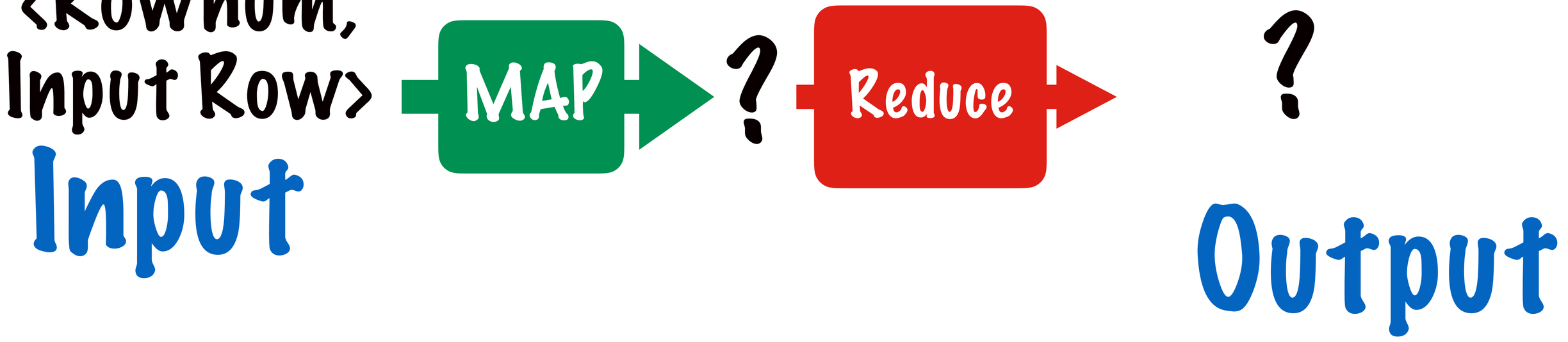
Calculate the max Opening Price

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES

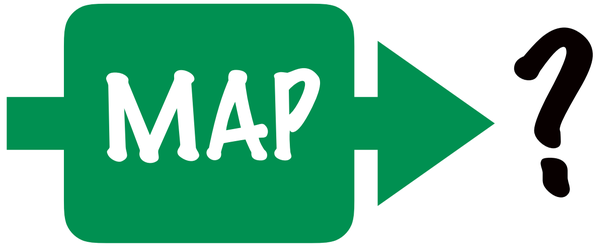
```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

Write out only those values that satisfy this condition

```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```



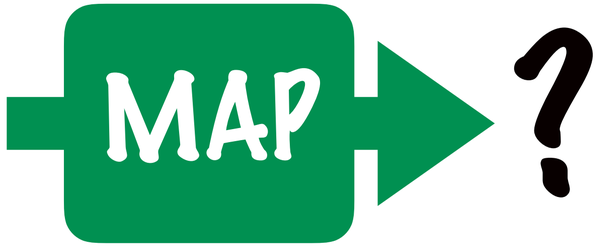
<Rownum, Input Row>



The input row is a line in the csv file

20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

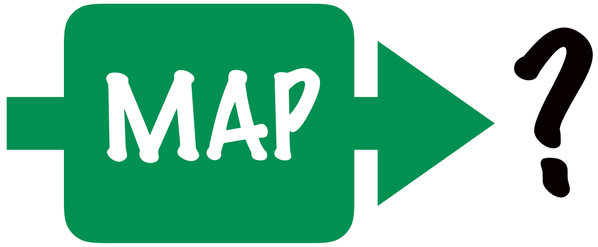
<Rownum, Input Row>



20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The line comes to the Mapper as a Text object

<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The Mapper should parse the Text to extract all the fields

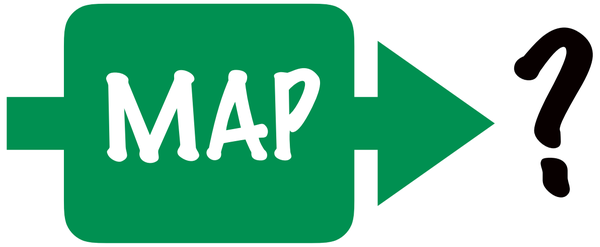
<Rownum, Input Row>

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

The Mapper will first check if the Where conditions are satisfied

WHERE SERIES = "EQ"

<Rownum, Input Row>

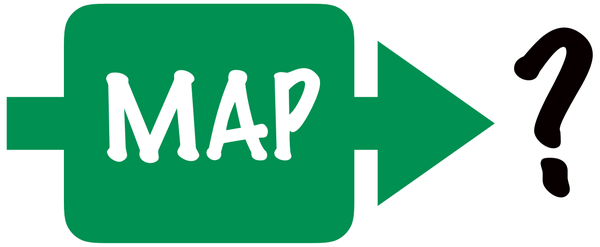


SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

If they are satisfied, then a Text object is constructed using the columns in select

SELECT SYMBOL, max(OPEN)

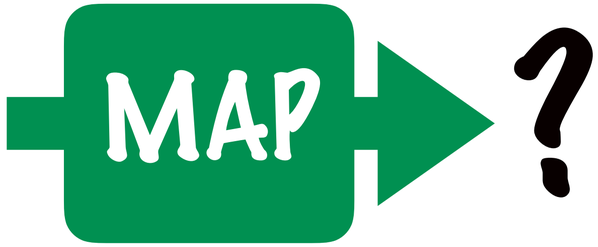
<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS		31.8									
20MICRONS		30.75									
20MICRONS		30.85									

In this case, we actually want to send the Reducer values grouped by their Symbol

<Rownum, Input Row>



?

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS		31.8									
20MICRONS		30.75									
20MICRONS		30.85									

Grouping is an operation that's like a natural part of MapReduce

<Rownum, Input Row>

?

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS		31.8									
20MICRONS		30.75									
20MICRONS		30.85									

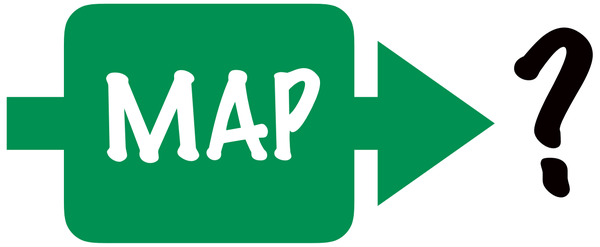
The Reduce operations are usually remarkably like grouping in SQL

<Rownum, Input Row>

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS		31.8									
20MICRONS		30.75									
20MICRONS		30.85									

So we want the reducer to do the grouping for us

<Rownum, Input Row>



SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST	PREVCLOSE	TOTTRDQTY	TOTTRDVAL	TIMESTAMP	TOTALTRADES
20MICRONS		31.8									
20MICRONS		30.75									
20MICRONS		30.85									

The key that we group by should be the key of Map output

GROUP BY SYMBOL

<Rownum, Input Row>

<Group by Key, Other Columns>

The aggregation operation naturally fits inside the reducer

```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

The Reducer will simply use the specified aggregation function to combine values that share a key

```
SELECT SYMBOL, Max(OPEN)
WHERE  SERIES = "EQ"
GROUP BY SYMBOL
HAVING Max(OPEN) >20
```

The Having condition will be checked inside the reducer before output is written

groupBy.Map

```
public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");

        if(data[0].equals("SYMBOL")){
            return;
        }


        if ( data[1].equals("EQ")){

            String symbol = data[0];
            Double openPrice = Double.parseDouble(data[2]);
            context.write(new Text(symbol), new DoubleWritable(openPrice));
        }


            }
        }
```

groupBy.Map

```
public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");

            if(data[0].equals("SYMBOL")){
                return;
            }
     if ( data[1].equals("EQ")){

        String symbol = data[0];
        Double openPrice = Double.parseDouble(data[2]);
        context.write(new Text(symbol), new DoubleWritable(openPrice));
    }


        }
    }
```

Parse the line and check if it's a header row

groupBy.Map

```
public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");

        if(data[0].equals("SYMBOL"))
            return;
        }
```

WHERE SERIES = "EQ"

```
if ( data[1].equals("EQ")){


            String symbol = data[0];
            Double openPrice = Double.parseDouble(data[2]);
            context.write(new Text(symbol), new DoubleWritable(openPrice));
        }


            }
        }
```

groupBy.Map

```
public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");

        if(data[0].equals("SYMBOL")){
            ret
        }


         if ( data[1].equals("EQ")){


           String symbol = data[0];
                 Double openPrice = Double.parseDouble(data[2]);
                 context.write(new Text(symbol), new DoubleWritable(openPrice))
              }


           }
        }
```

SELECT SYMBOL, max(OPEN)

groupBy.Map

```
public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String line = value.toString();
        String[] data = line.split(",");

        if(data[0].equals("SYMBOL")){
            return;
        }



         if ( data[1].equals("EQ")){


           String symbol = data[0];
           Double openPrice = Double.parseDouble(data[2]);
          context.write(new Text(symbol), new DoubleWritable(op
            }


           }
        }
```

GROUP BY SYMBOL

groupBy.Reduce

```
public class Reduce extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {
    @Override
    public void reduce(final Text key, final Iterable<DoubleWritable> values,
                       final Context context) throws IOException, InterruptedException {

        Double maxValue = Double.MIN_VALUE;
        for (DoubleWritable value : values) {
            maxValue = Math.max(maxValue,value.get());
        };


        if (maxValue>20) {
            context.write(key, new DoubleWritable(maxValue));

        }
    }
}
```

groupBy.Reduce

```
public class Reduce extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {
    @Override
    public void reduce(final Text key, final Iterable<DoubleWritable> values,
                       final Context context) throws IOException, InterruptedException {

        Double maxValue = Double.MIN_VALUE;
         for (DoubleWritable value : values) {
             maxValue = Math.max(maxValue,value.get());
         };


        if (maxValue>20) {
            context.write(key, new DoubleWritable(maxValue));

        }
    }
}
```

max(OPEN)

GROUP BY SYMBOL

groupBy.Reduce

```
public class Reduce extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {
    @Override
    public void reduce(final Text key, final Iterable<DoubleWritable> values,
                       final Context context) throws IOException, InterruptedException {

        Double maxValue = Double.MIN_VALUE;
        for (DoubleWritable value : values) {
            maxValue = Math.max(maxValue,value.get());
        };


            if (maxValue>20) {
            context.write(key, new DoubleWritable(maxValue));

        }
    }
}
```

HAVING Max(OPEN) >20