Design Document

SQL Query processing using Map Reduce and Spark

1. Table Schema

The text file amazon-meta.txt is divided into 4 .csv files (products.csv, similar.csv, categories.csv, reviews.csv). Each line of a file represents a tuple containing data separated by '^' delimiter.

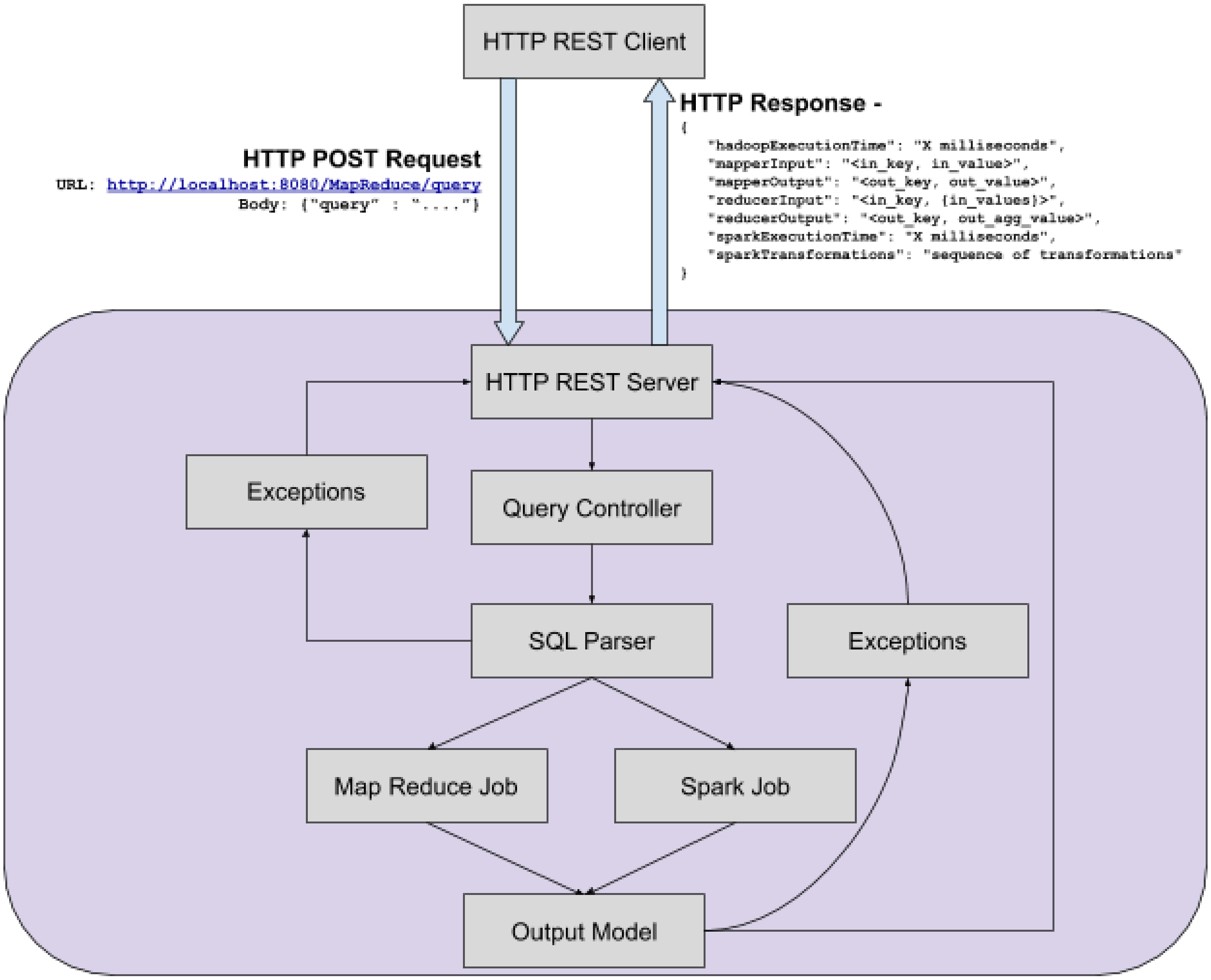
- products.csv Schema
  `{id^asin^title^group^salesrank^nosimilar^nocategories^totalreviews^downloaded^avgrating}`
  asin :- amazon standard identification number
  nosimilar :- total number of similar products
  nocategories :- total no of category hierarchy it has
  EXAMPLE:
  1^0827229534^Patterns of Preaching: A Sermon Sampler^Book^396585^5^2^2^2^5

- simiral.csv Schema
  `{id^asin^similarasin}`
  similarasin :- asin number of product similar to corresponding asin
  EXAMPLE:
  1^0827229534^0804215715

- categories.csv Schema
  `{id^asin^category}`
  category :- is the individual category (value is stored as single category between two pipes and not the whole hierarchy)
  EXAMPLE:
  1^0827229534^Books[283155]
  1^0827229534^Subjects[1000]
  1^0827229534^Religion & Spirituality[22]

- reviews.csv Schema
  `{id^asin^reviewdate^customerid^rating^votes^helpful}`
  EXAMPLE:
  1^0827229534^2000-7-28^A2JW67OY8U6HHK^5^10^9

2. Overall REST API Design

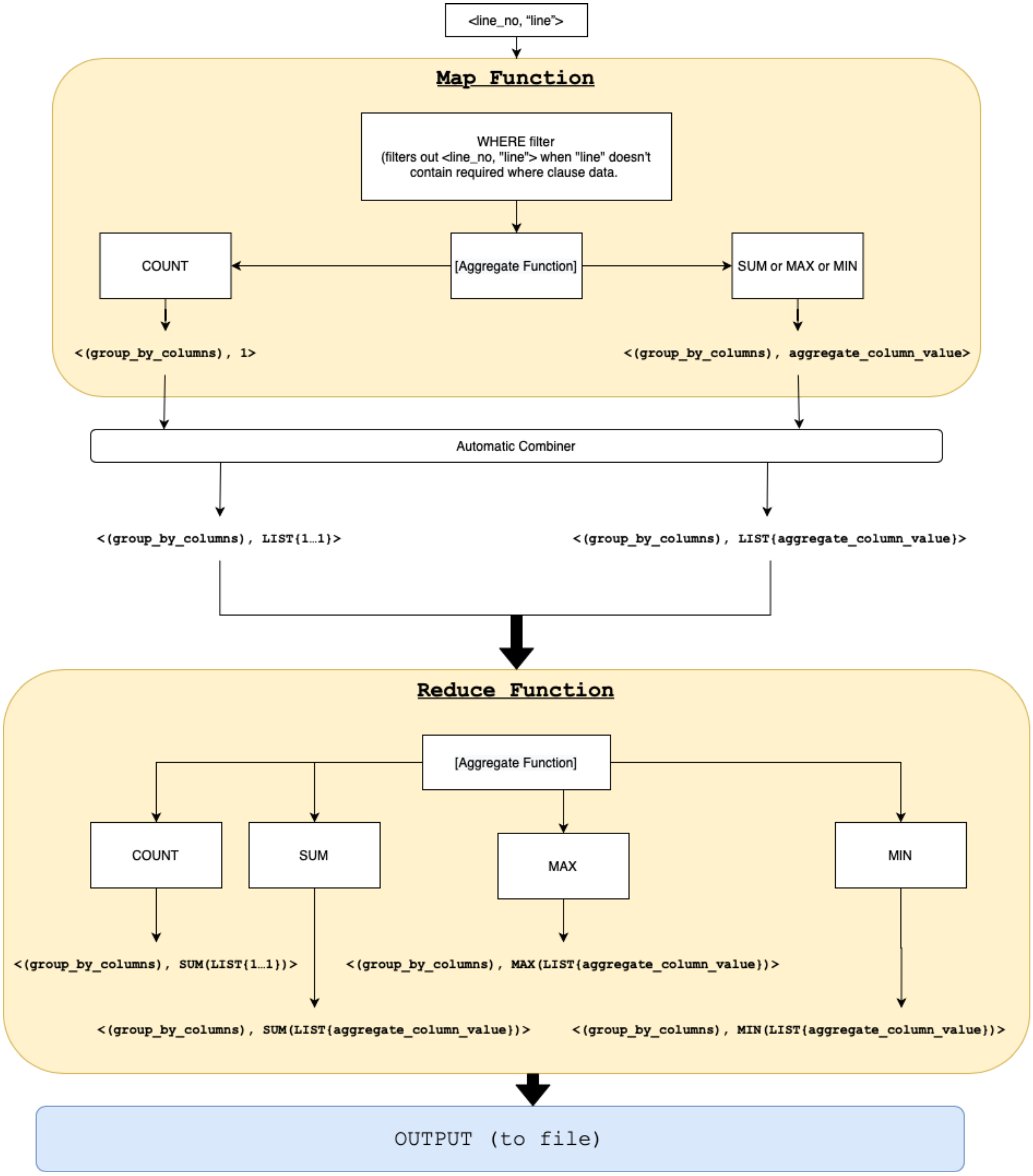


```
URL: http://localhost:8080/MapReduce/query
Body: {"query" : "...."}
```

```
{
   "hadoopExecutionTime": "X milliseconds",
   "mapperInput": "<in_key, in_value>",
   "mapperOutput": "<out_key, out_value>",
   "reducerInput": "<in_key, {in_values}>",
   "reducerOutput": "<out_key, out_agg_value>",
   "sparkExecutionTime": "X milliseconds",
   "sparkTransformations": "sequence of transformations"
}
```

- Http Rest Server is hosted on Tomcat and the project is using Maven and JAX-WS dependencies in java to host web services.
- POST request is made through Postman and a URL mapped to controller of the project which in turn first calls to SQL Parser (SqlQueryParser.java) which parse the query string and generates subsequent tokens encapsulated in PoJo (QueryModel.java) calls. Now Map Reduce and Spark functions are called with this parsed query token as input parameters.
- Output is returned encapsulated in PoJo (OutputModel.java) and then mapped to JSON using jersey json binding libraries. This JSON data is returned as response to POST requests.
- Contracts classes (ProductsTable.java, SimilarTable.java, ….etc) are used for each table to specific names of column and index for index to column mapping and also column to index mapping.

3. Map Reduce Design


<line_no, “line”>
Map Function
WHERE filter
(filters out <line_no, "line"> when "line" doesn't contain required where clause data.
[Aggregate Function]
COUNT
SUM or MAX or MIN
<(group_by_columns), 1>
<(group_by_columns), aggregate_column_value>
Automatic Combiner
<(group_by_columns), LIST{1…1}>
<(group_by_columns), LIST{aggregate_column_value}>
Reduce Function
[Aggregate Function]
COUNT
SUM
MAX
MIN
<(group_by_columns), SUM(LIST{1…1})>
<(group_by_columns), MAX(LIST{aggregate_column_value})>
<(group_by_columns), SUM(LIST{aggregate_column_value})>
<(group_by_columns), MIN(LIST{aggregate_column_value})>
OUTPUT (to file)

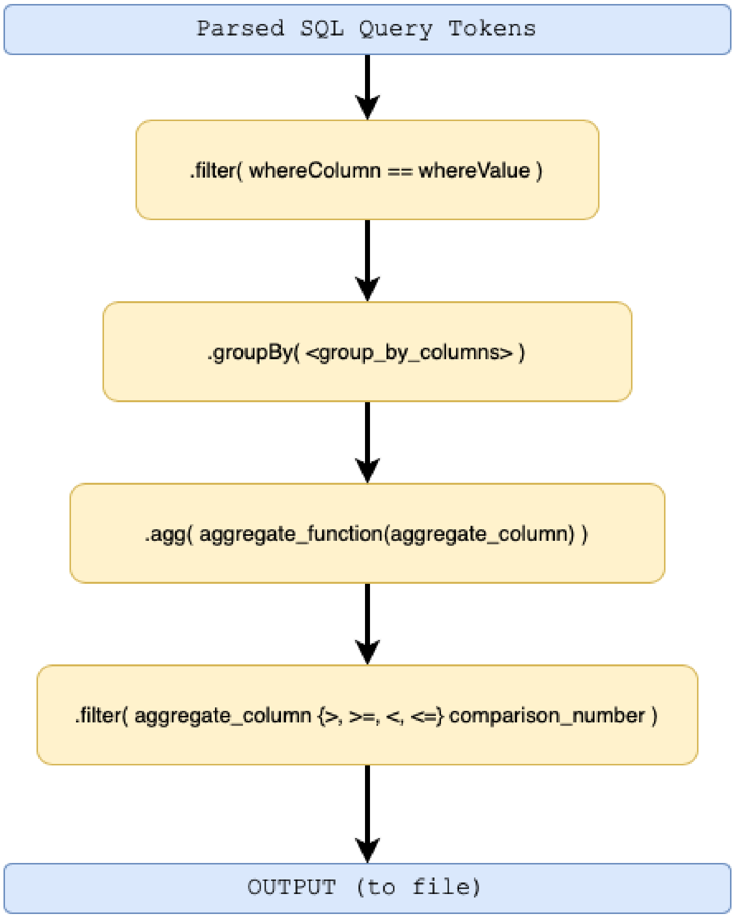
- Input to **Map** is of the form of tuple <line_no, “line”> where “line” is a single line that mapper reads one by one containing data/list of columns separated by “^” delimiter.
- Map function then splits the line based on the delimiter and stores in array names line[] and discards the line if where clause is not satisfied (by checking if line[where_col_index] == where_col_value_from_query). Map keeps the where clause satisfied lines and processes it to generate output key/value pair.
- Output key is - “all_group_by_columns_seperated_by_comma” and output value is - “1” if aggregate function is COUNT otherwise “line[aggregate_column_index]”.

- Automatic combiner of Hadoop combines all the values of keys and values as List of {values}.
- Input to **Reduce** is -

< “all_group_by_columns_seperated_by_comma”, LIST{1….1}/LIST{aggregate_column_value} >

- Then this reduce function applies an appropriate aggregate function on List of values and grouped by keys and writes to an output file as shown in above design diagram.

4. Spark Design


Parsed SQL Query Tokens
.filter(whereColumn == whereValue)
.groupBy(<group_by_columns>)
.agg(aggregate_function(aggregate_column))
.filter(aggregate_column {>, >=, <, <=} comparison_number)
OUTPUT (to file)

- Parsed query token is sent again to SparkJob.scala. It reads from the input file and converts the whole .csv file data to its own dataframe.
- First filter is applied to this data frame to discard rows which do not conform to where clause.
- Then the groupBy function is applied with all the groups by columns of query as its parameter to tell the spark to use these columns to group.
- Then the corresponding aggregate function is applied to the aggregate column.
- Then the final filter is applied to discard rows which do not conform to HAVING clause.

4. File Tree Structure of Code (MapReduce Folder)

```
├── MapReduce
│   ├── Data
│   │   ├── categories.csv
│   │   ├── products.csv
│   │   ├── reviews.csv
│   │   └── similar.csv
│   ├── Output
│   │   ├── mapreduce
│   │   └── spark
│   ├── pom.xml
│   └── src
│       └── main
│           ├── java
│           │   └── com
│           │       └── cloud
│           │           └── MapReduce
│           │               ├── Data
│           │               │   ├── CategoriesTable.java
│           │               │   ├── DataExtractor.java
│           │               │   ├── ProductsTable.java
│           │               │   ├── ReviewsTable.java
│           │               │   └── SimilarTable.java
│           │               ├── GlobalVariables.java
│           │               ├── Jobs
│           │               │   ├── MapReduceJob.java
│           │               │   └── SparkJob.scala
│           │               ├── Model
│           │               │   ├── InputModel.java
│           │               │   ├── OutputModel.java
│           │               │   └── QueryModel.java
│           │               ├── QueryController.java
│           │               └── SQL
│           │                   └── SqlQueryParser.java
│           ├── resources
│           ├── scala
│           └── webapp
│               ├── WEB-INF
│               │   └── web.xml
│               └── index.jsp
│
```

Data → Folder containing four .csv file

Output →Folder for output of MapReduce and Spark

CategoriesTable.java/ProductsTable.java/ReviewsTable.java/SimilarTable.java → They are contracts for each table, containing table name and column details.

GlobalVariables.java → Contains path values.

MapReduceJob.java → Map Reduce code

SparkJob.scala → Spark Code

InputModel.java → PoJo for encapsulating Input data.

OutputModel.java → PoJo for encapsulating Output data.

QueryModel.java → PoJo for storing tokens of parsed query.

QueryController.java → Main Controller class for calling jobs.

SqlQueryParser.java → Code for parsing SQL query into tokens.