**Homework5**

Now we will continue with aggregation. Aggregation is just like ‘group by’ in the SQL database. The aggregation pipeline allows MongoDB to provide native aggregation capabilities that corresponds with many common data aggregation operations in SQL. This is an instance about adopting group by in SQL:

**SELECT cust\_id,**

**count(\*)**

**FROM orders**

**GROUP BY cust\_id**

**HAVING count(\*) > 1**

This SQL statement can show all the customers’ IDs and the count of ordering records. This sentence can also be realized in MongoDB:

**result = orders.aggregate( [**

**{**

**$group: {**

**\_id: "$cust\_id",**

**count: { $sum: 1 }**

**}**

**},**

**{ $match: { count: { $gt: 1 } } }**

**] )**

We can have a look on it. For cust\_id with multiple records, the cust\_id and the corresponding record count would be returned. The output contain an \_id field which contains the distinct group by key. The attributions in the brace which is right after the sign ’$group’ are grouped. The operator ‘$match’ can match the specified conditions, which is like ‘having’ or ‘where’ in SQL.

Right now, we want to make a conclusion about the highest temperature. You need to calculate the average highest temperatures in a certain date range (start from 2000.01.01 to 2010.01.01) for every city.

**NOTE:** The result should be stored in the variable *pipeline* and the operator *$and* means logical and.

**Here is the answer:**

**pipeline = [{'$match':{'$and':[{"DATE":{'$lt':20100101}},{"DATE":{'$gt':20000101}}]}},**

**{'$group':{'\_id' :'$STATION\_NAME',"ave\_high\_temp":{'$avg':'$TMIN'}}}]**