Yiran Zheng
CS 127
zhengyr@brandeis.edu

PrimaryKeys, ForeignKeys

Following the rules for primary keys and foreign keys, we first set up the the primary keys and foreign keys for the tables so that we could execute the following queries. Notice that primary needs to be execute before foreign keys.

Problem 1

For this problem, we don't need trigger. We first dropped the foreign key we declared earlier and then add predicate on update

Problem 2

We do need trigger for this one. According to the description of the problem, each time there is a change in the retail price, the supply cost must change for the same amount. So we would do a trigger AFTER UPDATE ON table part. In side our function, we would have update table partsupp according to the changes in table part.

Problem 3

According to the description, every time there is an new order comes in, we need to check if it's status is open. If it's status is open, then if need to check the number of open orders in this customer has reached 14, if it reaches 14, then we would not insert the tuple into the table and raise exception to tell the user that he/she cannot add this new order into our table.

Problem 4

We do not need triggers for this one. We simply delete the old foreign key and add delete cascade this time.

Problem 5

For this one, there are some cases to consider.

Let t donate the tuple we want to add in table lineitem

Case 1: If every other tuples on lineitem that have the same order key as t does have the same status, open or fulfilled, when if we add an tuple with the same status, nothing needs to change in our order keep

Case 2: If t has different order status than any one of the tuples in table lineitem with the same orderkey, we would have a P status.

Only case 2 need to be handled.

Problem 6

No trigger is needed for this problem. We just deleted the old foreign and add cascade and set

null to update and deletion

Problem 7

For this table we have six cases, each case is the changes in supply cost from one region to another. In the each if statement, we first checks if our condition for regions is meet, once we've find the right regions, we would update the supply cost according to the table provided.

We would only do this procedure after there is an update in supplier and the update changes nation.

We could use nested if statement for the six cases, but this way of representing is more clear.

Part 1

In this problem, we are asked to draw five lines for five regions so that we would need loop through each region and then get the lines. Inside the loop, we need to get the data from the table first. In our query, we need to group lineitems by year and month so that after we calculate prices for each lineitem we are able to sum the results together for each month. After we get the data, we then plot them into lines and add the line to the graph. We repeat these procedures five times to get the graph for five regions.

Par 2:

For this problem, we need a graph that represents the consumer cost for from one region to another. Each region will need to buy items from the other four regions and it also buy things from itself. Since we have 5 regions, there are 5*5=25 edges to draw and 25 queries to write. Using nested for loops, we are able to loop though each region and then draw the edges for it. In the query, we only calculate the total amount of money from one region to another one and then added it as weight of edge from one node to the other in the graph. Each query will only return one tuple, which is the weight of the edge. After we finished all the 25 cases, we are done with our graph.

Part 3

For this problem, we want to get the order information first and then use the information to generate queries. We first have a scanner that reads the input file line by line. For each line, we separate the order number and the ordered parts using split method of string. After we get the order number and the parts we want to order, we need to get information for each ordered part. Using the string.split method again, we get the order for each part and generate the corresponding queries to get the suppliers who offers the part with cheapest price and enough availability. The query would use partsupp table. We will compare each supplier that offers the desired part and get the one that offers the cheapest price. The query will return one single tuple with the first column being the supplier and the second column being the cost for that part. After getting the result of the query, we put the result in our result set so that we could read from it. For each part, we first check if the supplier for this part already exists in our hashmap. If we've already ordered from this supplier, we would add the part we ordered to the supplier's part list. If not, we would add this supplier to our hash map as key and the key will point to a set of the parts this supplier supply. After we

generate a hash map of ordered parts, we would print the result out.