# Question1

**a.**

We know that a data cube of n dimensions contains $2^n$ cuboids.

So, given 10 dimensions, we can easily derive that there are $2^{10} = 1024$ cuboids in the full data cube.

**b.**

To calculate distinct aggregated cells. We need to:

1. get all the aggregated cells(non-base) in this data cube.
2. Delete all the duplicate ones.

1. There are 3 base cells in the base cuboid so each base cell can generate $\sum_{n=1}^{10} \binom{10}{n} = 2^{10} - 1$ aggregated cells. Then there are in all $3 \times (2^{10} - 1) = 3069$ aggregated cells.
2. Now we delete the duplicate ones. Note that in each base cells, there are 7 dimensions which are the same: $c_4, \ldots, c_9, c_{10}$.
   So when we roll up to $(*, *, *, c_4, \ldots, c_9, c_{10}) : 1$, all the cells that are left to be aggregated will be the same. Thus we can simply combine all those cells and the count for each of those cells is 3:
   $(*, *, *, *, c_5, \ldots c_{10}) : 3$
   $(*, *, *, *, *, c_6, \ldots, c_{10}) : 3$
   ......
   $(*, *, *, *, *, *, *, *, *, *) : 3$
   This leave us with $2 \times \sum_{n=1}^{7} \binom{7}{1} + 2 = 2 \times (2^7 - 1) + 2 = 256$ duplicate aggregated cells. (+2 means we have 2 duplicate $(*, *, *, c_4, \ldots c_{10})$ cells)

   Thus we get $3069 - 256 = 2813$ distinct aggregated cells

**c.**

The condition for iceberg cube here is count > 2. These cells are just the cells that we combined in part b. Namely:

$(*, *, *, c_4, \ldots c_{10}) : 3$
$(*, *, *, *, c_5, \ldots c_{10}) : 3$
$(*, *, *, *, *, c_6, \ldots, c_{10}) : 3$
.......
$(*, *, *, *, *, *, *, *, *, *) : 3$
Thus we have $\sum_{n=1}^{7} \binom{7}{1} + 1 = (2^7 - 1) + 1 = 128$ distinct aggregated cells.

**d.**
By definition:

- A cell, c, is a closed cell if there exists no cell: d, such that d is a specialization (descendant) of cell c and d has the same measure value as c.

Using this definition, we see there are only 4 closed cell in this data cube:

- $(a_1, a_2, a_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}) : 1$
- $(b_1, b_2, b_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}) : 1$
- $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}) : 1$
- $(*, *, *, c_4, \ldots c_{10}) : 3$
  Thus the closed cell with count = 3 has **7** non-star dimensions.

# Question 2

**a.** As this a cube with concept hierarchy, we use the following formula

- $T = \prod_{k=1}^{n} (L_k + 1)$
  Since Location dimension has two levels: we get
  $T = (2 + 1) \times (1 + 1) \times (1 + 1) \times (1 + 1) = 24$
  Hence, there are **24** cuboids in this cube.

**b.** I use pandas library to handle this problem which will be rather simple.

- I first find all the cells in the cuboid('City', 'Category', 'Price', 'Rating') and use the drop_duplicates() function to get the distinct cells of the cuboid.

In [1]:

```
import pandas as pd
header = ['Business id', 'State', 'City', 'Category', 'Price', 'Rating']
cube = pd.read_csv('Q2data.csv', names = header)

cuboid_1 = cube[['City', 'Category', 'Price', 'Rating']].drop_duplicates()
print(len(cuboid_1))
```

```
48
```

Thus, there are **48** cells in the cuboid (Location(city), Category, Rating, Price).

**c.** By the same token:

In [2]:

```
cuboid_2 = cube[['State', 'Category', 'Price', 'Rating']].drop_duplicates()
print(len(cuboid_2))
```

```
34
```

Thus, there are **34** cells in the cuboid (Location(State), Category, Rating, Price).

**d.**Also by the same token:

In [3]:

```
cuboid_3 = cube[['Category', 'Price', 'Rating']].drop_duplicates()
print(len(cuboid_3))
```

```
23
```

Thus, there are **23** cells in the cuboid (Category, Price, Rating).

**e.** Running the code get us the count:

In [4]:

```
print(cube.loc[(cube['State'] == 'Illinois') & (cube['Rating'] == 3) & (cube['Pr
ice'] == 'moderate')])
```

```
    Business id     State     City Category     Price  Rating
6             6  Illinois   Aurora  clothes  moderate       3
45           45  Illinois  Chicago     food  moderate       3
```

Thus, the count for the cell (Location(state) = 'Illinois' , , *rating = 3 , Price = 'Moderate') is* **2**.

**f.** Using the same method:

In [5]:

```
print(cube.loc[(cube['City'] == 'Chicago') & (cube['Category'] == 'food')])
```

```
    Business id     State     City Category      Price  Rating
31           31  Illinois  Chicago     food  expensive       3
45           45  Illinois  Chicago     food   moderate       3
```

Thus, the count for the cell (Location(city) = 'Chicago' , Category='food' , , ) is **2**.

# Question 3

**a.**

When the minimum support is 20, we want to find all patterns that has support >= 20.

In [6]:

```
import csv
import copy
import pprint
from collections import defaultdict
from itertools import combinations
reader = csv.reader(open('Q3data'), delimiter = ' ')
transaction_list = list(reader)


Li = []
countList = []
dict_list = []

for i in range(1, 5):
        new_dict = {}
        count = 0
        freq_item = []
        freq_dict = defaultdict(lambda : 0)
        transactions_temp = []
        for transaction in transaction_list:
                transaction = list(combinations(transaction,i))
                transactions_temp.append(transaction)

        for transaction in transactions_temp:
                for item in transaction:
                        freq_dict[item] += 1

        for item in freq_dict:
                if(freq_dict[item] >= 20):
                        freq_item.append(item)
                        count += 1
                        new_dict[item] = freq_dict[item]

        dict_list.append(new_dict)
        countList.append(count)
        Li.append(freq_item)


pprint.pprint(dict_list)
print('Number of frequent 1-itemset is {0}'.format(countList[0]))
print('Number of frequent 2-itemset is {0}'.format(countList[1]))
print('Number of frequent 3-itemset is {0}'.format(countList[2]))
print('Number of frequent 4-itemset is {0}'.format(countList[3]))
print('Number of frequent patterns is {0}'.format(sum(countList)))
```

```
[{('A',): 64,
  ('B',): 54,
  ('C',): 83,
  ('D',): 28,
  ('E',): 66,
  ('F',): 29,
  ('G',): 34},
 {('A', 'B'): 37,
  ('A', 'C'): 52,
  ('A', 'E'): 44,
  ('A', 'F'): 20,
  ('A', 'G'): 22,
  ('B', 'C'): 47,
  ('B', 'E'): 34,
  ('B', 'G'): 21,
  ('C', 'D'): 23,
  ('C', 'E'): 56,
  ('C', 'F'): 28,
  ('C', 'G'): 32,
  ('E', 'F'): 25,
  ('E', 'G'): 22},
 {('A', 'B', 'C'): 31,
  ('A', 'B', 'E'): 24,
  ('A', 'C', 'E'): 38,
  ('A', 'C', 'G'): 20,
  ('B', 'C', 'E'): 32,
  ('B', 'C', 'G'): 20,
  ('C', 'E', 'F'): 25,
  ('C', 'E', 'G'): 21},
 {('A', 'B', 'C', 'E'): 23}]
Number of frequent 1-itemset is 7
Number of frequent 2-itemset is 14
Number of frequent 3-itemset is 8
Number of frequent 4-itemset is 1
Number of frequent patterns is 30
```

Thus we get:

1. The number of frequent patterns is **30**
2. The number of frequent patterns with length 3 is **8**

By implementing the definition of max pattern (A pattern X is a max-pattern if X is frequent and there exists no frequent super-pattern Y containing X):

In [7]:

```
length = len(Li)
should_remove = []
for i in range(0, length):
        if(i+1<length):
                for j in range(0,len(Li[i+1])):
                                a = set(combinations(Li[i+1][j],i+1))
                                for it in Li[i]:
                                        if it in a:
                                                should_remove.append(it)

combined = [item for sublist in Li for item in sublist]
max_pattern = list(set(combined)- set(should_remove))
pprint.pprint(max_pattern)
print('The number of max patterns is {0}'.format(len(max_pattern)))
```

```
[('A', 'B', 'C', 'E'),
 ('C', 'E', 'F'),
 ('C', 'E', 'G'),
 ('B', 'C', 'G'),
 ('C', 'D'),
 ('A', 'C', 'G'),
 ('A', 'F')]
The number of max patterns is 7
```

**b.**

Now we want to find all patterns that have support >= 10. So repeating the above code and changing **minimum support** to 10:

In [8]:

```python
Li = []
countList = []
dict_list = []

for i in range(1, 6):
        new_dict = {}
        count = 0
        freq_item = []
        freq_dict = defaultdict(lambda : 0)
        transactions_temp = []
        for transaction in transaction_list:
                transaction = list(combinations(transaction,i))
                transactions_temp.append(transaction)

        for transaction in transactions_temp:
                for item in transaction:
                        freq_dict[item] += 1

        for item in freq_dict:
                if(freq_dict[item] >= 10):
                        freq_item.append(item)
                        count += 1
                        new_dict[item] = freq_dict[item]

        dict_list.append(new_dict)
        countList.append(count)
        Li.append(freq_item)
pprint.pprint(dict_list)
print('Number of frequent 1-itemset is {0}'.format(countList[0]))
print('Number of frequent 2-itemset is {0}'.format(countList[1]))
print('Number of frequent 3-itemset is {0}'.format(countList[2]))
print('Number of frequent 4-itemset is {0}'.format(countList[3]))
print('Number of frequent patterns is {0}'.format(sum(countList)))
```

```
[{('A',): 64,
  ('B',): 54,
  ('C',): 83,
  ('D',): 28,
  ('E',): 66,
  ('F',): 29,
  ('G',): 34},
 {('A', 'B'): 37,
  ('A', 'C'): 52,
  ('A', 'D'): 16,
  ('A', 'E'): 44,
  ('A', 'F'): 20,
  ('A', 'G'): 22,
  ('B', 'C'): 47,
  ('B', 'D'): 14,
  ('B', 'E'): 34,
  ('B', 'F'): 15,
```

```
  ('B', 'G'): 21,
  ('C', 'D'): 23,
  ('C', 'E'): 56,
  ('C', 'F'): 28,
  ('C', 'G'): 32,
  ('D', 'E'): 19,
  ('E', 'F'): 25,
  ('E', 'G'): 22},
 {('A', 'B', 'C'): 31,
  ('A', 'B', 'E'): 24,
  ('A', 'B', 'F'): 11,
  ('A', 'B', 'G'): 14,
  ('A', 'C', 'D'): 14,
  ('A', 'C', 'E'): 38,
  ('A', 'C', 'F'): 19,
  ('A', 'C', 'G'): 20,
  ('A', 'D', 'E'): 13,
  ('A', 'E', 'F'): 17,
  ('A', 'E', 'G'): 17,
  ('B', 'C', 'D'): 12,
  ('B', 'C', 'E'): 32,
  ('B', 'C', 'F'): 14,
  ('B', 'C', 'G'): 20,
  ('B', 'E', 'F'): 13,
  ('B', 'E', 'G'): 11,
  ('C', 'D', 'E'): 16,
  ('C', 'E', 'F'): 25,
  ('C', 'E', 'G'): 21},
 {('A', 'B', 'C', 'E'): 23,
  ('A', 'B', 'C', 'F'): 10,
  ('A', 'B', 'C', 'G'): 13,
  ('A', 'B', 'E', 'G'): 10,
  ('A', 'C', 'D', 'E'): 12,
  ('A', 'C', 'E', 'F'): 17,
  ('A', 'C', 'E', 'G'): 16,
  ('B', 'C', 'E', 'F'): 13,
  ('B', 'C', 'E', 'G'): 11},
 {('A', 'B', 'C', 'E', 'G'): 10}]
Number of frequent 1-itemset is 7
Number of frequent 2-itemset is 18
Number of frequent 3-itemset is 20
Number of frequent 4-itemset is 9
Number of frequent patterns is 55
```

So we get the following answers:

1. The number of frequent patterns is **55** .
2. The number of frequent patterns with length 3 is **20.**

Now we calculate the **number of max patterns** using the original code below:

In [9]:

```
length = len(Li)
should_remove = []
for i in range(0, length):
        if(i+1<length):
                for j in range(0,len(Li[i+1])):
                                a = set(combinations(Li[i+1][j],i+1))
                                for it in Li[i]:
                                        if it in a:
                                                should_remove.append(it)


combined = [item for sublist in Li for item in sublist]
max_pattern = list(set(combined)- set(should_remove))
pprint.pprint(max_pattern)
print('The number of max patterns is {0}'.format(len(max_pattern)))
```

```
[('A', 'B', 'C', 'E', 'G'),
 ('B', 'C', 'E', 'F'),
 ('A', 'C', 'D', 'E'),
 ('A', 'B', 'C', 'F'),
 ('A', 'C', 'E', 'F'),
 ('B', 'C', 'D')]
The number of max patterns is 6
```

1. The number of max patterns is **6**.

In order to calculate the confidence, we use the confidence measure of the **Association Rule**:

- $confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \bigcup B)}{support(A)} = \frac{support_{count}(A \bigcup B)}{support_{count}(A)}$

In [10]:

```
con1 = dict_list[2][('A', 'C', 'E')]/ dict_list[1][('C', 'E')]
print('4. The confidence measure of the association rule (C, E) -> A is {0}\n'.f
ormat(con1))
con2 = dict_list[3][('A', 'B', 'C', 'E')]/ dict_list[2][('A', 'B', 'C')]
print('5. The confidence measure of the association rule (A, B, C) -> E is {0}'.
format(con2))
```

```
4. The confidence measure of the association rule (C, E) -> A is 0.6
785714285714286

5. The confidence measure of the association rule (A, B, C) -> E is
0.7419354838709677
```

Hence:

- The confidence measure of the association rule (C, E) -> A is 0.679.
- The confidence measure of the association rule (A, B, C) -> E is 0.742.