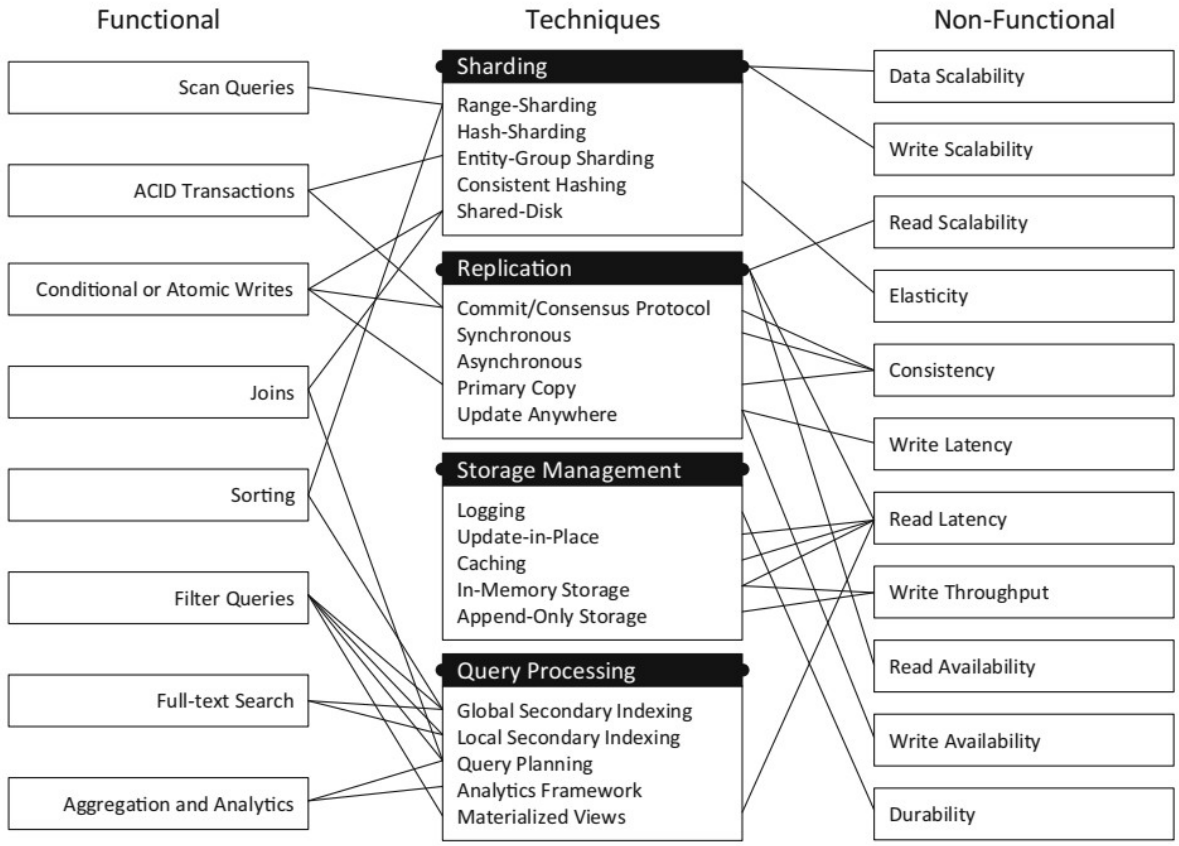
Question-1:

a)



b) 1- S, H, A, S
2- U, M, C, E
3- S, H, A, S
4- S, M, A/C, S/E

Question-2:

Design criteria

1. Completeness
– All aspects of the information needed by the applications should be covered
2. Soundness
– All information aspects and relationships are modeled correctly
3. Minimality
– No unnecessary or logically redundant information should be modeled
– (however, sometimes redundancy enhances performance)
4. Readability
– No complex encoding should be used to describe the information
– Entity and attribute names should be self-explanatory
5. Modifiability
– Changes to the data structure should be possible
6. Modularity
– The data model should be devided into subsets that form logically coherent entities
– Also helps with modifiability

Question-3:

a) Hash-based distribution of data

Dave explains the benefits of hash-based data distribution as follows: Hash-based data distribution provides efficient data retrieval by creating a one- to one mapping between keys and database locations, allowing direct access to data without searching, In addition, hash-based data distribution can handle complex queries involving joins, range-based access or sorting.

As often, he mixes up things and you need to correct this statement. Explain to him

why. Start your sentence with "Sorry Dave, but…"

Ans: Sorry Dave, but your statement is incorrect. Hash-based data distribution does not provide a one-to-one mapping between keys and database locations. In hash-based distribution, data is divided into multiple partitions or buckets based on the hash value of a key. Multiple keys can hash to the same location, resulting in a one-to-many mapping rather than a one-to-one mapping.

b) **Certainly! Let's tackle this problem step by step.**
Given the database table structure and data:

OrderID	CustomerName	Product	Quantity
1	John	Apples	20
2	Jane	Oranges	15
3	Mary	Bananas	10
4	John	Oranges	5
5	Jane	Apples	25
6	Mary	Bananas	30

We need to design a simple hash function that distributes the data based on the 'CustomerName' field into three available buckets using ASCII letter codes of strings. Here's how we can approach this:

1. Hash Function Design:
    - We'll create three buckets (let's call them A, B, and C).
    - For each customer name, we'll calculate the sum of the ASCII values of its characters.
    - Then, we'll take the modulo of this sum with 3 to determine which bucket the customer belongs to.
2. Hash Function Example:
    - Let's apply the hash function to the given data:

- John: ASCII sum = 74 + 111 + 104 + 110 = 399. Modulo 3 = 0 (Bucket A).
- Jane: ASCII sum = 74 + 97 + 110 + 101 = 382. Modulo 3 = 2 (Bucket C).
- Mary: ASCII sum = 77 + 97 + 114 + 121 = 409. Modulo 3 = 1 (Bucket B).

3. Uniform Distribution:

- Our hash function ensures uniform distribution because it uses the modulo operation with 3.
- Since there are three buckets, each customer name will be assigned to one of the buckets.
- The modulo operation evenly distributes the data across the buckets, preventing clustering.

In summary, our simple hash function based on ASCII letter codes ensures that the data is uniformly distributed among the three buckets. 🌟

c) Let's analyze the problem and demonstrate how a hash function distributes the sample data tuples using the provided ASCII table for calculations.
The given problem asks us to design a hash function that distributes sample data tuples into three buckets. We'll use the ASCII table below for our calculations:

```
A 65 B 66 C 67 D 68 E 69 F 70 G 71 H 72 I 73 J 74 K 75 L
76 M 77 N 78 O 79 P 80 Q 81 R 82 S 83 T 84 U 85 V 86 W 87
X 88 Y 89 Z 90
a 97 b 98 c 99 d 100 e 101 f 102 g 103 h 104 i 105 j 106
k 107 l 108 m 109 n 110 o 111 p 112 q 113 r 114 s 115 t
116 u 117 v 118 w 119 x 120 y 121 z 122
```

Now let's proceed step by step:

1. Hash Function Design:

- We'll use a simple hash function that calculates the sum of ASCII values for each character in a given string.
- The hash function will then take the modulo of this sum with the number of buckets (which is 3 in our case).

2. Sample Data Tuples:

- We have three sample data tuples: John, Jane, and Mary.

3. Calculations:

- For each name, we'll calculate the sum of ASCII values:
  - John: J=74, o=111, h=104, n=110. Total = 399.

- Jane: J=74, a=97, n=110, e=101. Total = 382.
- Mary: M=77, a=97, r=114, y=121. Total = 409.

4. Modulo Operation:
    - Now we'll take the modulo of each total with 3 (the number of buckets):
        - John: 399 % 3 = 0 (Bucket A).
        - Jane: 382 % 3 = 2 (Bucket C).
        - Mary: 409 % 3 = 1 (Bucket B).
5. Result:
    - John goes into Bucket A.
    - Jane goes into Bucket C.
    - Mary goes into Bucket B.

Our hash function ensures that the data is distributed evenly among the three buckets based on the ASCII values of the characters.

Question-4:

a) MongoDB Query Language

b) MongoDB

c) No

d) It will returns the first name with email body which are send by them.

e) Cypher

f) Neo4j DB

g) No, to get the desired result we have to change the matching attribute from "Email_To" to "Email_From". Following query can be the solution:

```
MATCH (n:Email) - [:EMAIL_FROM] - (p:Person{firstname: "Joe"})
WHERE n.email_date > "2001-09-01 00:00:00+0000" RETURN n, p
```

Question-5:

a) 4th

b) 4th

c) 1st

d) It will write on disk log, after that to memtable. And if memtable gets full, it will write those data to SDF and clear the memtable.

f) except 1st one.

Question-6:

a) Scalability, Sparseness, Scalability, Volume, *Modifiability, Self description ability, Modifiability, Reliability.*

b) CAP consistency refers to the requirement that all nodes in a distributed system have the same view of data at the same time. It emphasizes strong coordination and synchronization across nodes.

ACID consistency, on the other hand, ensures the correctness and integrity of data within individual transactions, enforcing integrity constraints and business rules.

c)

- improves performance
- Ensures better latency
- Higher Throughput
- Helps in writing durability