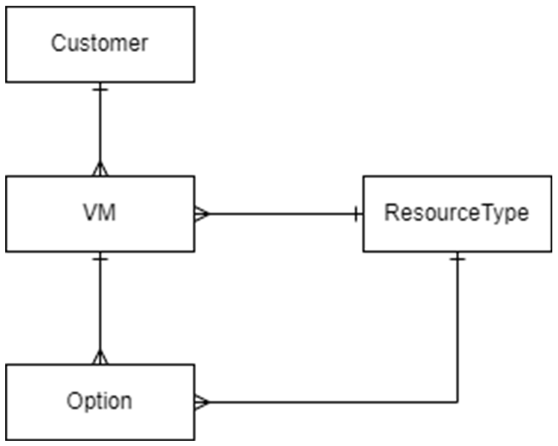
Solutions for 2023 SH2 Prelim Paper 1

1)

a)



All Relationships + Entities [4]

b)

1) Customer (<u>CustID</u>, CustName, Email, Contact, Address)

2)

   Resource = enum (VM, VD, VNet)
   - ResouceType (<u>TypeID</u>, Resource, Type, Description, Price)

OR
   - ResouceType (<u>Resource, Type</u>, Description, Price)
   -

3) VM (<u>VMID</u>, CustID*, TypeID*, Hours)

OR

   VM(<u>VMID</u>, CustID*, Resource*, Type*, Hours)

4)

   Option (<u>OptionID,</u> VMID*, Resource*, Type*)
   //The same vm can have multiple same resource and type, eg 1 vm can have 2 or more small vdisks.

OR

   Option (<u>OptionID</u>, VMID*, TypeID*)

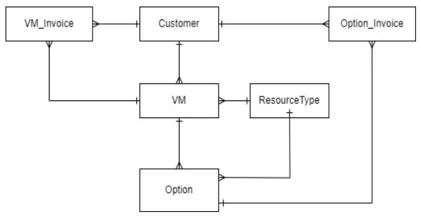
[4] Correct PK,FK

c) An unnormalised database is a database that **hasn't been structured** to reduce data redundancy. This means that the same piece of **data might be stored in multiple places**, which can lead to (update/delete) **inconsistency, update/delete/insert anomalies and inefficiencies [1]**
Potiential errors: **[2]**
- Update only a partial set of records
- Inserting a record with incomplete data (empty fields)
- Unintended deleting of records that are inter-related

d)
```
SELECT  VM.VMID, VM.Hours, ResourceType.Resource, ResourceType.Type,
ResourceType.Price
FROM VM
INNER JOIN ResourceType ON ResourceType.Resource = "VM" AND
ResourceType.Type = VM.Type
WHERE VM.CustID =876542
   -  JOIN [1]
   -  Correct attributes/table selected [1]
   -  WHERE [1]
```

e)
```
SELECT VM.VMID, Option.Resource, Option.Type, ResourceType.Price
FROM VM
INNER JOIN Option ON VM.VMID = Option.VMID
INNER JOIN ResourceType ON Option.Resource = ResourceType.Resource
AND Option.Type = ResourceType.Type
WHERE VM.CustID = 876542
   -  JOIN [2]
   -  Correct attributes/table selected [1]
   -  WHERE [1]
```

f)
Create additional tables :
- vm_invoice (year, month, custid*, vmid*, vm_price)
- option_invoice (year, month, custid*, optionid*, price)



Before the beginning of the billing cycle, the resources used by each customer will be created with the current price.

OR update the change in price only after the invoice for the current price has been sent to the customer.

g)
Data Security : (C.I.A) [1]
- (C) Ensure that proper authentication is enforced when a customer tries to access the virtual disk.
- (C) Ensure that access control is enforced to allow only authorised user to access the virtual disk
- (I, A) Ensure that there are backup of the virtual disk created by the user.
- Ensure that fault tolerance hardware are used to provide continuous accessibility

Data Retention period: [1]
- Ensure that customers' data, including backups are completely removed when customer has unsubscribed from CloudCompute's services.

Data Portability/Transfer [1]
- Ensure that the data stored in CloudCompute can be migrated out from the platform.

h) [2] Any
- RDB has a fixed schema, NoQL has no fixed schema and store unstructured data
- NoSQL can scale horizontally, RDB can scale vertically
- NoSQL does not check integrity rules during transactions, RDB perform transaction integrity check

i) React quickly to business demands by providing new services. A NoSQL supports changes to new data requirements quickly [1]

2)

(a)

```
01 def ternary_search(A, target):
02     lb = 0
03     ub = len(A)-1
04     while lb <= ub:
05         increment = (ub-lb)//3
06         index_1 = lb + increment
07         index_2 = index_1  + increment
08         if target == A[index_1] :
09             return index_1
10         elif target < A[index_1]:
11             ub = index_1 -1
12         elif target == A[index_2]:
13             return index_2
14         elif target < A[index_2]:
15             lb = index_1+1
16             ub = index_2-1
17         else:
18             lb = index_2 + 1
19     else:
20         return -1
```

- **calculate the 2 indexes [2]**
- **check for termination conditions line 8,12,20 [2]**
- **check for iteration consitions line 10, 14,18 [2]**

(b)

Array ←

0	1	2	3	4	5	6
6	10	27	57	82	83	98

target ← 10

lb<=ub	lb	ub	increment	index1	index2	Line 8	Line 10	Line 12	Line 14	line 17	return
True	0	6									
True	0	6	2	2	4		True				
True	0	1	0	0	0					True	
True	1	1	0	1	1	True					1

table contains
- variables for loop termination, lb, ub, index1, index2, return [2]
- correct flow [3]
    - indexes updated
    - line 10 update ub value
    - line 17 update lb value
    - line 8 return index value

(c)
In each iteration, the target is either found (line 8, line line 12) or **the search space is reduced by one third [1]** till it converges to a single element in the search space (lb>ub). Therefore the number of iteration in the worst case is $\mathbf{\log_3 N}$ **,** where N is the size of the array. The time complex is therefore $\mathbf{O(\log N)}$**[1]**

(d)
In each iteration, the **binary search reduces the search space by a factot of 2** where as the **ternary search reduces it by a factor of 3**. **Therefore the number of iterations is less for ternary search ($\log_3 N$) compare to $\log_2 N$ for binary search. [1]**

However inside each iterations, **the binary seach only makes 2 comparison whereas the ternary search needs to make 4 comparisons.[1]**
But for large value of N, they will have the **same performance $O(\log N)$ [1]**

3)
(a) [2]
- **Peer to Peer**
- **Client Server**

(b) [2]
- **Message to be transmitted are broken into packets**
- **Packets are routed independently to iheir destination**

(c) [2]
- **Resilience, able to sustain network devices failures by using a different route. Dedicated connection will fail if any devices used in the connection failed.**
- **Network resources are shared and are used on demand compare to a resources that are reserved and used in a dedicated connection.**

(d) [2]
- **Encryption/Decryption requires a key that must be provided to initiate the encryption/decryption process. Its purpose is to provide confidentiality.**
- **Encoding/Decoding does not need a key, it transform the input into a output in a different form that is more practical for transmission or storage**

(e) [2]
- **Symetric key: Uses the same key for both encryption/decryption**
- **Asymetric keys: A pair of keys (public and private) are used. If a public key is used for encryption, its corresponding private key must be used for decryption and vice versa.**

f) [2]
- **Compress a file into a smaller size before transmitting**
- **Low latency applications.**
- **The same encoding/decoding algorithm is used without requiring any keys to be acquired.**

g) [3]
- **Private key for Device A install on Device A**
- **Private key for Device B install on Device B**
- **Public keys for devices A and B should be used to enrol to be stored on certificates from a Certificate Authority.**

h)

- **Authenticating and binding the public key to a subject (device or person) [1]**
- **Contains [2]**
    - **public key, subject, expiry date, digital signature of issuer(CA)**

i) [2]

- **enrol a certificate from a certificate authority with the public key and identity of device A.**
- **a certificate Is then digitally signed with the private key of the trusted Certificate Authority.**

4)

(a) Prints the linked list in reverse order giving the integer value stored.[1]

(b) line 02 and 06 is the base case, line 09 is the recursive case [2]

(c)

```
class Node:
    def __init__(self):
        self.next = None
        self.data = None

    def __repr__(self):
        return f"{self.data}"

    def insert(self, data):
        ## make sure data is a single digit
        if data//10 != 0:
            print("NOT a single digit")
            return
        if self.data  == None: #empty linked list
            self.data = data
            return
        if self.next == None:
            self.next = Node() # Base case
            self.next.insert(data)
        else:
            self.next.insert(data)

def add(list1, list2)-> Node:
    carry = 0
    tmp_node = Node(None)
    while list1 or list2:
        a = list1.data if list1 else 0
        b = list2.data if list2 else 0
        carry, digit = divmod (a+b+carry, 10)
        insert(tmp_node,digit)
        list1 = list1.next if list1 else None
        list2 = list2.next if list2 else None

if carry:
        insert(tmp_node,carry)
    return tmp_node
```

- variables carry, new node created [1]
- while loop with termination [1]
- adding the two least significant digits with carry [2]
- add digit to new node [1]
- traverse the 2 linked list [1]
- check for carry [1]

(d)
On an embedded computer system with **limited memory resources**, the system can only allow an integer to be stored **in fixed size 8 bits of binary data**. This means that non negative integer can only have the **values from 0 to 255.**Two advantages of using a linked list to store the binary digits as nodes is that we can **allocate the nodes on demand [1]** and we can **extend the size of the integer value [1]**. The disadvantage is that there is still **memory overhead to store the pointers [1]**and **arithmetic operations may be more complex to implement.[1]**

OR

On a IoT embedded system, the sensor needs to detect certain events and keep a counter. If the embedded system uses 8-bit to represent integers, it can only count up to 255. Using a linked list allows the counter to store values larger than 255.

(e) Encapsulation, Inheritance and Polymorphism. [1]

(f)
The procedure Z uses the ADT Node and **access its properties directly [1].** The ADT is designed such that its properties can be accessed directly and modified by code not written by the creator of the ADT. This violates the **principle of encapsulation,[1]** where **the state/data of the ADT can only be changed and access by the provided operations of the ADT**.

(5)
(a) uses only 1 byte to store a a value from 0 to 127, last bit in the bytes is used for parity bit **[1]**

(b)- Lookup up table that allows any characters to be mapped to an integer value know as a code point.
- 3 Encoding scheme where the code point can be stored as
UTF-8 : Variable 1 byte, 2-bytes, 3-bytes or 4 bytes per code point
UTF-16: Variable 2 bytes or 4 bytes per code point
UTF-32 fixed 4 bytes per code point
[2]

(b)
(c)
UTF-16 uses 2 bytes or 4 bytes to store a character in the file. ASCII system **can only read 1 bytte as a character**. **[2]**

ASCII is compatible with UTF-8, as **the first 128 code points maps to the the 128 ascii values [1] ,** therefore all the characters stored as ASCII values can be decoded by UTF-8

(d) Alice to save as UTF-8
[1]

6)
a) Any 3 **[3m]**
- Deterministic: For the same input, the hash function should always generate the same hash value.

- Efficient to compute: The hash function should be capable of returning the hash value quickly to ensure the overall efficiency of the hash table operations.

- Uniform distribution: The hash function should provide a uniform distribution across the hash table to minimize collisions. That is, every hash bucket should be equally likely for any input.

- Minimize collisions: While collisions are inevitable, a good hash function should minimize them. Two different inputs producing the same output (a collision) can slow down lookup time.

- Use all the input data: The hash function should use all the data in the input. Ignoring parts of the input can lead to a higher likelihood of collisions.

b)
Collision happens when the hash function returns an address that has already being used.

Separate Chaining: In this method, each slot in the hash table is associated with a linked list. All elements that hash to the same index are placed in the same linked list. If a collision occurs, the new item is simply added to the end of the list. **[2]**

Linear Probing: In this method, if a collision occurs, we look for the next available slot in the hash table by sequentially look for the next available slot.When the sequential search cannot find an abilable slot after having probing the entire array, the hash table is full. **[2]**

c) Any 2 **[2]**

Fixed Size: Regardless of the size of the input data, the output hash length remains the same.

Collision Resistance: It should be extremely difficult to find two different inputs that produce the same hash output.

One-way function, meaning that the function cannot be used in reverse to obtain the original input.

d) It is used to protect the **integrity** of the message and provide **non-repudiation**. **[1]**

A digital signature is a fixed size output **obtained by hashing an input message**. It is then **encrypted with the message sender's private key.[2]**

e)
Passwords **are stored in hash form** in the web app database after a user has register his user id and password**.[1]** Client code on the web browser perform the hashing so that password is never transmitted in the clear on the Internet.

Hash function is used to **transform the entered password and then compared with the stored hashed password [1]** in the database.