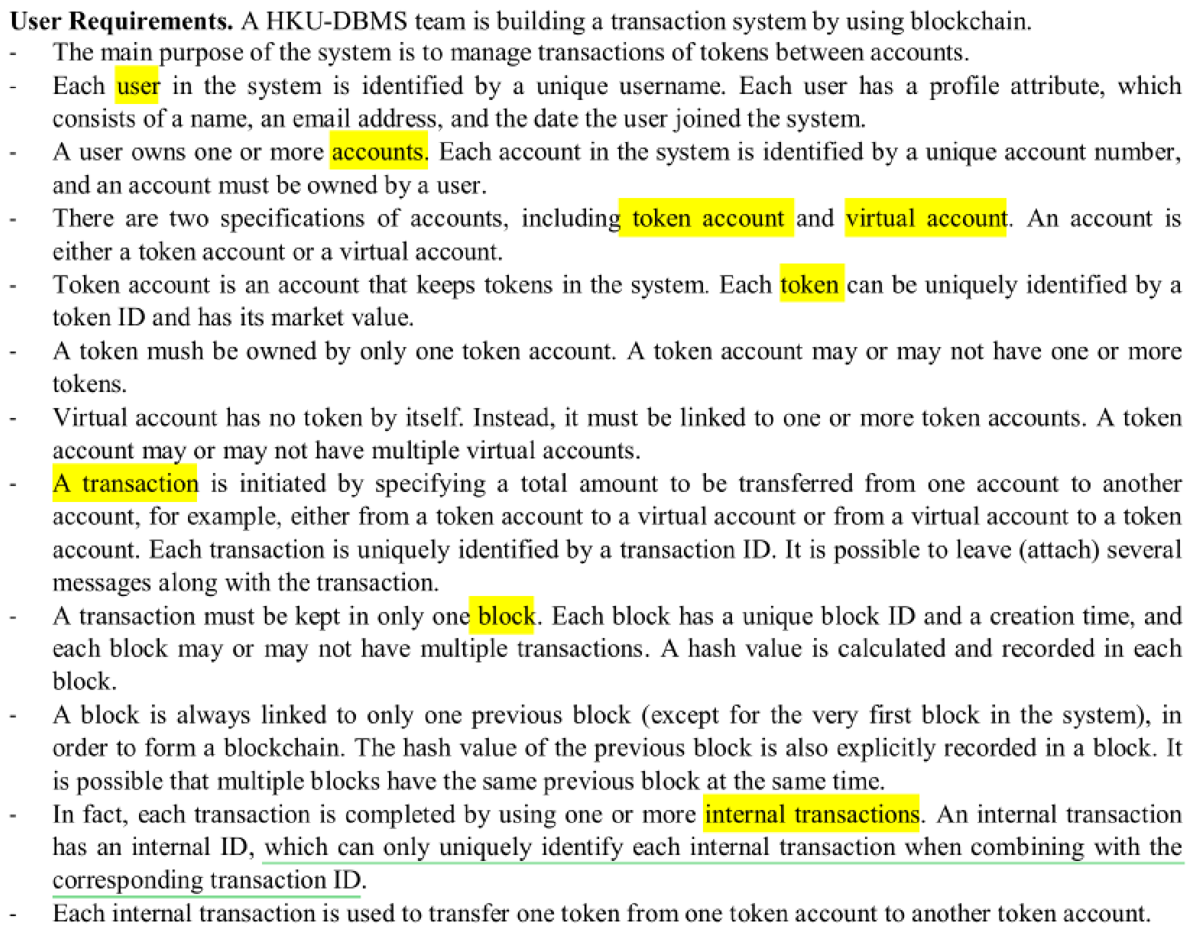
Assignment 1

COMP3278B 2020

Question 1 – (a)

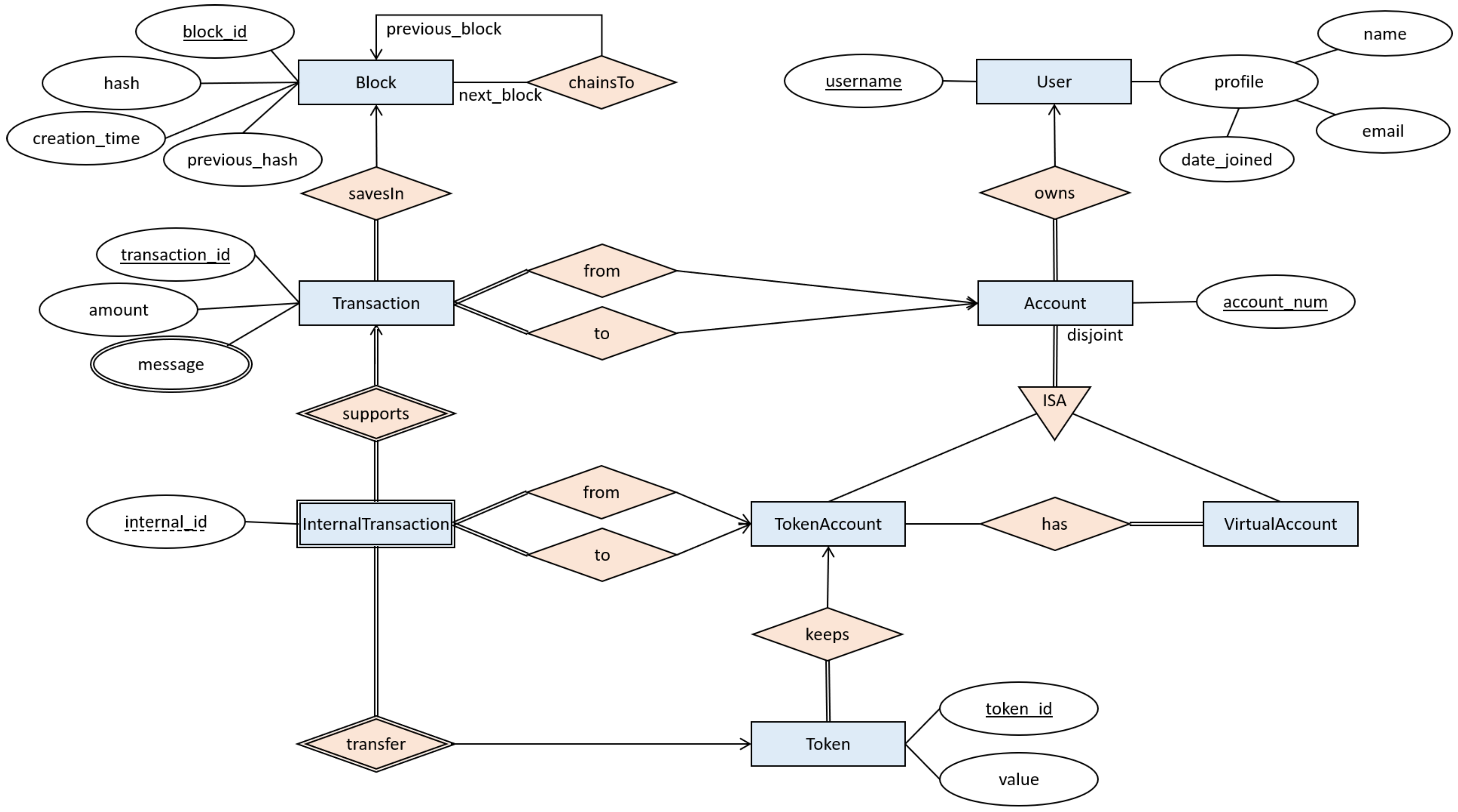
- Please model the users' requirements by drawing the corresponding E-R Diagram.



User Requirements. A HKU-DBMS team is building a transaction system by using blockchain.

- The main purpose of the system is to manage transactions of tokens between accounts.
- Each user in the system is identified by a unique username. Each user has a profile attribute, which consists of a name, an email address, and the date the user joined the system.
- A user owns one or more accounts. Each account in the system is identified by a unique account number, and an account must be owned by a user.
- There are two specifications of accounts, including token account and virtual account. An account is either a token account or a virtual account.
- Token account is an account that keeps tokens in the system. Each token can be uniquely identified by a token ID and has its market value.
- A token mush be owned by only one token account. A token account may or may not have one or more tokens.
- Virtual account has no token by itself. Instead, it must be linked to one or more token accounts. A token account may or may not have multiple virtual accounts.
- A transaction is initiated by specifying a total amount to be transferred from one account to another account, for example, either from a token account to a virtual account or from a virtual account to a token account. Each transaction is uniquely identified by a transaction ID. It is possible to leave (attach) several messages along with the transaction.
- A transaction must be kept in only one block. Each block has a unique block ID and a creation time, and each block may or may not have multiple transactions. A hash value is calculated and recorded in each block.
- A block is always linked to only one previous block (except for the very first block in the system), in order to form a blockchain. The hash value of the previous block is also explicitly recorded in a block. It is possible that multiple blocks have the same previous block at the same time.
- In fact, each transaction is completed by using one or more internal transactions. An internal transaction has an internal ID, which can only uniquely identify each internal transaction when combining with the corresponding transaction ID.
- Each internal transaction is used to transfer one token from one token account to another token account.

Question 1 – (a)


block_id
hash
creation_time
previous_hash
Block
previous_block
next_block
chainsTo
savesIn
transaction_id
amount
message
Transaction
from
to
supports
internal_id
InternalTransaction
from
to
transfer
username
User
profile
name
email
date_joined
owns
Account
account_num
disjoint
ISA
TokenAccount
has
VirtualAccount
keeps
Token
token_id
value

Question 1 – (b)

- Relationship
- Multi-valued Attributes
- Weak Entity Set
- Specialization

User (username, profile.name, profile.email profile.date_joined)

Account (account_num, username)
Foreign keys: username refers **User**

TokenAccount (account_num)
Foreign keys: account_num refers **Account**

VirtualAccount (account_num, token_account_num)
Foreign keys: token_account_num refers **TokenAccount**

Token (token_id, value, account_num)
Foreign keys: account_num refers **TokenAccount**

TokenAccount (account_num)
Foreign keys: account_num refers **Account**

VirtualAccount (account_num)
Foreign keys: account_num refers **Account**

Link_TokenAccount_VirtualAccount(token_account_num, virtual_account_num)
Foreign keys: token_account_num refers **TokenAccount**, virtual_account_num refers **VirtualAccount**

Question 1 – (b)

Block (block_id, hash, creation_time, previous_hash, previous_id)
Foreign keys: previous_id refers **Block**

Transaction (transaction_id, amount, from_account_num, to_account_num, block_id)
Foreign keys: from_account_num refers **Account**, to_account_num refers **Account**, block_id refers **Block**

TransactionMessage (transaction_id, message)
Foreign keys: transaction_id refers **Transaction**

InternalTransaction (transaction_id, internal_id, from_account_num, to_account_num, token_id)
Foreign keys: transaction_id refers **Transaction**, from_account_num refers **TokenAccount**, to_account_num refers **TokenAccount**, token_id refers **Token**

- Relationship
- Multi-valued Attributes
- Weak Entity Set
- Specialization

Question 2

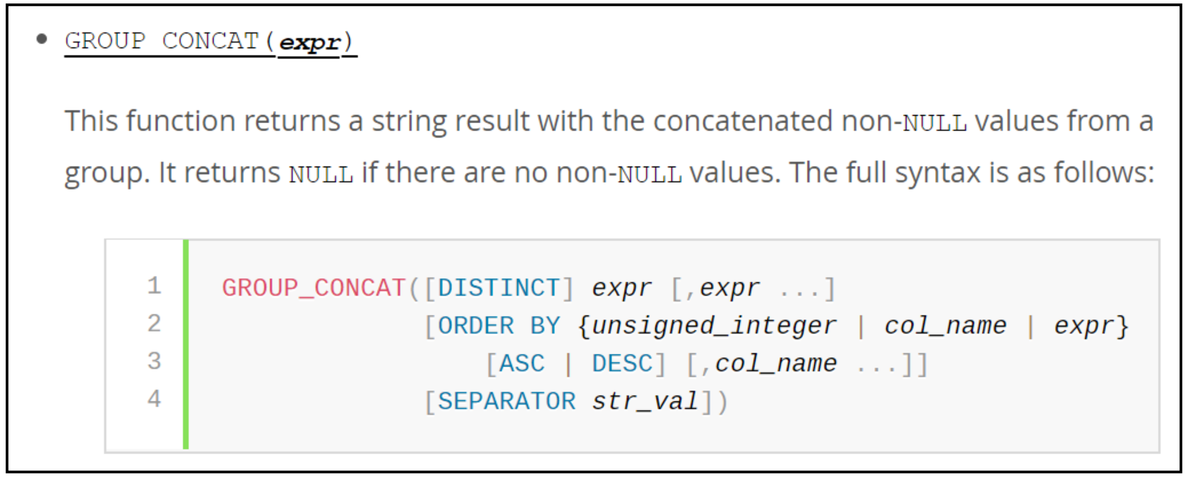
- APC Printing Company (APC) is a company specialized in producing prints on different media. The company wants to setup an online printing ordering system to allow customer to order their prints online. Their in-house developers have designed the ER diagram and produced a relational table schema with data type assigned.
- You are hired to join them in preparing the database for the system.

Question 2 – (c)

- Find the customer name, phone numbers, and order ID of all orders that has “New” in the status. If the customer has multiple phone numbers, concatenate them by commas in the result.

Question 2 – (3)

- This question require you to read the MySQL documentation on `GROUP_CONCAT`



- GROUP_CONCAT(*expr*)

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

This is NOT standard SQL, this is MySQL specific.

Question 2 – (3)

```
SELECT name, GROUP_CONCAT(phone SEPARATOR ','), PO.order_id
FROM PrintOrder PO, AppUser AU, CustomerPhone CP
WHERE PO.user_id = AU.user_id
      AND AU.user_id = CP.user_id
      AND PO.status LIKE '%New%'
GROUP BY PO.order_id, AU.user_id
```

1. Join table

2. Filter result

3. Group result (by order_id first)

4. Apply GROUP_CONCAT

Question 2 – (4) & (5)

- List all items of order ID 3, showing the item ID, quantity, product description, and the subtotal. Subtotal equals to the product price times quantity.
- List all orders by customer with user ID 5, showing the order ID, status, number of items (sum of quantity), and the total amount.

Question 2 – (4) & (5)

- The two questions are related, part (5) needs to sum up the results from (4).
  - Which implies that the query in (4) could be a subquery in (5) except for the columns selected.

(4)

```
SELECT item_id, quantity, description, quantity * price AS subtotal
FROM Item It, Product Pr
WHERE It.product_id = Pr.product_id AND order_id = 3
```

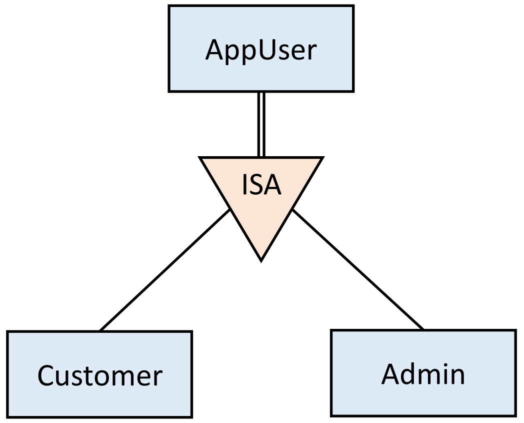
(5)

```
SELECT PO.order_id, status, SUM(quantity) AS items, SUM(subtotal) AS amount
FROM PrintOrder PO, (
  SELECT order_id , quantity, quantity * price AS subtotal
  FROM Item It, Product Pr
  WHERE It.product_id = Pr.product_id
) T1
WHERE user_id = 5 AND PO.order_id = T1.order_id
GROUP BY PO.order_id;
```

Question 2 – (6)

- Find the names of all users that is a customer and an admin user.

Question 2 – (6)



- From the ER-D, the ISA relationship is not disjoint
    - Otherwise there won’t be any result for this question.

```
SELECT name
FROM AppUser
WHERE user_id IN (
  SELECT user_id
  FROM Customer
) AND user_id IN (
  SELECT user_id
  FROM Admin
)
```

Alternatives: use INNER JOIN

Note: There is no INTERSECT in MySQL

Question 2 – (7)

- Find all admin users is not a supervisor.

Question 2 – (7)

Find the users who are supervisors

```
SELECT supervisor_id
FROM Admin
WHERE supervisor_id IS NOT NULL;
```

```
SELECT *
FROM Admin
WHERE user_id NOT IN (
      SELECT supervisor_id
      FROM Admin
      WHERE supervisor_id IS NOT NULL
);
```

Question 2 – (8)

- List all albums owned by admin users, in a descending order of the number of images in the album. If there exist two albums with the same number of images, order them in descending order of the creation date.

Question 2 – (8)

```
SELECT Al.*
FROM Album Al, Image Im
WHERE Al.album_id = Im.album_id
      AND user_id IN (
           SELECT user_id FROM Admin
      )
GROUP BY Al.album_id
ORDER BY COUNT(*) DESC, creation_date DESC;
```

2. Join tables

1. Filter results

3. Apply GROUP BY, ORDER BY

Question 2 – (9)

- Find the number of times each of the images in album_ID 1 is included in an order. An image that is placed in the same order multiple times should be counted as once only. Images that has never been ordered must also be shown.

Question 2 – (9)

Attempt 1

```
SELECT Im.image_id, COUNT(*)
FROM Image Im, Item It
WHERE Im.album_id = It.album_id AND Im.image_id = It.image_id AND Im.album_id = 1
GROUP BY Im.image_id;
```

✗

- INNER JOIN is not enough for this question
  - We need to show zero count

Attempt 2

```
SELECT Im.image_id, COUNT(*)
FROM Image Im
LEFT JOIN Item It ON Im.album_id = It.album_id AND Im.image_id = It.image_id
WHERE AND Im.album_id = 1
GROUP BY Im.image_id;
```

✗

- LEFT JOIN is good, but COUNT(*) will double count
  - If an image is put in the same order twice, the count will be two

Question 2 – (9)

- Possible solution: COUNT DISTINCT
  - Here we cound only distinct order_id for each item.

```
SELECT Im.image_id, COUNT(DISTINCT It.order_id)
FROM Image Im
LEFT JOIN Item It ON Im.album_id = It.album_id AND Im.image_id = It.image_id
WHERE Im.album_id = 1
GROUP BY Im.image_id;
```

Alternatives:
In a subquery, generate a list of distinct order_id and image_id, then find the COUNT in the main query.

Question 2 – (10)

- Find all customer name of orders that has the greatest number of order items (not considering quantity)

Question 2 – (10)

- This is another typical question that find records with a maximum/minimum count.
  - You should now be familiar with this with the previous practices.

```
SELECT name
FROM AppUser
WHERE user_id IN (
  SELECT user_id
  FROM PrintOrder PO, Item It
  WHERE PO.order_id = It.order_id
  GROUP BY PO.order_id
  HAVING COUNT(*) >= ALL(
    SELECT COUNT(*)
    FROM PrintOrder PO, Item It
    WHERE PO.order_id = It.order_id
    GROUP BY PO.order_id
  )
)
```