Question 01

1. Suppose you are a manufacturer of product ABC, which is composed of parts A, B, and C. Each time a new product is created, it must be added to the product inventory, using the PROD_QOH in a table named PRODUCT. And each time the product ABC is created, the parts inventory, using PART_QOH in a table named PART, must be reduced by one each of parts A, B, and C. The sample database contents are shown in Table P10.1

Table P10.1 The Database for Problem 1

Table name: PRODUCT

PROD_CODE	PROD_QOH
ABC	1,205

Table name: PART

PART_CODE	PART_QOH
A	567
B	498
C	549

Given that information, answer Questions a through e.

a. How many database requests can you identify for an inventory update for both PRODUCTand PART?

Depending in how the SQL statements are written, there are two correct answers: 4 or 2.

b. Using SQL, write each database request you identified in problem 1.

The database requests are shown in the following table.

Four SQL statements	Two SQL statements
UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'B' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'C'	UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' OR PART_CODE = 'B' OR PART_CODE = 'C'

c. Write the complete transaction(s).

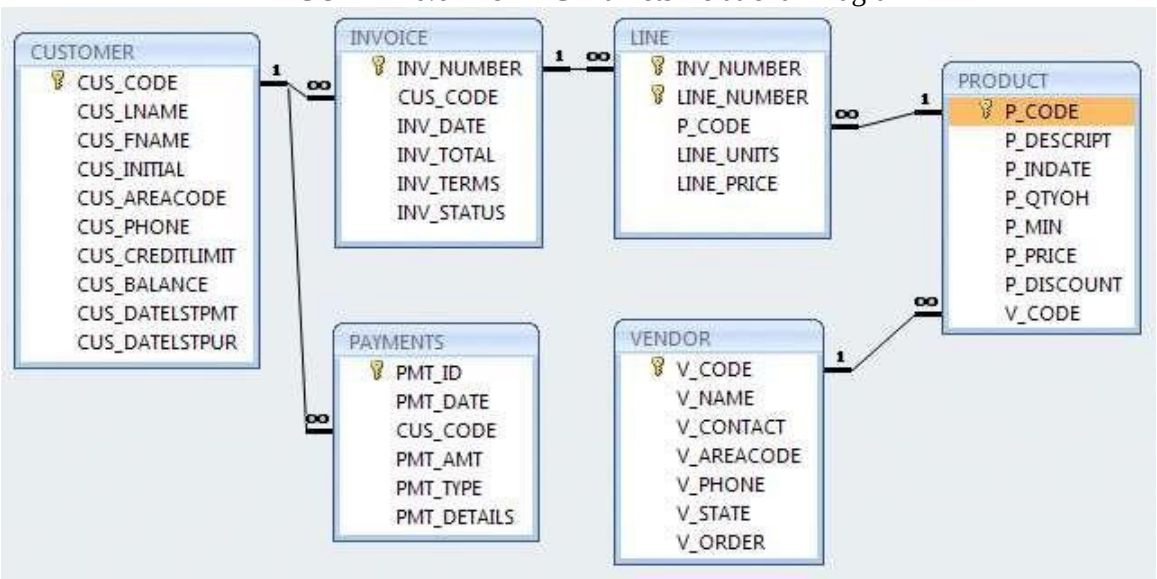
The transactions are shown in the following table.

Four SQL statements	Two SQL statements
BEGIN TRANSACTION UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'B' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'C' COMMIT;	BEGIN TRANSACTION UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' OR PART_CODE = 'B' OR PART_CODE = 'C' COMMIT;

Question 2

2. ABC Markets sell products to customers. The relational diagram shown in Figure P10.6 represents the main entities for ABC's database.

FIGURE P10.6 The ABC Markets Relational Diagram



Note the following important characteristics:

- A customer may make many purchases, each one represented by an invoice.
  - The CUS_BALANCE is updated with each credit purchase or payment and represents the amount the customer owes.
  - The CUS_BALANCE is increased (+) with every credit purchase and decreased (-) with every customer payment.

- The date of last purchase is updated with each new purchase made by the customer.
- The date of last payment is updated with each new payment made by the customer.

- An invoice represents a product purchase by a customer.
  - An INVOICE can have many invoice LINEs, one for each product purchased.
  - The INV_TOTAL represents the total cost of invoice including taxes.
  - The INV_TERMS can be "30," "60," or "90" (representing the number of days of credit) or "CASH," "CHECK," or "CC."
  - The invoice status can be "OPEN," "PAID," or "CANCEL."
- A product's quantity on hand (P_QTYOH) is updated (decreased) with each product sale.
- A customer may make many payments. The payment type (PMT_TYPE) can be one of the following:
  - "CASH" for cash payments.
  - "CHECK" for check payments
  - "CC" for credit card payments
- The payment details (PMT_DETAILS) are used to record data about check or credit card payments:
  - The bank, account number, and check number for check payments
  - The issuer, credit card number, and expiration date for credit card payments.

Note: Not all entities and attributes are represented in this example. Use only the attributes indicated.
Using this database, write the SQL code to represent each one of the following transactions. Use BEGIN TRANSACTION and COMMIT to group the SQL statements in logical transactions.

a. On May 11, 2012, customer '10010' makes a credit purchase (30 days) of one unit of product '11QER/31' with a unit price of $110.00; the tax rate is 8 percent. The invoice number is 10983, and this invoice has only one product line.

a. BEGIN TRANSACTION
b. INSERT INTO INVOICE
    i. VALUES (10983, '10010', '11-May-2012', 118.80, '30', 'OPEN');
c. INSERT INTO LINE
    i. VALUES (10983, 1, '11QER/31', 1, 110.00);
d. UPDATE PRODUCT
    i. SET P_QTYOH = P_QTYOH – 1
    ii. WHERE P_CODE = '11QER/31';
e. UPDATE CUSTOMER
f. SET CUS_DATELSTPUR = '11-May-2012', CUS_BALANCE = CUS_BALANCE +118.80
g. WHERE CUS_CODE = '10010';
h. COMMIT;

b. On June 3, 2012, customer '10010' makes a payment of $100 in cash. The payment ID is 3428.

a. BEGIN TRANSACTION
b. INSERT INTO PAYMENTS
VALUES (3428, '03-Jun-2012', '10010', 100.00, 'CASH', 'None');
UPDATE CUSTOMER;
SET CUS_DATELSTPMT = '03-Jun-2012', CUS_BALANCE = CUS_BALANCE -100.00
WHERE CUS_CODE = '10010';
COMMIT

Question 03

Conflict serializability is a concept that checks if a schedule is serializable by examining the conflict operations. Two operations are said to be in conflict if they belong to different transactions, operate on the same data item, and at least one of them is a write operation. A schedule is conflict-serializable if it is conflict- equivalent to a serial schedule. To determine if a schedule is conflict-serializable, we must examine the operations within it to identify conflicting operations. We can then analyze if those conflicts preserve their order relative to the transactions in the equivalent serial schedule.

Schedule a: $r_1(X)$; $r_3(X)$; $w_1(X)$; $r_2(X)$; $w_3(X)$;
Conflicts:

$w_1(X)$ and $r_2(X)$ are in conflict

$w_1(X)$ and $w_3(X)$ are in conflict

$w_3(X)$ and $r_2(X)$ are in conflict

a. The conflicting operations preserve their relative order for the transactions involved. Thus, Schedule a is conflict-serializable. The equivalent serial schedules could be $T_1 \rightarrow T_3 \rightarrow T_2$ or $T_3 \rightarrow T_1 \rightarrow T_2$.

Schedule b: $r_1(X)$; $r_3(X)$; $w_3(X)$; $w_1(X)$; $r_2(X)$;

Conflicts:

$w_3(X)$ and $w_1(X)$ are in conflict

$w_1(X)$ and $r_2(X)$ are in conflict

$w_3(X)$ and $r_2(X)$ are in conflict

b. The conflicting operations do not preserve their order for the transactions involved. Thus, Schedule b is not conflict-serializable.
Schedule c: $r_3(X)$; $r_2(X)$; $w_3(X)$; $r_1(X)$; $w_1(X)$;
Conflicts:

$w_3(X)$ and $r_1(X)$ are in conflict

$w_3(X)$ and $w_1(X)$ are in conflict

$w_1(X)$ and $r_2(X)$ are in conflict

c. The conflicting operations preserve their relative order for the transactions involved. Thus, Schedule *c* is conflict-serializable. The equivalent serial schedules could be $T_3 \rightarrow T_1 \rightarrow T_2$ or $T_3 \rightarrow T_2 \rightarrow T_1$.

Schedule d: $r_3(X)$; $r_2(X)$; $r_1(X)$; $w_3(X)$; $w_1(X)$;
Conflicts:

$w_3(X)$ and $w_1(X)$ are in conflict

$w_3(X)$ and $r_1(X)$ are in conflict

$w_1(X)$ and $r_2(X)$ are in conflict

d. The conflicting operations preserve their relative order for the transactions involved. Thus, Schedule d is conflict-serializable. The equivalent serial schedules could be $T_3 \rightarrow T_1 \rightarrow T_2$ or $T_3 \rightarrow T_2 \rightarrow T_1$.

Question 04

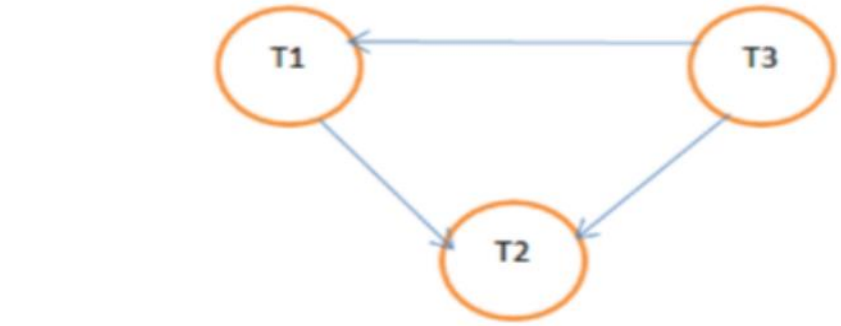
Step 01

Given schedule S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y);

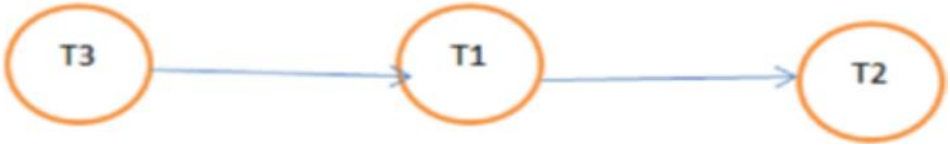
T1	T2	T3
r1(x)		
	r2(z)	
r1(z)		
		r3(x)
		r3(y)
w1(x)		
		w3(y)
	r2(y)	
	w2(z)	
	w2(y)	

Conflicts in the above transactions are $= r3(X) \rightarrow w1(X)$, $r3(Y) \rightarrow w2(Y)$, $r1(Z) \rightarrow w2(Z)$,$w3(y) \rightarrow w2(y)$

The precedence for the above transactions is given below-



OR



Step 02

The equivalent serializable schedule for the above schedule S1 is -

T1	T2	T3
r1(x)		
r1(z)		
	r2(z)	
		r3(x)
w1(x)		
	r2(y)	
		r3(y)
		w3(y)
	w2(z)	
	w2(y)	

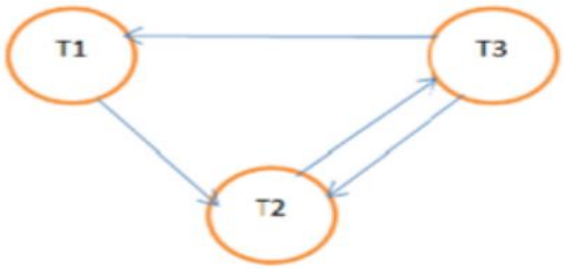
Step 03

Given schedule S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y); w2 (Y);

T1	T2	T3
r1(x)		
	r2(z)	
		r3(x)
r1(z)		
	r2(y)	
		r3(y)
w1(x)		
	w2(z)	
	w2(z)	
		w3(y)
	w2(y)	

Conflicts in the above transactions are $= r3(X) \rightarrow w1(x)$, $r1(z) \rightarrow w2(z)$, $r2(y) \rightarrow w3(y)$,$r3(y) \rightarrow w2(y)$

The precedence for the above transactions is given below-



The above schedule cannot be made serializable as there exist the cycle in the precedence graph.

Result

If precedence graph shows cycle, the schedule is non-serializable otherwise precedence graph with no cycle is serializable.

Question 05

Schedule S3

The operations in S_3 occur in the following sequence:

$r_1(X);r_2(Z);r_1(Z);r_3(X);r_3(Y);w_1(X);c_1;w_3(Y);c_3;r_2(Y);w_2(Z);w_2(Y);c_2$

- Strict: It's not strict because $r_3(X)$ reads X before $w_1(X)$ and c_1 are executed.
- Cascadeless: It's not cascadeless as $r_2(Y)$ reads Y that $w_3(Y)$ writes before c_3 is executed.
- Recoverable: It is recoverable as no transaction commits before the transactions whose data they read commit.

So, the strictest condition S_3 satisfies is Recoverable.

Schedule S4

The operations in S_4 occur in the following sequence:

$r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y); c_1 ; c_2; c_3$

- Strict: It's not strict.
- Cascadeless: It's not cascadeless as $r_2(Y)$ reads Y that $w_3(Y)$ writes before c_3 is executed..
- Recoverable: It is recoverable.

So, the strictest condition S_4 satisfies is Recoverable.

Schedule S5

The operations in S_5 occur in the following sequence:

$r_1(X);r_2(Z);r_3(X);r_1(Z);r_2(Y);r_3(Y);w_1(X);c_1;w_2(Z);w_3(Y);w_2(Y);c_3;c_2$

- Strict: It's not strict.
- Cascadeless: It's not cascadeless.
- Recoverable: It is recoverable.

So, the strictest condition S_5 satisfies is Recoverable.

Question 06

Step 01

```
read_lock (X):
    B: if LOCK (X)="unlocked"
       then begin LOCK (← X) "read-locked"; no_of_reads(X) 1
            end
       else if LOCK(X)="read-locked"
          then no_of_reads(X) ← no_of _reads(X) + 1
          else begin wait (until LOCK (X)="unlocked" and
                    the lock manager wakes up the transaction);
                    go to B
               end;
```

Step 02

```
write_lock (X):
   B: if LOCK (X)="unlocked"
      then LOCK (X) ← "write-locked"
      else begin
           wait (until LOCK(X)="unlocked" and
           he lock manager wakes up the transaction);
           go to B
        end;
```

Step 03

```
unlock_item (X):
    if LOCK (X)="write-locked"
    then begin LOCK (X) "unlocked;“ wakeup one of the waiting transactions, if any
         end
    else if LOCK(X)="read-locked"
         then begin
            no_of_reads(X) ← no_of_reads(X) – 1;
            if no_of_reads(X)=0
            then begin LOCK (X)="unlocked"; wakeup one of the waiting transactions, if any
                 end
             end;
```