**Database Assignment 4**

**AIMAN  
K21-3906**

# **Question 01**

## **Part a:**

To update the inventory, either 2 or 4 requests are needed.

1. One request for the product table.
2. One or three requests for the part table, either using three specific fields for XYZ or a combination of three with the 'OR' condition.

## **Part b:**

### **Using 4 database requests:**

****Update the PRODUCT table:**UPDATE PRODUCT**

**SET PROD\_QOH = PROD\_QOH + 1**

**WHERE PROD\_CODE = 'XYZ';**

****Update the PART table for each part code:****

1. UPDATE PART
2. SET PART\_QOH = PART\_QOH - 1
3. WHERE PART\_CODE = 'X';
4. UPDATE PART
5. SET PART\_QOH = PART\_QOH - 1
6. WHERE PART\_CODE = 'Y';
7. UPDATE PART
8. SET PART\_QOH = PART\_QOH - 1
9. WHERE PART\_CODE = 'Z';

### **Using 2 database requests:**

****Update the PRODUCT table:****

1. UPDATE PRODUCT
2. SET PROD\_QOH = PROD\_QOH + 1
3. WHERE PROD\_CODE = 'XYZ';

****Update the PART table for multiple part codes:****

1. UPDATE PART
2. SET PART\_QOH = PART\_QOH - 1
3. WHERE PART\_CODE IN ('X', 'Y', 'Z');

## **Part c:**

**4 database requests transaction:**BEGIN TRANSACTION;

UPDATE PRODUCT

SET PROD\_QOH = PROD\_QOH + 1

WHERE PROD\_CODE = 'XYZ';

UPDATE PART

SET PART\_QOH = PART\_QOH - 1

WHERE PART\_CODE = 'X';

UPDATE PART

SET PART\_QOH = PART\_QOH - 1

WHERE PART\_CODE = 'Y';

UPDATE PART

SET PART\_QOH = PART\_QOH - 1

WHERE PART\_CODE = 'Z';

COMMIT;

**2 database requests transaction:**BEGIN TRANSACTION;

UPDATE PRODUCT

SET PROD\_QOH = PROD\_QOH + 1

WHERE PROD\_CODE = 'XYZ';

UPDATE PART

SET PART\_QOH = PART\_QOH - 1

WHERE PART\_CODE IN ('X', 'Y', 'Z');

COMMIT;

**Question 02:**

In the above-given programming statement, it is required to implement two transactions

in SQL against an event taking place against the database.

1. **Start transaction**
2. **Get latest invoice**
3. **Insert new credit purchase in invoice table**
4. **Update CUS\_balance in customer table**
5. **Get latest line number**
6. **Insert into line table**
7. **Decrease P\_QTOH in product table**
8. **Commit changes**

**QUERY:**

BEGIN TRANSACTION

SELECT invoice-number:=MAX(INV\_NUMBER)+1 FROM INVOICE;

INSERT INTO INVOICE(INV\_NUMBER,CUS\_CODE, INV\_DATE, INV\_TOTAL, INV\_TERMS, INV\_STATUS) VALUES(invoice-number,10010,"11/05/2018", 110, 30, "OPEN");

UPDATE CUSTOMER AS custom (

SELECT INV\_TOTAL

FROM INVOICE

WHERE INV\_NUMBER = invoice-Number ) as invoice\_Total

SET custom.CUS\_BALANCE = custom.CUS\_BALANCE + invoice\_Total

WHERE custom.CUST\_CODE = 10010;

SELECT line-Number:=MAX(LINE\_NUMBER)+1

FROM LINE;

INSERT INTO LINE(INV\_NUMBER, LINE\_NUMBER, P\_CODE, LINE\_UNITS, LINE\_PRICE)

VALUES( invoice-Number, line-Number, 11QER/31, 1, 110);

UPDATE PRODUCT

SET P\_QTYOH = P\_QTYPH - 1

WHERE P\_CODE = 11QER/31;

COMMIT;

B:

1. Start new Transaction
2. Get latest payment number
3. Insert new payment details in payment table
4. Update balance in customer table
5. Commit

**Query:**

BEGIN TRANSACTION;

SELECT payment-Number:=MAX(PMT\_ID)+1 FROM PAYMENTS;

INSERT INTO INVOICE

(PMT\_ID, PMT\_DATE, CUS\_CODE, PMT\_AMT, PMT\_TYPE,PMT\_DETAILS)

VALUES(invoice-Number, payment-Number, "03/06/2018", 10010, 100,

"CASH Goldsmiths, 113241, 121342");

UPDATE CUSTOMER AS custom (

SELECT PMT\_AMT FROM PAYMENTS

WHERE PMT\_ID = payment-Number ) as pmt\_Total

SET custom.CUS\_BALANCE = custom.CUS\_BALANCE + pmt\_Total

WHERE custom.CUST\_CODE = 10010;

COMMIT;

**Question 03:**

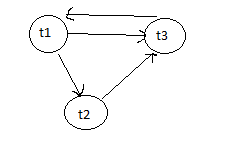
By Definition schedule conflict:

Pair should be equal to any serial schedule. Where as conflict pair has at least one write operation to some data item but at some different transaction.

Analyze,

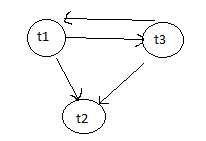
**1-Schedule [ r1(X); r3(X); w1(X); r2(X); w3(X);**

Precedence graph: there are 3 transactions t1,t2,t3 which produce schedule



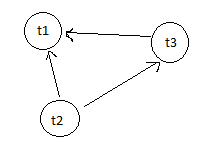
Creates cycle not conflict serializable schedule.

**2-r1(X); r3(X); w3(X); w1(X); r2(X);**



Creates cycle not conflict serializable schedule

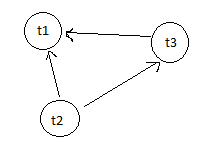
1. **r3(X); r2(X); w3(X); r1(X); w1(X);**

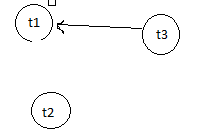


Creates no cycle is a serializable schedule

For equivalent schedule:

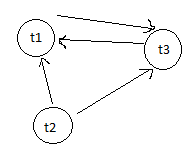
Consider the node with 0 in-degrees t2 and deleting out-degree vertices.





Conflict equivalent schedule is t2->t3->t1

1. **r3(X); r2(X); r1(X); w3(X); w1(X);**



Contains cycle so the given schedule is not conflict serializable schedule.

The correct option Is C

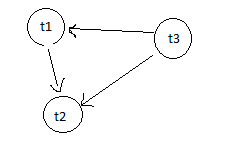
**Question 04:**

1-S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y);

Conflict points:

1. r1 (Z) - w2 (Z)[t1->12]
2. r3 (X)- w1(x) [t3 ->t1]
3. r3(Y) - w2(Y)[t3 ->t2]
4. w3(y) -r2(y) [t3 ->t2]
5. w3(y)- w2(Y)[t3 ->t2]

Precedence graph:



No cycle

S1 is serializable schedule

Topological order of precedence of graph of s1

Topological Precedence graph:

T3-> t1->t2

1. S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y); w2 (Y);

Conflict points:

1. r3(x)-w3(x)[t3-t1]

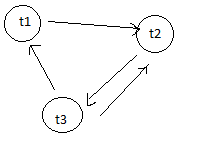
2- r1 (Z) - w2 (Z)[t1->t2]

3- r2 (X)- w3(x) [t2 ->t3]

4- r3(Y) - w2(Y)[t3 ->t2]

5- w3(y)- w2(Y)[t3 ->t2]

Precedence graph:



No cycle

2nd method:

View serializability:

|  |  |
| --- | --- |
| Given schedule | Serial schedule |
| 1. Final writes of each data item:   X: t1  Y:t2 t3  Z: t2 | T3 ->t2 |
| 1. Initial reads:      |  |  |  | | --- | --- | --- | | Data item | Transaction of initial reads | Transaction of write | | x | T1 | T1 | | y | T2 | T3,T2 | | z | T2 | T2 | | T2 ->t3 |

Update read:

No such conflict

Form t3->t2, t2->t3 can’t we conclude any schedule(serial).

S2 isn’t serializable schedule.

**Question 05:**

Strict schedule:

A schedule is strict if it satisfies the following conditions:

Tj reads a data item X after Ti has written to X and Ti is terminated means aborted or committed.

Tj writes a data item X after Ti has written to X and Ti is terminated means aborted or committed.

S3 is not strict because In a strict schedule T3 must read X after C1 but here T3 reads X (r3(X)) before Then T1 has written to X (w1(X)) and T3 commits after T1.

S4 is not strict because In a strict schedule T3 must read X after C1, but here T3 reads X (r3(X)) before T1 has written to X (w1(X)) and T3 commits after T1.

S5 is not strict because T3 reads X (r3(X)) before T1 has written to X (w1(X))

but T3 commits after T1. In a strict schedule T3 must read X after C1.

Cascade less schedule:

Cascade less schedule follows the below condition:

Tj reads X only? after Ti has written to X and terminated means aborted or committed.

S3 is not cascade less schedule because T3 reads X (r3(X)) before T1 commits.

S4 is not cascade less schedule because T3 reads X (r3(X)) before T1 commits.

S5 is not cascade less schedule because T3 reads X (r3(X)) before T1 commits or T2 reads Y (r2(Y)) before T3 commits. cascade less schedules S3, S4, and S4 are not cascade less, and T3 is not affected if T1 is rolled back in any of the schedules, that is,

T3 does not have to roll back if T1 is rolled back. The problem occurs because these

schedules are not serializable.

Recoverable schedule:

.

Ci > Cj means that Ci happens before Cj.

Ai denotes abort Ti. To test if a schedule is recoverable one has to include abort operations. Thus in testing the recover ability abort operations will have to used in place of commit one at a time. Also the strictest condition is,

where a transaction neither reads nor writes to a data item, which was written to by a transaction that has not committed yet.

1. A1?>C3>C2, then schedule S3 is recoverable because rolling back of T1 does not affect T2 and

T3.

1. C1>A3>C2. schedule S3 is not recoverable because T2 read the value of Y (r2(Y)) after T3 wrote X (w3(Y)) and T2 committed but T3 rolled back. Thus, T2 used non- existent value of Y.
2. C1>C3>A3, then S3 is recoverable because roll back of T2 does not affect T1 and T3.

Strictest condition of schedule S3 is C3>C2.

1. A1?>C2>C3, then schedule S4 is recoverable because roll back of T1 does not affect T2 and T3.
2. 5-C1>A2>C3, then schedule S4 is recoverable because the roll back of T2 will restore the value of Y that was read and written to by T3 (w3(Y)). It will not affect T1.
3. C1>C2>A3, then schedule S4 is not recoverable because T3 will restore the value of Y which was not read by T2.

**Question 06:**

We assume that a List of transaction ids that have read-locked an item is maintained, as well as the (single) transaction id that has write-locked an item. Only read\_lock and write\_lock are shown below.

read\_lock (X, Tn):

B:

if lock (X) = "unlocked";

then begin lock (X) <- "read\_locked, List(Tn)";

no\_of\_reads (X) <- 1

end,

else if lock(X) = "read\_locked, List"

then begin

(\* add Tn to the list of transactions that have read\_lock on X \*)

lock (X) <-"read\_locked, Append(List,Tn)";

no\_of\_reads (X) <- no\_of\_reads (X) + 1

end,

else if lock (X) = "write\_locked, Tn"

(\* downgrade the lock if write\_lock on X is held by Tn itself \*)

then begin lock (X) <-"read\_locked, List(Tn)";

no\_of\_reads (X) <- 1

end,

else begin

wait (until lock (X) = "unlocked" and

the lock manager wakes up the transaction);

go to B;

end;

write\_lock (X,Tn);

**B:** if lock (X) = "unlocked"

then lock (X) <- "write\_locked, Tn"

Else if ( (lock (X) ="read\_locked, List") and (no\_of\_reads (X) = 1)

and (transaction in List = Tn) )

(\* upgrade the lock if read\_lock on X is held only by Tn itself \*)

then lock (X) = "write\_locked, Tn"

else begin

wait (until ( [ lock (X) = "unlocked" ] or

[ (lock (X) = "read\_locked, List") and (no\_of\_reads (X) = 1)

and (transaction in List = Tn) ] ) and

the lock manager wakes up the transaction);

go to B;

end;