1.

There are 4 database requests for the inventory update for both PRODUCT and PART.

The Database requests are as follows:

UDPATE PRODUCT

SET PRODD\_QOH = PRO\_QOH + 1

WHERE PROD\_CODE = ‘ABC’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘A’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘B’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘C’

Here are the complete transactions which just add BEGIN TRANSACTION and COMMIT; to the whole thing that is already written above.

BEGIN TRANSACTION

UDPATE PRODUCT

SET PRODD\_QOH = PRO\_QOH + 1

WHERE PROD\_CODE = ‘ABC’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘A’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘B’

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE = ‘C’

COMMIT;

While assuming some of the numbers, this is the transaction log;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRL ID | TRX NUM | PREV PTR | NEXT PTR | OPERATION | TABLE | ROW ID | ATTRIBUTE | BEFORE VALUE | AFTER VALUE |
| 1 | 101 | NULL | 2 | START | \*\*\*\*START TRANSACTION |  |  |  |  |
| 2 | 101 | 1 | 3 | UPDATE | PRODUCT | ‘ABC’ | PROD\_QOH | 1205 | 1206 |
| 3 | 101 | 2 | 4 | UPDATE | UPDATE | A’ | PART\_QOH | 567 | 566 |
| 4 | 101 | 3 | 5 | UPDATE | PART | ‘B’ | PART\_QOH | 98 | 97 |
| 5 | 101 | 4 | 6 | UPDATE | PART | ‘C’ | PART\_QOH | 549 | 548 |
| 6 | 101 | 5 | NULL | COMMIT | \*\*\*\*END TRANSACTION |  |  |  |  |

TRL\_ID = Transaction log record ID

TRX\_NUM = Transaction number

PTR = Pointer to a transaction log record ID

(Note: The transaction number is automatically assigned by the DBMS)

Database will be restored to its earlier point right before this transaction only if the ROLLBACK is issued before the termination of the transaction. Committed transactions are not Rolled back.

2. The Three main problems are lost updates, uncommitted data, and inconsistent retrievals.

The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment. To achieve this goal, most concurrency control techniques are oriented toward preserving the isolation property of concurrently executing transactions. Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.

3. The scheduler is a special DBMS process that establishes the order in which the operations are executed within concurrent transactions. The scheduler interleaves the execution of database operations to ensure serializability and isolation of transactions. To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms, such as locking or time stamping methods.

4. A binary lock has only two states: locked (1) or unlocked (0). If an object such as a database, table, page, or row is locked by a transaction, no other transaction can use that object. If an object is unlocked, any transaction can lock the object for its use.

An exclusive lock exists when access is reserved specifically for the transaction that locked the object. The exclusive lock must be used when the potential for conflict exists. A shared lock exists when concurrent transactions are granted read access on the basis of a common lock. A shared lock produces no conflict as long as all the concurrent transactions are read-only.

5. When the recovery procedure uses a deferred-write technique (also called a deferred update), the transaction operations do not immediately update the physical database. Instead, only the transaction log is updated. The database is physically updated only with data from committed transactions, using information from the transaction log. If the transaction aborts before it reaches its commit point, no changes (no ROLLBACK or undo) need to be made to the database because it was never updated. The recovery process for all started and committed transactions (before the failure) follows these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
2. For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data is already saved.
3. For a transaction that performed a commit operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and update the database, using the “after” values in the transaction log. The changes are made in ascending order, from oldest to newest.
4. For any transaction that had a ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure occurred, nothing needs to be done because the database was never updated.

When the recovery procedure uses a write-through technique (also called an immediate update), the database is immediately updated by transaction operations during the transaction’s execution, even before the transaction reaches its commit point. If the transaction aborts before it reaches its commit point, a ROLLBACK or undo operation needs to be done to restore the database to a consistent state. In that case, the ROLLBACK operation will use the transaction log “before” values. The recovery process follows these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
2. For a transaction that started and was committed before the last checkpoint, nothing needs to be done because the data is already saved.
3. For a transaction that was committed after the last checkpoint, the DBMS re-does the transaction, using the “after” values of the transaction log. Changes are applied in ascending order, from oldest to newest.
4. For any transaction that had a ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure occurred, the DBMS uses the transaction log records to ROLLBACK or undo the operations, using the “before” values in the transaction log. Changes are applied in reverse order, from newest to oldest.

6.

a).

BEGIN TRANSACTION

INSERT INTO INVOICE VALUES (10983, ‘10010’, ‘11-May-2008’, 118.80, ‘30’, ‘OPEN’);

INSERT INTO LINE VALUES (10983, 1, ‘11QER/31’, 1, 110.00);

UPDATE PRODUCT

SET P\_QTYOH = P\_QTYOH – 1

WHERE P\_CODE = ‘11QER/31’;

UPDATE CUSTOMER

SET CUS\_DATELSTPUR = ‘11-May-2008’, CUS\_BALANCE = CUS\_BALANCE +118.80

WHERE CUS\_CODE = ‘10010’;

COMMIT;

b)

BEGIN TRANSACTION

INSERT INTO PAYMENTS VALUES (3428, ‘03-Jun-2008’, ‘10010’, 100.00, ‘CASH’, 'None');

UPDATE CUSTOMER

SET CUS\_DATELSTPMT = ‘03-Jun-2008’, CUS\_BALANCE = CUS\_BALANCE -100.00

WHERE CUS\_CODE = ‘10010’;

COMMIT;

7.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRL ID | TRX NUM | PREV PTR | NEXT PTR | OPERATION | TABLE | ROW ID | ATTRIBUTE | BEFORE VALUE | AFTER VALUE |
| 987 | 101 | NULL | 1023 | START | \*START TRX. |  |  |  |  |
| 1023 | 101 | 987 | 1026 | INSERT | INVOICE | 10983 |  |  | 10983, 10010  11-MAY-2012, 118.80, 30, OPEN |
| 1026 | 101 | 1023 | 1029 | INSERT | LINE | 10983, 1 |  |  | 10983, 1, 11QER/31, 1, 110.00 |
| 1029 | 101 | 1026 | 1031 | UPDATE | PRODUCT | 11QER/31 | P\_QTYOH | 47 | 46 |
| 1031 | 101 | 1029 | 1032 | UPDATE | CUSTOMER | 10010 | CUS\_BALANCE | 345.67 | 464.47 |
| 1032 | 101 | 1031 | 1034 | UPDATE | CUSTOMER | 10010 | CUS\_DATELSTPUR | 5-MAY-2010 | 11-MAY-2012 |
| 1034 | 101 | 1032 | NULL | COMMIT | \*END TRX. \* |  |  |  |  |
| 1089 | 102 | NULL | 1091 | START | \*START TRX |  |  |  |  |
| 1091 | 102 | 1089 | 1095 | INSERT | PAYMENT | 3428 |  |  | 3428, 3-JUN-2012, 10010, 100.00, CASH. NONE |
| 1095 | 102 | 1091 | 1096 | UPDATE | CUSTOMER | 10010 | CUS\_BALANCE | 464.47 | 364.47 |
| 1096 | 102 | 1095 | 1097 | UPDATE | CUSTOMER | 10010 | CUS\_DATELSTPMT | 2-MAY-2010 | 3-JUN-2012 |
| 1097 | 102 | 1096 | NULL | COMMIT | \*END TRX |  |  |  |  |

8.

1. LOCK INVOICE
2. INSERT ROW 10983 INTO INVOICE
3. UNLOCK INVOICE
4. LOCK LINE
5. INSERT ROW 10983, 1 INTO LINE
6. UNLOCK LINE
7. LOCK PRODUCT
8. UPDATE PRODUCT 11QER/31, P\_QTYOH FROM 47 TO 46
9. UNLOCK PRODUCT
10. UNLOCK CUSTOMER
11. UPDATE CUSTOMER 10010, CUS\_BALANCE FROM 345.67 TO 464.47
12. UPDATE CUSTOMER 10010, CUS\_DATELSPUR FROM 05-MAY-2010 TO 11-MAY-2012
13. UNLOCK CUSTOMER

9.

1. Lock INVOICE
2. Lock LINE
3. Lock PRODUCT
4. Lock CUSTOMER
5. Insert row 10983 into INVOICE
6. Insert tow 10983, 1 into LINE
7. Update PRODUCT 11QER/31, P\_QTYOH from 47 to 46
8. Update CUSTOMER 10010, CUS\_BALANCE from 345.67 to 464.47
9. Update CUSTOMER 10010, CUS\_DATELSTPUR from 05-May-2010 to 11-May-2012
10. Unlock INVOICE
11. Unlock LINE
12. Unlock PRODUCT
13. Unlock CUSTOMER

10.

1. Lock PAYMENT
2. Insert row 3428 into PAYMENT
3. Unlock PAYMENT
4. Lock CUSTOMER
5. Update CUSTOMER 10010, CUS\_BALANCE from 464.47 to 364.47
6. Update CUSTOMER 10010, CUS\_DATELSTPMT from 02-May-2010 to 03-Jun-2012
7. Unlock CUSTOMER

11.

1. Lock PAYMENT
2. Lock CUSTOMER
3. Insert row 3428 into PAYMENT
4. Update CUSTOMER 10010, CUS\_BALANCE from 464.47 to 364.47
5. Update CUSTOMER 10010, CUS\_DATELSTPMT from 02-May-2010 to 03-Jun-2012
6. Unlock PAYMENT
7. Unlock CUSTOMER91