The National Higher School of Artificial Intelligence

DATABASES

Chapter 6 : Transaction Management and Concurrency Control

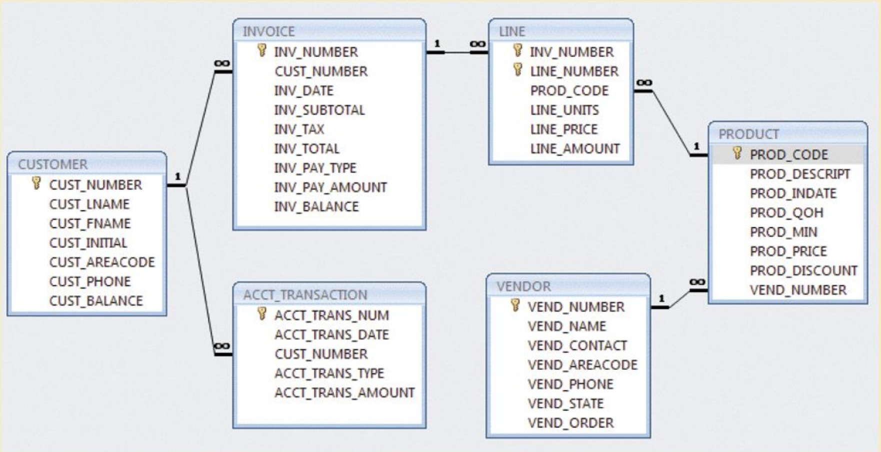
Pr. Kamel BOUKHALFA

Slides From the Textbook :
Carlos Coronel and Steven Morris, Database Systems: Design, Implementation, and Management
Tenth Edition

What Is a Transaction?

- Logical unit of work that must be either entirely completed or aborted
- Successful transaction changes database from one **consistent state** to another
  - One in which all data integrity constraints are satisfied
- Most real-world database transactions are formed by two or more database requests
  - Equivalent of a single SQL statement in an application program or transaction
  - Will involve at least one read and write action

FIGURE 10.1 The Ch10_SaleCo database relational diagram



SOURCE: Course Technology/Cengage Learning

Evaluating Transaction Results

- Not all transactions update database
- SQL code represents a transaction because database was accessed
- Improper or incomplete transactions can have devastating effect on database integrity
  - Some DBMSs provide means by which user can define enforceable constraints
  - Other integrity rules are enforced automatically by the DBMS

Evaluating Transaction Results

```
INSERT INTO INVOICE VALUES (…);
INSERT INTO LINE (…);
UPDATE PRODUCT SET …
UPDATE CUSTOMER SET CUST_BALANCE..
INSERT INTO ACCT_TRANSACTION VALUES (…);
COMMIT;
```

- Suppose the first 3 statements are executed and then power is lost with no backup –DBMS is in an inconsistent state unless transaction management is supported
  - Invoice and Line rows were added and the QOH of Product table updated but the customer balance has not been increased nor has a new transaction been added to Acct_Transaction

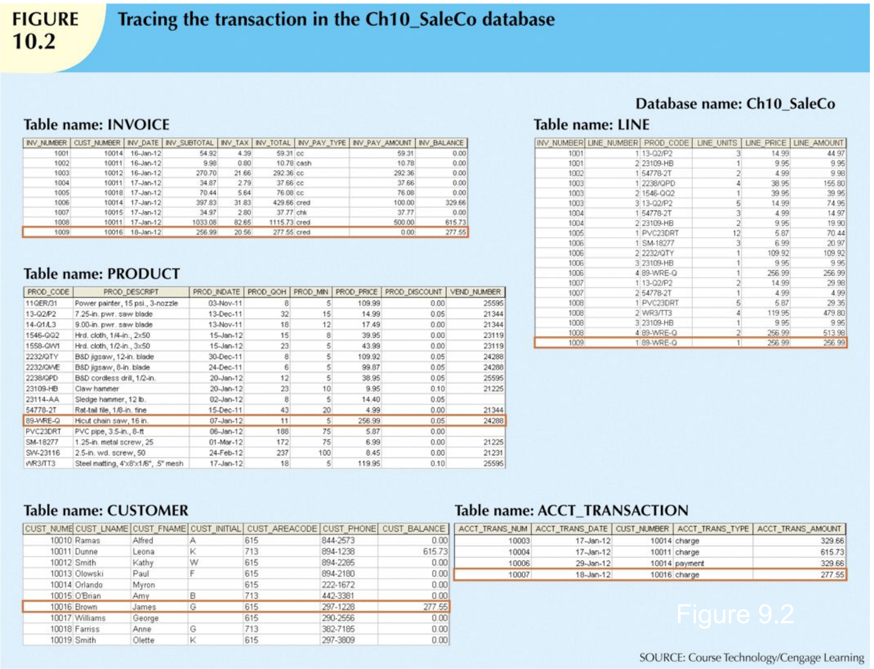


FIGURE 10.2 Tracing the transaction in the Ch10_SaleCo database

Database name: Ch10_SaleCo

Table name: INVOICE

INV_NUMBER	CUST_NUMBER	INV_DATE	INV_SUBTOTAL	INV_TAX	INV_TOTAL	INV_PAY_TYPE	INV_PAY_AMOUNT	INV_BALANCE
1001	10014	16-Jan-12	54.92	4.39	59.31	cc	59.31	0.00
1002	10011	16-Jan-12	9.98	0.80	10.78	cash	10.78	0.00
1003	10012	16-Jan-12	270.70	21.66	292.36	cc	292.36	0.00
1004	10011	17-Jan-12	34.87	2.79	37.66	cc	37.66	0.00
1005	10018	17-Jan-12	70.44	5.64	76.08	cc	76.08	0.00
1006	10014	17-Jan-12	397.83	31.83	429.66	cred	100.00	329.66
1007	10015	17-Jan-12	34.97	2.80	37.77	chk	37.77	0.00
1008	10011	17-Jan-12	1033.08	82.65	1115.73	cred	500.00	615.73
1009	10016	18-Jan-12	256.99	20.56	277.55	cred	0.00	277.55

Table name: LINE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE	LINE_AMOUNT
1001	1	13-Q2/P2	3	14.99	44.97
1001	2	23109-HB	1	9.95	9.95
1002	1	54778-2T	2	4.99	9.98
1003	1	2238/QPD	4	38.95	155.80
1003	2	1546-QQ2	1	39.95	39.95
1003	3	13-Q2/P2	5	14.99	74.95
1004	1	54778-2T	3	4.99	14.97
1004	2	23109-HB	2	9.95	19.90
1005	1	PVC23DRT	12	5.87	70.44
1006	1	SM-18277	3	6.99	20.97
1006	2	2232/QTY	1	109.92	109.92
1006	3	23109-HB	1	9.95	9.95
1006	4	89-WRE-Q	1	256.99	256.99
1007	1	13-Q2/P2	2	14.99	29.98
1007	2	54778-2T	1	4.99	4.99
1008	1	PVC23DRT	5	5.87	29.35
1008	2	WR3/TT3	4	119.95	479.80
1008	3	23109-HB	1	9.95	9.95
1008	4	89-WRE-Q	2	256.99	513.98
1009	1	89-WRE-Q	1	256.99	256.99

Table name: PRODUCT

PROD_CODE	PROD_DESCRIPT	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_NUMBER
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-11	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-11	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-11	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-12	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-12	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-11	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-11	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-12	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-12	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-12	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-11	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Jan-12	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	06-Jan-12	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-12	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-12	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-12	18	5	119.95	0.10	25595

Table name: CUSTOMER

CUST_NUME	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	615.73
10012	Smith	Kathy	W	615	894-2285	0.00
10013	Olowski	Paul	F	615	894-2180	0.00
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	277.55
10017	Williams	George		615	290-2556	0.00
10018	Farriss	Anne	G	713	382-7185	0.00
10019	Smith	Olette	K	615	297-3809	0.00

Table name: ACCT_TRANSACTION

ACCT_TRANS_NUM	ACCT_TRANS_DATE	CUST_NUMBER	ACCT_TRANS_TYPE	ACCT_TRANS_AMOUNT
10003	17-Jan-12	10014	charge	329.66
10004	17-Jan-12	10011	charge	615.73
10006	29-Jan-12	10014	payment	329.66
10007	18-Jan-12	10016	charge	277.55

Figure 9.2

SOURCE: Course Technology/Cengage Learning

Transaction Properties

- Atomicity
  - All operations of a transaction must be completed or else the entire transaction is aborted
- Consistency
  - Permanence of database's consistent state
- Isolation
  - Data used during transaction cannot be used by second transaction until the first is completed

Transaction Properties (cont'd.)

- Durability
  - Once transactions are committed, they cannot be undone
- Serializability
  - Concurrent execution of several transactions yields consistent results
- Multiuser databases are subject to multiple concurrent transactions

Transaction Management with SQL

- ANSI has defined standards that govern SQL database transactions
- Transaction support is provided by two SQL statements: COMMIT and ROLLBACK
- Transaction sequence must continue until:
  - COMMIT statement is reached
  - ROLLBACK statement is reached
  - End of program is reached
  - Program is abnormally terminated
- Some SQL implementations use BEGIN (or SET) TRANSACTION; to indicate the beginning of a new transaction

The Transaction Log

- Keeps track of all transactions that update the database
  - Used for a recovery triggered by a ROLLBACK, abnormal termination or system failure
- While the DBMS executes transactions that modify the database, it also automatically updates the transaction log
- The ability to recover a corrupted database is worth the Increased processing overhead

The Transaction Log

- Transaction log stores:
  - A record for the beginning of transaction
  - For each transaction component:
    - Type of operation being performed (update, delete, insert)
    - Names of objects affected by transaction
    - “Before” and “after” values for updated fields
    - Pointers to previous and next transaction log entries for the same transaction
  - Ending (COMMIT) of the transaction

The Transaction Log

- Next slide illustrates a simplified transaction log that reflects a transaction composed of two SQL UPDATE statements
- If a system failure occurs, the DBMS examines the log for all uncommitted or incomplete transactions and restores the database to its previous state
  - When the recovery process is completed, the DBMS will write in the log all committed transactions that were not physically written to the database before the failure occurred
  - Committed transactions are not rolled back

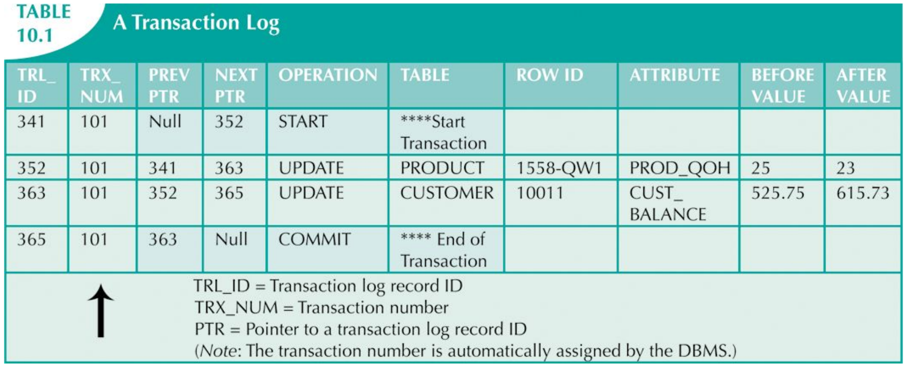


TABLE 10.1 A Transaction Log

TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_ BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of 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.)

Concurrency Control

- Coordination of simultaneous transaction execution in a multiprocessing database
- Objective is to ensure serializability of transactions in a multiuser environment
- The simultaneous execution of transactions over a shared database can create several data integrity and consistency problems
- Three main problems:
  - Lost updates
  - Uncommitted data
  - Inconsistent retrievals

Lost Updates

- Lost update problem:
  - Two concurrent transactions update same data element

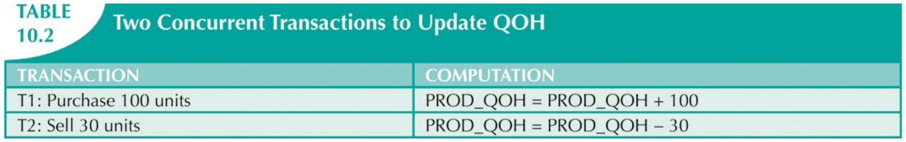


TABLE 10.2 Two Concurrent Transactions to Update QOH

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

  - Table 10.3 shows the serial execution of the transactions under normal circumstances.
  - Final result of PROD_QOH = 105 is correct

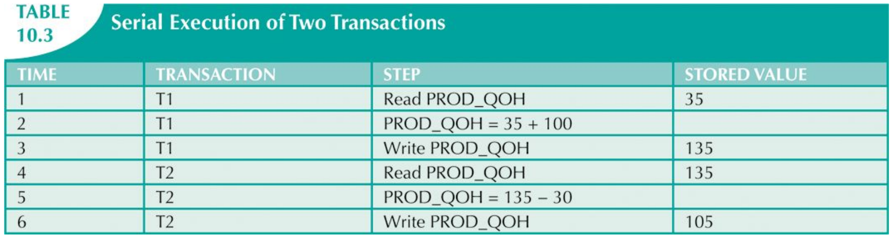


TABLE 10.3 Serial Execution of Two Transactions

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 – 30	
6	T2	Write PROD_QOH	105

Lost Updates

- Table 10.4 shows what happens when a second transaction can read the PROD_QOH value of a product before a previous transaction has been committed for that same product
- One of the updates is lost by being overwritten by the other transaction

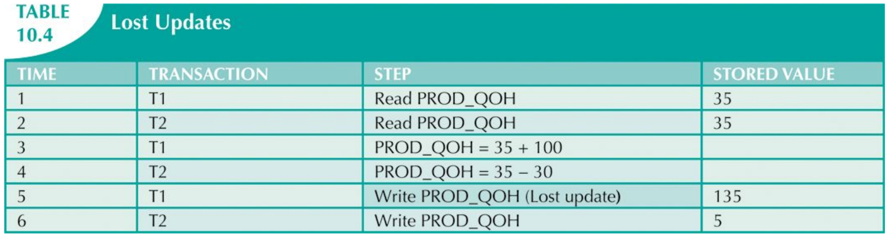


TABLE 10.4 Lost Updates

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 – 30	
5	T1	Write PROD_QOH (Lost update)	135
6	T2	Write PROD_QOH	5

Uncommitted Data

- Uncommitted data phenomenon:
  - Two transactions are executed concurrently
  - First transaction rolled back after second already accessed uncommitted data
    - Violates the isolation property

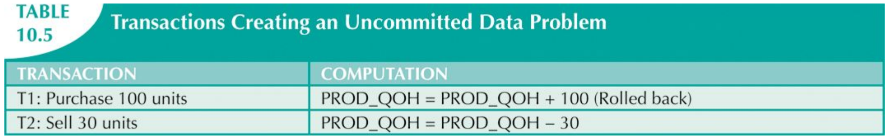


TABLE 10.5 Transactions Creating an Uncommitted Data Problem

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

Uncommitted Data

- Serial execution of the transactions yields the correct answer under normal circumstances

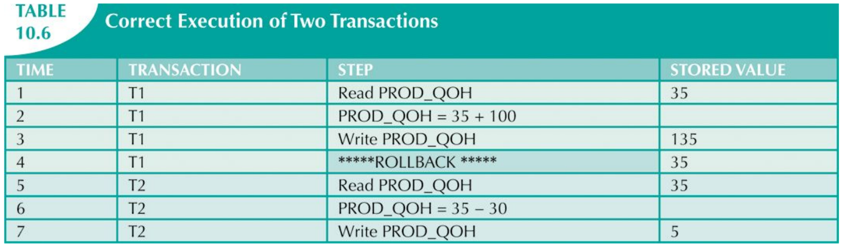


TABLE 10.6 Correct Execution of Two Transactions

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 – 30	
7	T2	Write PROD_QOH	5

Uncommitted Data

- Uncommitted data problem can arise when the ROLLBACK is completed after T2 has begun its execution

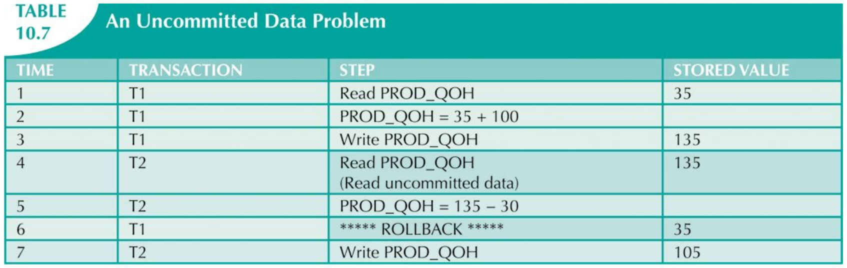


TABLE 10.7 An Uncommitted Data Problem

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 – 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Inconsistent Retrievals

- Inconsistent retrievals:
  - First transaction accesses data
  - Second transaction alters the data
  - First transaction accesses the data again
- Transaction might read some data before they are changed and other data after changed
- Yields inconsistent results

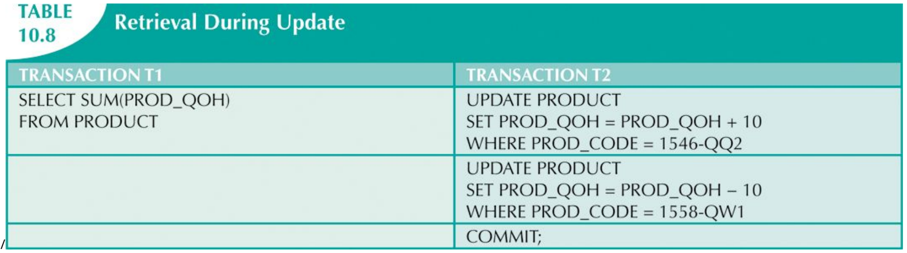


TABLE 10.8 Retrieval During Update

TRANSACTION T1	TRANSACTION T2

SELECT SUM(PROD_QOH)
FROM PRODUCT
	UPDATE PRODUCT
	SET PROD_QOH = PROD_QOH + 10
	WHERE PROD_CODE = 1546-QQ2

	UPDATE PRODUCT
	SET PROD_QOH = PROD_QOH – 10
	WHERE PROD_CODE = 1558-QW1

	COMMIT;

Inconsistent Retrievals

- A sample of the PRODUCT table, showing the records being updated as specified in the UPDATE code

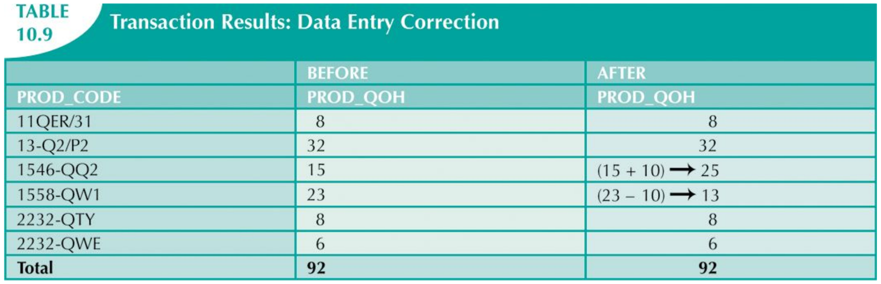


TABLE 10.9 Transaction Results: Data Entry Correction

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 – 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	**92**	**92**

Inconsistent Retrievals

- Inconsistent retrievals are possible during the transaction execution
  - The “after” summation shown below reflects that the value of 25 for product 1546-QQ2 was read after the WRITE statement was completed so the total up to that point is 65
  - The “before” total reflects that the value of 23 for 1558-QQ1 was read before the next WRITE was completed (in which case the value would have been 13 instead of 23)
    - After step 7, the total is 65+23 rather than 65+13
    - The final total will be 102 instead of 92

Inconsistent Retrievals

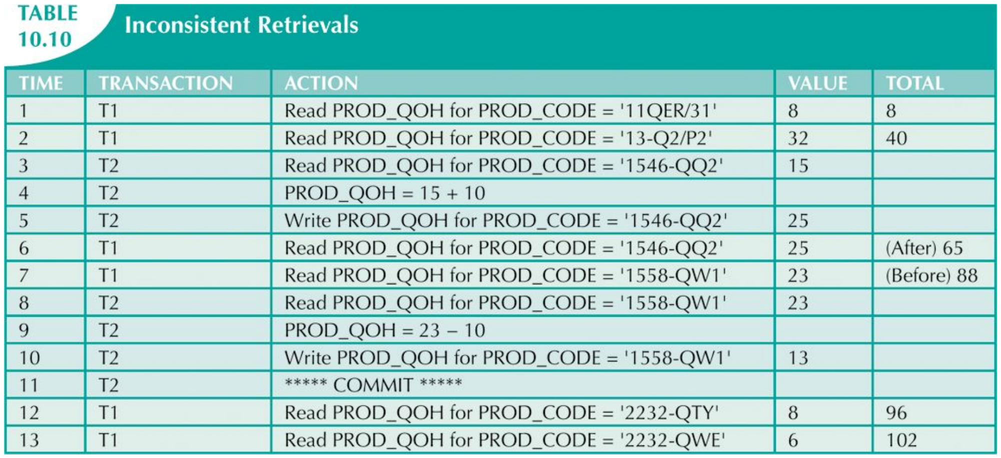


TABLE 10.10 Inconsistent Retrievals

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 – 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

The Scheduler

- Special DBMS program
  - Purpose is to establish order of operations within which concurrent transactions are executed
- Interleaves execution of database operations:
  - Ensures serializability
  - Ensures isolation
- Serializable schedule
  - Interleaved execution of transactions yields same results as serial execution
- Transactions that are not serializable with interleaving are executed on a first-come first-serve basis

The Scheduler

- FCFS is an inefficient method of allocating the CPU, since the CPU waits idly as a READ or WRITE operation completes
- The scheduler also facilitates data isolation to ensure that two transactions do not update the same data element at the same time
- If one or both of two concurrent transactions wants to perform a WRITE on the same data, then a conflict will result

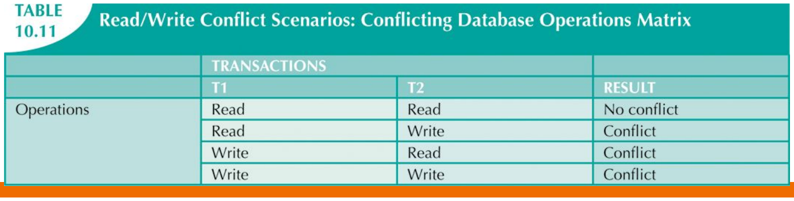


TABLE 10.11 Read/Write Conflict Scenarios: Conflicting Database Operations Matrix

	TRANSACTIONS		
	T1	T2	RESULT
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Concurrency Control with Locking Methods

- Lock
  - Guarantees exclusive use of a data item to a current transaction
    - Lock is acquired prior to access and released when the transaction is completed
  - Required to prevent another transaction from reading inconsistent data
  - Pessimistic locking
    - Use of locks based on the assumption that conflict between transactions is likely
  - Lock manager
    - Responsible for assigning and policing the locks used by transactions

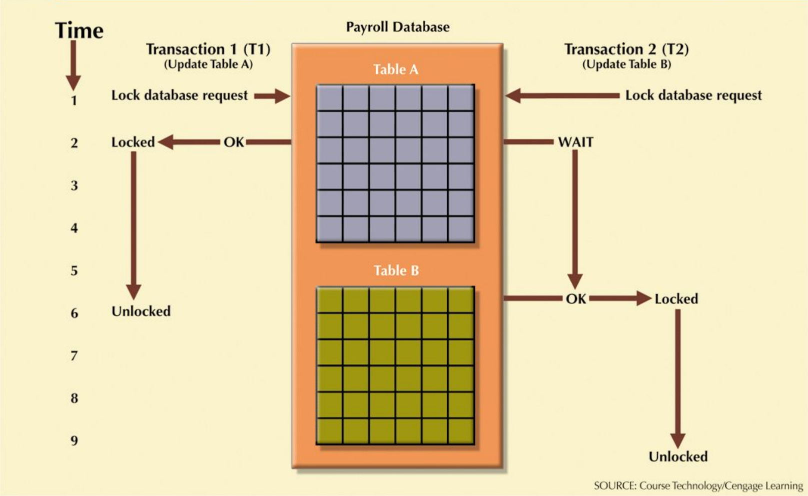
Lock Granularity

- Indicates level of lock use
- Locking can take place at following levels:
  - Database
  - Table
  - Page
  - Row
  - Field (attribute)

Lock Granularity

- Database-level lock
  - Entire database is locked, prevents the use of any tables by transaction T2 while T1 is being executed
  - Good for batch systems, not for multi-user systems

FIGURE 10.3 Database-level locking sequence

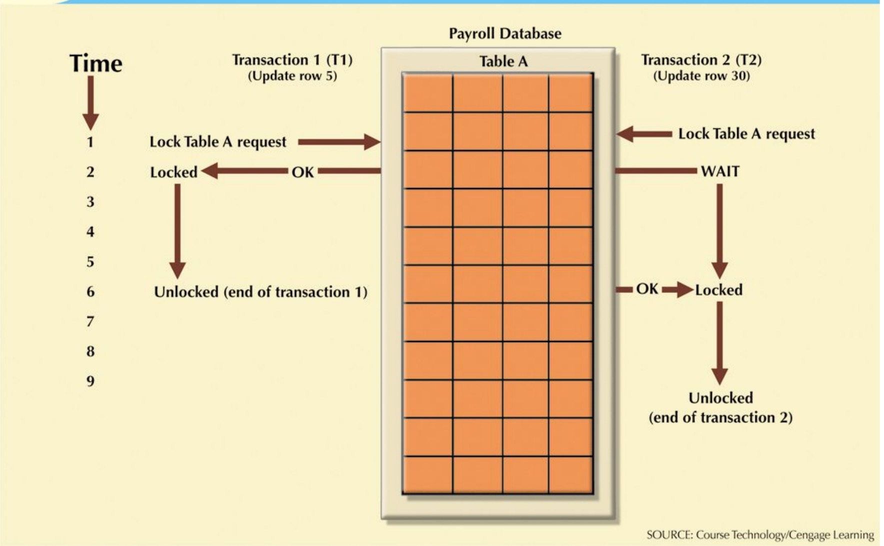


SOURCE: Course Technology/Cengage Learning

Lock Granularity

- Table-level lock
  - Entire table is locked
  - If a transaction requires access to several tables, each table may be locked
  - Two transactions can access the database as long as they access different tables
  - Not suitable for multi-user systems because all rows are locked even if different transactions access different rows

FIGURE 10.4 An example of a table-level lock

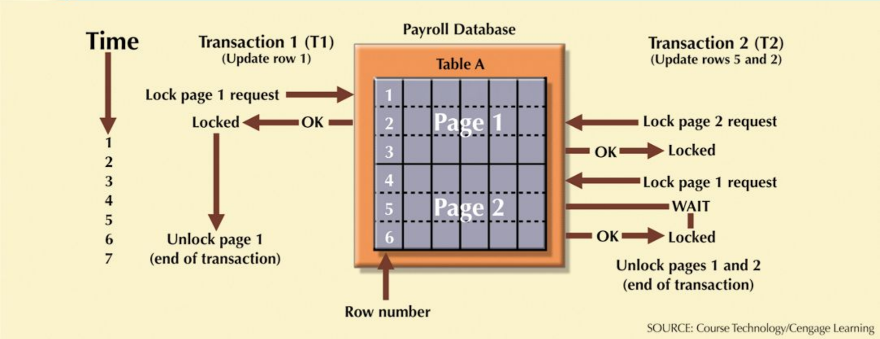


SOURCE: Course Technology/Cengage Learning

Lock Granularity

- Page-level lock
  - Entire diskpage (or page) is locked
    - Page is the equivalent of a diskblock - a directly addressable section of disk (e.g., 4K, 8K or 16K)
    - If you want to write 73 bytes to a 4K page, the entire page must be read from disk, updated in memory and written back to disk
      - A table can span multiple pages and a page can contain several rows of one or more tables
    - As long as two transactions do not access the same page, no waiting is required by the second transaction
    - currently the most frequently used locking method for multi-user DBMSs

FIGURE 10.5 An example of a page-level lock

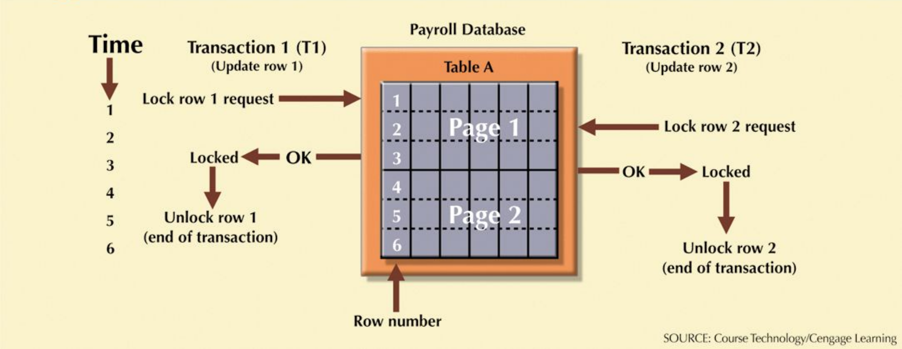


SOURCE: Course Technology/Cengage Learning

Lock Granularity

- Row-level lock
  - – Allows concurrent transactions to access different rows of same table
    - Even if rows are located on same page
    - High overhead because a lock exists for each row of a table
    - If the application requests multiple locks on the same, a row level lock is automatically escalated to a page level lock

FIGURE 10.6 An example of a row-level lock



SOURCE: Course Technology/Cengage Learning

Lock Granularity

- Field-level lock
  - – Allows concurrent transactions to access same row
    - Requires use of different fields (attributes) within the row
    - Most flexible type of lock but requires an extremely high level of overhead – rarely implemented

Lock Types

- Binary lock
  - Two states: locked (1) or unlocked (0)
  - Every DB operation requires that the affected object be locked
  - Transaction must unlock the object after its termination
  - Locks and unlocks are performed automatically by the DBMS, not the user

Lock Types

- Applying locks to the “Lost Update Problem”
  - Lock/unlock eliminates problem because the lock is not released until the WRITE is completed
  - PROD_QOH can not be used until it has been properly updated
  - Binary locks are considered too restrictive as the DBMS will not allow two transactions to read the same object even though neither updates the database and no concurrency problems an occur

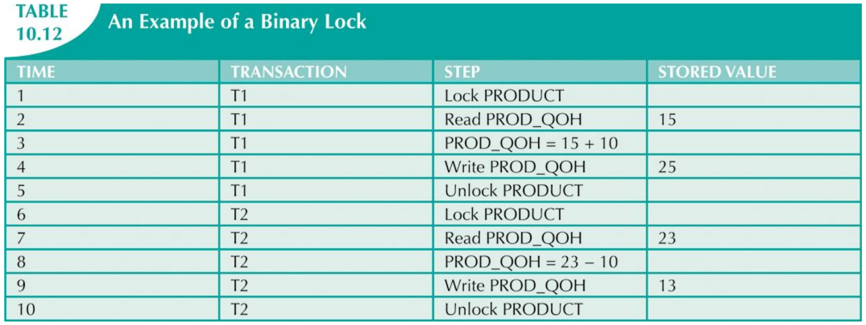


TABLE 10.12 An Example of a Binary Lock

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	15
3	T1	PROD_QOH = 15 + 10	
4	T1	Write PROD_QOH	25
5	T1	Unlock PRODUCT	
6	T2	Lock PRODUCT	
7	T2	Read PROD_QOH	23
8	T2	PROD_QOH = 23 − 10	
9	T2	Write PROD_QOH	13
10	T2	Unlock PRODUCT	

Lock Types

- Exclusive lock
  - Access is specifically reserved for transaction that locked object
  - Must be used when potential for conflict exists
    - i.e., when two operations are in conflict when they access the same data and at least one of them is performing a WRITE
- Shared lock
  - Concurrent transactions are granted read access on basis of a common lock
    - Produces no conflict as long as all the concurrent transactions are read-only

Lock Types

- A shared lock is issued when a transaction wants to read data from the DB and no exclusive lock is held on that data item
- An exclusive lock is issued when a transaction wants to update (write) a data item and no locks are currently held on that data item by any other transaction
- Thus, under these concepts, a lock can have three states
  - Unlocked
  - Shared (read)
  - Exclusive (write)

Lock Types

- If T1 has a shared lock on data item X, T2 may be issued a shared lock to read X
- If T2 wants to update X, an exclusive lock is required by T2
  - The exclusive lock is granted if and only if no other locks are held on the data item
  - If T1 has a shared lock on X. T2 can not get an exclusive lock on X – **mutual exclusive rule**
  - T2 must wait until T1 committs

Lock Types

- A shared/exclusive lock schema increases the lock manager's overhead
  - The type of lock held must be known before a lock can be granted
  - Three lock operations exist:
    - READ_LOCK to check the type of lock
    - WRITE_LOCK to issue the lock
    - UNLOCK to release the lock
    - The schema has been enhanced to allow a lock upgrade from shared to exclusive and a lock downgrade from exclusive to shared

Lock Types

- Although locks prevent serious data inconsistencies, they can lead to major problems
  - The resulting transaction schedule might not be serializable
  - The schedule might create deadlocks
    - A deadlock occurs when two transactions wait indefinitely for each other to unlock data
    - A database deadlock (similar to traffic gridlock) is caused when two or more transactions wait for each to unlock data

Two-Phase Locking to Ensure Serializability

- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but does not prevent deadlocks
  - Growing phase
    - Transaction acquires all required locks without unlocking any data
  - Shrinking phase
    - Transaction releases all locks and cannot obtain any new lock

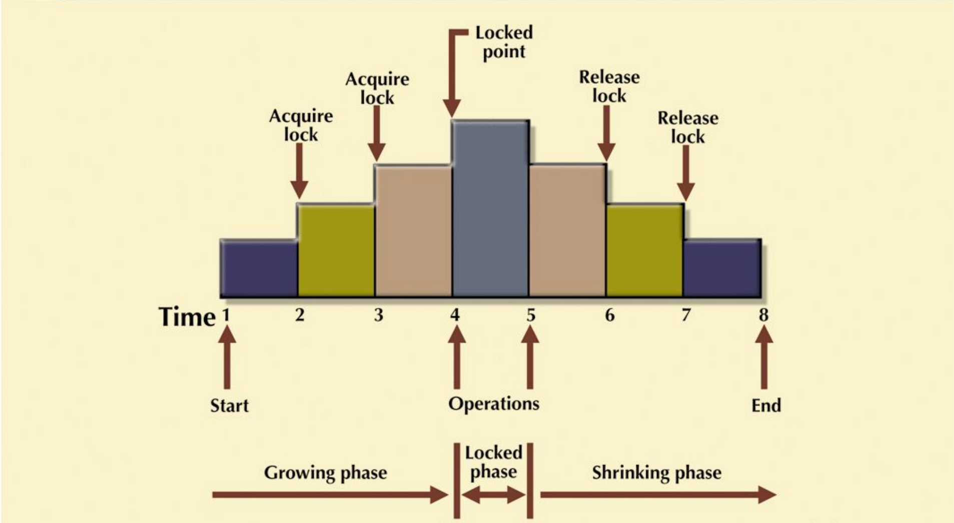
Two-Phase Locking to Ensure Serializability

- Governed by the following rules:
  - Two transactions cannot have conflicting locks
  - No unlock operation can precede a lock operation in the same transaction
  - No data are affected until all locks are obtained, meaning until the transaction is in its locked point

Two-Phase Locking to Ensure Serializability

- The transaction first acquires the two locks it needs
- When it has two locks, it reaches its locked point
- Next, the data are modified to conform to the transaction requirements
- Finally, the transaction is completed and it releases all of the locks it acquired in the first phase
- Two phase locking increases transaction processing and cost and might cause deadlocks

FIGURE 10.7 Two-phase locking protocol



SOURCE: Course Technology/Cengage Learning

Deadlocks

- Condition that occurs when two transactions wait for each other to unlock data
  - T1 = access data items X and Y
  - T2 = access data items Y and X
    - If T1 has not unlocked data item Y, T2 cannot begin
    - If T2 has not unlocked data item X, T1 cannot continue
    - T1 and T2 each wait for the other to unlock the required data item – **deadly embrace**
- Possible only if one of the transactions wants to obtain an exclusive lock on a data item
  - No deadlock condition can exist among shared locks

Deadlocks

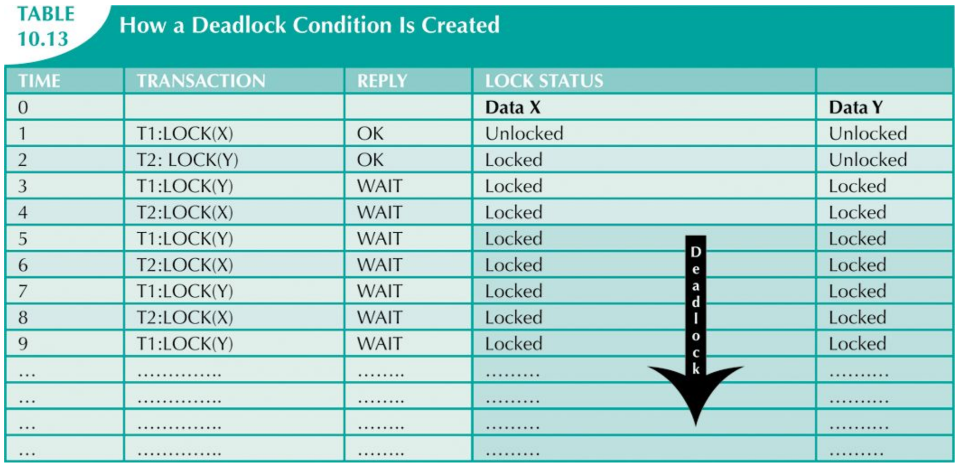


TABLE 10.13 How a Deadlock Condition Is Created

TIME	TRANSACTION	REPLY	LOCK STATUS	
0			**Data X**	**Data Y**
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....

Controlling Deadlock

- Prevention
  - A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur.
  - If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released
  - The transaction is then rescheduled for execution
  - It avoids the condition that leads to deadlocking

Controlling Deadlock

- Detection
  - The DBMS periodically tests the database for deadlocks
  - If found, the “victim” transaction is aborted (rolled back and restarted) and the other transaction continues
- Avoidance
  - The transaction must obtain all of the locks it needs before it can be executed
  - This avoids the rolling back of conflicting transactions by requiring that locks be obtained in succession
  - Serial lock assignment increases action response times

Deadlocks

- Choice of deadlock control method depends on database environment
  - Low probability of deadlock; detection recommended
  - High probability; prevention recommended
  - If response time is not high on the system’s priority list, deadlock avoidance might be employed
- All current DBMSs support deadlock detection while some use a blend of prevention and avoidance techniques for other types of data, such as data warehouses or XML data

Concurrency Control with Time Stamping Methods

- Assigns global unique time stamp to each transaction
- Produces explicit order in which transactions are submitted to DBMS
- Properties of timestamps
  - Uniqueness
    - Ensures that no equal time stamp values can exist
  - Monotonicity
    - Ensures that time stamp values always increase

Concurrency Control with Time Stamping Methods

- All database operations within the same transaction must have the same timestamp
- The DBMS executes conflicting operations in timestamp order, thereby ensuring serializability of the transactions
- If two transactions conflict, one is stopped, rolled back, rescheduled and assigned a new timestamp value

Concurrency Control with Time Stamping Methods

- Disadvantage - each value stored in the database requires two additional timestamp fields; one for the last time the field was read and one for the last update
  - This increases memory needs and the database's processing overhead
- Timestamping demands a lot of system resources because many transactions might have to be stopped, rescheduled and restamped

Wait/Die and Wound/Wait Schemes

- How to decide which transaction to rollback and which continues executing
  - **Wait/die**
    - Older transaction requests lock - it waits until younger is completes and locks released
    - Younger transaction requests lock - it dies (rolls back) and is rescheduled using the same timestamp
  - **Wound/wait**
    - Older transaction requests lock, it preempts (wounds) the younger transaction by rolling it back and rescheduling it using the same timestamp
    - Younger transaction requests lock, it will wait until the other transaction is completed and the locks released

Wait/Die and Wound/Wait Schemes

- Wait for lock request can cause some transactions to wait indefinitely, causing a deadlock
- To prevent a deadlock, each lock request has an associated time-out value
  - If the lock is not granted before the time-out expires, the transaction is rolled back

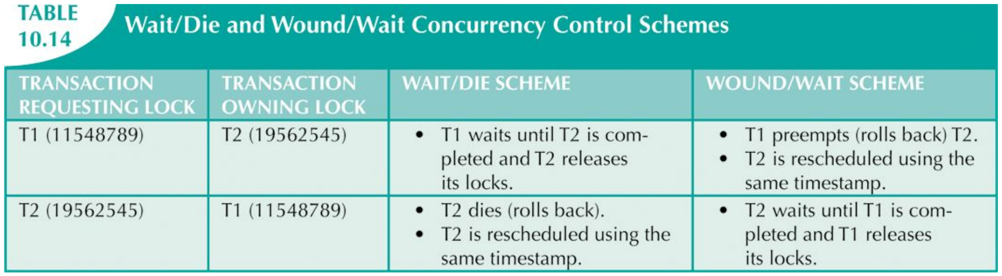


TABLE 10.14 Wait/Die and Wound/Wait Concurrency Control Schemes

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT/DIE SCHEME	WOUND/WAIT SCHEME
T1 (11548789)	T2 (19562545)	• T1 waits until T2 is completed and T2 releases its locks.	• T1 preempts (rolls back) T2. • T2 is rescheduled using the same timestamp.
T2 (19562545)	T1 (11548789)	• T2 dies (rolls back). • T2 is rescheduled using the same timestamp.	• T2 waits until T1 is completed and T1 releases its locks.

Concurrency Control with Optimistic Methods

- Optimistic approach
  - Based on assumption that majority of database operations do not conflict
  - Does not require locking or time stamping techniques
  - Transaction is executed without restrictions until it is committed
  - Each transaction moves through two or three phases - read, validation, and write

Concurrency Control with Optimistic Methods

- Read Phase
  - Transaction reads the database, executes computations, makes updates to a private copy of the database values (temporary update file) not accessed by the remaining transactions
- Validation Phase
  - Transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database
    - If validation test is positive, the transaction goes to the write phase
    - If validation test is negative, transaction is restarted and the changes discarded
- Write Phase
  - The changes are permanently applied to the database

Concurrency Control with Optimistic Methods

- This approach is acceptable for most read or query database systems that require few update transactions
- In a heavily used DBMS environment, one or more of the above techniques will be used
- However, it may be necessary at times to employ database recovery techniques to restore the database to a consistent state

Database Recovery Management

- Restores database to previous consistent state
- Based on atomic transaction property
  - All portions of transaction are treated as single logical unit of work
  - All operations are applied and completed to produce consistent database
- If transaction operation cannot be completed:
  - Transaction aborted
  - Changes to database are rolled back

Database Recovery Management

- In addition to recovering transactions, recovery techniques also apply to the database and system after some type of critical error
  - Hardware/software failures
  - Human caused incidents
  - Natural disasters

Transaction Recovery

- Use data in the transaction log to recover a database from an inconsistent state to a consistent state
- Four concepts that affect the recovery process
  - **Write-ahead-log protocol**: ensures transaction logs are written before database data is updated
    - In case of a failure, the DB can be recovered using data in the transaction log
  - **Redundant transaction logs**: ensure physical disk failure will not impair ability to recover

Transaction Recovery

- **Buffers**: temporary storage areas in primary memory
  - To improve processing time, the DBMS software reads the data from the physical disk and stores a copy of it on a “buffer” in primary memory
  - When a transaction updates data, it actually updates the copy of the data in the buffer because that’s faster than updating the physical disk
  - Later, all buffers that contain data are written to a physical disk during a single operation thereby saving significant processing time

Transaction Recovery

- **Checkpoints**: operations in which DBMS writes all its updated buffers to disk
  - While this is happening, the DBMS does not execute any other requests
  - Scheduled by the DBMS several times per hour
  - A checkpoint operation is also registered in the transaction log
  - The physical database and the transaction log are in synch
    - Synching is required because update operations update the copy of the data in the buffers and not in the physical database

Transaction Recovery

- **Deferred-write (deferred update) technique**
  - Only transaction log is updated
  - DB is physically updated only after the transaction reaches is commit point
  - If the transaction aborts before a commit, no rollback is needed

Transaction Recovery

- Recovery process for all started and committed transactions (before the failure) follows these steps
  - identify last checkpoint (i.e., when transaction data was physically saved to disk)
  - If transaction committed before checkpoint:
    - Do nothing, the data was already saved
  - If transaction committed after checkpoint:
    - Use transaction log to redo the transaction using the “after” values
      - Changes are made in oldest to newest order
  - If transaction had ROLLBACK operation after the last checkpoint or that was left active (with neither a COMMIT nor a ROLLBACK) before the failure:
    - Do nothing, DB was never updated

Transaction Recovery

- **Write-through technique**
  - Database is immediately updated by transaction operations during transaction’s execution even before the transaction reaches its commit point
- Recovery process: identify last checkpoint (when data were physically written to the disk)
  - If transaction committed before checkpoint:
    - Do nothing; data already saved
  - If transaction committed after last checkpoint:
    - DBMS redoes the transaction using “after” values in oldest to newest order
  - If transaction had ROLLBACK or was left active (with neither a COMMIT nor a ROLLBACK) before failure occurred:
    - DBMS uses the transaction log to ROLLBACK using the “before” values in newest to oldest order

Transaction Recovery

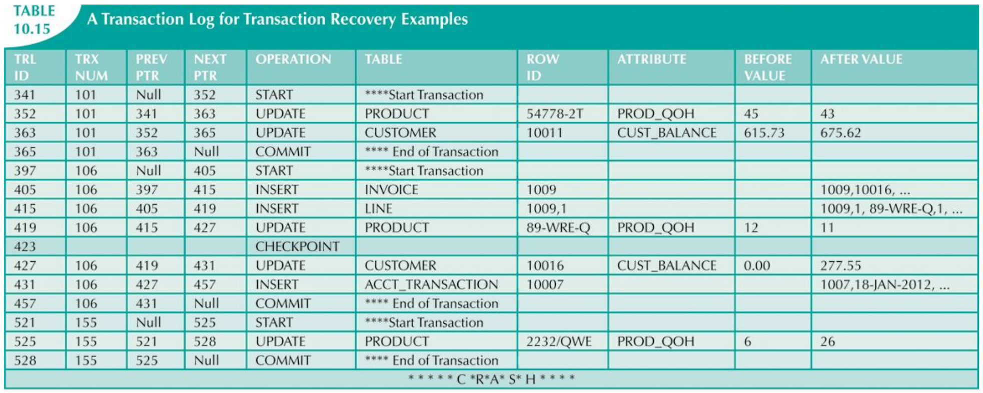


TABLE 10.15 A Transaction Log for Transaction Recovery Examples

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	****Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ...
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ...
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2012, ...
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	****Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				
* * * * * C *R*A* S* H * * * *									

Transaction Recovery

- Transaction 101 consists of two UPDATE statements that reduce the QOH for 54778-2T and increase the customer balance for customer 10011 for a credit sale of 2 units of product 54778-27
- Transaction 106 represents the credit sale of 1 unit of 89-WRE-Q to customer 10016 for $277.55. This transaction consists of 5 DML statement – 3 INSERTs and 2 UPDATEs
- Transaction 155 consists of 1 UPDATE that increases the QOH of 2232/QWE from 6 to 26
- A database checkpoint writes all updated database buffers for previously uncommitted transaction to disk. In this case, all changes made by 101.

Transaction Recovery

- Using the deferred update method, the following will happen during a recovery
  1. Identify the last checkpoint – TRL ID 423
  2. Transaction 101 started and ended before the last checkpoint so no action needs to be taken
  3. For each transaction committed after TRL ID 423, the DBMS uses the transaction log to write changes to disk using the “after” values. For transaction 106
     a. Find COMMIT (TRL ID 457)
     b. Use the previous pointer values to locate the start of the transaction (TRL ID 397)
     c. Use the next pointer values to locate each DML and apply the “after” values (TRL ID 405, 415, 419, 427 and 431)
     d. Repeat the process for transaction 155
  4. Any other transaction will be ignored as no changes were written to disk