Chapter 13
Managing Transactions and Concurrency
Objectives

- In this chapter, you will learn:
  - What a database transaction is and what its properties are
  - What concurrency control is and what role it plays in maintaining the database’s integrity
  - What locking methods are and how they work
  - How stamping methods are used for concurrency control
  - How optimistic methods are used for concurrency control
  - How database recovery management is used to maintain database integrity
What is a Transaction?

Figure 10.1: The Ch10_SaleCo database relational diagram

[Diagram showing the database schema for a sales company, including entities such as CUSTOMER, INVOICE, LINE, PRODUCT, AND ACCOUNT_TRANSACTION with relationships indicated between them.]
What is a Transaction? (continued)

• Any action that reads from and/or writes to a database may consist of:
  ▫ Simple SELECT statement to generate list of table contents
  ▫ Series of related UPDATE statements to change values of attributes in various tables
  ▫ Series of INSERT statements to add rows to one or more tables
  ▫ Combination of SELECT, UPDATE, and INSERT statements
What is a Transaction? (continued)

- Transaction is 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
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 (continued)

**Figure 10.2** Tracing the transaction in the Ch10_SaleCo database

<table>
<thead>
<tr>
<th>PROD_CODE</th>
<th>PROD_DESCRPT</th>
<th>PROD_QTY</th>
<th>PROD_MNT</th>
<th>PROD_PRICE</th>
<th>PROD_DISCOUNT</th>
<th>PROD_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1000001</td>
<td>Power pole, 3 mm, reliable</td>
<td>50</td>
<td>148.50</td>
<td>2.97</td>
<td></td>
<td>148.50</td>
</tr>
<tr>
<td>P1000002</td>
<td>Steel cable, 5 mm, 8-flame</td>
<td>100</td>
<td>0.95</td>
<td>0.95</td>
<td></td>
<td>95.00</td>
</tr>
<tr>
<td>1100001</td>
<td>Steel cable, 10 mm, 14-flame</td>
<td>500</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td>500.00</td>
</tr>
<tr>
<td>1100002</td>
<td>Steel cable, 16 mm, 18-flame</td>
<td>1000</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td>1000.00</td>
</tr>
</tbody>
</table>

**Table 10.2** SaleCo database table

<table>
<thead>
<tr>
<th>CUST_NAME</th>
<th>CIST поболее</th>
<th>CUST_ADDR</th>
<th>CUST_PHONE</th>
<th>CUST_BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>1234</td>
<td>5678</td>
<td>90123</td>
<td>1234567</td>
</tr>
<tr>
<td>Brown</td>
<td>2345</td>
<td>6789</td>
<td>34567</td>
<td>7890123</td>
</tr>
<tr>
<td>Davis</td>
<td>3456</td>
<td>7890</td>
<td>45678</td>
<td>8901234</td>
</tr>
</tbody>
</table>

Transaction Properties

- **Atomicity**
  - Requires that all operations (SQL requests) of a transaction be completed

- **Consistency**
  - Indicates the permanence of database’s consistent state
Transaction Properties (continued)

- Isolation
  - Data used during execution of a transaction cannot be used by second transaction until first one is completed

- Durability
  - Indicates permanence of database’s consistent state
Transaction Properties (continued)

• Serializability
  ▫ Ensures that concurrent execution of several transactions yields consistent results
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 Management with SQL (continued)

- ANSI standards require that, when a transaction sequence is initiated by a user or an application program, it must continue through all succeeding SQL statements until one of four events occurs:
  - COMMIT statement is reached
  - ROLLBACK statement is reached
  - End of program is reached
  - Program is abnormally terminated
The Transaction Log

- Transaction log stores:
  - A record for the beginning of transaction
  - For each transaction component (SQL statement):
    - 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 (continued)

### Table 10.1: A Transaction Log

<table>
<thead>
<tr>
<th>TRL_ID</th>
<th>TRX_NUM</th>
<th>PREV_PTR</th>
<th>NEXT_PTR</th>
<th>OPERATION</th>
<th>TABLE</th>
<th>ROW ID</th>
<th>ATTRIBUTE</th>
<th>BEFORE VALUE</th>
<th>AFTER VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>101</td>
<td>Null</td>
<td>352</td>
<td>START</td>
<td><strong>Start Transaction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>101</td>
<td>341</td>
<td>363</td>
<td>UPDATE</td>
<td>PRODUCT</td>
<td>1558-QW1</td>
<td>PROD_QOH</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>363</td>
<td>101</td>
<td>352</td>
<td>365</td>
<td>UPDATE</td>
<td>CUSTOMER</td>
<td>10011</td>
<td>CUST_BALANCE</td>
<td>525.75</td>
<td>615.73</td>
</tr>
<tr>
<td>365</td>
<td>101</td>
<td>363</td>
<td>Null</td>
<td>COMMIT</td>
<td><strong>End of Transaction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **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 system
- Objective is to ensure serializability of transactions in a multiuser database environment
Concurrent Control (continued)

• Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems
  ▫ Lost updates
  ▫ Uncommitted data
  ▫ Inconsistent retrievals
## Lost Updates

### TABLE 10.2
Normal Execution of Two Transactions

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>STEP</th>
<th>STORED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>PROD_QOH = 35 + 100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Write PROD_QOH</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>T2</td>
<td>Read PROD_QOH</td>
<td>135</td>
</tr>
<tr>
<td>5</td>
<td>T2</td>
<td>PROD_QOH = 135 - 30</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T2</td>
<td>Write PROD_QOH</td>
<td>105</td>
</tr>
</tbody>
</table>
## Lost Updates (continued)

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>STEP</th>
<th>STORED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>T2</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>PROD_QOH = 35 + 100</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T2</td>
<td>PROD_QOH = 35 - 30</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T1</td>
<td>Write PROD_QOH (Lost update)</td>
<td>135</td>
</tr>
<tr>
<td>6</td>
<td>T2</td>
<td>Write PROD_QOH</td>
<td>5</td>
</tr>
</tbody>
</table>
## Uncommitted Data

### Table 10.4
Correct Execution of Two Transactions

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>STEP</th>
<th>STORED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>PROD_QOH = 35 + 100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Write PROD_QOH</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>T1</td>
<td>*****ROLLBACK *****</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>T2</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>T2</td>
<td>PROD_QOH = 35 - 30</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T2</td>
<td>Write PROD_QOH</td>
<td>5</td>
</tr>
</tbody>
</table>
### Uncommitted Data (continued)

#### TABLE 10.5

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>STEP</th>
<th>STORED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Read PROD_QOH</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>PROD_QOH = 35 + 100</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>Write PROD_QOH</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>T2</td>
<td>Read PROD_QOH (Read uncommitted data)</td>
<td>135</td>
</tr>
<tr>
<td>5</td>
<td>T2</td>
<td>PROD_QOH = 135 – 30</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>T1</td>
<td>***** ROLLBACK *****</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>T2</td>
<td>Write PROD_QOH</td>
<td>105</td>
</tr>
</tbody>
</table>
## Inconsistent Retrievals

<table>
<thead>
<tr>
<th>TRANSACTION T1</th>
<th>TRANSACTION T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT SUM(PROD_QOH) FROM PRODUCT</td>
<td>UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 10 WHERE PROD_CODE = ‘1546-QQ2’</td>
</tr>
<tr>
<td></td>
<td>UPDATE PRODUCT SET PROD_QOH = PROD_QOH - 10 WHERE PROD_CODE = ‘1558-QW1’</td>
</tr>
<tr>
<td></td>
<td>COMMIT;</td>
</tr>
</tbody>
</table>
### Inconsistent Retrievals (continued)

#### Table 10.7: Transaction Results: Data Entry Correction

<table>
<thead>
<tr>
<th>PROD_CODE</th>
<th>BEFORE PROD_QOH</th>
<th>AFTER PROD_QOH</th>
</tr>
</thead>
<tbody>
<tr>
<td>11QER/31</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>13-Q2/P2</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>1546-QQ2</td>
<td>15</td>
<td>(15 + 10) → 25</td>
</tr>
<tr>
<td>1558-QW1</td>
<td>23</td>
<td>(23 - 10) → 13</td>
</tr>
<tr>
<td>2232-QTY</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2232-QWE</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>92</strong></td>
<td><strong>92</strong></td>
</tr>
</tbody>
</table>
Inconsistent Retrievals (continued)

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>ACTION</th>
<th>VALUE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’11QER/31’</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’13-Q2/P2’</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>T2</td>
<td>Read PROD_QOH for PROD_CODE = ’1546-QQ2’</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T2</td>
<td>PROD_QOH = 15 + 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T2</td>
<td>Write PROD_QOH for PROD_CODE = ’1546-QQ2’</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’1546-QQ2’</td>
<td>25</td>
<td>(After) 65</td>
</tr>
<tr>
<td>7</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’1558-QW1’</td>
<td>23</td>
<td>(Before) 88</td>
</tr>
<tr>
<td>8</td>
<td>T2</td>
<td>Read PROD_QOH for PROD_CODE = ’1558-QW1’</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>T2</td>
<td>PROD_QOH = 23 − 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>T2</td>
<td>Write PROD_QOH for PROD_CODE = ’1558-QW1’</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>T2</td>
<td>***** COMMIT *****</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’2232-QTY’</td>
<td>8</td>
<td>96</td>
</tr>
<tr>
<td>13</td>
<td>T1</td>
<td>Read PROD_QOH for PROD_CODE = ’2232-QWE’</td>
<td>6</td>
<td>102</td>
</tr>
</tbody>
</table>
The Scheduler

- Special DBMS program
  - Purpose is to establish order of operations within which concurrent transactions are executed
- Interleaves execution of database operations to ensure serializability and isolation of transactions
The Scheduler (continued)

- Bases its actions on concurrency control algorithms
- Ensures computer’s central processing unit (CPU) is used efficiently
- Facilitates data isolation to ensure that two transactions do not update same data element at same time
## The Scheduler (continued)

### Table 10.9

**Read/Write Conflict Scenarios: Conflicting Database Operations Matrix**

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>T1</th>
<th>T2</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
<td>No conflict</td>
</tr>
<tr>
<td>Read</td>
<td>Write</td>
<td></td>
<td>Conflict</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td></td>
<td>Conflict</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
<td>Conflict</td>
</tr>
</tbody>
</table>

Concurrency Control with Locking Methods

• **Lock**
  - Guarantees exclusive use of a data item to a current transaction
  - Required to prevent another transaction from reading inconsistent data

• **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 (continued)

- Database-level lock
  - Entire database is locked
- Table-level lock
  - Entire table is locked
- Page-level lock
  - Entire diskpage is locked
Lock Granularity (continued)

- **Row-level lock**
  - Allows concurrent transactions to access different rows of same table, even if rows are located on same page

- **Field-level lock**
  - Allows concurrent transactions to access same row, as long as they require use of different fields (attributes) within that row
Lock Granularity (continued)
Lock Granularity (continued)

FIGURE 10.4 An example of a table-level lock

Time
1
2
3
4
5
6
7
8
9

Transaction 1 (T1) (Update row 5)

Transaction 2 (T2) (Update row 30)

Payroll Database

Table A

Lock Table A request

Locked

OK

Unlocked (end of transaction 1)

Unlocker (end of transaction 2)

Lock Table A request

WAIT

OK

Locked
Lock Granularity (continued)

FIGURE 10.5
An example of a page-level lock

Time
1
2
3
4
5
6
7

Transaction 1 (T1)
(Update row 1)

Lock page 1 request

Locked

OK

Transaction 2 (T2)
(Update rows 5 and 2)

Lock page 2 request

OK

Locked

Row number

Unlock page 1
(end of transaction)

Unlock pages 1 and 2
(end of transaction)
Lock Granularity (continued)

An example of a row-level lock

Time
1
2
3
4
5
6

Transaction 1 (T1)
(Update row 1)

Transaction 2 (T2)
(Update row 2)

Payroll Database

Table A
1
2
3
4
5
6

Page 1

Page 2

Row number

Lock row 1 request
Locked
OK
Unlock row 1 (end of transaction)

Lock row 2 request
OK
Locked
Unlock row 2 (end of transaction)
Lock Types

- **Binary lock**
  - Has only two states: locked (1) or unlocked (0)

- **Exclusive lock**
  - Access is specifically reserved for transaction that locked object
  - Must be used when potential for conflict exists

- **Shared lock**
  - Concurrent transactions are granted Read access on basis of a common lock
Lock Types (continued)

### An Example of a Binary Lock

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>STEP</th>
<th>STORED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>Lock PRODUCT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>Read PROD_QOH</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>PROD_QOH = 15 + 10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T1</td>
<td>Write PROD_QOH</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>T1</td>
<td>Unlock PRODUCT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T2</td>
<td>Lock PRODUCT</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T2</td>
<td>Read PROD_QOH</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>T2</td>
<td>PROD_QOH = 23 - 10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>T2</td>
<td>Write PROD_QOH</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>T2</td>
<td>Unlock PRODUCT</td>
<td></td>
</tr>
</tbody>
</table>
Two-Phase Locking to Ensure Serializability

- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but it 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 (continued)

- 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—that is, until transaction is in its locked point
Two-Phase Locking to Ensure Serializability (continued)

FIGURE 10.7 Two-phase locking protocol

Time 1 2 3 4 5 6 7 8
Start Operations Release lock Release lock
Growing phase Locked phase Shrinking phase

Deadlocks

- Condition that occurs when two transactions wait for each other to unlock data
- 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 (continued)

- Control through:
  - Prevention
  - Detection
  - Avoidance
Deadlocks (continued)

<table>
<thead>
<tr>
<th>TIME</th>
<th>TRANSACTION</th>
<th>REPLY</th>
<th>LOCK STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T1:LOCK(X)</td>
<td>OK</td>
<td>Data X: Data Y</td>
</tr>
<tr>
<td>1</td>
<td>T2:LOCK(Y)</td>
<td>OK</td>
<td>Unlocked: Unlocked</td>
</tr>
<tr>
<td>2</td>
<td>T1:LOCK(Y)</td>
<td>WAIT</td>
<td>Locked: Unlocked</td>
</tr>
<tr>
<td>3</td>
<td>T2:LOCK(X)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>4</td>
<td>T1:LOCK(Y)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>5</td>
<td>T2:LOCK(X)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>6</td>
<td>T1:LOCK(Y)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>7</td>
<td>T2:LOCK(X)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>8</td>
<td>T1:LOCK(Y)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
<tr>
<td>9</td>
<td>T2:LOCK(X)</td>
<td>WAIT</td>
<td>Locked: Locked</td>
</tr>
</tbody>
</table>

...
Concurrency Control with Time Stamping Methods

• Assigns global unique time stamp to each transaction
• Produces explicit order in which transactions are submitted to DBMS
• Uniqueness
  ▫ Ensures that no equal time stamp values can exist
• Monotonicity
  ▫ Ensures that time stamp values always increase
Wait/Die and Wound/Wait Schemes

- **Wait/die**
  - Older transaction waits and younger is rolled back and rescheduled

- **Wound/wait**
  - Older transaction rolls back younger transaction and reschedules it
## Wait/Die and Wound/Wait Schemes (continued)

<table>
<thead>
<tr>
<th>TRANSACTION REQUESTING LOCK</th>
<th>TRANSACTION OWNING LOCK</th>
<th>WAIT/DIE SCHEME</th>
<th>WOUND/WAIT SCHEME</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (11548789)</td>
<td>T2 (19562545)</td>
<td>• T1 waits until T2 is completed and T2 releases its locks.</td>
<td>• T1 preempts (rolls back) T2. T2 is rescheduled using the same time stamp.</td>
</tr>
<tr>
<td>T2 (19562545)</td>
<td>T1 (11548789)</td>
<td>• T2 dies (rolls back). T2 is rescheduled using the same time stamp.</td>
<td>• T2 waits until T1 is completed and T1 releases its locks.</td>
</tr>
</tbody>
</table>
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
  - Phases are read, validation, and write
Database Recovery Management

- Database recovery
  - Restores database from given state, usually inconsistent, to previously consistent state
  - Based on atomic transaction property
    - All portions of transaction must be treated as single logical unit of work, so all operations must be applied and completed to produce consistent database
    - If transaction operation cannot be completed, transaction must be aborted, and any changes to database must be rolled back (undone)
Transaction Recovery

- Makes use of deferred-write and write-through techniques
- Deferred write
  - Transaction operations do not immediately update physical database
  - Only transaction log is updated
  - Database is physically updated only after transaction reaches its commit point using transaction log information
Transaction Recovery (continued)

• **Write-through**
  ▪ Database is immediately updated by transaction operations during transaction’s execution, even before transaction reaches its commit point
## Transaction Recovery (continued)

### Table 10.13

<table>
<thead>
<tr>
<th>TRL ID</th>
<th>TRX NUM</th>
<th>PREV PTR</th>
<th>NEXT PTR</th>
<th>OPERATION</th>
<th>TABLE</th>
<th>ROW ID</th>
<th>ATTRIBUTE</th>
<th>BEFORE VALUE</th>
<th>AFTER VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>101</td>
<td>Null</td>
<td>352</td>
<td>START</td>
<td>**** Start Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>101</td>
<td>341</td>
<td>363</td>
<td>UPDATE</td>
<td>PRODUCT</td>
<td>54778-2T</td>
<td>PROD_QOH</td>
<td>45</td>
<td>43</td>
</tr>
<tr>
<td>363</td>
<td>101</td>
<td>352</td>
<td>365</td>
<td>UPDATE</td>
<td>CUSTOMER</td>
<td>10011</td>
<td>CUST_BALANCE</td>
<td>615.73</td>
<td>675.62</td>
</tr>
<tr>
<td>365</td>
<td>101</td>
<td>363</td>
<td>Null</td>
<td>COMMIT</td>
<td>**** End of Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>397</td>
<td>106</td>
<td>Null</td>
<td>405</td>
<td>START</td>
<td>**** Start Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>405</td>
<td>106</td>
<td>397</td>
<td>415</td>
<td>INSERT</td>
<td>INVOICE</td>
<td>1009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>415</td>
<td>106</td>
<td>405</td>
<td>419</td>
<td>INSERT</td>
<td>LINE</td>
<td>1009,1</td>
<td></td>
<td>1009,1, 89-WRE-Q,1, ...</td>
<td></td>
</tr>
<tr>
<td>419</td>
<td>106</td>
<td>415</td>
<td>427</td>
<td>UPDATE</td>
<td>PRODUCT</td>
<td>89-WRE-Q</td>
<td>PROD_QOH</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>423</td>
<td></td>
<td></td>
<td></td>
<td>CHECKPOINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>427</td>
<td>106</td>
<td>419</td>
<td>431</td>
<td>UPDATE</td>
<td>CUSTOMER</td>
<td>10016</td>
<td>CUST_BALANCE</td>
<td>0.00</td>
<td>277.55</td>
</tr>
<tr>
<td>431</td>
<td>106</td>
<td>427</td>
<td>457</td>
<td>INSERT</td>
<td>ACCT_TRANSACTION</td>
<td>10007</td>
<td></td>
<td>1007,18-JAN-2004, ...</td>
<td></td>
</tr>
<tr>
<td>457</td>
<td>106</td>
<td>431</td>
<td>Null</td>
<td>COMMIT</td>
<td>**** End of Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>521</td>
<td>155</td>
<td>Null</td>
<td>525</td>
<td>START</td>
<td>**** Start Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>525</td>
<td>155</td>
<td>521</td>
<td>528</td>
<td>UPDATE</td>
<td>PRODUCT</td>
<td>2232/QWE</td>
<td>PROD_QOH</td>
<td>6</td>
<td>26</td>
</tr>
<tr>
<td>528</td>
<td>155</td>
<td>525</td>
<td>Null</td>
<td>COMMIT</td>
<td>**** End of Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

* **** C *R*A* S* H ****

Summary

• Transaction
  ▫ **Sequence of database operations that access database**
  ▫ **Represents real-world events**
  ▫ **Must be logical unit of work**
    • No portion of transaction can exist by itself
  ▫ **Takes database from one consistent state to another**
    • One in which all data integrity constraints are satisfied
Summary (continued)

• Transactions have five main properties: atomicity, consistency, isolation, durability, and serializability
• SQL provides support for transactions through the use of two statements: COMMIT and ROLLBACK
• SQL transactions are formed by several SQL statements or database requests
Summary (continued)

- Transaction log keeps track of all transactions that modify database
- Concurrency control coordinates simultaneous execution of transactions
- Scheduler is responsible for establishing order in which concurrent transaction operations are executed
Summary (continued)

• Lock guarantees unique access to a data item by transaction
• Two types of locks can be used in database systems: binary locks and shared/exclusive locks
• Serializability of schedules is guaranteed through the use of two-phase locking
Summary (continued)

- When two or more transactions wait indefinitely for each other to release lock, they are in deadlock, or deadly embrace
- Three deadlock control techniques: prevention, detection, and avoidance
Concurrency control with time stamping methods assigns unique time stamp to each transaction and schedules execution of conflicting transactions in time stamp order.
Summary (continued)

• Concurrency control with optimistic methods assumes that the majority of database transactions do not conflict and that transactions are executed concurrently, using private copies of the data
• Database recovery restores database from given state to previous consistent state