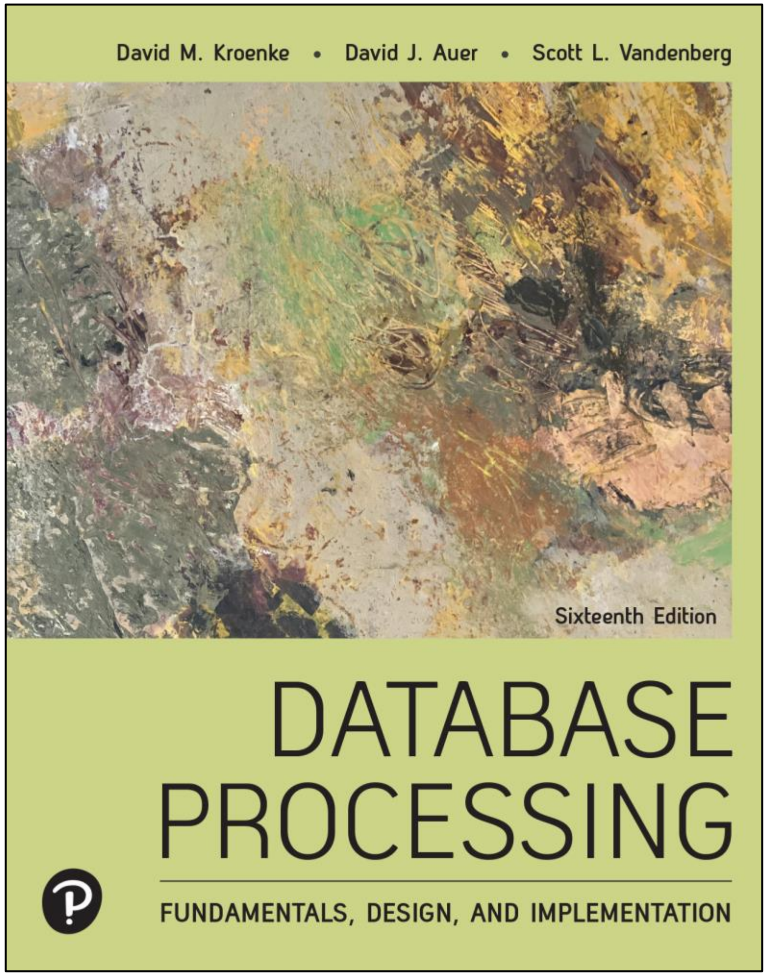
Database Processing: Fundamentals, Design, and Implementation

Sixteenth Edition


David M. Kroenke • David J. Auer • Scott L. Vandenberg
Sixteenth Edition
DATABASE PROCESSING
FUNDAMENTALS, DESIGN, AND IMPLEMENTATION

Chapter 9

Managing Enterprise Databases

Learning Objectives (1 of 2)

9.1 To understand the need for and importance of database administration

9.2 To understand the need for concurrency control, security, and backup and recovery

9.3 To learn about typical problems that can occur when multiple users process a database concurrently

9.4 To understand the use of locking and the problem of deadlock

9.5 To learn the difference between optimistic and pessimistic locking

9.6 To know the meaning of an ACID transaction

9.7 To learn the four ANSI standard isolation levels

Learning Objectives (2 of 2)

9.8 To learn different ways of processing a database using cursors

9.9 To understand the need for security and specific tasks for improving database security

9.10 To know the difference between recovery via reprocessing and recovery via rollback/rollforward

9.11 To understand the nature of the tasks required for recovery using rollback/rollforward

9.12 To know basic administrative and managerial DBA functions

9.13 To lean some basic terms and concepts involved in physical database design and query optimization

Database Administration

- All large and small databases need database administration.
- **Data administration** refers to a function that applies to an entire organization.
- **Database administration (DBA)** refers to a more technical function that is specific to a particular database, including the applications that process that database.

Figure 9-1: Summary of Database Administration Tasks

Summary of Database Administration Tasks

Manage database structure

Control concurrent processing

Manage processing rights and responsibilities

Develop database security

Provide for database recovery

Manage the DBMS

Maintain the data repository

Figure 9-2: DBA Responsibilities

Participate in Database and Application Development

Assist in the requirements analysis stage and data model creation

Play an active role in database design and creation

Facilitate Changes to Database Structure

Seek communitywide solutions

Assess impact on all users

Provide configuration control forum

Be prepared for problems after changes are made

Maintain documentation

Physical Database Design and Optimization

- An **index** is a secondary structure that can speed up access to data for certain types of queries.
  - An index is one component of what is called a **physical database design**.
- A DBA:
  - Must have a good understanding of the relative sizes and speeds of primary and secondary memory to create the best physical database design.
  - May also be able to influence the amount of replication of data.
    - **Disk mirroring** – maintaining two copies of data so a disk failure need not result in unavailable data.
    - Various forms of **redundant arrays of independent disks** (**RAID**) storage
- A **query optimizer** evaluates ways to execute a query and picks the one with the lowest expected execution time.

Concurrency Control

- **Concurrency control** ensures that one user’s work does not inappropriately influence another user’s work.
  - No single concurrency control technique is ideal for all circumstances.
  - Trade-offs need to be made between level of protection and throughput.

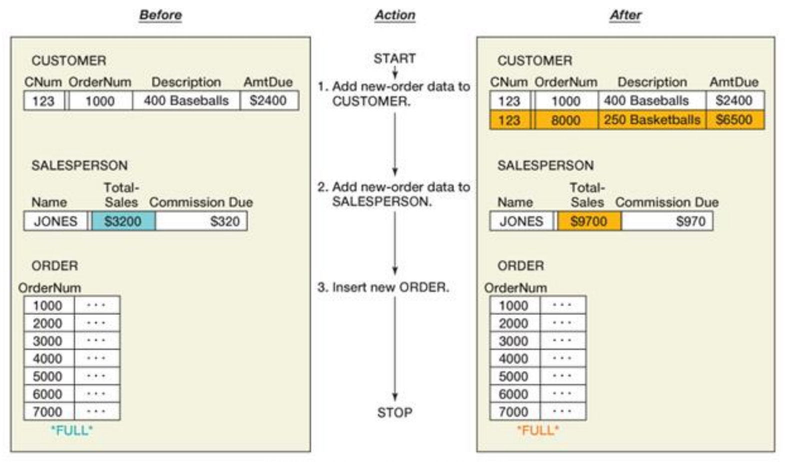
Atomic Transactions

- A **transaction**, or **logical unit of work (LUW),** is a series of actions taken against the database that occurs as an atomic unit:
  - Either all actions in a transaction occur or none of them do.
- Examples include:
  - Change a customer’s row, increasing AmountDue.
  - Change a salesperson’s row, increasing CommissionsDue.
  - Insert a new order row into the database.

Figure 9-3: Transaction Processing System

(a) Errors Introduced Without Transaction

(b) Atomic Transaction Prevents Errors



Before
Action
After
START
1. Add new-order data to CUSTOMER.
2. Add new-order data to SALESPERSON.
3. Insert new ORDER.
STOP
"FULL"
"FULL"

Before:

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	...
2000	...
3000	...
4000	...
5000	...
6000	...
7000	...

After:

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400
123	8000	250 Basketballs	$6500

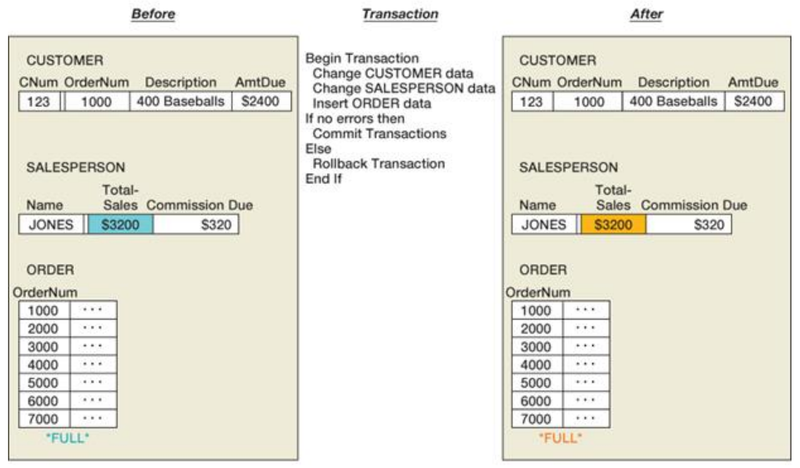
SALESPERSON

Name	Total-Sales	Commission Due
JONES	$9700	$970

ORDER

OrderNum	
1000	...
2000	...
3000	...
4000	...
5000	...
6000	...
7000	...

(a) Errors Introduced Without Transaction



```
Begin Transaction
  Change CUSTOMER data
  Change SALESPERSON data
  Insert ORDER data
If no errors then
  Commit Transactions
Else
  Rollback Transaction
End If
```

Before:

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	...
2000	...
3000	...
4000	...
5000	...
6000	...
7000	...

After:

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	...
2000	...
3000	...
4000	...
5000	...
6000	...
7000	...

(b) Atomic Transaction Prevents Errors

Concurrent Transactions

- **Concurrent transactions** refer to two or more transactions that appear to users as they are being processed against a database at the same time.
- In reality, the CPU can execute only one instruction at a time.
  - **Transactions are interleaved**:
    - The operating system quickly switches CPU services among tasks so that some portion of each of them is carried out in a given interval.
- Concurrency problems include lost updates and inconsistent reads.

Figure 9-4: Concurrent-Processing Example

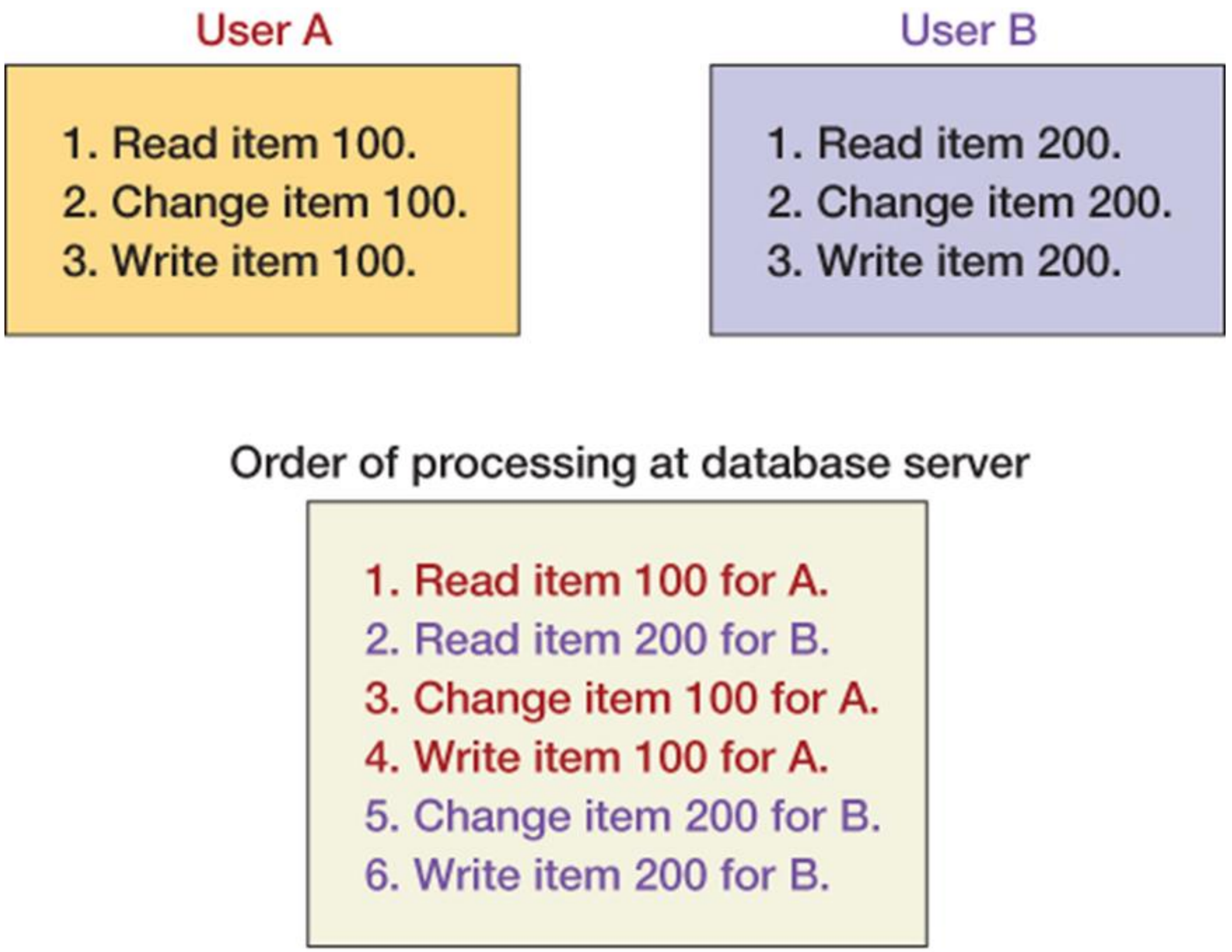
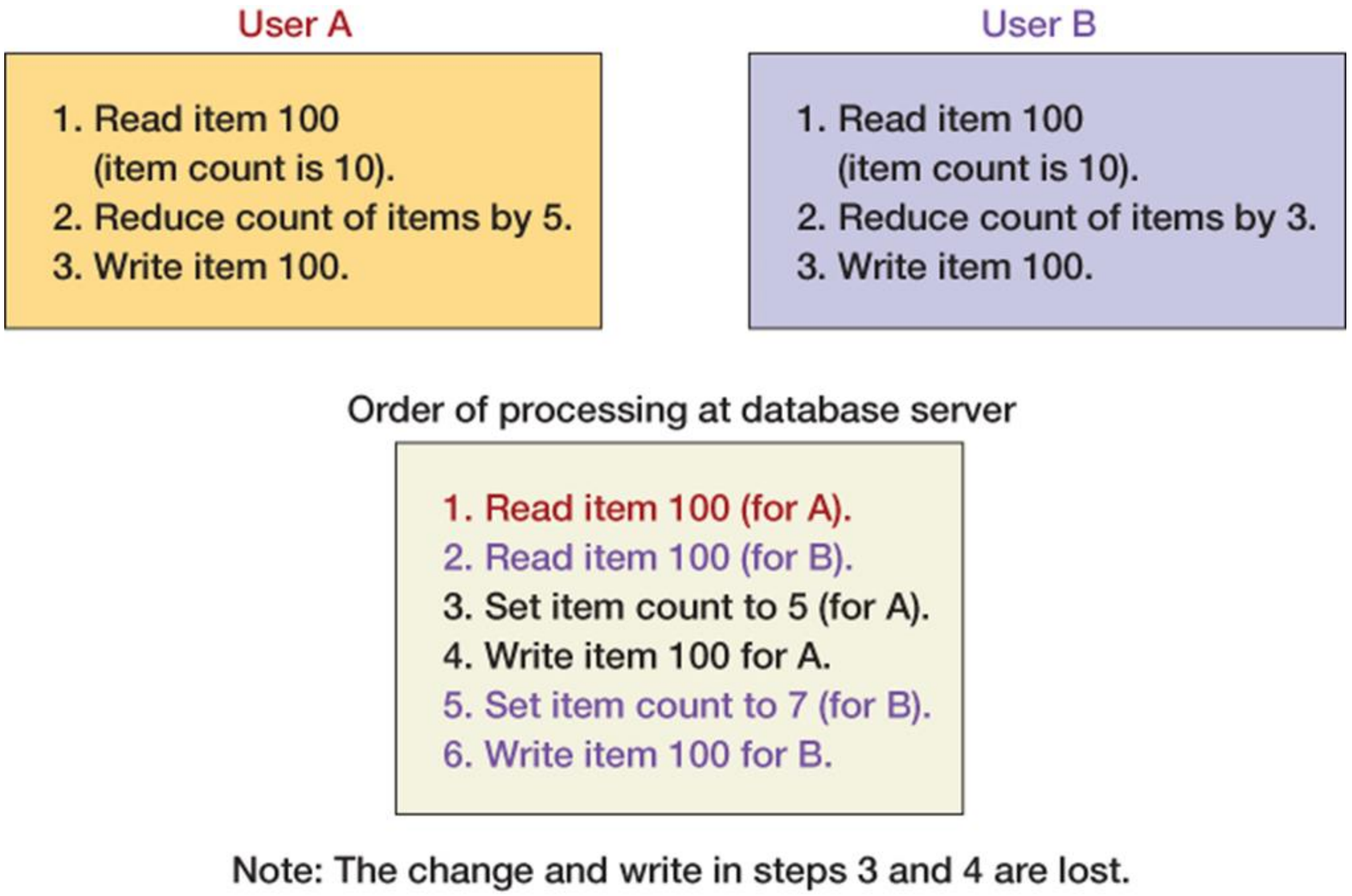


Figure 9-5: Lost Update Problem


User A
1. Read item 100 (item count is 10).
2. Reduce count of items by 5.
3. Write item 100.
User B
1. Read item 100 (item count is 10).
2. Reduce count of items by 3.
3. Write item 100.
Order of processing at database server
1. Read item 100 (for A).
2. Read item 100 (for B).
3. Set item count to 5 (for A).
4. Write item 100 for A.
5. Set item count to 7 (for B).
6. Write item 100 for B.
Note: The change and write in steps 3 and 4 are lost.

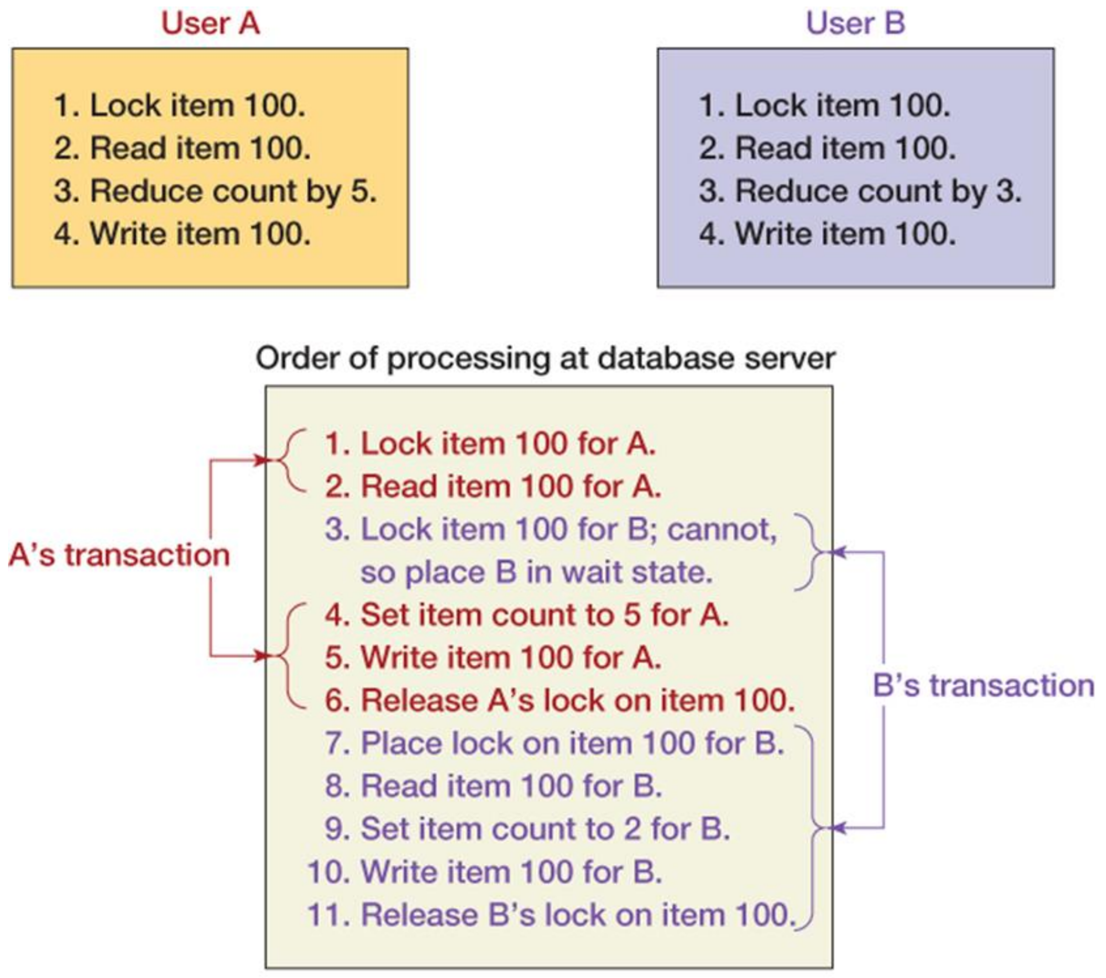
Resource Locking

- **Resource locking** prevents multiple applications from obtaining copies of the same record when the record is about to be changed.

Locking Terminology

- **Implicit locks** are locks placed by the DBMS.
- **Explicit locks** are issued by the application program.
- **Lock granularity** refers to size of a locked resource:
  - Rows, page, table, and database level
- Large granularity is easy to manage but frequently causes conflicts.
- Types of lock:
  - An **exclusive lock** prohibits other users from reading the locked resource.
  - A **shared lock** allows other users to read the locked resource, but they cannot update it.

Figure 9-6: Concurrent Processing with Explicit Locks



Serializable Transactions

- **Serializable transactions** refer to two transactions that run concurrently and generate results that are consistent with the results that would have occurred if they had run separately.
- **Two-phase locking** refers to one of the techniques used to achieve serializability.

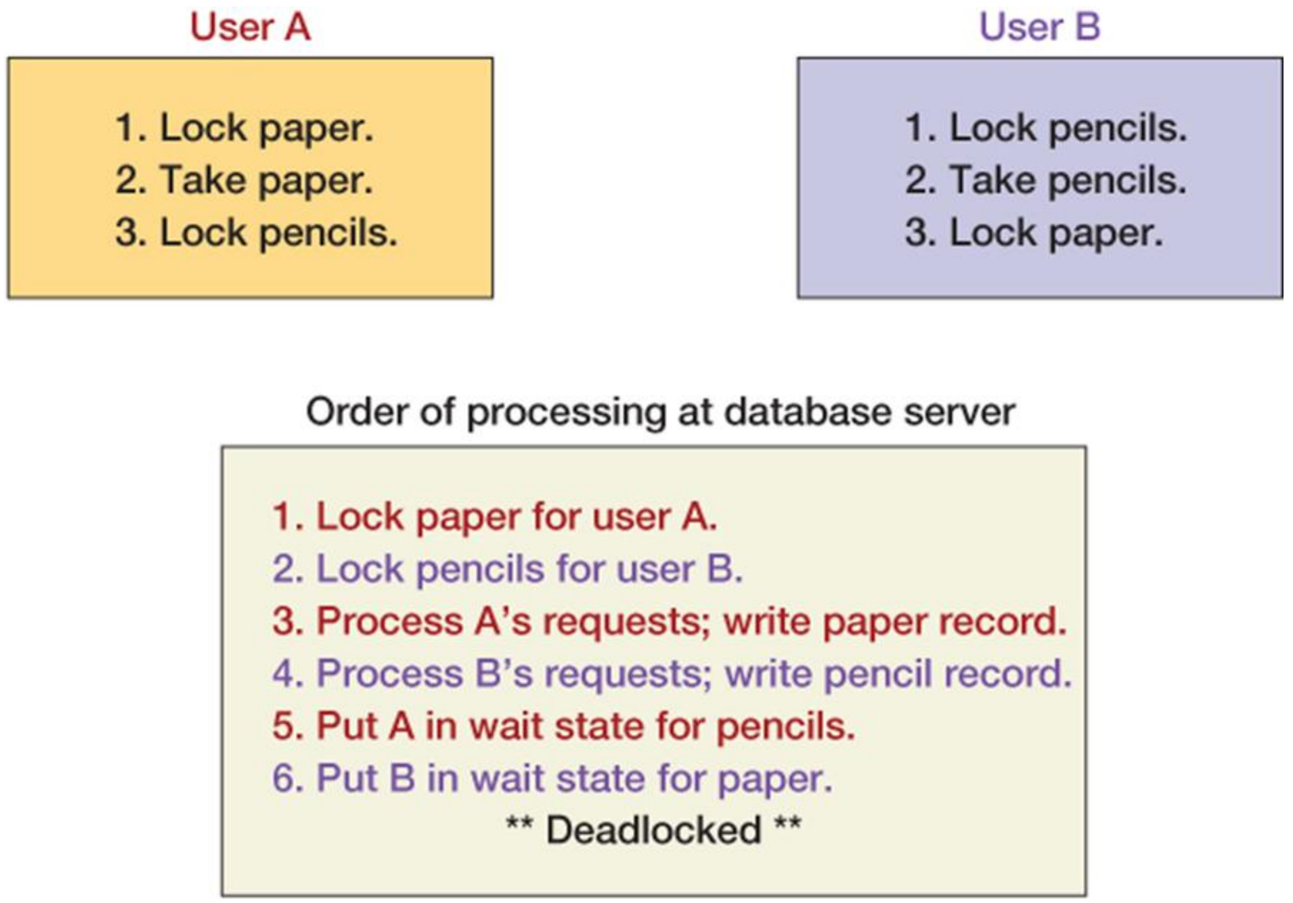
Two-Phased Locking

- **Two-phased locking**
  - Transactions are allowed to obtain locks as necessary (**growing phase**).
  - Once the first lock is released (**shrinking phase**), no other lock can be obtained.
- A special case of two-phased locking.
  - Locks are obtained throughout the transaction.
  - No lock is released until the COMMIT or ROLLBACK command is issued.
  - This strategy is more restrictive but easier to implement than two-phase locking.

Deadlock

- Deadlock, or the deadly embrace, occurs when two transactions are each waiting on a resource that the other transaction holds.
- Preventing deadlock:
  - Allow users to issue all lock requests at one time.
  - Require all application programs to lock resources in the same order.
- Breaking deadlock:
  - Almost every DBMS has algorithms for detecting deadlock. When deadlock occurs, DBMS aborts one of the transactions and rolls back partially completed work.

Figure 9-7: Deadlock Example



User A

1. Lock paper.
2. Take paper.
3. Lock pencils.

User B

1. Lock pencils.
2. Take pencils.
3. Lock paper.

Order of processing at database server

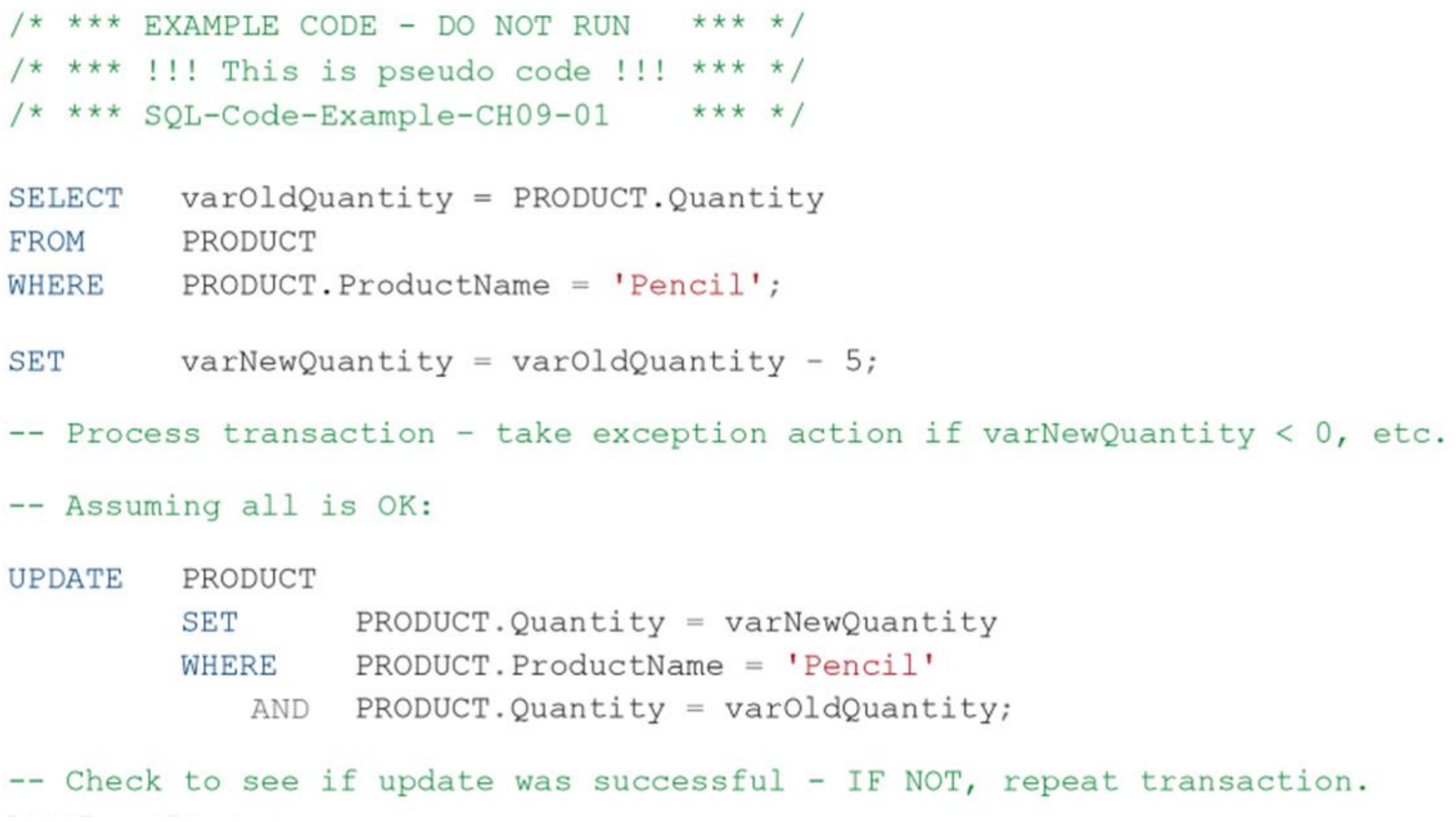
1. Lock paper for user A.
2. Lock pencils for user B.
3. Process A's requests; write paper record.
4. Process B's requests; write pencil record.
5. Put A in wait state for pencils.
6. Put B in wait state for paper.

** Deadlocked **

Optimistic Versus Pessimistic Locking

- **Optimistic locking** assumes that no transaction conflict will occur.
  - DBMS processes a transaction; checks whether conflict occurred:
    - If not, the transaction is finished.
    - If so, the transaction is repeated until there is no conflict.
- **Pessimistic locking** assumes that conflict will occur.
  - Locks are issued while the transaction is processed, and then the locks are released.
- Optimistic locking is preferred for the Internet and for many intranet applications.

Figure 9-8: Optimistic Locking



```
/* *** EXAMPLE CODE - DO NOT RUN    *** */
/* *** !!! This is pseudo code !!! *** */
/* *** SQL-Code-Example-CH09-01    *** */

SELECT    varOldQuantity = PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.ProductName = 'Pencil';

SET       varNewQuantity = varOldQuantity - 5;

-- Process transaction - take exception action if varNewQuantity < 0, etc.

-- Assuming all is OK:

UPDATE    PRODUCT
          SET       PRODUCT.Quantity = varNewQuantity
          WHERE     PRODUCT.ProductName = 'Pencil'
              AND   PRODUCT.Quantity = varOldQuantity;

-- Check to see if update was successful - IF NOT, repeat transaction.
```

Figure 9-9: Pessimistic Locking

```
/* *** EXAMPLE CODE - DO NOT RUN    *** */
/* *** !!! This is pseudo code !!! *** */
/* *** SQL-Code-Example-CH09-02    *** */

LOCK      PRODUCT EXCLUSIVE MODE;

SELECT    varOldQuantity = PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.ProductName = 'Pencil';

SET       varNewQuantity = varOldQuantity - 5;

-- Process transaction - take exception action if varNewQuantity < 0, etc.

-- Assuming all is OK:

UPDATE    PRODUCT
          SET       PRODUCT.Quantity = varNewQuantity
          WHERE     PRODUCT.ProductName = 'Pencil'
             AND    PRODUCT.Quantity = varOldQuantity;

UNLOCK    PRODUCT;

-- No need to check to see if update was successful.
```

Declaring Lock Characteristics

- Most application programs do not explicitly declare locks due to its complication.
- Instead, they mark **transaction boundaries** and declare locking behavior they want the DBMS to use.
  - Transaction boundary markers (syntax varies with DBMS):
    - **SQL BEGIN TRANSACTION statement**
    - **SLCOMMIT TRANSACTION statement**
    - **SQL ROLLBACK TRANSACTION statement**
- Advantage:
  - If the locking behavior needs to be changed, only the lock declaration need be changed, not the application program.

Figure 9-10: Marking Transaction Boundaries

```
/* *** EXAMPLE CODE - DO NOT RUN    *** */
/* *** !!! This is pseudo code !!! *** */
/* *** SQL-Code-Example-CH09-03     *** */

BEGIN TRANSACTION;

SELECT     varOldQuantity = PRODUCT.Quantity
FROM       PRODUCT
WHERE      PRODUCT.ProductName = 'Pencil';

SET        varNewQuantity = varOldQuantity - 5;

-- Process transaction - take exception action if varNewQuantity < 0, etc.

UPDATE     PRODUCT
           SET           PRODUCT.Quantity = varNewQuantity
           WHERE         PRODUCT.ProductName = 'Pencil';

-- Continue processing the transaction.

IF /* {Transaction has completed normally}  */
       THEN
          COMMIT TRANSACTION;
       ELSE
          ROLLBACK TRANSACTION;
       END IF;

-- Continue processing other actions not part of this transaction.
```

Implicit and Explicit COMMIT TRANSACTION

- Microsoft SQL Server 2014 and MySQL 8.0 automatically commit changes after a transaction.
  - This is an **implicit COMMIT**.
- Oracle Database requires an **explicit COMMIT** statement:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-UPDATE-CH09-02 *** */
UPDATE CUSTOMER
SET AreaCode = '425'
WHERE ZIPCode = '98050';
COMMIT;
```

ACID Transaction (1 of 3)

- An **ACID** transaction is one that is **A**tomic, **C**onsistent, **I**solated, and **D**urable.
- **Atomic** means either all or none of the database actions occur.
- **Durable** means database committed changes are permanent.

ACID Transaction (2 of 3)

- **Consistency** means either statement level or transaction level consistency.
  - **Statement level consistency**: each statement independently processes rows consistently.
  - **Transaction level consistency**: all rows impacted by any of the SQL statements are protected from changes during the entire transaction.
    - With transaction level consistency, a transaction may not see its own changes.

ACID Transaction (3 of 3)

- **Isolation** means application programmers are able to declare the type of isolation level and to have the DBMS manage locks so as to achieve that level of isolation.
- SQL - 92 defines four **transaction isolation levels**:
  - **Read uncommitted**
  - **Read committed**
  - **Repeatable read**
  - **Serializable**

Figure 9-11: Summary of Data Read Problems

Data Read Problem Type	Definition
Dirty Read	The transaction reads a row that has been changed, but the change has not been committed. If the change is rolled back, the transaction has incorrect data.
Nonrepeatable Read	The transaction rereads data that has been changed and finds updates or deletions due to committed transactions.
Phantom Read	The transaction rereads data and finds new rows inserted by a committed transaction.

Figure 9-12: Summary of Transaction Isolation Levels

Problem Type	Isolation Level Read Uncommitted	Isolation Level Read Committed	Isolation Level Repeatable Read	Isolation Level Serializable
Dirty Read	Possible	Not Possible	Not Possible	Not Possible
Nonrepeatable Read	Possible	Possible	Not Possible	Not Possible
Phantom Read	Possible	Possible	Possible	Not Possible

SQL Cursors

- A **cursor** is a pointer into a set of records.
- It can be defined using SELECT statements.
- Four cursor types:
  - **Forward only**: the application can only move forward through the recordset.
  - Scrollable cursors can be scrolled forward and backward through the recordset.
    - **Static**: processes a snapshot of the relation that was taken when the cursor was opened.
    - **Keyset**: combines some features of static cursors with some features of dynamic cursors.
    - **Dynamic**: a fully featured cursor.
- Choosing appropriate isolation levels and cursor types is critical to database processing.

Figure 9-13: Summary of SQL Cursors

Cursor Type	Description	Comments
Forward only	Application can only move forward through the recordset.	Changes made by other cursors in this transaction or in other transactions will be visible only if they occur on rows ahead of the cursor.
Static	Application sees the data as they were at the time the cursor was opened.	Changes made by this cursor are visible. Changes from other sources are not visible. Backward and forward scrolling allowed.
Keyset	When the cursor is opened, a primary key value is saved for each row in the recordset. When the application accesses a row, the key is used to fetch the current values for the row.	Updates from any source are visible. Inserts from sources outside this cursor are not visible (there is no key for them in the keyset). Inserts from this cursor appear at the bottom of the recordset. Deletions from any source are visible. Changes in row order are not visible. If the isolation level is read-uncommitted, then uncommitted updates and deletions are visible; otherwise, only committed updates and deletions are visible.
Dynamic	Changes of any type and from any source are visible.	All inserts, updates, deletions, and changes in recordset order are visible. If the isolation level is dirty read, then uncommitted changes are visible. Otherwise, only committed changes are visible.

Database Security

- **Database security** ensures that only authorized users can perform authorized activities at authorized times.
- Developing database security:
  - Determine users’ processing rights and responsibilities.
  - Enforce security requirements using security features from both DBMS and application programs.

Figure 9-14: Processing Rights at View Ridge Gallery

	CUSTOMER	TRANSACTION	WORK	ARTIST
Sales personnel	Insert, change, query	Insert, query	Query	Query
Management personnel	Insert, change, query	Insert, change, query	Insert, change, query	Insert, change, query
System administrator	Grant rights, modify structure	Grant rights, modify structure	Grant rights, modify structure	Grant rights, modify structure

DBMS Security

- DBMS products provide security facilities.
- They limit certain **actions** on certain **objects** to certain **users** or **groups** (also called **roles**).
- Almost all DBMS products use some form of username and password security.
- SQL Data Control Language (DCL) permissions:
  - The **SQL GRANT statement** is used to assign permissions to users and groups so that the users or groups can perform various operations on the data in the database.
  - The **SQL REVOKE statement** is used to take existing permissions away from users and groups.

Figure 9-15: A Model of DBMS Security

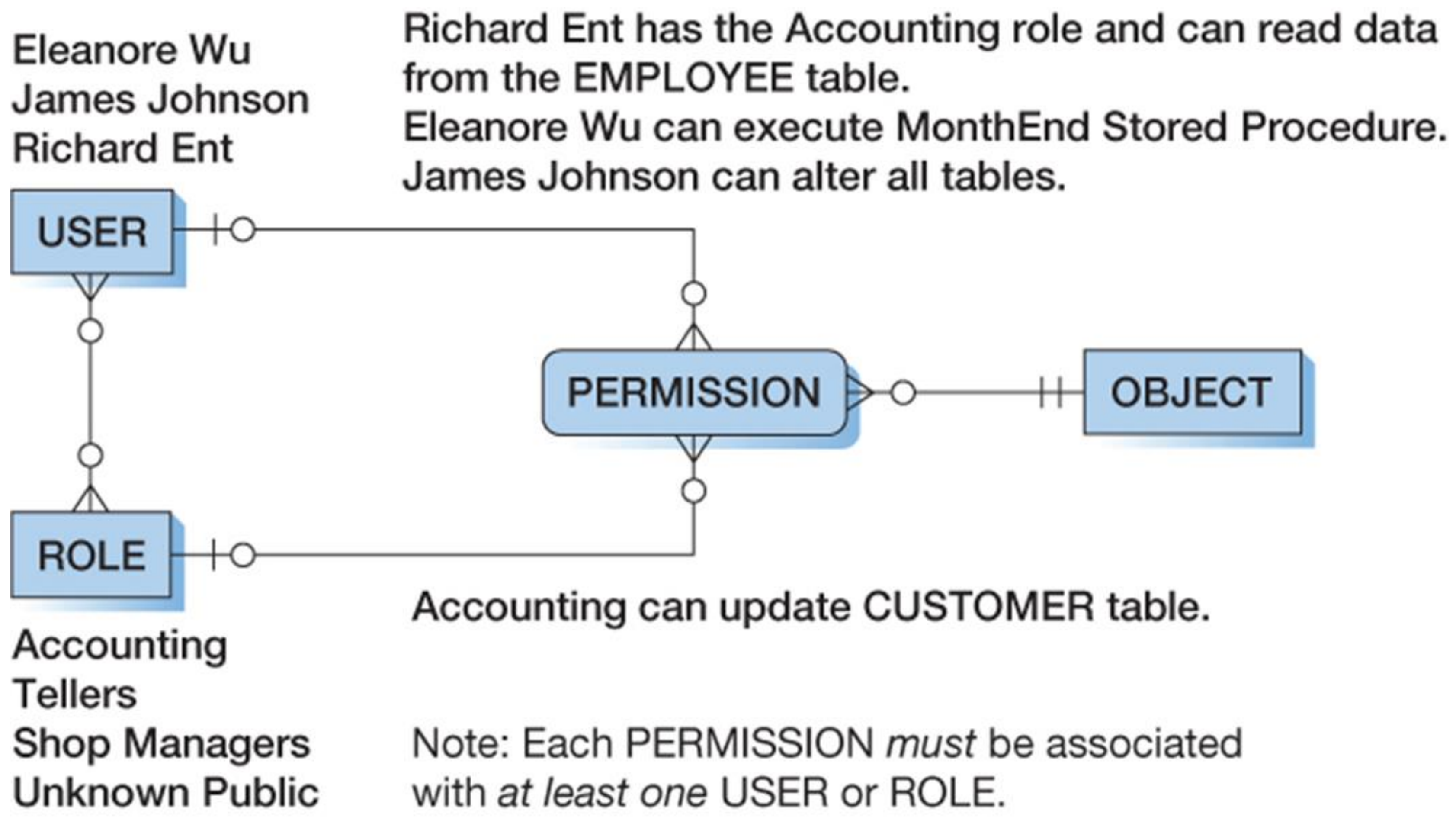

Eleanore Wu
James Johnson
Richard Ent
Richard Ent has the Accounting role and can read data from the EMPLOYEE table.
Eleanore Wu can execute MonthEnd Stored Procedure.
James Johnson can alter all tables.
USER
PERMISSION
OBJECT
ROLE
Accounting can update CUSTOMER table.
Accounting
Tellers
Shop Managers
Unknown Public
Note: Each PERMISSION must be associated with at least one USER or ROLE.

Figure 9-16: Summary of DBMS Security (1 of 2)

Run DBMS behind a firewall, but plan as though the firewall has been breached

Apply the latest operating system and DBMS service packs and fixes

Use the least functionality possible

- Support the fewest network protocols possible
- Delete unnecessary or unused system stored procedures
- Disable default logins and guest users, if possible
- Unless required, never allow users to log on to the DBMS interactively

Protect the computer that runs the DBMS

- No user allowed to work at the computer that runs the DBMS
- DBMS computer physically secured behind locked doors
- Visits to the room containing the DBMS computer should be recorded in a log

Figure 9-16: Summary of DBMS Security (2 of 2)

Manage accounts and passwords

- Use a low-privilege user account for the DBMS service
- Protect database accounts with strong passwords
- Monitor failed login attempts
- Frequently check group and role memberships
- Audit accounts with null passwords
- Assign accounts the lowest privileges possible
- Limit DBA account privileges

Planning

- Develop a security plan for preventing and detecting security problems
- Create procedures for security emergencies and practice them

Application Security

- If DBMS security features are inadequate, additional security code could be written in application program.
  - Application security in Internet applications is often provided on the Web server computer.
- However, you should use the DBMS security features first.
  - The closer the security enforcement is to the data, the less chance there is for infiltration.
  - DBMS security features are faster, cheaper, and probably result in higher quality results than developing your own.

SQL Injection Attack

- An **SQL injection attack** occurs when data from the user is used to modify an SQL statement.
- User input that can modify an SQL statement must be carefully edited to ensure that only valid input has been received and that no additional SQL syntax has been entered.

SQL Injection Attack Example

- Example: users are asked to enter their names into a Web form textbox:
  - User input:

    Benjamin Franklin' OR 'x' = 'x
  - Resulting effective user input:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-Code-Example-CH09-09 *** */
SELECT *
FROM EMPLOYEE
WHERE EMPLOYEE.Name = 'Benjamin Franklin' OR 'x'='x';
```

  - Result is that **TRUE** is true for every row, so every row is returned!

Recovery Via Reprocessing

- **Recovery via reprocessing**: the database goes back to a known point (database save) and reprocesses the workload from there.
- Unfeasible strategy because:
  - The recovered system may never catch up if the computer is heavily scheduled.
  - Asynchronous events, although concurrent transactions, may cause different results.

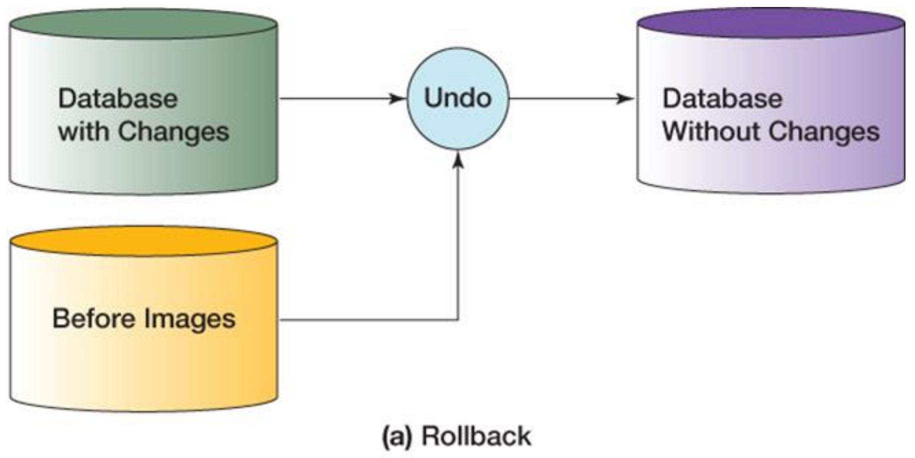
Rollback/Rollforward

- Recovery via rollback/rollforward:
  - Periodically save the database and keep a database change log since the save.
    - Database log contains records of the data changes in chronological order.
- When there is a failure, either rollback or rollforward is applied.
  - **Rollback**: undo the erroneous changes made to the database and reprocess valid transactions.
  - **Rollforward**: restore database using saved data and valid transactions since the last save.

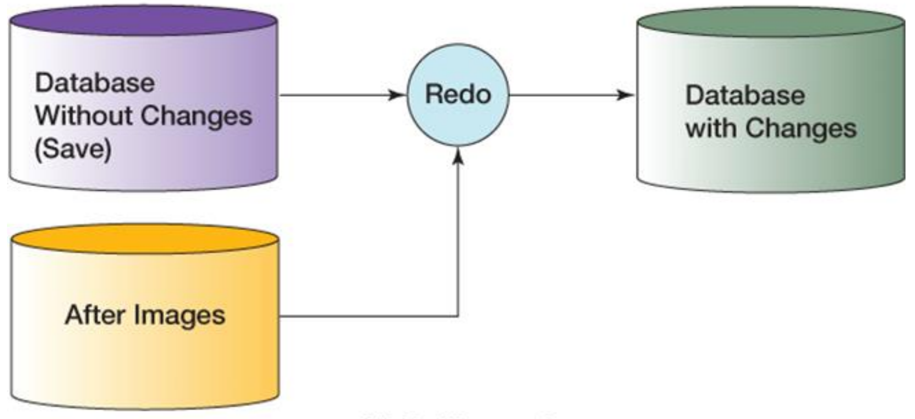
Figure 9-17: Undo and Redo Transactions

- **Before-image**: a copy of every database record (or page) before it was changed (Rollback)

- **After-image**: a copy of every database record (or page) after it was changed (Rollforward)

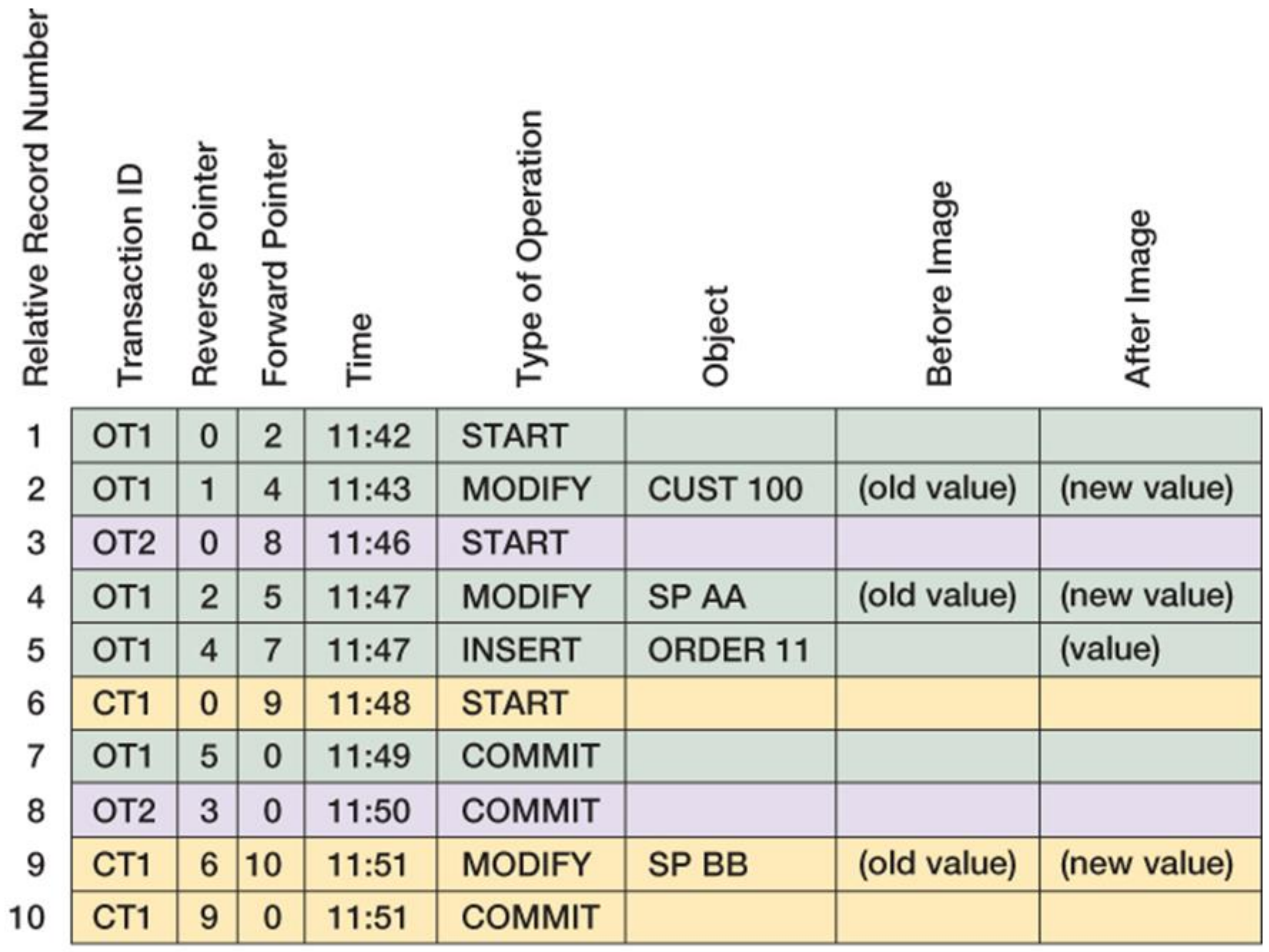


(a) Rollback



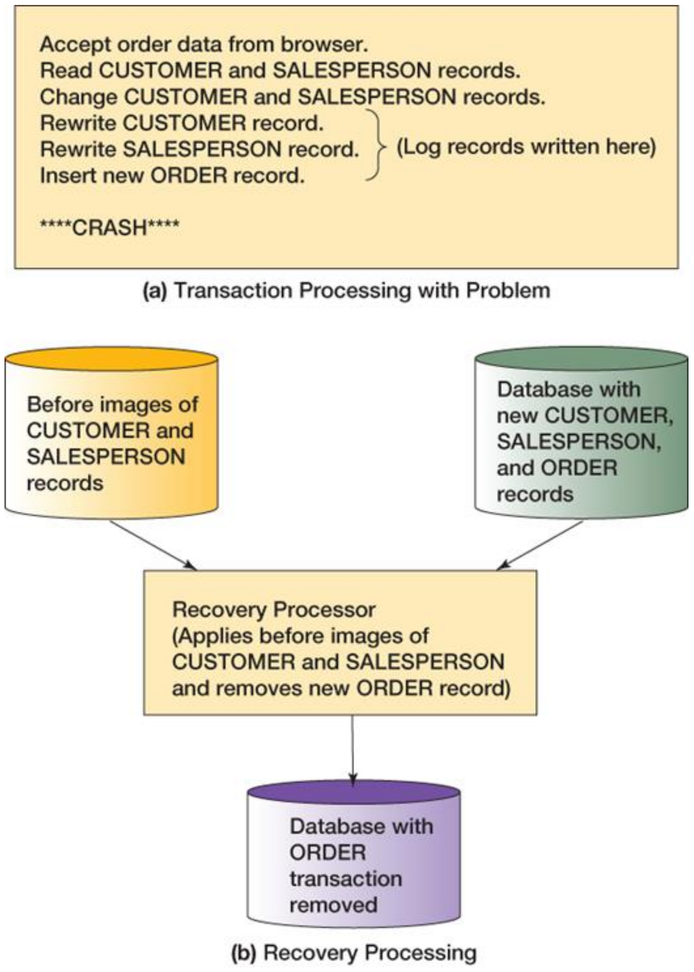
(b) Rollforward

Figure 9-18: Example Transaction Log



Relative Record Number	Transaction ID	Reverse Pointer	Forward Pointer	Time	Type of Operation	Object	Before Image	After Image
1	OT1	0	2	11:42	START			
2	OT1	1	4	11:43	MODIFY	CUST 100	(old value)	(new value)
3	OT2	0	8	11:46	START			
4	OT1	2	5	11:47	MODIFY	SP AA	(old value)	(new value)
5	OT1	4	7	11:47	INSERT	ORDER 11		(value)
6	CT1	0	9	11:48	START			
7	OT1	5	0	11:49	COMMIT			
8	OT2	3	0	11:50	COMMIT			
9	CT1	6	10	11:51	MODIFY	SP BB	(old value)	(new value)
10	CT1	9	0	11:51	COMMIT			

Figure 9-19: Recovery Example


Accept order data from browser.
Read CUSTOMER and SALESPERSON records.
Change CUSTOMER and SALESPERSON records.
Rewrite CUSTOMER record.
Rewrite SALESPERSON record.
Insert new ORDER record.
(Log records written here)
****CRASH****
(a) Transaction Processing with Problem
Before images of CUSTOMER and SALESPERSON records
Database with new CUSTOMER, SALESPERSON, and ORDER records
Recovery Processor (Applies before images of CUSTOMER and SALESPERSON and removes new ORDER record)
Database with ORDER transaction removed
(b) Recovery Processing

Checkpoint

- A **checkpoint** is a point of synchronization between the database and the transaction log.
  - DBMS refuses new requests, finishes processing outstanding requests, and writes its buffers to disk.
  - The DBMS waits until the writing is successfully completed.

  → The log and the database are synchronized.
- Checkpoints speed up database recovery process.
  - Database can be recovered using after-images since the last checkpoint.
  - Checkpoint can be done several times per hour.
- Most DBMS products automatically checkpoint themselves.

Figure 9-20: Summary of the DBA's Responsibilities for Managing the DBMS

- Generate database application performance reports
- Investigate user performance complaints
- Assess need for changes in database structure or application design
- Modify database structure
- Evaluate and implement new DBMS features
- Tune DBMS performance using physical DB design, query optimization, and other techniques

Maintaining the Data Repository

- DBA is responsible for maintaining the data repository.
- **Data repositories** are collections of metadata about users, databases, and its applications.
- The repository may be:
  - Virtual, as it is composed of metadata from many different sources: DBMS, code libraries, Web page generation and editing tools, etc.
  - An integrated product from a CASE tool vendor or from other companies.
  - **Active** – part of the systems development process.
  - **Passive** – documentation only made when someone has time.
- The best repositories are active, and they are part of the system development process.

Copyright