MANAGING MULTIUSER DATABASES

revised by 김 태 연

OBJECTIVES

- To understand the need for concurrency control
- To learn about typical problems that can occur when multiple users process a database concurrently
- To understand the use of locking and the problem of deadlock
- To learn the difference between optimistic and pessimistic locking
- To know the meaning of an ACID transaction
- To learn the four 1992 ANSI standard isolation levels

THREE SERVICES THAT SHOULD BE PROVIDED BY DBMS

- Concurrency control services
  - A DBMS must furnish a mechanism to ensure that the database is updated correctly when multiple users are updating the database concurrently.
- Transaction support
  - A DBMS must furnish a mechanism that will ensure either that all the updates corresponding to a given transaction are made or that none of them is made.
- Recovery services
  - A DBMS must furnish a mechanism for recovering the database in the event that the database is damaged in any way.

Thomas Connolly, Carolyn Begg, Database Systems: A Practical Approach to Design, Implementation, and Management 6th, Pearson, 2014

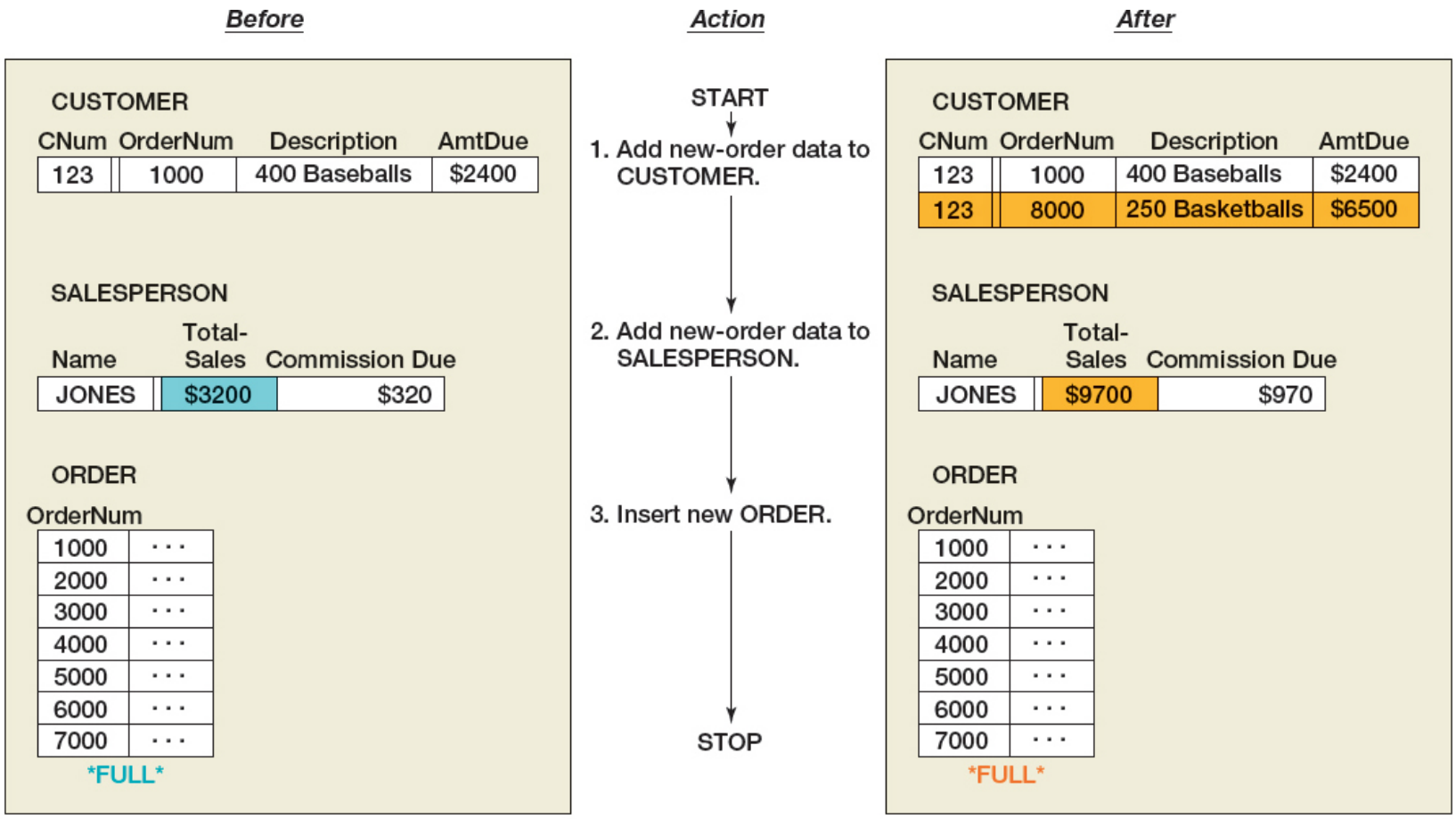
CONCURRENCY CONTROL

- **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

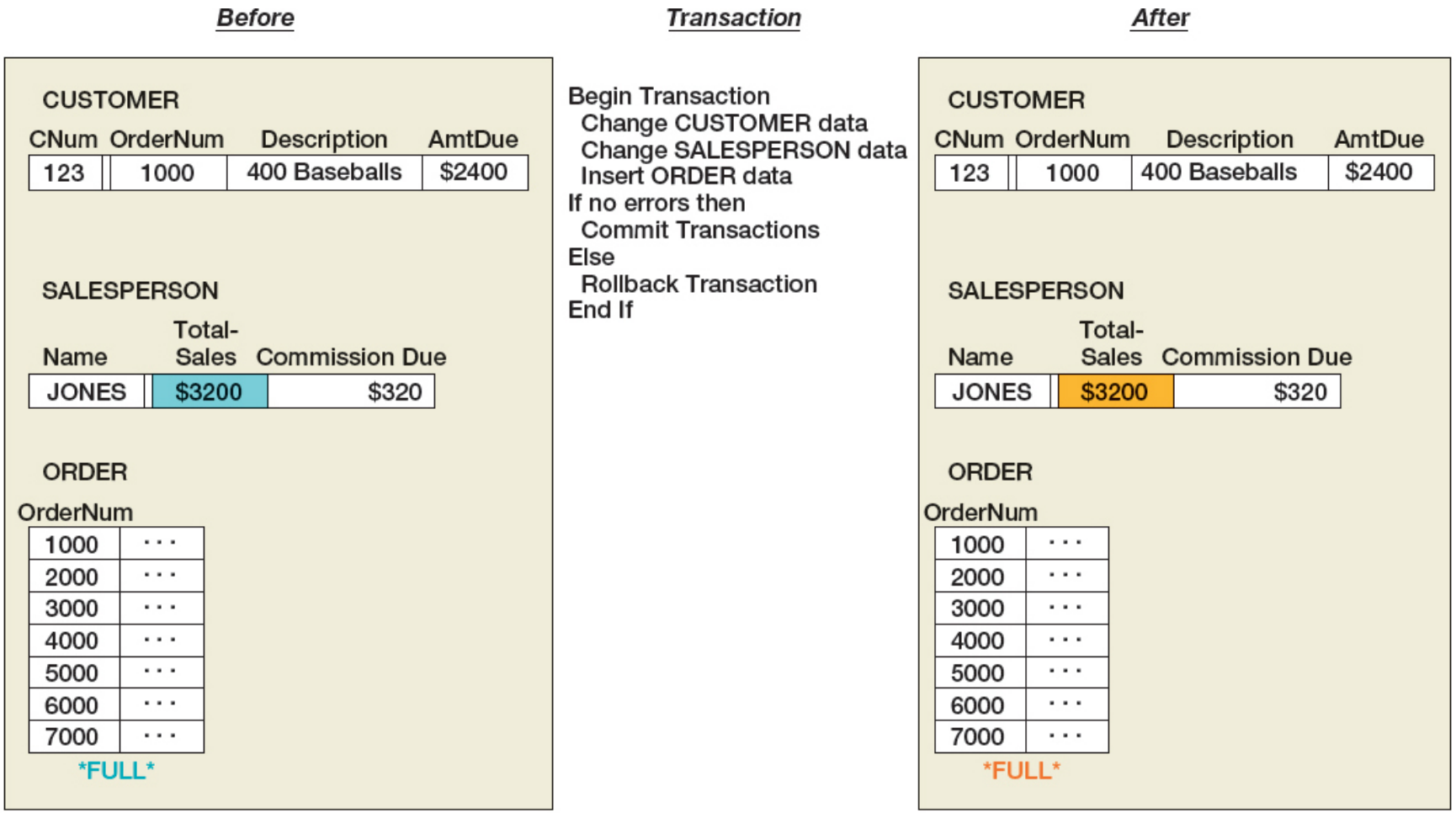
- A **transaction**, or logical unit of work (LUW), is a series of actions taken against the database that occurs as an atomic unit:
  - Either all actions in a transaction occur or none of them do.
- Consider the following sequence of database actions that could occur when recording a new order:
  - Change a customer's row, increasing AmountDue.
  - Change a salesperson's row, increasing CommissionDue.
  - Insert a new order row into the database.

ERRORS INTRODUCED WITHOUT ATOMIC TRANSACTION



(a) Errors Introduced Without Transaction

ERRORS PREVENTED WITH ATOMIC TRANSACTION

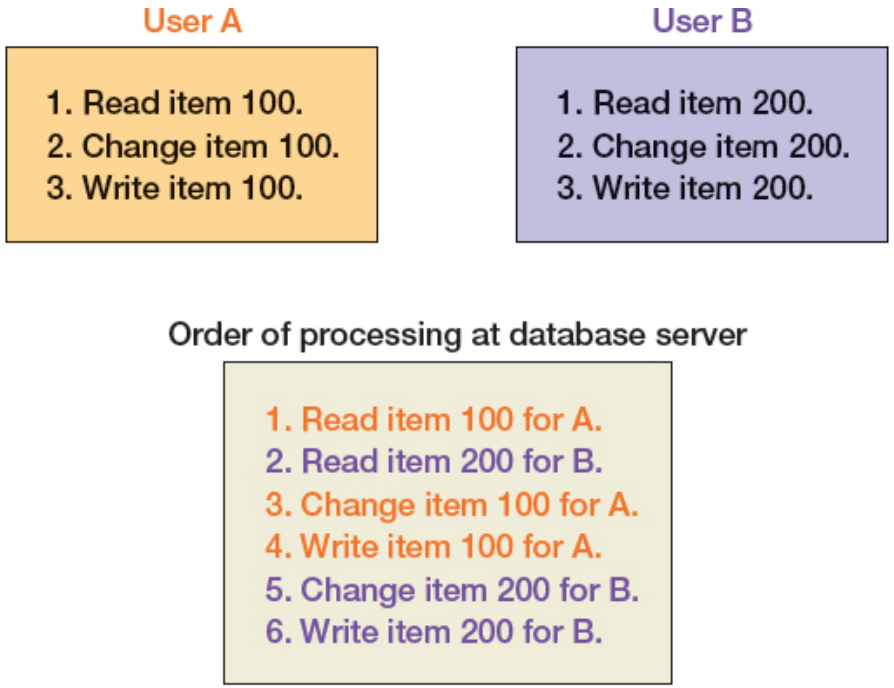


(b) Atomic Transaction Prevents Errors

CONCURRENT TRANSACTION

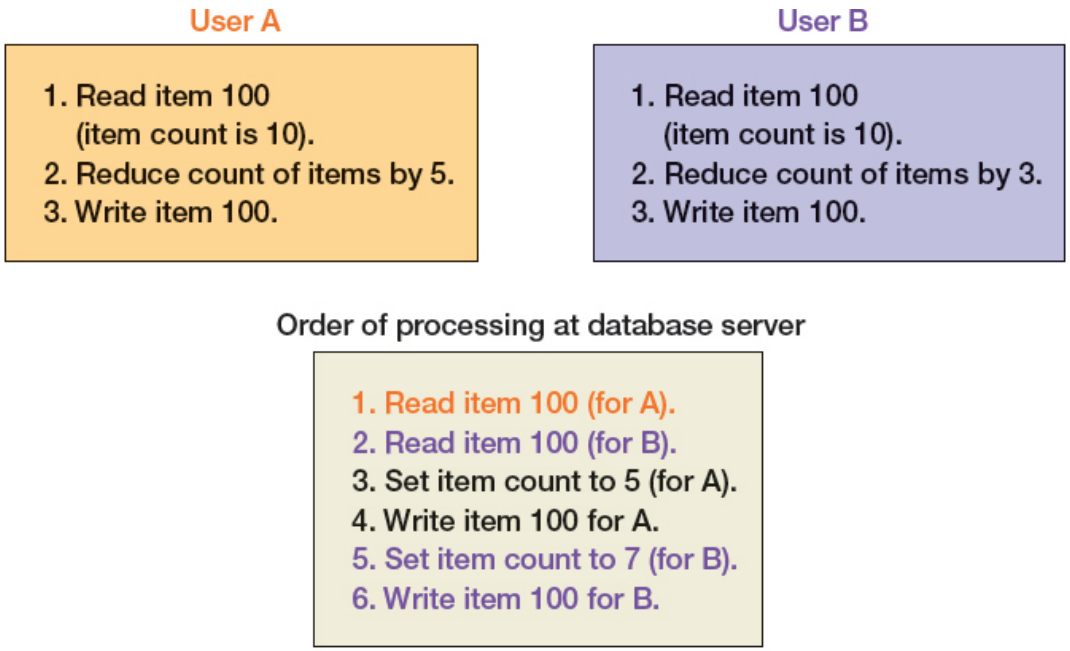
- **Concurrent transactions** refer to two or more transactions that appear to users as they are being processed against a database at the same time.
- In reality, CPU can execute only one instruction at a time.
  - Transactions are interleaved:
    - The operating system quickly switches CPU services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

LOST UPDATE PROBLEM



(a) Concurrent Transaction Processing

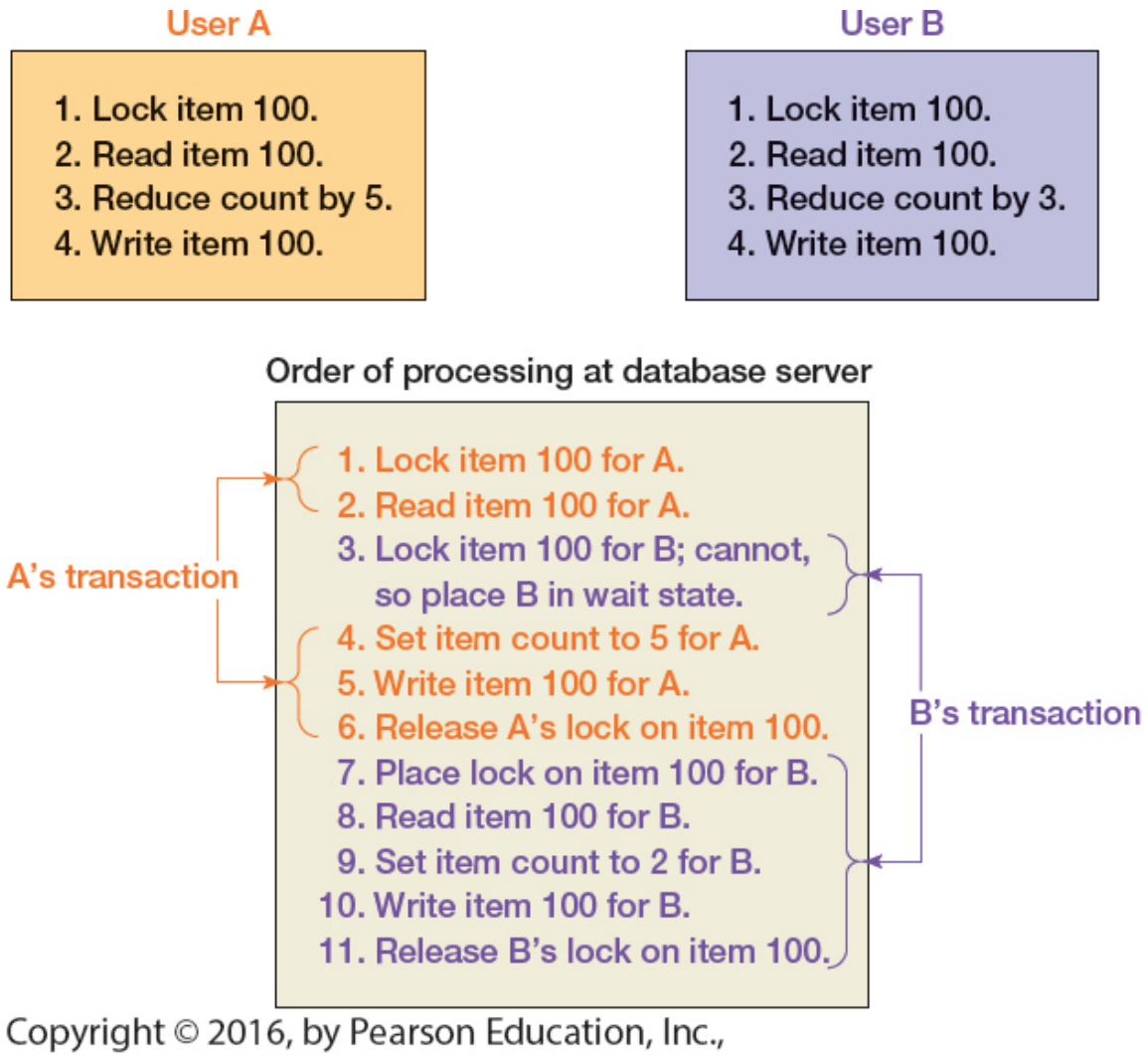
(b) Lost Update Problem



RESOURCE LOCKING

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

CONCURRENT PROCESSING WITH EXPLICIT LOCKS



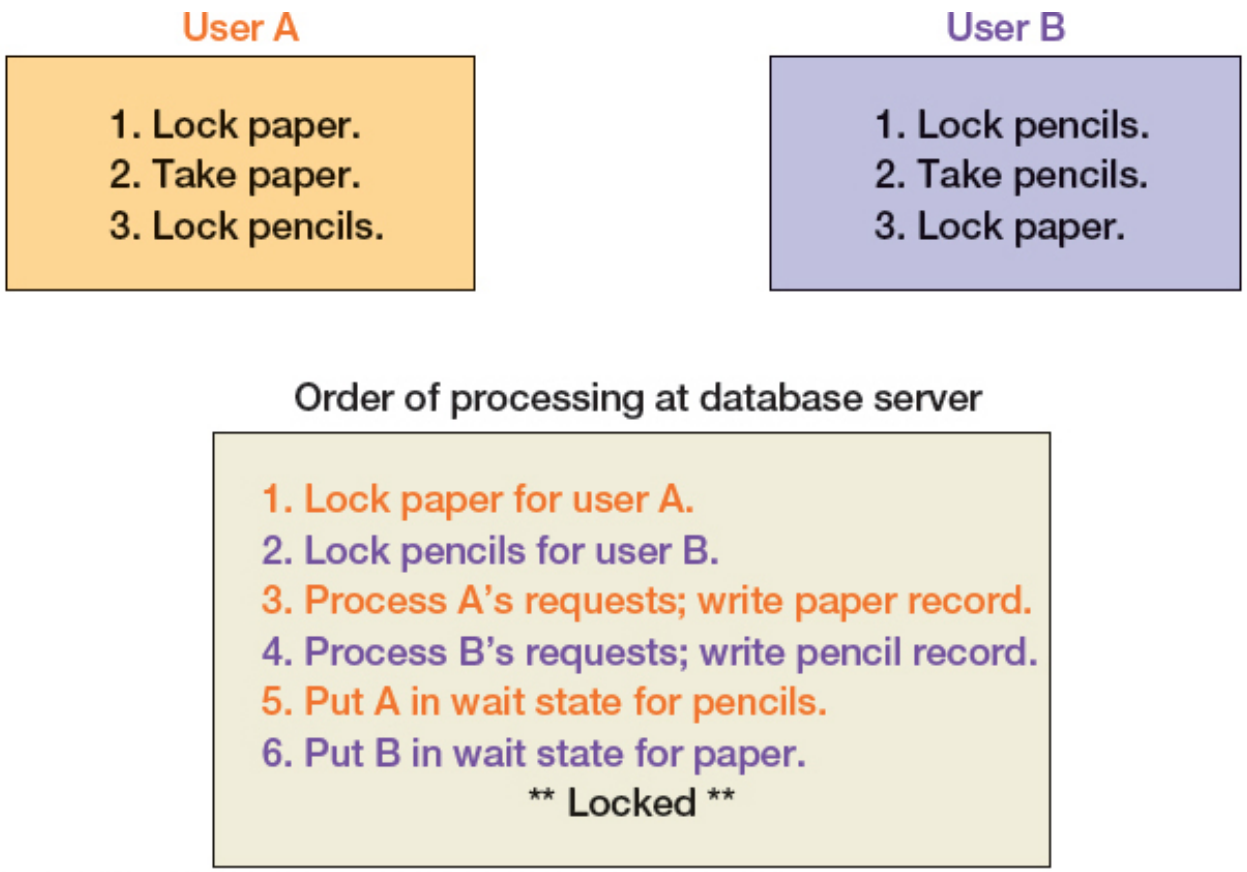
SERIALIZABLE TRANSACTIONS

- **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-phased locking** is one of the techniques used to achieve serializability.
  - 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.

DEADLOCK



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 before transaction is processed, and then the locks are released.
- Optimistic locking is preferred for the Internet and for many intranet applications.

OPTIMISTIC LOCKING

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH09-01 *** */

SELECT   PRODUCT.Name, PRODUCT.Quantity
FROM     PRODUCT
WHERE    PRODUCT.Name = 'Pencil';

Set NewQuantity = PRODUCT.Quantity - 5;

{process transaction - take exception action if NewQuantity < 0, etc.

Assuming all is OK: }

LOCK     PRODUCT;

UPDATE   PRODUCT
SET      PRODUCT.Quantity = NewQuantity
WHERE    PRODUCT.Name = 'Pencil'
     AND PRODUCT.Quantity = OldQuantity;

UNLOCK   PRODUCT;

{check to see if update was successful;
if not, repeat transaction}
```

PESSIMISTIC LOCKING

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH09-02 *** */

LOCK     PRODUCT;

SELECT   PRODUCT.Name, PRODUCT.Quantity
FROM     PRODUCT
WHERE    PRODUCT.Name = 'Pencil';

Set NewQuantity = PRODUCT.Quantity - 5;

{process transaction - take exception action if NewQuantity < 0, etc.

Assuming all is OK: }

UPDATE   PRODUCT
SET      PRODUCT.Quantity = NewQuantity
WHERE    PRODUCT.Name = 'Pencil';

UNLOCK   PRODUCT;

{no need to check if update was successful}
```

AUTOCOMMIT

- By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.
- Implicit Commit
  - Data definition language (DDL) statements that define or modify database objects
  - Statements that implicitly use or modify tables in the mysql database
  - Transaction-control and locking statements.
  - http://dev.mysql.com/doc/refman/5.6/en/implicit-commit.html

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):
    - BEGIN TRANSACTION
    - COMMIT TRANSACTION
    - ROLLBACK TRANSACTION
- Advantage:
  - If the locking behavior needs to be changed, only the lock declaration need be changed, not the application program.

SQL TRANSACTION CONTROL LANGUAGE (TCL)

- Application programs mark transaction boundaries using SQL Transaction Control Language (TCL), and then declare the type of locking behavior they want the DBMS to use.
- SQL BEGIN TRANSACTION statement
  - explicitly marks the start of a new transaction
- SQL COMMIT TRANSACTION statement
  - makes any database changes made by the transaction permanent and marks the end of the transaction.
- SQL ROLLBACK TRANSACTION statement
  - is used to undo all transaction changes, if there is a need to undo the changes made during the transaction due to an error in the process, and return the database to the state it was in before the transaction was attempted.
  - also marks the end of the transaction, but with a very different outcome.

MARKING TRANSACTION BOUNDARIES

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH09-03 *** */

BEGIN TRANSACTION;

SELECT     PRODUCT.Name, PRODUCT.Quantity
FROM       PRODUCT
WHERE      PRODUCT.Name = 'Pencil';

Set NewQuantity = PRODUCT.Quantity - 5;

{process transaction - take exception action if NewQuantity < 0, etc.}

UPDATE     PRODUCT
SET        PRODUCT.Quantity = NewQuantity
WHERE      PRODUCT.Name = 'Pencil';

{continue processing transaction} . . .


IF {transaction has completed normally} THEN

     COMMIT TRANSACTION;

ELSE

     ROLLBACK TRANSACTION;

END IF;

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

ACID TRANSACTIONS

- Acronym ACID transaction is one that is Atomic, Consistent, Isolated, and Durable.
- **Atomic** means either all or none of the database actions occur.
- **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 either of the SQL statements are protected from changes during the entire transaction
    - With transaction level consistency, a transaction may not see its own changes.
- **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
- **Durable** means database committed changes are permanent.

TRANSACTION ISOLATION LEVEL AND DATA READ PROBLEMS

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

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.

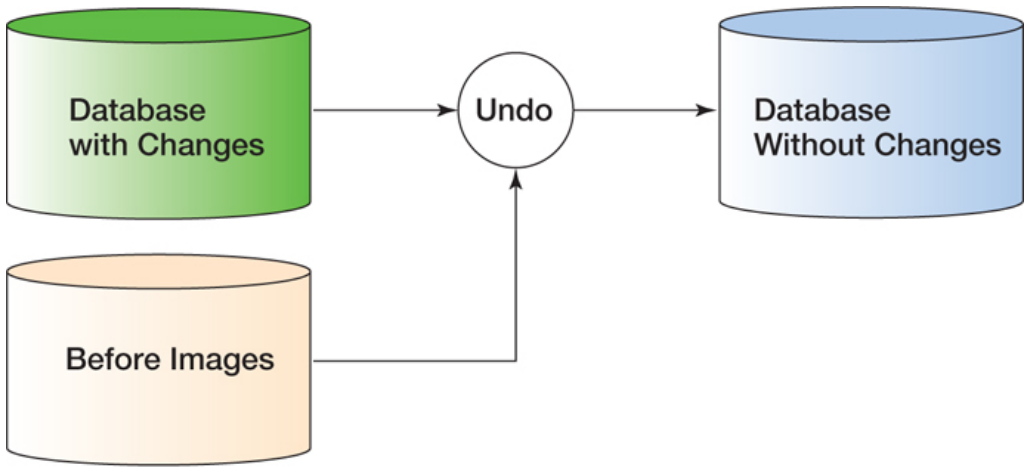
INNODB LOCKING AND TRANSACTION MODEL

- In the InnoDB transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking.
- InnoDB performs locking at the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle.
- The lock information in InnoDB is stored space-efficiently so that **lock escalation is not needed**.
- Typically, several users are permitted to lock every row in InnoDB tables, or any random subset of the rows, without causing InnoDB memory exhaustion.
- http://dev.mysql.com/doc/refman/5.6/en/innodb-locking-transaction-model.html

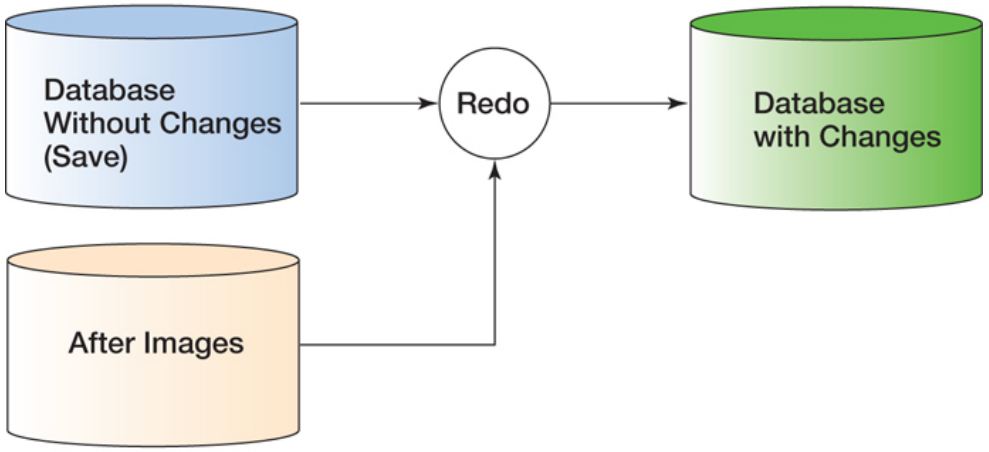
DATABASE RECOVERY

- In the event of system failure, that database must be restored to a usable state as soon as possible.
- Two recovery techniques:
  - **Recovery via reprocessing**: the database goes back to a known point (database save) and reprocesses the workload from there.
    - The recovered system may never catch up if the computer is heavily scheduled.
    - Asynchronous events, although concurrent transactions, may cause different results.
  - **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.
    - 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

ROLLBACK AND ROLLFORWARD



(a) Rollback



(b) Rollforward

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.

TRANSACTION LOG

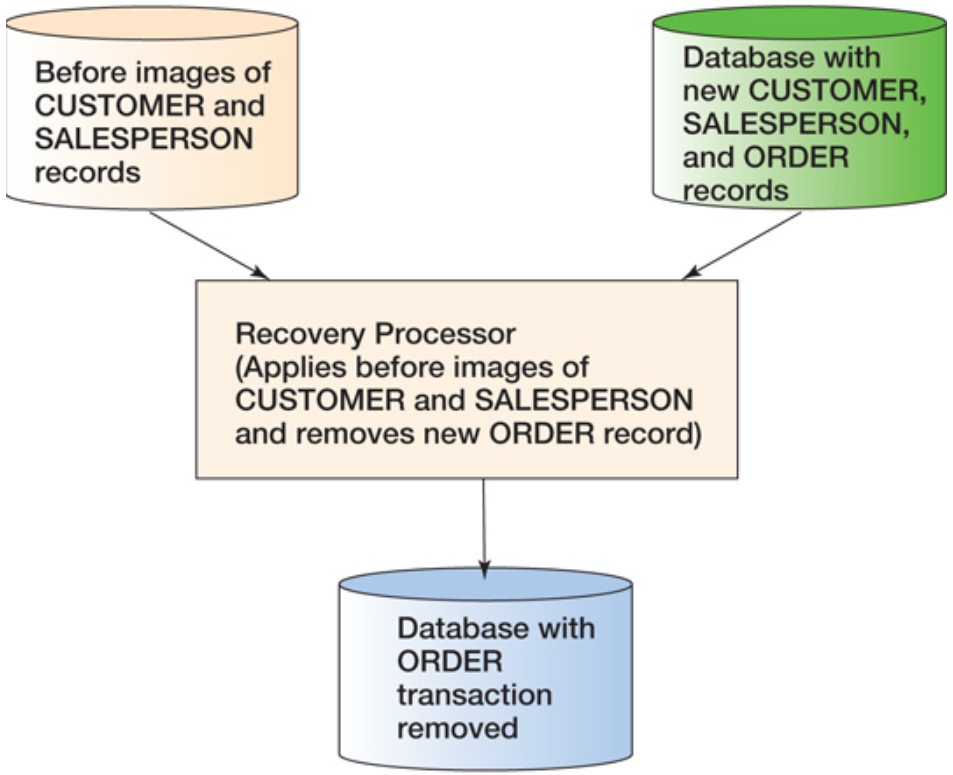
Relative Record Number	Transaction ID	Reverse Pointer	Forward Pointer	Time	Type of Operation	Object	Before Image	After Image
1	OT1	0	2	11:42	START			
2	OT1	1	4	11:43	MODIFY	CUST 100	(old value)	(new value)
3	OT2	0	8	11:46	START			
4	OT1	2	5	11:47	MODIFY	SP AA	(old value)	(new value)
5	OT1	4	7	11:47	INSERT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

DATABASE RECOVERY

Accept order data from browser.
Read CUSTOMER and SALESPERSON records.
Change CUSTOMER and SALESPERSON records.
Rewrite CUSTOMER record.
Rewrite SALESPERSON record. } (Log records written here)
Insert new ORDER record.

****CRASH****

(a) Processing with Problem



(b) Recovery Processing