

MANAGING MULTIUSER DATABASES

revised by 김태연

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.

CONCURRENCY CONTROL

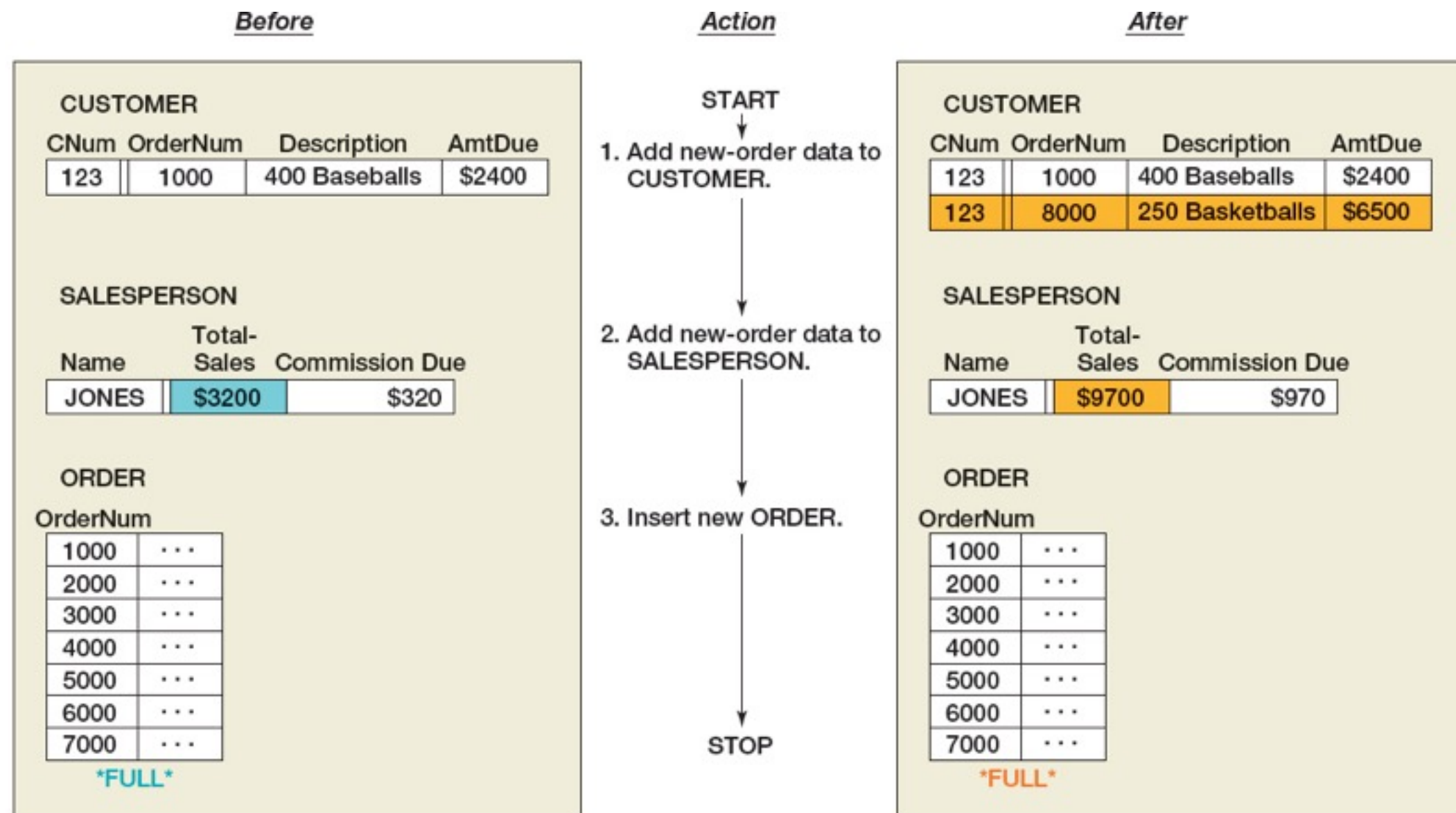
- **Concurrency control** ensures that one user's work does not inappropriately influence another user's work.
- No single concurrency control technique is ideal for all circumstances.
- Trade-offs need to be made between level of protection and throughput.

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

Copyright © 2016, by Pearson Education, Inc.,

ERRORS PREVENTED WITH ATOMIC TRANSACTION

.....

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	...

FULL

Transaction

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

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	...

FULL

(b) Atomic Transaction Prevents Errors

Copyright © 2016, by Pearson Education, Inc.,

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

User A

1. Read item 100.
2. Change item 100.
3. Write item 100.

User B

1. Read item 200.
2. Change item 200.
3. Write item 200.

Order of processing at database server

1. Read item 100 for A.
2. Read item 200 for B.
3. Change item 100 for A.
4. Write item 100 for A.
5. Change item 200 for B.
6. Write item 200 for B.

Copyright © 2016, by Pearson Education, Inc.,

(a) Concurrent Transaction Processing

(b) 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.

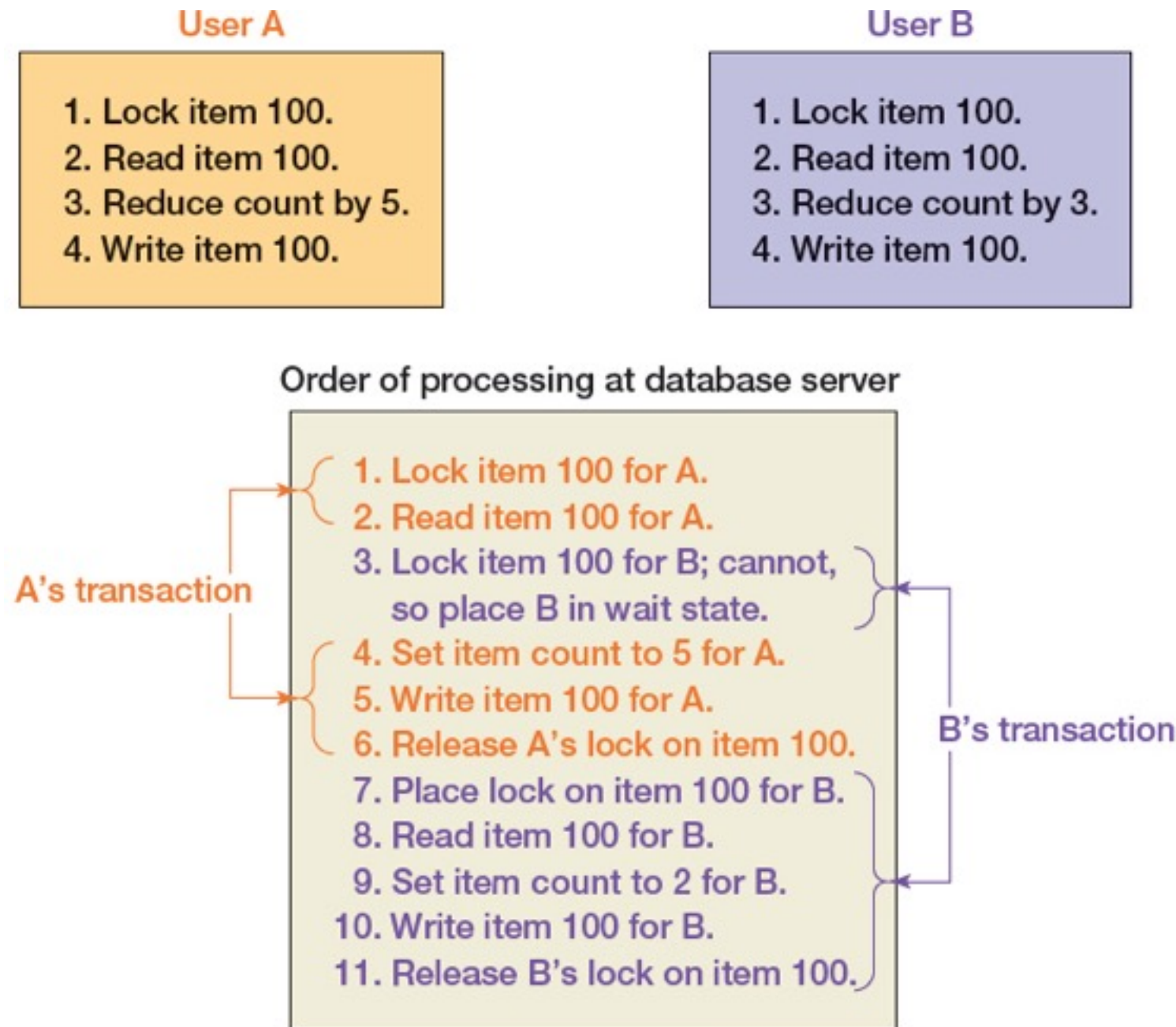
Copyright © 2016, by Pearson Education, Inc.,

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

.....



Copyright © 2016, by Pearson Education, Inc.,

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

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.

**** Locked ****

Copyright © 2016, by Pearson Education, Inc.,

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}
Copyright © 2016, by Pearson Education, Inc.,
```


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}
```

```
Copyright © 2016, by Pearson Education, Inc.,
```

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 . . .
Copyright © 2016, by Pearson Education, Inc.,
```

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

Copyright © 2016, by Pearson Education, Inc.,

Data Read Problem Type	Definition
Dirty Read	The transaction reads a row that has been changed, but the change has <i>not</i> 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.

Copyright © 2016, by Pearson Education, Inc.,

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>