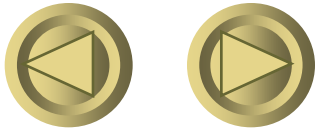
Chapter 6 The database Language SQL –as a tutorial

- **About SQL**

  SQL is a standard database language, adopted by many commercial systems.

  ANSI SQL, SQL-92 or SQL2, SQL99 or SQL3 extends SQL2 with object-relational features. SQL2003 is the collection of extensions to SQL3.

- **How to query the database**
- **How to make modifications on database**
- **Transactions in SQL**



Transactions

What is transactions?

Why do we need transactions?

How to set transaction with different isolation level?

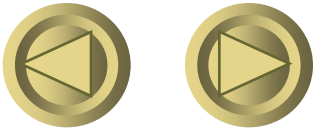
Why Transactions?

- **Concurrent database access**

  Execute *sequence of SQL statements* so they appear to be running in isolation

- **Resilience to system failures**

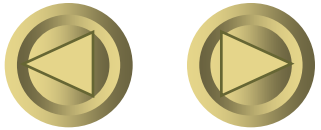
  Guarantee all-or-nothing execution, regardless of failures



Concurrent Control

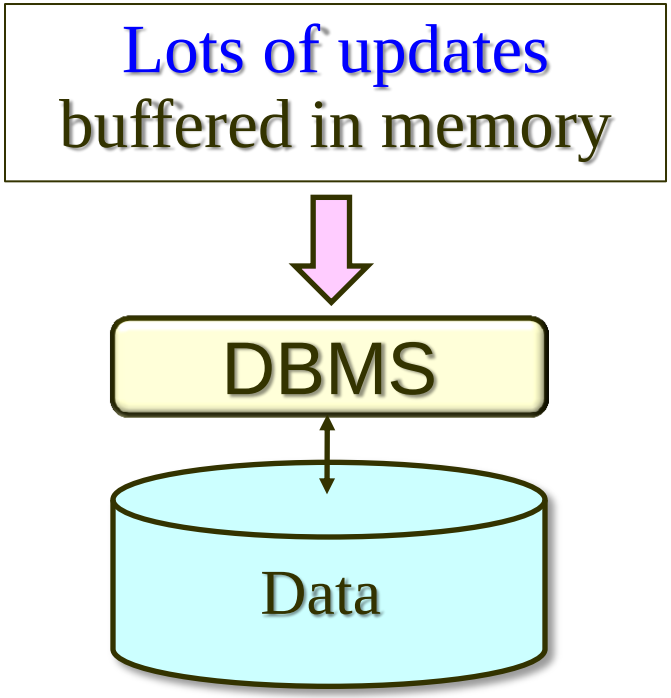
- **Database systems are normally being accessed by many users or processes at the same time.**
  - **Both queries and modifications.**

Serializability
Operations may be interleaved, but execution must be equivalent to *some* sequential (serial) order of all transactions



Resilience to system failures

- **Failures may happen at any time.**
- **All or nothing done, never half done.**



Transfer money from one account into another account.

Update accounts set balance = balance - 1000 where accounts.number=123;

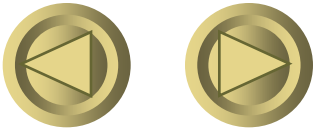
Update accounts set balance =balance +1000 where account.number= 456;

Solution for both concurrency and failures

Transactions

A transaction is a sequence of one or more SQL operations treated as a unit

- Transactions appear to run in isolation
- If the system fails, each transaction's changes are reflected either entirely or not at all

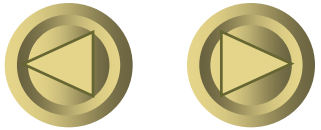


Transactions: SQL standard

- **Normally with some strong properties regarding concurrency.**
- **Formed in SQL from single statements or explicit programmer control, i.e.**

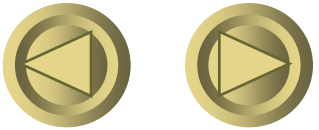
Transaction begins automatically on first SQL statement

- **On "`commit`" transaction ends and new one begins.**
- **Current transaction ends on session termination.**
- **"`Autocommit`" turns each statement into transaction.**



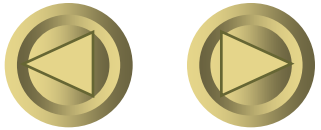
ACID Transactions

- ***ACID transactions*** **are:**
  - ***Atomic*** **: Whole transaction or none is done.**
  - ***Consistent*** **: Database constraints preserved.**
  - ***Isolated*** **: It appears to the user as if only one process executes at a time.**
  - ***Durable*** **: Effects of a process survive a crash.**



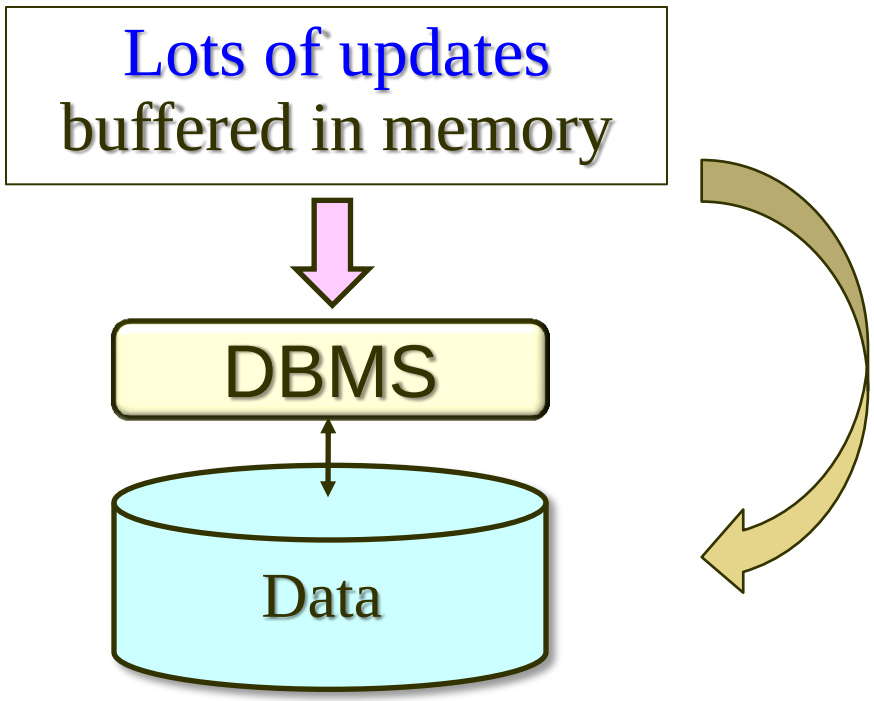
Consistency and isolation

- **Application defines consistency.**
- **Application requires isolation to achieve consistent results, there are four *isolation levels.***
- **Locking typically used to achieve isolation.**



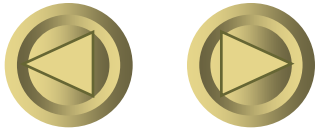
COMMIT

- **The SQL statement COMMIT causes a transaction to complete.**
  - **It's database modifications are now permanent in the database.**



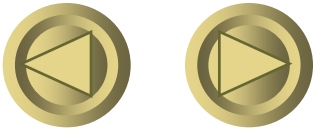
ROLLBACK

- **The SQL statement ROLLBACK also causes the transaction to end, but by *aborting*.**
  - **No effects on the database.**
- **Failures like division by 0 or a constraint violation can also cause rollback, even if the programmer does not request it.**
- **Application versus system-generated rollbacks.**



Example: Interacting Processes

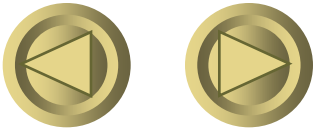
- **Assume the usual Sells(bar,beer,price) relation, and suppose that Joe's Bar sells only Bud for $2.50 and Miller for $3.00.**
- **Sally is querying Sells for the highest and lowest price Joe charges.**
- **Joe decides to stop selling Bud and Miller, but to sell only Heineken at $3.50.**



Sally's Program

- **Sally executes the following two SQL statements called (min) and (max) to help us remember what they do.**

```
(max)SELECT MAX(price) FROM Sells
        WHERE bar = 'Joe''s Bar';
(min)SELECT MIN(price) FROM Sells
        WHERE bar = 'Joe''s Bar';
```



Joe's Program

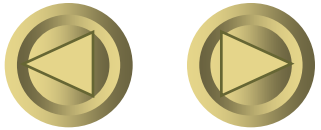
- **At about the same time, Joe executes the following steps: (del) and (ins).**

(del) **DELETE FROM Sells**

WHERE bar = 'Joe''s Bar';

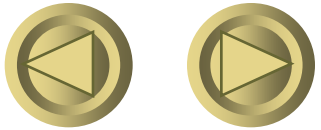
(ins) **INSERT INTO Sells**

VALUES('Joe''s Bar', 'Heineken', 3.50);



Interleaving of Statements

- **Although (max) must come before (min), and (del) must come before (ins), there are no other constraints on the order of these statements, unless we group Sally's and/or Joe's statements into transactions.**



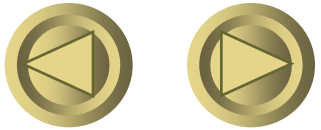
Example: Strange Interleaving

- **Suppose the steps execute in the order (max)(del)(ins)(min).**

Joe's Prices:

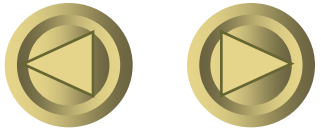
	{2.50,3.00}	{2.50,3.00}		{3.50}
Statement:	(max)	(del)	(ins)	(min)
Result:	3.00			3.50

- **Sally sees MAX < MIN!**



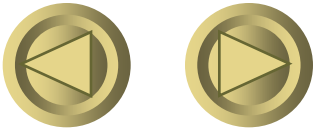
Fixing the Problem by Using Transactions

- **If we group Sally's statements (max)(min) into one transaction, then she cannot see this inconsistency.**
- **She sees Joe's prices at some fixed time.**
  - **Either before or after he changes prices, or in the middle, but the MAX and MIN are computed from the same prices.**



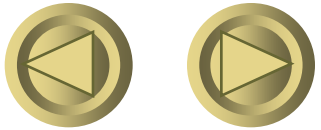
Another Problem: Rollback

- **Suppose Joe executes (del)(ins), not as a transaction, but after executing these statements, thinks better of it and issues a ROLLBACK statement.**
- **If Sally executes her statements after (ins) but before the rollback, she sees a value, 3.50, that never existed in the database.**



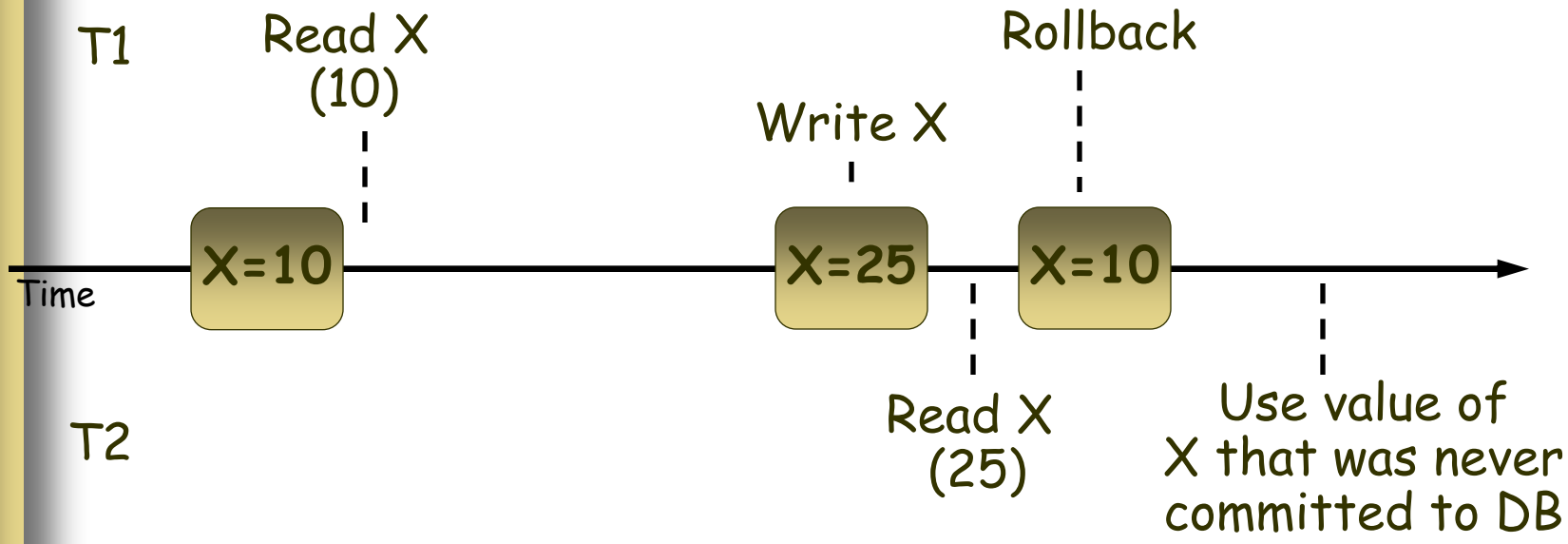
Solution

- **If Joe executes (del)(ins) as a transaction, its effect cannot be seen by others until the transaction executes COMMIT.**
  - **If the transaction executes ROLLBACK instead, then its effects can *never* be seen.**



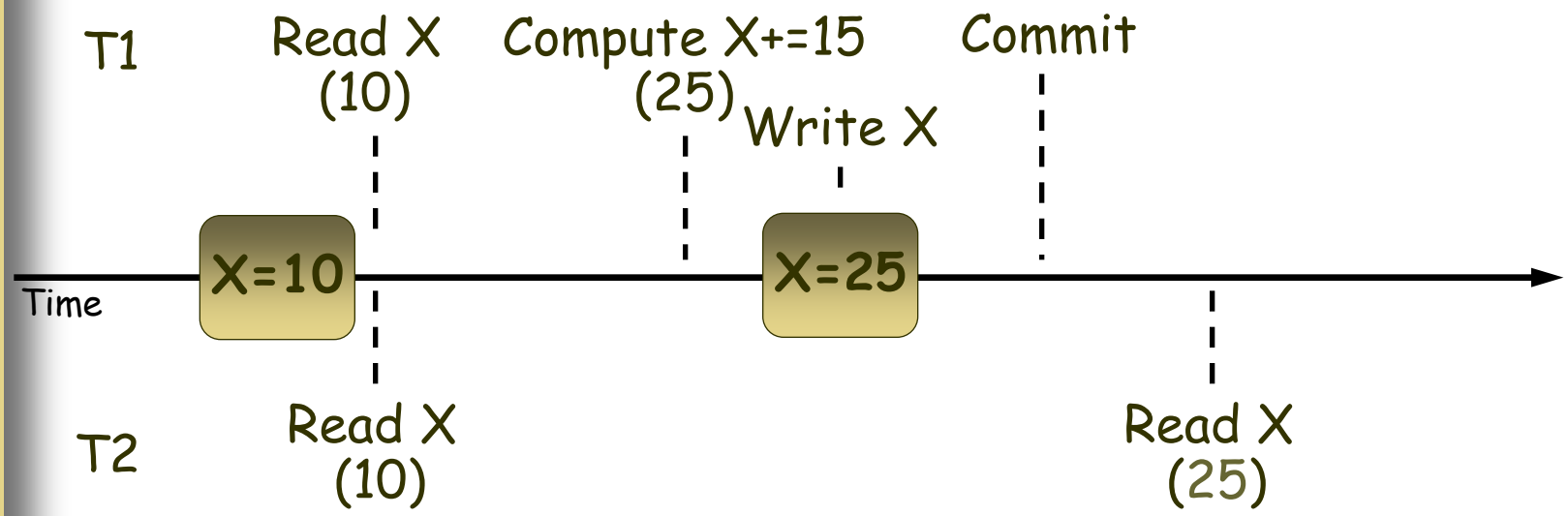
Summarize of problems caused by multiple users accessing (1)

- **Dirty read**



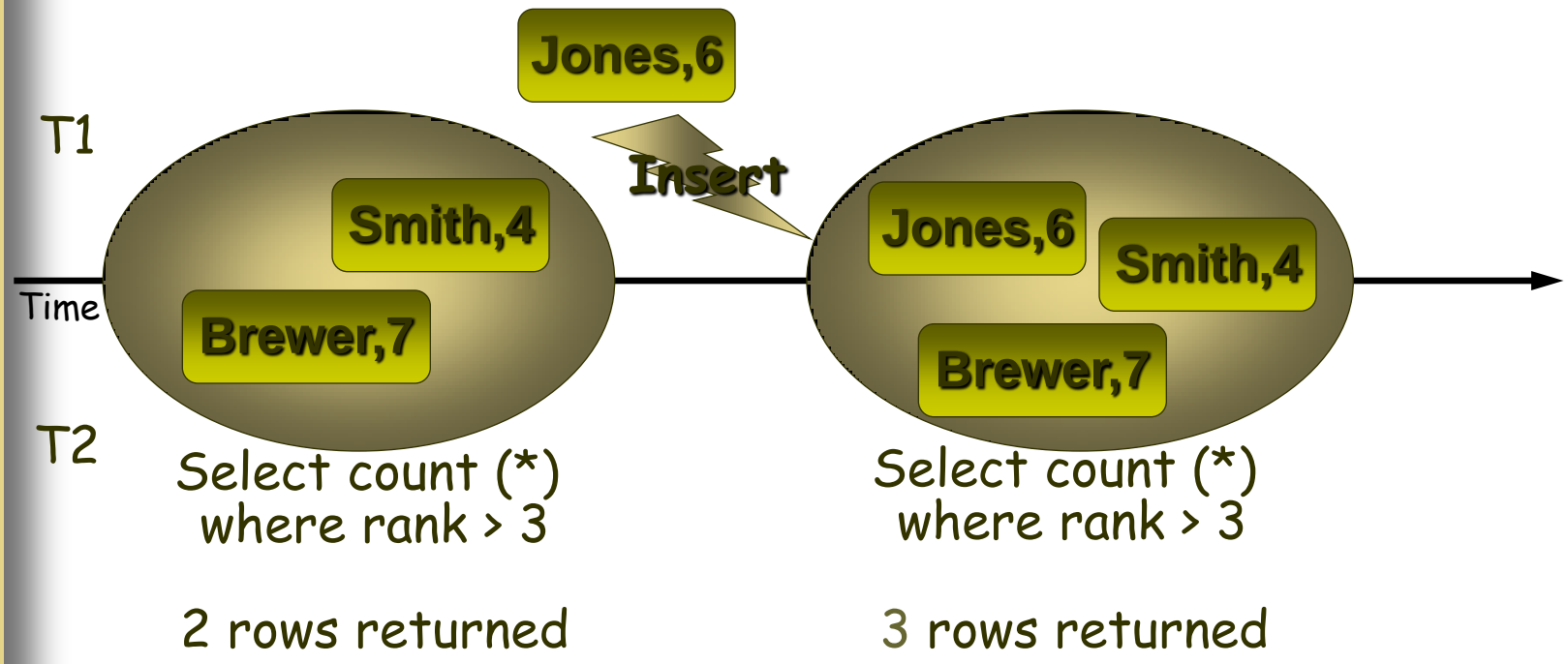
Summarize of problems caused by multiple users accessing (2)

- **Non-Repeatable Read**



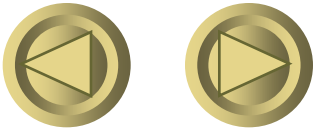
Summarize of problems caused by multiple users accessing (3)

- **The "Phantom" Problem**



Isolation Levels

- **SQL defines four *isolation levels* = choices about what interactions are allowed by transactions that execute at about the same time.**
- **Only one level ("serializable") = ACID transactions.**
- **Each DBMS implements transactions in its own way.**



Choosing the Isolation Level

- **Within a transaction, we can say:**

SET TRANSACTION ISOLATION LEVEL *X*

where *X* =

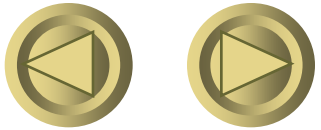
1. **SERIALIZABLE** ⇒ Overhead ⇒ Reduction in concurrency
2. **REPEATABLE READ**
3. **READ COMMITTED**
4. **READ UNCOMMITTED**

↓ Overhead ↑ Concurrency

↓ **Consistency Guarantees**

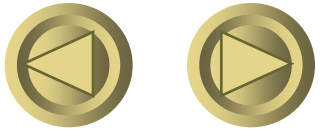
Serializable Transactions

- **If Sally = (max)(min) and Joe = (del)(ins) are each transactions, and Sally runs with isolation level SERIALIZABLE, then she will see the database either before or after Joe runs, but not in the middle.**



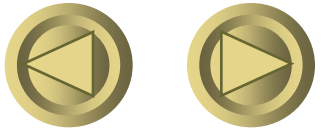
Isolation Level Is Personal Choice

- **Your choice, e.g., run serializable, affects only how *you* see the database, not how others see it.**
- **Example: If Joe Runs serializable, but Sally doesn't, then Sally might see no prices for Joe's Bar.**
  - **i.e., it looks to Sally as if she ran in the middle of Joe's transaction.**



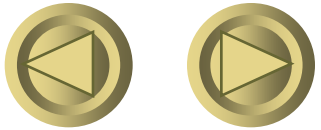
Read-Commited Transactions

- **If Sally runs with isolation level READ COMMITTED, then she can see only committed data, but not necessarily the same data each time.**
- **Example: Under READ COMMITTED, the interleaving (max)(del)(ins)(min) is allowed, as long as Joe commits.**
  - **Sally sees MAX < MIN.**



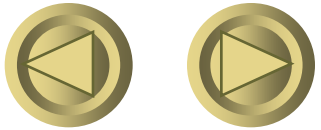
Repeatable-Read Transactions

- **Requirement is like read-committed, plus: if data is read again, then everything seen the first time will be seen the second time.**
  - **But the second and subsequent reads may see *more* tuples as well.**



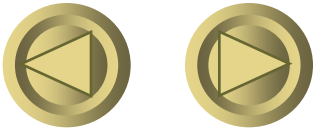
Example: Repeatable Read

- **Suppose Sally runs under REPEATABLE READ, and the order of execution is (max)(del)(ins)(min).**
  - **(max) sees prices 2.50 and 3.00.**
  - **(min) can see 3.50, but must also see 2.50 and 3.00, because they were seen on the earlier read by (max).**



Read Uncommitted

- **A transaction running under READ UNCOMMITTED can see data in the database, even if it was written by a transaction that has not committed (and may never).**
- **Example: If Sally runs under READ UNCOMMITTED, she could see a price 3.50 even if Joe later aborts.**



From weakest to strongest and the read behaviors they permit:

isolation level	dirty reads	nonrepeatable reads	phantoms
READ UNCOMMITTED	Y	Y	Y
READ COMMITTED	N	Y	Y
REPEATABLE READ	N	N	Y
SERIALIZABLE	N	N	N

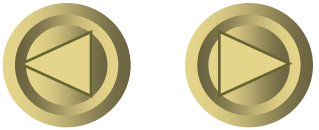
- True isolation is expensive in terms of concurrency
  - Many systems allow application to choose the phenomena they will live with
  - Trade off between correctness and concurrency

Homework

- **exercise 6.6.4**


Summary

- **SQL: The language is the principal query language for relational database systems. (SQL2, SQL3)**
- **Select-From-Where Queries**
- **Subqueries: The operators EXISTS, IN,ALL and ANY may be used to express boolean-valued conditions about the relations that are the result of a subquery**
- **Set Operations on Relations: UNION, INTERSECT, EXCEPT**

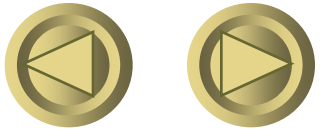


Summary(cont.)

- **The bag model for SQL, DISTINCT elimination of duplicate tuples; ALL allows the result to be a bag.**
- **Aggregations: SUM,AVG,MIN,MAX,COUNT**

GROUP BY, HAVING

- **Modification Statements: INSERT, DELETE, UPDATE**



SUMMARY(cont.)

- **Transactions: ACID**
- **Isolation levels :**

1. **Serializable: the transaction must appear to run either completely before or completely after each other transaction**
2. **Repeatable read: every tuple read in response to a query will reappear if the query is repeated.**
3. **read-committed: only tuples written by transactions that have already committed may be seen by the transaction.**
4. **Read-uncommitted: no constraint.**