

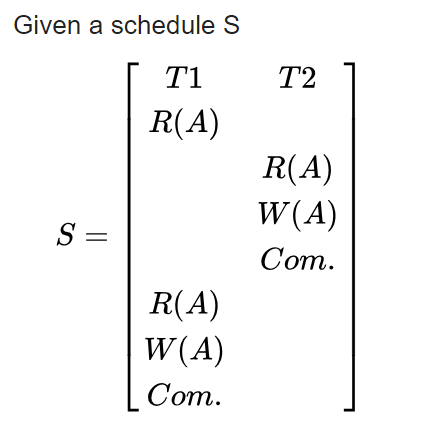
**Dirty read (WRITE-READ Conflict) Read other’s uncommitted WRITE data**

# Dirty read (WRITE-READ Conflict)

A transaction reads data written by a concurrent uncommitted transaction.

# Nonrepeatable read (READ-WRITE Conflict)

A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).



[Alice and Bob](https://en.wikipedia.org/wiki/Alice_and_Bob) are using a website to book tickets for a specific show. Only one ticket is left for the specific show. Alice signs on first to see that only one ticket is left, and finds it expensive. Alice takes time to decide. Bob signs on and also finds one ticket left, and orders it instantly. Bob purchases and logs off. Alice decides to buy a ticket, to find there are no tickets. This is a typical read-write conflict situation.

T1 T2

**R(X)**

R(Y)

R(X)

R(Y)

**W(X)**

**(READ-WRITE conflict)**

# Phantom read

A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently committed transaction.

For example: Select count(\*) from supplier; (READ)

Prevention of the Phantom Read (Make it SERIALIZABLE-issuing locks that prevent any insertion, or update while it is trying to read)

## Situation 1

Insert into emp values (‘111’, ‘John’, ‘AAA’)

It does not need any locks in order to insert a new record

Pick one below

**1.** **READ UNCOMMITTED, 2. READ COMMITTED and**

**3. SERIALIZABLE**

## Situation 2

Update emp set full\_name=’Jack’ where emp\_id = ‘111’;

Note: this requires an update to an existing record, need to issue a lock.

Pick one below

**1.** **READ UNCOMMITTED, 2. READ COMMITTED and**

**3. SERIALIZABLE**

## Situation 3

Select count(\*) from emp;

To prevent other transactions from inserting or updating the table emp while we are reading from it (known as the phantom problem), we would need to use

Pick one below

**1.** **READ UNCOMMITTED, 2. READ COMMITTED and 3. SERIALIZABLE**

In databases and transaction processing, two-phase locking (2PL) is a concurrency control method that **guarantees serializability**

According to the two-phase locking protocol, a transaction handles its locks in two distinct, consecutive phases during the transaction's execution:

1. Expanding phase (aka Growing phase): locks are acquired and no locks are released (the number of locks can only increase).
2. Shrinking phase (aka Contracting phase): locks are released and no locks are acquired.

The two-phase locking rule can be summarized as: never acquire a lock after a lock has been released. The serializability property is guaranteed for a schedule with transactions that obey this rule.

# Conservative two-phase locking

The difference between 2PL and C2PL is that C2PL's transactions obtain all the locks they need before the transactions begin. This is to ensure that a transaction that already holds some locks will not block waiting for other locks. Conservative 2PL prevents deadlocks.

# Strict two-phase locking

To comply with the S2PL protocol, a transaction needs to comply with 2PL, and release its write (exclusive) locks only after it has ended, i.e., being either committed or aborted. On the other hand, read (shared) locks are released regularly during phase 2. In contrast to 2PL, **Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.**

# Atomicity

Atomicity means all or nothing. Transactions often contain multiple separate actions. For example, a transaction may insert data into one table, delete from another table, and update a third table. Atomicity ensures that either all of these actions occur or none at all.

# Consistency

Consistency means that transactions always take the database from one consistent state to another. So, if a transaction violates the databases consistency rules, then the entire transaction will be rolled back.

# Isolation

Isolation means that **concurrent** transactions, and the changes made within them, are not visible to each other until they complete. This avoids many problems, including those that could lead to violation of other properties. The implementation of isolation is quite different in different DBMS’. This is also the property most often related to locking problems.

# Durability

Durability means that committed transactions will not be lost, even in the event of abnormal termination. That is, once a user or program has been notified that a transaction was committed, they can be certain that the data will not be lost.

# Sample test question:

1. The acronym ACID stands for A(tomic)C(onsistent)I(solated)D(urable).

Which of the above initials refers to the ability to run transactions concurrently? \_\_\_\_\_

Answer is? I

1. For ONE of the other terms, give a short description of a problem it solves.

A: All or none of transaction goes through

C: Integrity constraints preserved at end of transaction

D: Effects not lost after commit

1. What is the purpose of multiple levels of isolation in SQL? A two word answer please:

Increase concurrency