

[Log In](#)

[Join](#)

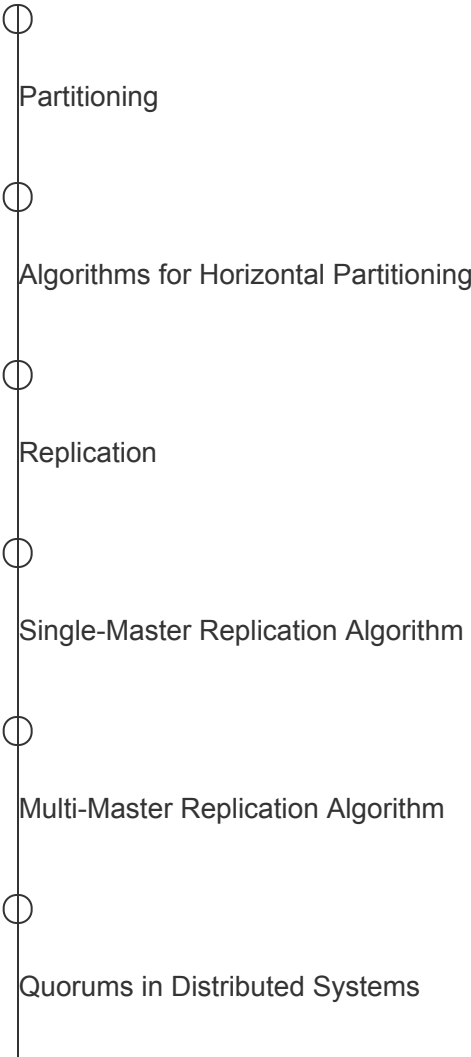
[Back To Module Home](#)


# Distributed Systems

0% completed

## Introduction to Distributed Systems

### Basic Concepts and Theorems





○	Safety Guarantees in Distributed Systems
○	ACID Transactions
○	The CAP Theorem
○	Consistency Models
○	CAP Theorem's Consistency Model
○	Isolation Levels and Anomalies
○	Prevention of Anomalies in Isolation Levels
○	Consistency and Isolation
○	Hierarchy of Models
○	Why All the Formalities?
○	Quiz

# Conclusion

Mark Module as Completed

# Isolation Levels and Anomalies

Let's see a list of Isolation levels and a detailed explanation of anomalies that may occur in distributed systems due to inherent concurrency.

We'll cover the following

- Anomalies
  - Dirty write
  - Dirty read
  - Fuzzy or non-repeatable read
  - Phantom read
  - Lost update
  - Read skew
  - Write skew

The inherent **concurrency** in distributed systems creates the potential for **anomalies** and unexpected behaviors. Specifically, transactions that comprise multiple operations and run concurrently can lead to different results depending on how their operations are interleaved.

As a result, there is still a need for some formal models that define what is possible and what is not in a system's behavior. These are called **isolation levels**.

We will study the most common ones here, which are the following:

- Serializability
- Repeatable read
- Snapshot isolation
- Read committed
- Read uncommitted

Unlike the consistency models presented in the Consistency Models lesson, some of these isolation levels do not define what is possible via some formal specification. Instead, they define what is not possible, i.e., which anomalies of the already known ones are prevented.

Of course, stronger isolation levels prevent more anomalies at the cost of performance. Let's first have a look at the possible anomalies before examining the various levels.

The origin of the isolation levels above and the associated anomalies was essentially the ANSI SQL-92 standard. However, the definitions in this standard were ambiguous and missed some possible anomalies.

Subsequent research examines more anomalies extensively and attempts a stricter definition of these levels. The basic parts will be covered in this lesson, but please refer to this paper for a deeper analysis.

# Anomalies#

The anomalies covered here are the following:

- Dirty writes
- Dirty reads
- (Fuzzy) non-repeatable reads
- Phantom reads
- Lost updates
- Read skew
- Write skew

## Dirty write#

A **dirty write** occurs when a transaction overwrites a value that was previously written by another transaction that is still in-flight and has not been committed yet.

One reason dirty writes are problematic is they can violate **integrity constraints**. For example, there are two transactions A and B, where transaction A runs the operations  $[x=1, y=1]$  and transaction B runs the operations  $[x=2, y=2]$ . Then, the serial execution of them would always result in a situation where x and y have the same value. However, this is not necessarily true in a concurrent execution where dirty writes are possible.

### Example

An example could be the following execution  $[x=1, x=2, y=2, \text{commit B}, y=1, \text{commit A}]$  that would result in  $x=2$  and  $y=1$ .

Another problem with dirty writes is they make it impossible for the system to automatically rollback to a previous image of the database. As a result, this is an anomaly we need to prevent in most cases.

# Dirty read#

A **dirty read** occurs when a transaction reads a value that has been written by another transaction that has not yet been committed.

This is problematic since the system might make decisions depending on these values, even though the associated transactions might be rolled back subsequently. Even in the case where these transactions eventually commit, though, this can still be a problem.

## Example

An example is the classic scenario of a bank transfer, where the total amount of money should be observed to be the same at all times. For example, imagine transaction A is able to read the balance of two accounts involved in a transfer during another transaction (B) that performs the transfer from account 1 to account 2. During the transfer, it will look as if some money has been lost from account 1.

However, there are a few cases where allowing dirty reads can be useful if done with care. One such case is to generate a big aggregate report on a full table when we can tolerate some inaccuracies on the numbers of the report.

It can also be useful when we troubleshoot an issue and want to inspect the state of the database in the middle of an ongoing transaction.

# Fuzzy or non-repeatable read#

A **fuzzy or non-repeatable read** occurs when a value is retrieved twice during a transaction (without it being updated in the same transaction), and the value is different.

This can lead to problematic situations similar to the example presented above for dirty reads.

Other cases where this can lead to problems are when the first read of the value is used for some conditional logic, and the second is used to update data. In this case, the transaction might act on stale data.

# Phantom read#

A **phantom read** occurs when a transaction does a predicate-based read, and another transaction writes or removes a data item matched by that predicate while the first transaction is still in flight. If that happens, then the first transaction might be acting again on stale data or inconsistent data.

## Example

For example, transaction A runs 2 queries to calculate the maximum and the average age of a specific set of employees. However, between the two queries, transaction B is interleaved and inserts a lot of old employees in this set, thus making transaction A return an average that is larger than the maximum.

Allowing phantom reads can be safe for an application that does not make use of predicate-based reads, i.e., performs only the reads that select records using a primary key.

# Lost update#

A **lost update** occurs when two transactions read the same value and then try to update it to two different values. The end result is that one of the two updates survives, but the process executing the other update is not informed that its update did not take effect. Thus it is called a lost update.

## Example

Imagine a warehouse with various controllers that update the database when new items arrive. The transactions are rather simple. They involve reading the number of items currently in the warehouse, adding the number of new items to this number, and then storing the result back in the database.

This anomaly could lead to the following problem:

- Transactions A and B read the current inventory size simultaneously (say, 100 items), add the number of new items to this (say, 5 and 10 respectively), and then store this back to the database. Let's assume that transaction B was the last one to write. This means that the final inventory is 110 instead of 115. Thus, five new items are *not* recorded!

See the following illustration for a visualization of this example.

#### Example of a lost update

Depending on the application, it might be safe to allow lost updates in some cases. For example, consider an application that allows multiple administrators to update specific parts of an internal website used by employees of a company.

## Read skew#

A **read skew** occurs when there are integrity constraints between two data items that seem to be violated because a transaction can only see partial results of another transaction.

### Example

Let's imagine an application that contains a table of persons, where each record represents a person and contains a list of all the friends of this person. The main integrity constraint is that friendships are mutual; if person B is included in person A's list of friends, then A must also be included in B's list. Every time someone (say, P1) wants to unfriend someone else (say, P2), a transaction is executed that removes P2 from P1's list



and also removes P1 from P2's list in a single go.

Now, let's also assume that some other part of the application allows people to view friends of multiple people simultaneously. This is done by a transaction that reads the friends list of these people. If the second transaction reads the friends list of P1 before the first transaction has started, but reads the friends list of P2 after the second transaction has been committed, it will notice an integrity violation. P2 will be in P1's list of friends, but P1 will not be in P2's list of friends.

Note that this case is not a dirty read, because any values written by the first transaction are read-only after it has been committed.

See the following illustration for a visualization of this example.

#### Example of a read skew

A strict requirement to prevent read skew is quite rare, as we might have guessed already. For example, a common application of this type might allow a user to view the profile of only one person at a time along with their friends, thus not requiring the integrity constraint described above.

## Write skew#

A **write skew** occurs when two transactions read the same data, but then modify disjoint sets of data.

### Example

Imagine an application that maintains the on-call rota of doctors in a hospital. A table contains one record for every doctor with a field indicating whether they are on-call. The application allows a doctor to remove themselves from the on-call rota if another doctor is also registered. This is done via a transaction that reads the number of doctors that are on-call from this table. If the number is greater than one, it updates the record corresponding to this doctor to not be on-call.

Now, let's look at the problems that can arise from the write skew phenomenon. Let's say two doctors, Alice and Bob, are on-call currently, and they both decide to see if they can remove themselves. Two transactions running concurrently might read the state of the database, see there are two doctors, and remove the associated doctor from being on-call. In the end, the system ends up with no doctors on-call!

See the following illustration for visualization of this example.

Example of a write skew

It is evident by now that there are many different anomalies for us to consider. On top of that, different applications manipulate data in different ways. So, we have to analyze each case separately to see which anomalies can create problems.

**Back**

CAP Theorem's Consistency Model

**Next**

Prevention of Anomalies in Isolation L...

Mark as Completed

---

Report an Issue