19.1 Concurrency

Locking

The ***concurrency system*** is a database component that manages concurrent transactions. The concurrency system implements isolation levels while attempting to optimize overall database performance. Tradeoffs between isolation levels and performance are complex, and many techniques have been implemented in concurrency systems. The most common technique is locking.

A ***lock*** is permission for one transaction to read or write data. Concurrent transactions are prevented from reading or writing the same data. A transaction takes locks when the transaction needs to read or write data. A transaction releases locks when the transaction is committed or no longer needs the locked data.

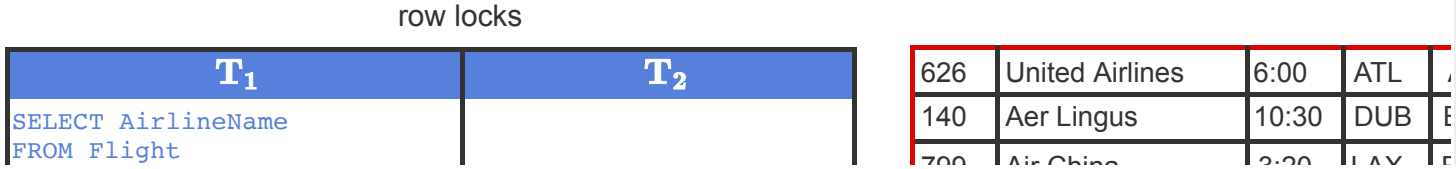
A ***shared lock*** allows a transaction to read, but not write, data. An ***exclusive lock*** allows a transaction to read and write data. Concurrent transactions can hold shared locks on the same data. When one transaction holds an exclusive lock, however, no concurrent transaction can take a shared or exclusive lock on the same data.

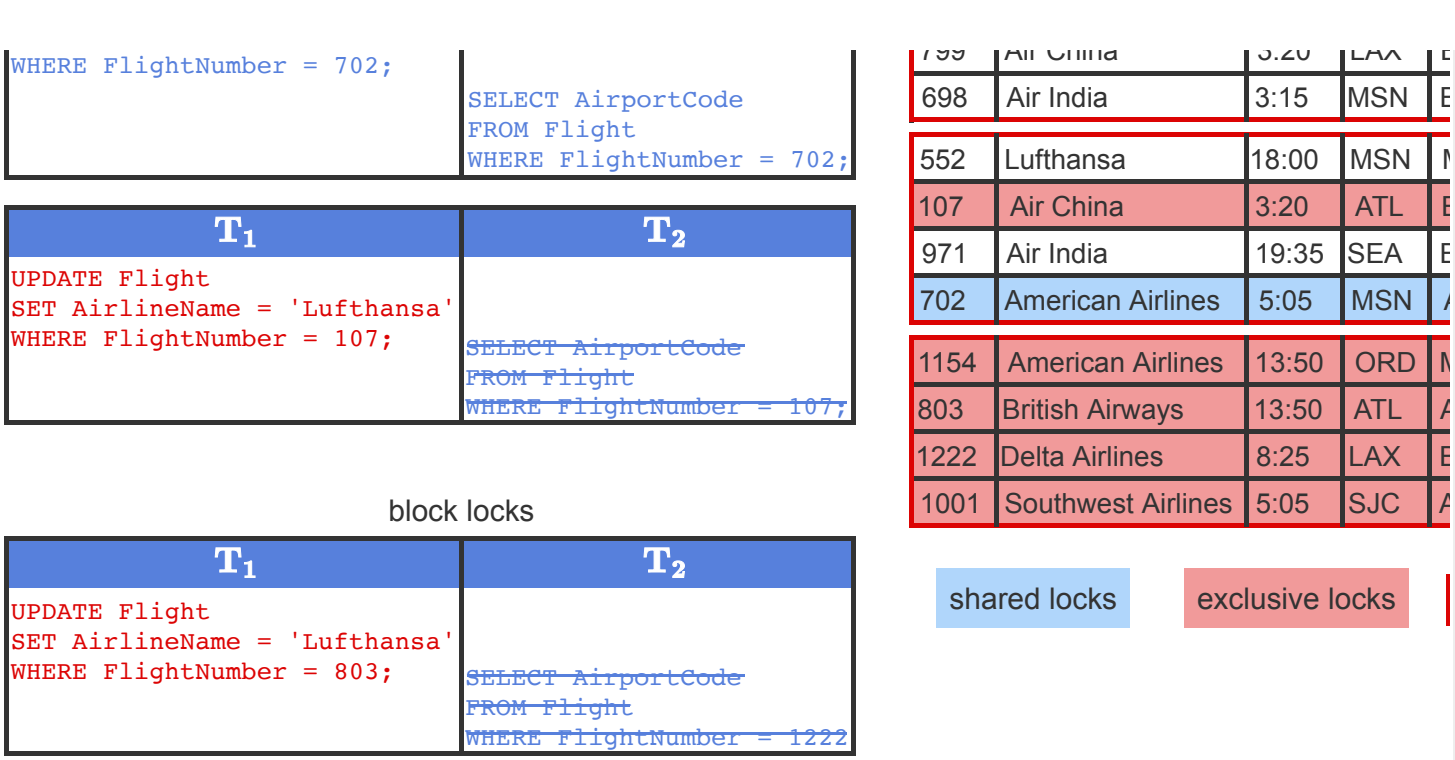
Lock scope is the collection of data reserved by a lock. Lock scope is often a single row, allowing other transactions to access other rows in the same table. If a transaction needs access to multiple rows, lock scope might be a block or the entire table. Since transactions also read and write indexes, lock scope might be an index entry, index block, or entire index.

The concurrency system monitors active transactions, determines when locks are needed, and issues requests to grant and release locks. Requests are sent to the ***lock manager***, a component of the concurrency system that tracks, grants, and releases locks.

By requesting shared and exclusive locks as needed, the concurrency system implements the isolation level specified for each transaction. By minimizing lock scope and duration, the concurrency system reduces the duration of transactions that are waiting for locked data.

PARTICIPATION ACTIVITY 19.1.1: Shared and exclusive locking.





Animation content:

Static figure:
Three transaction schedules appear.

The first schedule has caption row lock and actions in the following sequence:
T1: SELECT AirlineName FROM Flight WHERE FlightNumber = 702;
T2: SELECT AirportCode FROM Flight WHERE FlightNumber = 702;

The second schedule has caption row lock and actions in the following sequence:
T1: UPDATE Flight SET AirlineName = 'Lufthansa' WHERE FlightNumber = 107;
T2: SELECT AirportCode FROM Flight WHERE FlightNumber = 107;
The last action in this schedule is struck out.

The third schedule has caption block lock and actions in the following sequence:
T1: UPDATE Flight SET AirlineName = 'Lufthansa' WHERE FlightNumber = 803;
T2: SELECT AirportCode FROM Flight WHERE FlightNumber = 1222;
The last action in this schedule is struck out.

Three blocks of data appear, with columns flight number, airline name, departure time, airport code, and aircraft type. Each block has four rows of data. Rows are sorted on airport code. Highlights indicate a shared lock on one row and exclusive locks on another row and one block.

Step 1: The concurrent transactions T1 and T2 can take a shared lock on the same row. The

shared lock allows the transactions to read the row with FlightNumber 702. The first schedule appears. The fourth row of the second block takes a shared lock:
702, American Airlines, 5:05, MSN, Airbus 323

Step 2: If T1 takes an exclusive lock on the row with FlightNumber 107, T2 cannot access the row until the lock is released. The second schedule appears. The second row of the second block takes an exclusive lock:
107, Air China, 3:30, AT, Boeing 787
The T2 SELECT action, which attempts to read flight number 107, is struck out.

Step 3: If T1 takes an exclusive lock on a block, T2 cannot access any row in the block until the lock is released. The third schedule appears. The third block contains the following rows:
803, British Airways, 13:50, ATL, Airbus 323
1222, Delta Airlines, 8:25, LAX, Boeing 747
The third block takes an exclusive lock. The T2 SELECT action, which attempts to read fight number 1222, is struck out.

Animation captions:

1. The concurrent transactions T_1 and T_2 can take a shared lock on the same row. The shared lock allows the transactions to read the row with FlightNumber 702.
2. If T_1 takes an exclusive lock on the row with FlightNumber 107, T_2 cannot access the row until the lock is released.
3. If T_1 takes an exclusive lock on a block, T_2 cannot access any row in the block until the lock is released.

PARTICIPATION ACTIVITY 19.1.2: Locking.

1) Several transactions can hold concurrent shared locks on the same row.
   - ○ True
   - ○ False

2) Several transactions can hold concurrent exclusive locks on the same row.
   - ○ True

- False

3) One transaction can hold an exclusive lock while other transactions hold shared locks on the same row.
   - True
   - False

4) Several transactions can hold concurrent exclusive locks on the same block, as long as the transactions access different rows in the block.
   - True
   - False

5) When a transaction takes a shared lock on a block, other transactions may be delayed.
   - True
   - False

Two-phase locking

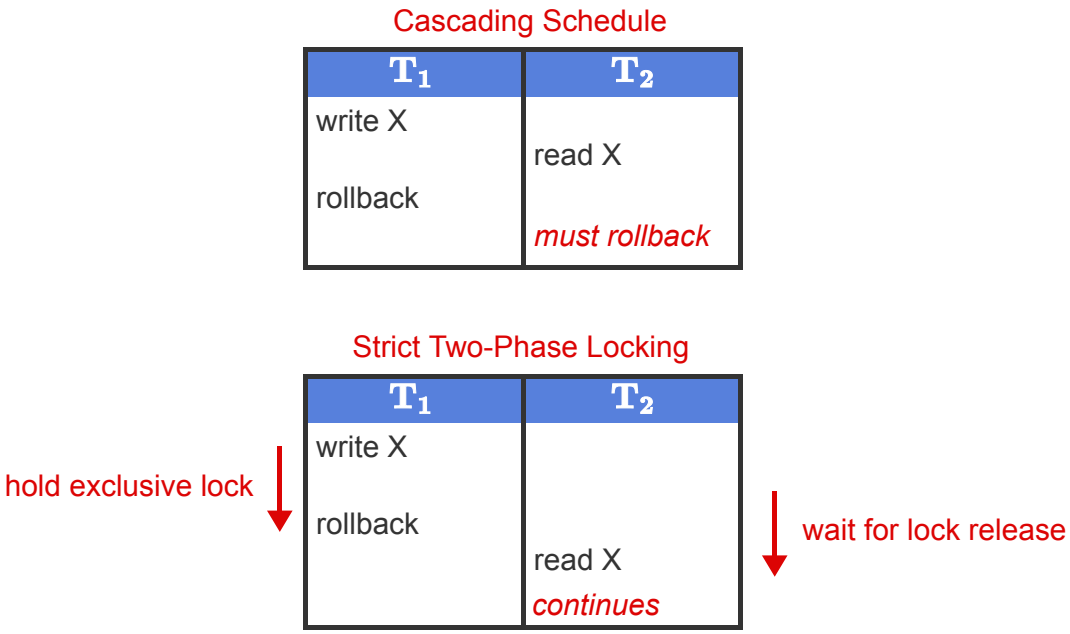
When a transaction's isolation level is set to SERIALIZABLE, a transaction is fully isolated from other transactions. ***Two-phase locking*** is a specific locking technique that ensures serializable transactions. Three variations of two-phase locking are common in relational databases:

- ***Basic two-phase locking*** has expand and contract phases for each transaction, also known as grow and shrink phases. In the expand phase, the transaction can take, but not release, locks. In the contract phase, the transaction can release, but not take, locks.
- ***Strict two-phase locking*** holds all exclusive locks until the transaction commits or rolls back. The expand phase is the same as in basic two-phase locking, but the contract phase releases only shared locks.
- ***Rigorous two-phase locking*** holds both shared and exclusive locks until the transaction commits or rolls back. In effect, rigorous two-phase locking has no contract phase.

All three variations prevent conflicts and ensure serializable transactions. Strict and rigorous also prevent cascading rollbacks while basic does not. Rigorous is easier to implement than strict, but less efficient, since rigorous holds shared locks longer.

Concurrency systems in most relational databases implement strict two-phase locking.

PARTICIPATION ACTIVITY 19.1.3: Strict two-phase locking.



Animation content:

Static figure:
Two transaction schedules appear.

The first schedule has caption cascading schedule and actions in the following sequence:
T1: write X
T2: read X
T1: rollback
T2: must rollback (highlighted and italicized)

The second schedule has caption strict two-phase locking and actions in the following sequence:
T1: write X
T1: rollback
T2: read X
T2: continues (highlighted and italicized)
An arrow, with caption hold exclusive lock, extends from the write X action to the rollback action.
An arrow, with caption wait for lock release, extends from the rollback action to the continues action.

Step 1: Rollback of T1 forces rollback of T2, so the schedule is cascading. The cascading schedule appears.

Step 2: With strict two-phase locking, exclusive locks are not released during the contract phase. The exclusive lock is held until T1 rolls back. The strict two-phase locking schedule appears, with actions in the same sequence as the cascading schedule. An arrow, with caption hold exclusive lock, appears. The arrow extends from the write X action to the rollback action.

Step 3: T2 read must wait until the exclusive lock is released. The strict two-phase locking actions change to:
T1: write X
T1: rollback
T2: read X
An arrow, with caption wait for lock release, appears. The arrow extends from the rollback action to the read X action.

Step 4: Rollback of T1 does not cascade to T2. The T2: continues action is added to the end of the strict two-phase locking schedule.

Animation captions:

1. Rollback of T_1 forces rollback of T_2, so the schedule is cascading.
2. With strict two-phase locking, exclusive locks are not released during the contract phase. The exclusive lock is held until T_1 rolls back.
3. T_2 read must wait until the exclusive lock is released.
4. Rollback of T_1 does not cascade to T_2.

PARTICIPATION ACTIVITY 19.1.4: Two-phase locking.

1) Which two-phase locking technique results in the longest wait times for concurrent transactions?

   - ○ Basic
   - ○ Strict
   - ○ Rigorous

2) Which two-phase locking technique has, in effect, just one phase?

has, in effect, just one phase?

- Basic
- Strict
- Rigorous

3) A database uses *basic* two-phase locking. Transaction A takes an exclusive lock on X. Transaction B requests a shared lock on X. When is the shared lock granted?

- During the contract phase of A
- Immediately, when B requests the shared lock
- After A commits or rolls back

4) A database uses *strict* two-phase locking. Transaction A takes an exclusive lock on X. Transaction B requests a shared lock on X. When is the shared lock granted?

- During the contract phase of A
- Immediately, when B requests the shared lock
- After A commits or rolls back

Deadlock

Deadlock is a state in which a group of transactions are frozen. In a deadlock, all transactions request access to data that is locked by another transaction in the group. None of the transactions can proceed. Ex:

1. T_1 takes an exclusive lock on X.
2. T_2 takes an exclusive lock on Y.
3. T_1 requests a shared lock on Y.
4. T_2 requests a shared lock on X.

T_1 waits for T_2 to release the lock on Y. T_2 waits for T_1 to release the lock on X. The transactions are deadlocked.

A ***dependent transaction*** is waiting for data locked by another transaction. A ***cycle*** of dependent transactions indicates deadlock has occurred. Ex: T_1 depends on T_2, which depends on T_3, which

transactions indicates deadlock has occurred. Ex: T_1 depends on T_2, which depends on T_3, which depends on T_1. The transactions are deadlocked.

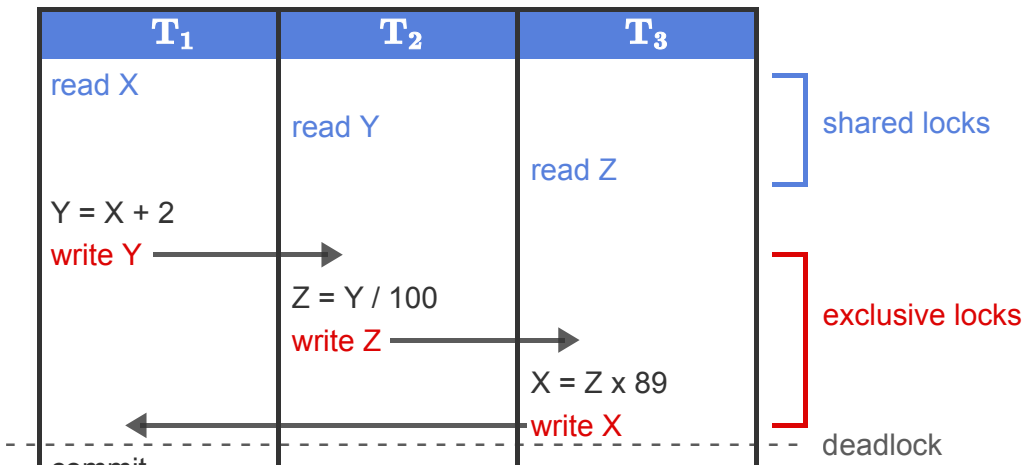
Concurrency systems manage deadlock with a variety of techniques:

- ***Aggressive locking***. Each transaction requests all locks when the transaction starts. If all locks are granted, the transaction runs to completion. If not, the transaction waits until other transactions release locks. Either way, the transaction cannot participate in a deadlock.
- ***Data ordering***. All data needed by concurrent transactions is ordered, and each transaction takes locks in order. Taking locks in order prevents deadlock. Ex: The sequence above is modified so that locks on X precede locks on Y:

  1. T_1 takes an exclusive lock on X.
  2. T_2 requests a shared lock on X.
  3. T_1 takes a shared lock on Y.
  4. T_2 takes an exclusive lock on Y.

  Request 2 waits for T_1 to release the exclusive lock on X. T_1 proceeds and eventually releases the lock on X. Now request 2 is granted and T_2 proceeds.
- ***Timeout***. When waiting time for a lock exceeds a fixed period, the transaction requesting the lock rolls back. Alternatively, the concurrency system compares transaction start times and rolls back the later transaction. The timeout period is set by the database or configured by the database administrator.
- ***Cycle detection***. The concurrency system periodically checks for cycles of dependent transactions. When a cycle is detected, the concurrency system selects and rolls back the 'cheapest' transaction. The cheapest transaction might, for example, have the fewest rows locked or most recent start time. The rollback breaks the deadlock.

PARTICIPATION ACTIVITY 19.1.5: Cycle of dependent transactions.



commit		
	commit	
		commit

Animation content:

Static figure:
A schedule has three transactions and actions in the following sequence:
T1: read X
T2: read Y
T3: read Z
T1: Y = X + 2
T1: write Y
T2: Z = Y / 100
T2: write Z
T3: X = Z x 89
T3: write X
T1: commit
T2: commit
T3: commit

The three read actions have caption shared locks. The three write actions have caption exclusive locks. A dashed line above the commit actions has caption deadlock. Arrows indicate each exclusive lock blocks another transaction:
T1: write Y blocks T2
T2: write Z blocks T3
T3: write X blocks T1

Step 1: The schedule has three transactions that read and write X, Y, and Z. The schedule appears.

Step 2: Transactions first take shared locks when reading, then request exclusive locks when writing. The read actions are labeled shared locks. The write actions are labeled exclusive locks.

Step 3: Since T2 has a shared lock on Y, the T1 exclusive lock request for Y waits for T2 to commit and release the shared lock. An arrow appears, indicating T1: write Y blocks T2.

Step 4: Similarly, T2 waits for T3 to release the shared lock on Z, and T3 waits for T1 to release the shared lock on X. Two more arrows appear, indicating T2: write Z blocks T3 and T3: write X blocks T1.

Step 5: The dependency cycle causes deadlock. A dashed line appears above the commit actions, with caption deadlock.

Animation captions:

1. The schedule has three transactions that read and write X, Y, and Z.
2. Transactions first take shared locks when reading, then request exclusive locks when writing.
3. Since T_2 has a shared lock on Y, the T_1 exclusive lock request for Y waits for T_2 to commit and release the shared lock.
4. Similarly, T_2 waits for T_3 to release the shared lock on Z, and T_3 waits for T_1 to release the shared lock on X.
5. The dependency cycle causes deadlock.

PARTICIPATION ACTIVITY 19.1.6: Deadlock.

1) Refer to the above animation. How many transactions must roll back to break the deadlock?
   - ○ One
   - ○ Two
   - ○ Three

2) Whenever deadlock occurs, a cycle of dependent transactions always exists.
   - ○ True
   - ○ False

3) Deadlock can occur in a serializable schedule.
   - ○ True
   - ○ False

4) A transaction with isolation level SERIALIZABLE can participate in a deadlock.
   - ○ True

- False

5) Which deadlock management technique delays the fewest possible transactions?

- Aggressive locking
- Data ordering
- Timeout
- Cycle detection

Snapshot isolation

Exclusive locks prevent concurrent transactions from accessing data and therefore increase transaction duration. For this reason, many databases support alternative concurrency techniques. ***Optimistic techniques*** execute concurrent transactions without locks and, instead, detect and resolve conflicts when transactions commit. Optimistic techniques are effective when conflicts are infrequent, as in analytic applications with many reads and few updates.

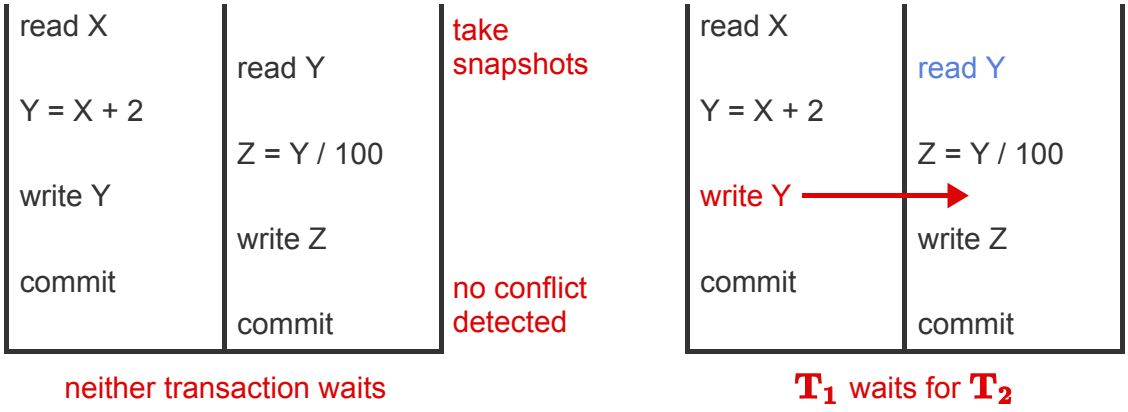
One leading optimistic technique is snapshot isolation. ***Snapshot isolation*** creates a private copy of all data accessed by a transaction, called a ***snapshot***, as follows:

1. Create a snapshot when the transaction starts.
2. Apply updates to the snapshot rather than the database.
3. Prior to commit, check for conflicts. A conflict occurs when concurrent transactions write the same data. Reads do not cause conflicts.
4. If no conflict is detected, write snapshot updates to the database (commit).
5. If a conflict is detected, discard the snapshot and restart the transaction (roll back).

Since snapshot isolation does not take locks, transactions never wait. However, transactions occasionally restart after all operations are processed.

Two-phase locking is the most common concurrency technique, implemented in most relational databases. Snapshot isolation is a widely used alternative, available in Oracle and SQL Server.

PARTICIPATION ACTIVITY 19.1.7: Snapshot isolation vs. two-phase locking.



Animation content:

Static figure:
Two schedules appear, with captions snapshot isolation and two-phase locking. Both schedules have the same sequence of actions:
T1: read X
T2: read Y
T1: Y = X + 2
T2: Z = Y / 100
T1: write X
T2: write Z
T1: commit
T2: commit

The snapshot isolation schedule has caption neither transaction waits. The read actions have caption take snapshots. The commit actions have caption no conflict detected.

The two-phase locking schedule has caption T1 waits for T2. An arrow indicates T1: write X blocks T2.

Step 1: Under snapshot isolation, the transactions take snapshots and process in parallel. The snapshot isolation schedule appears. The take snapshots caption appears.

Step 2: Since the transactions write different data, no conflict is detected. Both transactions commit. The write actions are highlighted. The no conflict detected caption appears.

Step 3: Under snapshot isolation, neither transaction waits. The caption neither transaction waits appears.

Step 4: Under two-phase locking, T1 waits for T2 to commit and release the shared lock on Y. The two-phase locking schedule, caption, and arrow appear.

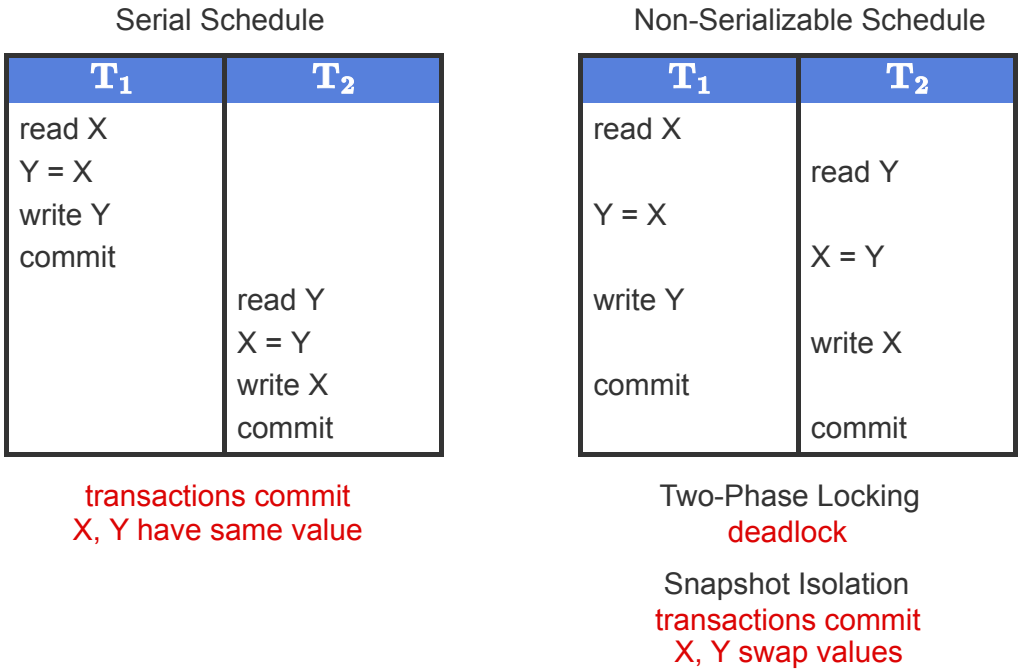
Animation captions:

1. Under snapshot isolation, the transactions take snapshots and process in parallel.
2. Since the transactions write different data, no conflict is detected. Both transactions commit.
3. Under snapshot isolation, neither transaction waits.
4. Under two-phase locking, T_1 waits for T_2 to commit and release the shared lock on Y.

When isolation level is set to SERIALIZABLE, two-phase locking ensures serializable schedules. Snapshot isolation does not ensure serializable schedules and may generate unexpected results. As a result, some databases implement a modified version of snapshot isolation.

Serializable snapshot isolation extends standard snapshot isolation and ensures serializable schedules when isolation level is set to SERIALIZABLE. Postgres supports serializable snapshot isolation.

PARTICIPATION ACTIVITY 19.1.8: Snapshot isolation allows non-serializable schedule.



Animation content:

Static figure:

Two schedules appear, with captions serial schedule and non-serializable schedule.

The serial schedule has actions in the following sequence:
T1: read X
T1: Y = X
T1: write Y
T1: commit
T2: read Y
T2: X = Y
T2: write X
T2: commit

The serial schedule has one caption:
Transactions commit: X, Y have same value

The non-serializable schedule has actions in the following sequence:
T1: read X
T2: read Y
T1: Y = X
T2: X = Y
T1: write Y
T2: write X
T1: commit
T2: commit

The non-serializable schedule has two captions:
Two-phase locking: deadlock
Snapshot isolation: transactions commit / X, Y swap values

Step 1: The serial schedule commits and results in the same value for X and Y. The serial schedule appears, with caption Transactions commit.

Step 2: The non-serializable schedule contains conflicting operations in a different order than the serializable schedule. The non-serializable schedule appears. Actions T1: write Y and T2: read Y are highlighted, indicating the order has reversed.

Step 3: The non-serializable schedule contains a cycle and results in deadlock under two-phase locking. In the non-serializable schedule, the Two-phase locking caption appears. An arrow indicates T1: write Y blocks T2. Another arrow indicates T2: write X blocks T1.

Step 4: Under snapshot isolation, no conflict is detected, and the transactions commit. In the non-serializable schedule, the Snapshot isolation caption appears. The arrows disappear.

Step 5: Snapshot isolation results in different values for X and Y. The caption X, Y swap values appears.

Animation captions:

1. The serial schedule commits and results in the same value for X and Y.
2. The non-serializable schedule contains conflicting operations in a different order than the serializable schedule.
3. The non-serializable schedule contains a cycle and results in deadlock under two-phase locking.
4. Under snapshot isolation, no conflict is detected, and the transactions commit.
5. Snapshot isolation results in different values for X and Y.

PARTICIPATION ACTIVITY 19.1.9: Snapshot isolation.

Place snapshot isolation steps in the correct order.

If unable to drag and drop, refresh the page.

Write updates to a private copy of data

Determine if any updates conflict with other transactions

Make a private copy of data accessed by the transaction

Write updates to the database or roll back the transaction

	Step 1
	Step 2
	Step 3
	Step 4

Reset

Reset

CHALLENGE ACTIVITY | 19.1.1: Concurrency.

544874.3500394.qx3zqy7

Start

T_1	T_2

`SELECT CountryID`
`FROM Country`
`WHERE CountryID = 818;`
	`UPDATE Country`
	`SET Area = 384362.3937`
	`WHERE CountryID = 818;`

If lock scope is a single row, and T_2 takes an exclusive lock on CountryID 818, what happe

Pick

If lock scope is a block, and T_2 takes an exclusive lock on CountryID 818, select the row(s)

☐	760	Syria	Damascus	70899.93937	Asia
☐	417	Kyrgyzstan	Bishkek	74054.39401	Asia
☐	524	Nepal	Kathmandu	55347.74443	Asia
☐	608	Philippines	Manila	115124.0806	Asia
☐	818	Egypt	Cairo	384345.3937	Africa
☐	840	United States	Washington	3531838.607	NAmerica
☐	380	Italy	Rome	113568.0889	Europe
☐	554	New Zealand	Wellington	101664.5594	Oceania
☐	716	Zimbabwe	Harare	149363.62	Africa

1 2 3 4

Check Next