7.3 Concurrency

This section has been set as optional by your instructor.

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.

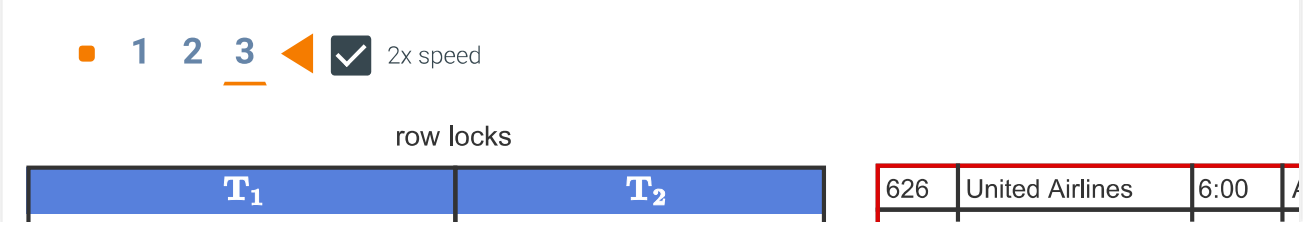
A ***shared lock*** allows a transaction to read, but not write, data. An ***exclusive lock*** allows a transaction to read and write data. Concurrent transactions can hold shared locks on the same data. When one transaction holds an exclusive lock, however, no concurrent transaction can take a shared 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 7.3.1: Shared and exclusive locking.



T_1	T_2
SELECT AirlineName FROM Flight WHERE FlightNumber = 702;	SELECT AirportCode FROM Flight WHERE FlightNumber = 702;

T_1	T_2
UPDATE Flight SET AirlineName = 'Lufthansa' WHERE FlightNumber = 107;	SELECT AirportCode FROM Flight WHERE FlightNumber = 107;

block locks

T_1	T_2
UPDATE Flight SET AirlineName = 'Lufthansa' WHERE FlightNumber = 803;	SELECT AirportCode FROM Flight WHERE FlightNumber = 1222

140	Aer Lingus	10:30	D
799	Air China	3:20	L
698	Air India	3:15	M
552	Lufthansa	18:00	M
107	Air China	3:20	A
971	Air India	19:35	S
702	American Airlines	5:05	M
1154	American Airlines	13:50	C
803	British Airways	13:50	A
1222	Delta Airlines	8:25	L
1001	Southwest Airlines	5:05	S

shared locks exclusive loc

If T_1 takes an exclusive lock on a block, T_2 cannot access any row in the block until the lock is

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.

Feedback?

PARTICIPATION ACTIVITY 7.3.2: Locking.

1) Several transactions can hold concurrent shared locks on the same row.

- (●) True
- ( ) False

Correct

Shared locks are taken when transactions read data. Any number of transactions can read the same data concurrently.

2) Several transactions

Correct

can hold concurrent exclusive locks on the same row.

- ○ True
- ◉ False

Exclusive locks are taken when transactions write data. Conflicts may arise when multiple transactions write the same data concurrently, so the lock manager grants an exclusive lock on a given row to one transaction at a time.

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

- ○ True
- ◉ False

Correct

Conflicts may arise when one transaction reads data while another is writing the data, so the lock manager does not grant shared locks on a given row while any transaction holds an exclusive lock.

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

Correct

A block-level lock restricts access to all rows on the block. While a transaction holds an exclusive lock on a block, no other transaction can access any row on the block.

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

- ◉ True
- ○ False

Correct

While a transaction holds a shared lock on a block, other transactions may take shared locks but not exclusive locks. Transactions requesting exclusive locks on the block, or any row in the block, must wait for release of the shared lock.

Feedback?

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:

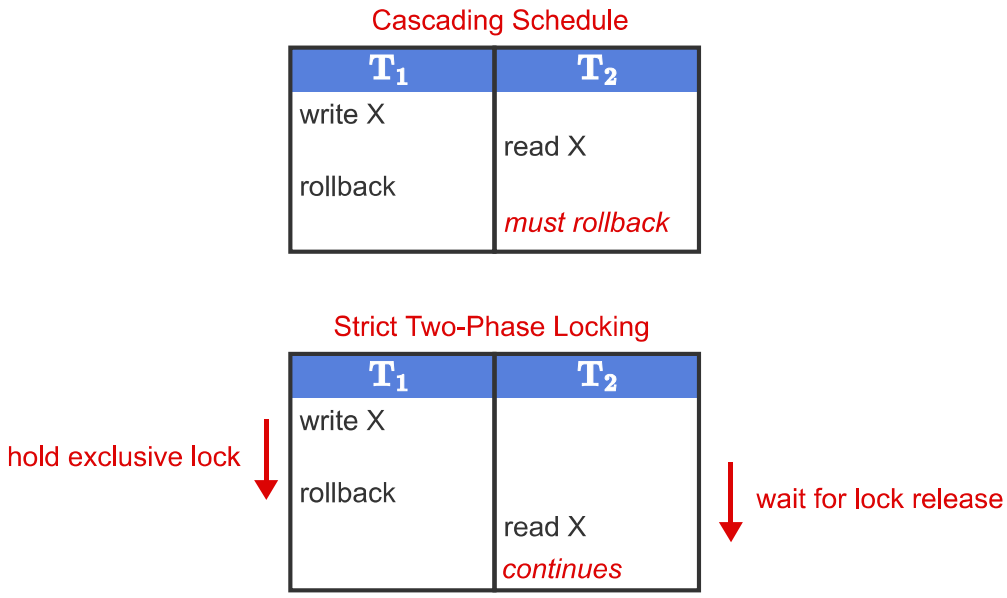
- ***Basic two-phase locking*** has expand and contract phases for each transaction, also known as grow and shrink phases. In the expand phase, the transaction can take, but not release, locks. In the contract phase, the transaction can release, but not take, locks.
- ***Strict two-phase locking*** holds 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 7.3.3: Strict two-phase locking.

1 2 3 4 2x speed



Rollback of T_1 does not cascade to T_2.

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.

Feedback?

PARTICIPATION ACTIVITY 7.3.4: Two-phase locking.

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

- Basic
- Strict
- Rigorous (selected)

Correct

Rigorous holds both shared and exclusive locks until a transaction commits or rolls back. Any other transaction requesting locks on the same data must wait until commit or rollback.

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

- Basic
- Strict
- Rigorous (selected)

Correct

Rigorous holds all locks until a transaction commits or rolls back. Although rigorous is considered a type of two-phase locking, rigorous has only an expand phase.

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 (selected)
- Immediately, when B requests the shared lock
- After A commits or rolls back

Correct

With basic two-phase locking, the exclusive lock is released during the contract phase of A. B is granted a shared lock when the exclusive lock is released.

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

Correct

With strict two-phase locking, all exclusive locks are released at the time of commit or rollback. B's shared lock request must wait until A commits or rolls back.

Feedback?

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 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.

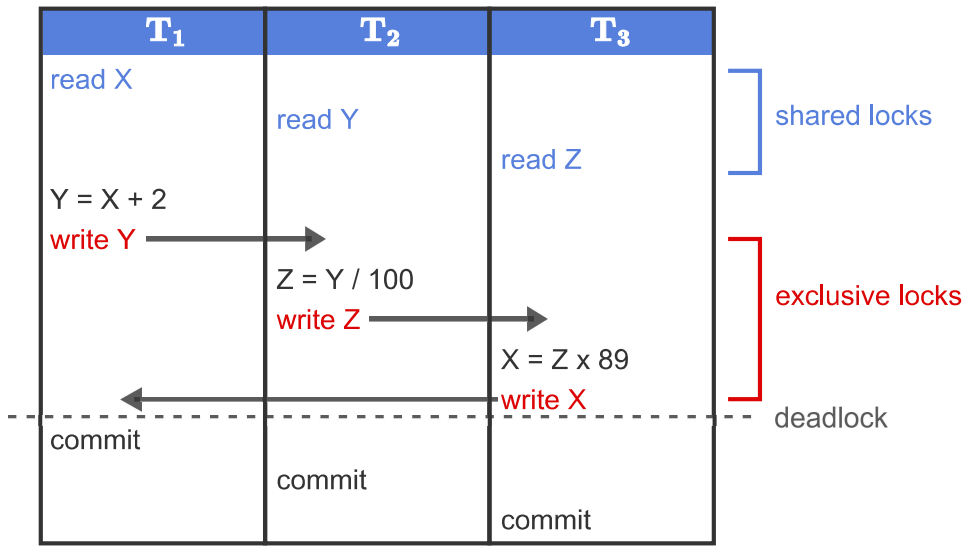
  Request 2 waits for T_1 to release the exclusive lock on X. T_1 proceeds and eventually releases the lock on X. Now request 2 is granted and T_2 proceeds.

- ***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 7.3.5: Cycle of dependent transactions.

1 2 3 4 5 2x speed



The dependency cycle causes deadlock.

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.

Feedback?

PARTICIPATION ACTIVITY 7.3.6: Deadlock.

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

Correct

Rolling back any one of the three transactions eliminates the cycle and breaks the deadlock. Ex: If T_3 rolls back, T_2 is no longer dependent and can proceed. When T_2 commits, T_1 can proceed.

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

Correct

Deadlock always involves a cycle. Without a cycle, some transaction is not dependent and can proceed, so deadlock cannot exist.

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

Correct

Serial schedules have no concurrent transactions and therefore cannot deadlock. Since a serializable schedule is equivalent to a serial schedule, serializable schedules also cannot deadlock.

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

Correct

In principle, SERIALIZABLE transactions run in serializable schedules and cannot deadlock. In practice, however, databases often enforce SERIALIZABLE transactions by allowing deadlock to occur and then rolling back one of the participating transactions.

- False

5) Which deadlock management technique delays the fewest possible transactions?
   - Aggressive locking
   - Data ordering
   - Timeout
   - Cycle detection

Correct

Cycle detection does not unnecessarily delay any transactions. Under cycle detection, transactions roll back only when participating in deadlock.

Feedback?

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).

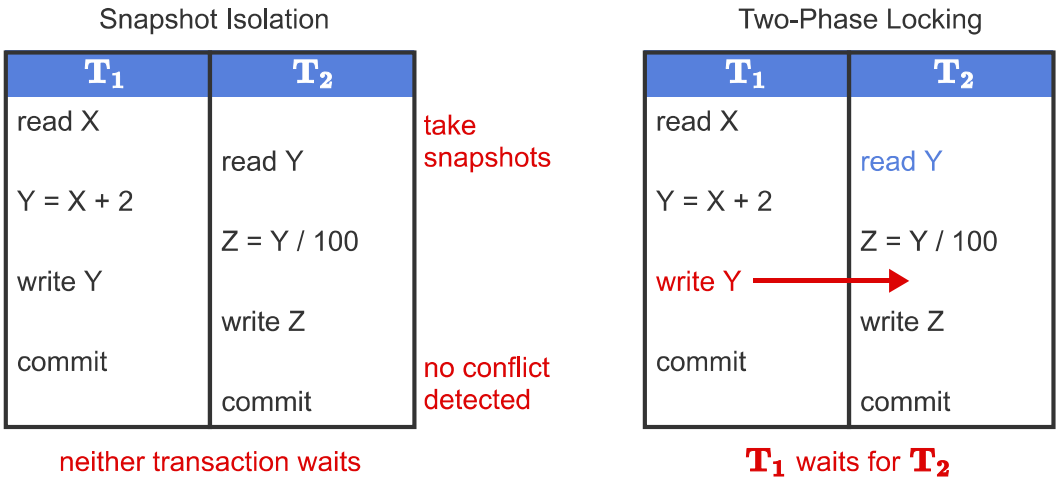
Since snapshot isolation does not take locks, transactions never wait. However, transactions occasionally restart after all operations are processed.

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 7.3.7: Snapshot isolation vs. two-phase locking.

1 2 3 4

1 2 3 4 2x speed



Under two-phase locking, T_1 waits for T_2 to commit and release the shared lock on Y.

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.

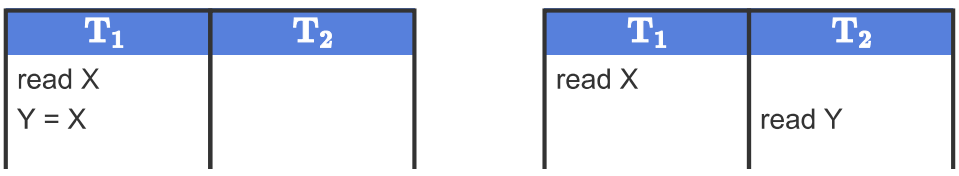
Feedback?

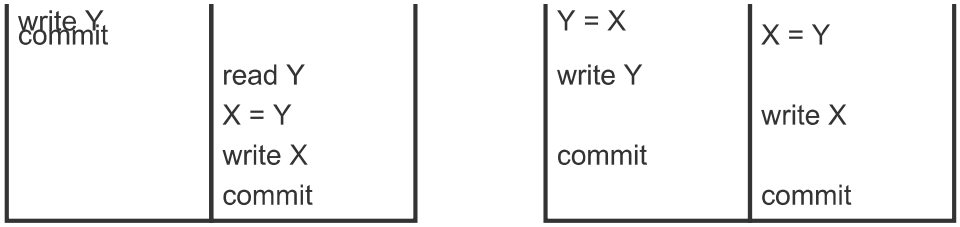
When isolation level is set to SERIALIZABLE, two-phase locking ensures serializable schedules. Snapshot isolation does not ensure serializable schedules and may generate unexpected results. As a result, some databases implement a modified version of snapshot isolation.

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 7.3.8: Snapshot isolation allows non-serializable schedule.

1 2 3 4 5 2x speed





transactions commit
X, Y have same value

Two-Phase Locking
deadlock

Snapshot Isolation
transactions commit
X, Y swap values

Snapshot isolation results in different values for X and Y.

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.

Feedback?

PARTICIPATION ACTIVITY 7.3.9: Snapshot isolation.

Place snapshot isolation steps in the correct order.

Make a private copy of data accessed by the transaction	Step 1 Snapshot isolation begins by making a private copy of all data read and written by the transaction.	Correct
Write updates to a private copy of data	Step 2 During transaction execution, updates are applied to a private copy of data rather than the database, allowing the transaction	Correct

	to proceed without waiting for locks.	
Determine if any updates conflict with other transactions	Step 3 Prior to saving updates to the database, snapshot isolation checks if any other transactions updated the same data after the private copy was made.	Correct
Write updates to the database or roll back the transaction	Step 4 If updates do not conflict with other transactions, updates in the private copy are saved in the database. If updates conflict, the transaction rolls back.	Correct

Reset

Feedback?

CHALLENGE ACTIVITY 7.3.1: Concurrency.

379958.2369558.qx3zqy7

Jump to level 1

1 2 3 4

Consider the following concurrent transactions:

T_1	T_2
read Z	
	read Y
X = Z * 41	
	X = Y - 11
write X	
	write X
commit	
	commit

What occurs under:

snapshot isolation?

(c) T1 writes X, commits, and T2 rolls back.

rigorous two-phase locking?

(a) T1 writes X, commits, and T2 waits until T1 commits.

1 2 3 4

Check Next

Done. Click any level to practice more. Completion is preserved.

Expected:
(c) T1 writes X, commits, and T2 rolls back.
(a) T1 writes X, commits, and T2 waits until T1 commits.

Under snapshot isolation:
Transactions create a private copy of data when starting and check for write conflict
Since T_1 writes X first, T_1 writes X and commits. T_2 tries to write X second, so a cor
rolls back.

Under rigorous two-phase locking:
A transaction holds all shared and exclusive locks until the transaction commits or r
before T_2 and immediately takes an exclusive lock on X. T_2 must wait until T_1 comr
exclusive lock on X.

Feedback?

How was this section? **Provide feedback**