18.1 Transactions

Transactions

A ***transaction*** is a sequence of database operations that must be either completed or rejected as a whole. Partial execution of a transaction results in inconsistent or incorrect data.

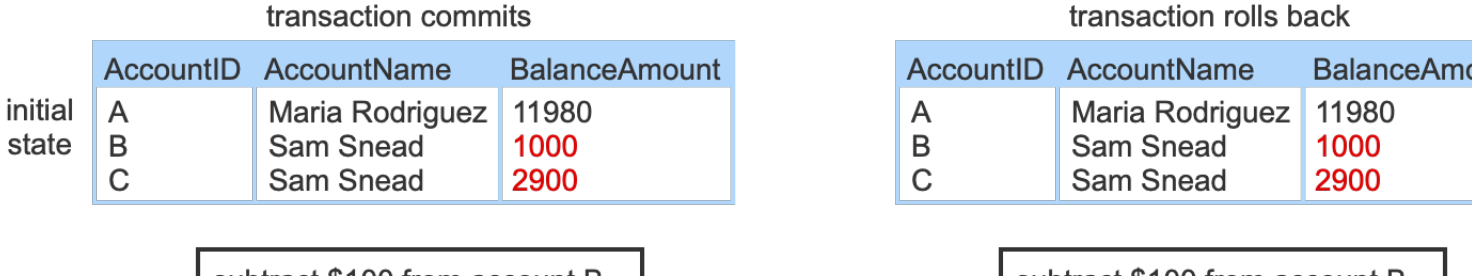
Ex: The debit-credit transaction transfers funds from one bank account to another. The first operation removes funds, say $100, from one account, and the second operation deposits $100 in another account. If the first operation succeeds but the second fails, $100 is mysteriously lost. The transaction must complete and save either both operations or neither operation.

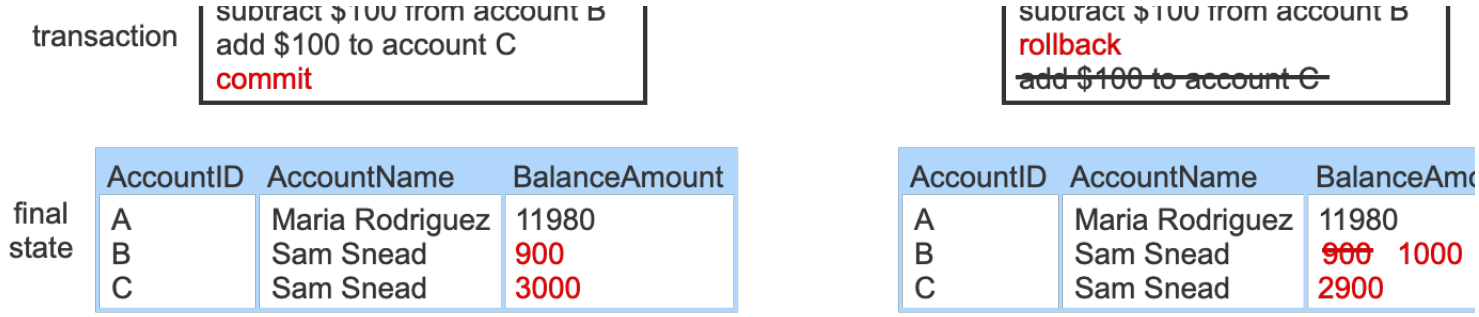
Saving complete transaction results in the database is called a ***commit***. Rejecting an incomplete transaction is called a ***rollback***. A rollback reverses the transaction and resets data to initial values. A variety of circumstances cause a rollback:

- The operating system detects a device failure. Ex: Magnetic disk fails during execution of a transaction, and transaction results cannot be written to the database.
- The database detects a conflict between concurrent transactions. Ex: Two airline customers attempt to reserve the same seat on a flight.
- The application program detects an unsuccessful database operation. Ex: In the debit-credit transaction, funds are removed from the debit account, but the credit account is deleted prior to deposit.

When a failure occurs, the database is notified and rolls back the transaction. If the failure is temporary, such as intermittent network problems, the database attempts to restart the transaction. If the failure is persistent, such as a deleted bank account, the database 'kills' the transaction permanently.

PARTICIPATION ACTIVITY 18.1.1: Commit and rollback.





Animation content:

Static figure:
Two diagrams appear, with captions transaction commits and transaction rolls back. Both diagrams show a transaction between two unnamed tables.

The first diagram, transaction commits, appears as follows:

The upper table has caption initial state. The table has columns AccountID, AccountName, and BalanceAmount, and three rows:
A, Maria Rodriquez, 11980
B, Sam Snead, 1000
C Sam Snead, 2900
Values 1000 and 2900 are highlighted.

The transaction is a rectangle containing three actions:
subtract $100 from account B
add $100 to account C
commit
The commit action is highlighted.

The lower table has caption final state. The table has the same columns as the upper table and three rows:
A, Maria Rodriquez, 11980
B, Sam Snead, 900
C Sam Snead, 3000
Values 900 and 3000 are highlighted.

The second diagram, transaction rolls back, appears as follows:

The upper table is the same as the upper table in the first diagram.

The transaction is a rectangle containing three actions:
subtract $100 from account B
roll back
add $100 to account C
The roll back action is highlighted. The add action is struck out.

The lower table has caption final state. The table has the same columns as the upper table and three rows:
A, Maria Rodriquez, 11980
B, Sam Snead, 1000
C Sam Snead, 2900
Values 1000 and 2900 are highlighted.

Step 1: Initially, Sam Snead has $1,000 in account B and $2,900 in account C. The caption transaction commits appears. The upper table appears

Step 2: The transaction changes balances and commits. Changes are saved in the database. The transaction and lower table appear. The subtract action and the value 900 in the lower table are highlighted. The add action and the value 3000 in the lower table are highlighted.

Step 3: If the operating system, database, or application fails during transaction, the transaction must roll back. The caption transaction rolls back appears. The upper table appears.

Step 4: The database updates account B, detects failure, and restores account B to the initial value $1,000. The transaction and lower table appear. The subtract action and the value 900 in the lower table are highlighted. The rollback action is highlighted and the value 900 is replaced by 1000.

Step 5: After rollback, the transaction terminates. The final operation is not executed, so account C remains $2,900. The add action and the value 2900 in the lower table are highlighted.

Animation captions:

1. Initially, Sam Snead has $1,000 in account B and $2,900 in account C.
2. The transaction changes balances and commits. Changes are saved in the database.
3. If the operating system, database, or application fails during transaction, the transaction must roll back.
4. The database updates account B, detects failure, and restores account B to the initial value $1,000.
5. After rollback, the transaction terminates. The final operation is not executed, so account C remains $2,900.

PARTICIPATION ACTIVITY 18.1.2: Transactions.

1) How many SQL statements must be in one transaction?
   - Exactly one
   - At least one
   - At least two

2) After a transaction commits, the transaction can be rolled back:
   - Always
   - Sometimes
   - Never

3) After a rollback, the database restarts a transaction:
   - Always
   - Sometimes
   - Never

ACID properties

All transactions must be atomic, consistent, isolated, and durable, commonly called the ***ACID*** properties:

- In an ***atomic*** transaction, either all or none of the operations are executed and applied to the database. Partial or incomplete results are rolled back, and the database returns to its state prior to execution of the transaction.
- In a ***consistent*** transaction, all rules governing data are valid when the transaction is committed. Completed transactions that violate any rules are rolled back.

  Consistency applies to both universal and business rules. Universal rules apply to all relational data. Ex: Primary keys must be unique and not NULL. Business rules are particular to a specific database or application. Ex: Funds must not be lost in a debit-credit transaction.
- An ***isolated*** transaction is processed without interference from other transactions. Isolated transactions behave as if each transaction were executed one at a time, or serially, when in

transactions behave as if each transaction were executed one at a time, or serially, when in fact the transactions are processed concurrently.

Computers usually process multiple transactions concurrently. Multiple processors, or cores, in a single computer might work on multiple transactions in parallel. A single processor might switch to a new transaction while waiting for an active transaction to read or write data.

Concurrent transactions that access the same data might conflict. Ex: One transaction sums all salaries while another increases all salaries by 10%. If both transactions run concurrently, the sum might include some increased salaries but not others, and thus the sum might be invalid. To ensure transactions are isolated, databases must prevent conflicts between concurrent transactions.

- A ***durable*** transaction is permanently saved in the database once committed, regardless of system failures.

  System failures potentially cause the loss of transaction data after the transaction is committed. Ex: An application commits a transaction, the transaction data is written to blocks in memory, but hardware fails before the blocks are saved on magnetic disk. Because transaction results are lost, the transaction is not durable.

The ACID properties are supported in two database subsystems. The ***recovery system*** enforces atomic and durable transactions. The ***concurrency system*** enforces isolated transactions. Both the recovery and concurrency systems, along with other database components, support consistency.

Table 18.1.1: Subsystem support for ACID properties.

	Atomic	Consistent	Isolated	Durable
Concurrency system	supporting	supporting	*primary*	none
Recovery system	*primary*	supporting	supporting	*primary*

PARTICIPATION ACTIVITY 18.1.3: ACID properties.

Each example violates an ACID property. Match the property with the violation.

If unable to drag and drop, refresh the page.

Durable **Atomic** **Consistent** **Isolated**

	A transaction increases all employee salaries by 10%. Due to a system failure, increases for only half of the employees are written to the database.
	A transaction saves a row with a foreign key. The foreign key is not NULL and does not match any values of the corresponding primary key.
	Two transactions running in parallel reserve the same seat for different passengers.
	A transaction withdraws $500 from account A and deposits $500 in account B. The withdrawal and deposit are written in the database, but due to a disk drive failure, the information is permanently lost.

Reset

Isolation

Concurrent transactions T_1 and T_2 can conflict in many ways. Common conflicts include dirty read, nonrepeatable read, and phantom read.

In a ***dirty read***, a transaction reads data that has been updated in a second, uncommitted transaction. A dirty read creates invalid results when the second transaction rolls back or makes additional updates to the data. Ex:

1. T_2 updates data X.
2. T_1 reads the updated value of X before T_2 commits.
3. T_2 fails and is rolled back.

Since T_1 reads a value that is eventually rolled back, the result of T_1 is invalid.

In a ***nonrepeatable read***, a transaction repeatedly reads changing data. Ex:

1. T_1 reads data X.
2. T_2 updates X.
3. T_1 rereads X.

If T_1 incorrectly assumes the value of X is stable, the result of T_1 is invalid.

In a ***phantom read***, one transaction inserts or deletes a table row that another transaction is reading. Ex:

1. T_1 begins reading table rows.
2. T_2 inserts a new row into the table.
3. T_1 continues reading table rows.

Since T_1 sees or misses the new row, depending on precisely when T_2 writes the row to the database, the result of T_1 is unpredictable.

To ensure concurrent transactions are isolated, the concurrency system must prevent dirty reads, nonrepeatable reads, phantom reads, and other potential conflicts. Strict prevention of all conflicts, however, increases transaction duration and resource utilization. Most databases can be configured for relaxed enforcement, producing greater efficiency but occasional violations of isolation.

PARTICIPATION ACTIVITY 18.1.4: Dirty, nonrepeatable, and phantom reads.

dirty read

T_1	T_2
upgrade seat	
	read seat class write ticket cost
rollback	
	commit

nonrepeatable read

T_1	T_2
	read seat class write ticket cost
upgrade seat	
	read seat class assign boarding priority
commit	
	commit

phantom read

T_1	T_2
read first class seats	
	reserve first class seat
read economy seats write total seats commit	
	commit

Animation content:

Static figure:
Three examples of concurrent transactions appear, with captions dirty read, nonrepeatable read, and phantom read. The concurrent transactions are named T1 and T2.

The dirty read example shows actions in the following sequence:
T1: upgrade seat
T2: read seat class
T2: write ticket cost
T1: rollback
T2: commit

The nonrepeatable read example shows actions in the following sequence:
T2: read seat class
T2: write ticket cost
T1: upgrade seat
T2: read seat class
T2: assign boarding priority
T1: commit
T2: commit

The phantom read example shows actions in the following sequence:
T1: read first class seats
T2: reserve first class seat
T1: read economy seats
T1: write total seats
T1: commit
T2: commit

Step 1: After T1 upgrades the seat, T2 determines the ticket cost. However, the upgrade is rolled back, so the customer is overcharged. The dirty read example appears.

Step 2: T2 writes the ticket cost based on an economy seat. T1 later upgrades the seat, so the passenger gets first class priority. Cost and priority are inconsistent. The nonrepeatable read example appears.

Step 3: T2 reserves a first class seat after T1 reads the first class seats, so T1 writes an incorrect seat total. The phantom read example appears.

Animation captions:

1. After T_1 upgrades the seat, T_2 determines the ticket cost. However, the upgrade is rolled back, so the customer is overcharged.
2. T_2 writes the ticket cost based on an economy seat. T_1 later upgrades the seat, so the passenger gets first class priority. Cost and priority are inconsistent.
3. T_2 reserves a first class seat after T_1 reads the first class seats, so T_1 writes an incorrect seat total.

PARTICIPATION ACTIVITY | 18.1.5: Conflicting transactions.

Match the conflict type with the corresponding transactions.

If unable to drag and drop, refresh the page.

Nonrepeatable read **Phantom read** **Dirty read**

	T_1 reads salaries of some Accounting department employees T_2 transfers Maria Rodriguez from Accounting to Development T_2 commits T_1 reads salaries of remaining Accounting employees T_1 computes and writes total salary of Accounting employees T_1 commits
	T_2 increases Sam Snead's salary by 20% T_1 reads Sam Snead's salary T_2 rolls back T_1 computes and writes Sam Snead's bonus based on his salary T_1 commits
	T_1 computes total salary for the entire company

T_2 increases Sam Snead's salary by 20%
T_2 commits
T_1 computes total salary by department
T_1 writes (department total / company total) for each department
T_1 commits

Reset

External resources

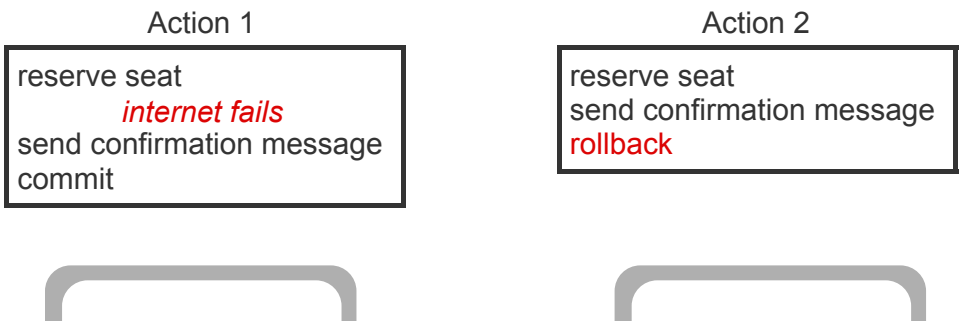
A transaction is a sequence of database operations. A transaction does not affect external resources, such as laptop computers, email systems, or files that are not managed by the database.

An ***action*** is a sequence of software operations that affect the database or external resources. Because application software has limited control over external resources, an action may violate the ACID properties.

Ex: An application executes an action that reserves a seat on a flight. The action commits the reservation in the database and then sends an email confirmation to the customer. If the internet fails, the customer does not receive the confirmation, and the database and email system have different information. The action is not consistent.

Ex: Suppose the action sends the email confirmation to a message server. The message server repeatedly sends the email until the internet is available and the email system confirms receipt. In this scenario, the action is temporarily inconsistent but eventually consistent.

PARTICIPATION ACTIVITY 18.1.6: Actions with external resources.



Animation content:

Static figure:
Two diagrams appear, with captions Action 1 and Action 2. Both diagrams show a transaction and a computer monitor.

The Action 1 diagram shows a transaction with three actions:
Reserve seat
Internet fails
Send confirmation message
Commit
The internet fails action is highlighted.

The computer monitor displays the message "no confirmation".

The Action 2 diagram shows a transaction with three actions:
Reserve seat
Send confirmation message
Rollback
The rollback action is highlighted.

The computer monitor displays the message "invalid confirmation".

Step 1: Action 1 reserves a seat, but the internet fails before a confirmation message is sent. Action 1 appears without the commit action and "no confirmation" message.

Step 2: The database changes commit, but the confirmation is not delivered. The commit action and "no confirmation" message appear.

Step 3: Action 2 reserves a seat, and the confirmation message is delivered. Action 2 appears without the rollback action. The message is "confirmation".

Step 4: The database changes are rolled back. The seat is not reserved, so the confirmation is invalid. The rollback action appears. The message changes to "invalid confirmation".

Animation captions:

Animation captions:

1. Action 1 reserves a seat, but the internet fails before a confirmation message is sent.
2. The database changes commit, but the confirmation is not delivered.
3. Action 2 reserves a seat, and the confirmation message is delivered.
4. The database changes are rolled back. The seat is not reserved, so the confirmation is invalid.

Terminology

*In informal communications, **transaction** occasionally means an **action** that affects the database or external resources. In this material, **transaction** always refers to a sequence of database operations.*

PARTICIPATION ACTIVITY 18.1.7: Actions with external resources.

A bank stores checking account data in a database. The bank stores account owner data in a file on a different computer. An action creates a new owner in the file and assigns the owner to a checking account in the database.

1) The action is always atomic.
   - ○ True
   - ○ False

2) The action is always consistent.
   - ○ True
   - ○ False

3) The action is always isolated.
   - ○ True
   - ○ False

4) The action is always durable.
   - ○ True
   - ○ False

Exploring further:

- Historical perspective on transaction management
- The classic reference book on transaction management, by Jim Gray

CHALLENGE ACTIVITY | 18.1.1: Transactions.

544874.3500394.qx3zqy7

Start

Select the ACID property violated in each example.

Pick (a) A transaction updates all accounts with a BalanceAmount of NUI written in the database, but due to a drive failure, the information

Pick (b) A transaction withdraws $25 from each account. However, a syst withdrawals for only half of the accounts to be written to the data

Pick (c) A transaction updates an account with a foreign key. The foreign values of the corresponding primary key and is not NULL.

Pick (d) Two transactions run in parallel to set an account's balance to dif

1 2

Check Next

18.2 Schedules

Schedules

Often, a database interleaves operations of multiple transactions, in order to utilize computer resources efficiently. A transaction ***schedule*** is a sequential order of operations in multiple transactions.

Schedules must preserve the original order of operations within each transaction, as the order within a transaction may affect the result. Operations in different transactions ***conflict*** when the order of the operations may affect the result. Two operations in different transactions conflict when:

- Both operations write the same data.
- One operation reads and another writes the same data.

Two read operations in different transactions never conflict, since the order of reads does not affect the result.

Equivalent schedules contain the same transactions with all conflicting operations in the same order. ***Conflicting schedules*** contain the same transactions with some conflicting operations in different order. Equivalent schedules always have the same result. Conflicting schedules can potentially have different results.

PARTICIPATION ACTIVITY 18.2.1: Equivalent and conflicting schedules.

Conflicting Schedule

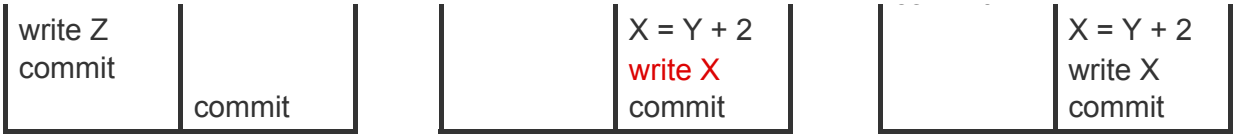
T_1	T_2
	read Y
	X = Y + 2
	write X
read X	
Z = X / 3	

Schedule

T_1	T_2
read X	
Z = X / 3	
write Z	
commit	
	read Y

Equivalent Schedule

T_1	T_2
	read Y
read X	
Z = X / 3	
write Z	
commit	



Animation content:

Static figure:
Three transaction schedules appear, with captions conflicting schedule, schedule, and equivalent schedule. Each schedule shows two concurrent transactions, named T1 and T2. Individual actions of these transactions appear sequentially.

The schedule shows actions in the following sequence:
T1: read X (highlighted)
T1: Z = X / 3
T1: write Z
T1: commit
T2: read Y (highlighted)
T2: X = Y + 2
T2: write X (highlighted)
T2: commit

The conflicting schedule shows actions in the following sequence:
T2: read Y
T2: X = Y + 2
T2: write X (highlighted)
T1: read X (highlighted)
T1: Z = X / 3
T1: write Z
T1: commit
T2: commit

The equivalent schedule shows actions in the following sequence:
T2: read Y (highlighted)
T1: read X (highlighted)
T1: Z = X / 3
T1: write Z
T1: commit
T2: X = Y + 2
T2: write X
T2: commit

Step 1: The schedule has transactions T1 and T2, each with a sequence of operations. The schedule appears without action highlights.

Step 2: Operations read X and write X conflict because the order of the read and write operations affects the final outcome. In schedule, actions T1: read X and T2: read Y are highlighted.

Step 3: In a conflicting schedule, conflicting operations are in a different order. The final value of Z may be different in the conflicting schedule. The conflicting schedule appears with highlighted actions.

Step 4: Operations read X and read Y do not conflict. In schedule, the two read actions are highlighted and the highlight for T2: write X disappears.

Step 5: In an equivalent schedule, non-conflicting operations may be in a different order, but conflicting operations are in the same order. The final value of Z is the same. The equivalent schedule appears with highlights. In schedule, the highlight for T2: write X reappears.

Animation captions:

1. The schedule has transactions T_1 and T_2, each with a sequence of operations.
2. Operations read X and write X conflict because the order of the read and write operations affects the final outcome.
3. In a conflicting schedule, conflicting operations are in a different order. The final value of Z may be different in the conflicting schedule.
4. Operations read X and read Y do not conflict.
5. In an equivalent schedule, non-conflicting operations may be in a different order, but conflicting operations are in the same order. The final value of Z is the same.

PARTICIPATION ACTIVITY 18.2.2: Equivalent and conflicting schedules.

Refer to the schedules in the above animation. Use the initial values below for each question:

X	Y	Z
9	4	0

1) After Schedule executes, what is the value of Z?

Check Show answer

2) After Equivalent Schedule executes, what is the value of Z?

Check Show answer

3) After Conflicting Schedule executes, what is the value of Z?

Check Show answer

4) The _____ Schedule has the same result as Schedule.

Check Show answer

5) The _____ Schedule has a different result than Schedule.

Check Show answer

Schedules and concurrency

A ***serial schedule*** is a schedule in which transactions are executed one at a time. Serial schedules have no concurrent transactions. Every transaction begins, executes, and commits or rolls back before the next transaction begins. All transactions in a serial schedule are isolated.

Any schedule that is equivalent to a serial schedule is a ***serializable schedule***. A serializable schedule can be transformed into a serial schedule by switching the relative order of reads in different transactions. The order of all operations within a single transaction, and reads and writes

of the same data in different transactions, cannot be changed.

Serializable schedules generate the same result as the equivalent serial schedule. Therefore, concurrent transactions in a serializable schedule are isolated.

PARTICIPATION ACTIVITY 18.2.3: Serial and serializable schedules.

Serial Schedule

T_1	T_2
read X Z = X / 3 write Z commit	
	read Y X = Y + 2 write X commit

Serializable Schedule

T_1	T_2
	read Y
read X Z = X / 3	
	X = Y + 2 write X commit
write Z commit	

Animation content:

Static figure:
Two transaction schedules appear, with captions serial schedule and serializable schedule. Each schedule shows two concurrent transactions, named T1 and T2. Individual actions of these transactions appear sequentially.

The serial schedule shows actions in the following sequence:
T1: read X (highlighted)
T1: Z = X / 3
T1: write Z
T1: commit
T2: read Y
T2: X = Y + 2
T2: write X
T2: commit

The serializable schedule shows actions in the following sequence:
T2: read Y
T1: read X (highlighted)

T1: read X (highlighted)
T1: Z = X / 3
T2: X = Y + 2
T2: write X (highlighted)
T2: commit
T1: write Z
T1: commit

Step 1: Serial schedules have no concurrent transactions. The T1 transaction commits before T2 starts. The serial schedule appears.

Step 2: The serial schedule is transformed to a new schedule by changing the order of non-conflicting operations. The serial schedule is duplicated. In the duplicate, actions move to the serializable schedule sequence. The first action and last two actions are highlighted.

Step 3: The relative order of all conflicting operations (read X and write X) is unchanged. The new schedule is equivalent to the serial schedule. In the duplicate, the serializable schedule caption appears and highlights revert to the static figure.

Animation captions:

1. Serial schedules have no concurrent transactions. The T_1 transaction commits before T_2 starts.
2. The serial schedule is transformed to a new schedule by changing the order of non-conflicting operations.
3. The relative order of all conflicting operations (read X and write X) is unchanged. The new schedule is equivalent to the serial schedule.

PARTICIPATION ACTIVITY 18.2.4: Serial and serializable schedules.

Match the schedule type to the example schedule.

If unable to drag and drop, refresh the page.

Serial schedule **Non-serializable schedule** **Serializable schedule**

	T_1	T_2
	read X	
	Y = X + 4	

write Y
commit
read X
X = X / 8
write X
commit

T_1 T_2
read X
Y = X + 4
read X
X = X / 8
write X
write Y
commit
commit

T_1 T_2
read X
X = X + 4
read X
X = X / 8
write X
write X
commit
commit

Reset

Isolation levels

Relational databases allow database administrators and application programmers to specify strict or relaxed levels of isolation for each transaction. The SQL standard defines four isolation levels:

1. ***SERIALIZABLE*** transactions run in a serializable schedule with concurrent transactions. Isolation is guaranteed.
2. ***REPEATABLE READ*** transactions read only committed data. After the transaction reads data, other transactions *cannot* update the data. REPEATABLE READ prevents most types of isolation violations but allows phantom reads.
3. ***READ COMMITTED*** transactions read only committed data. After the transaction reads data,

other transactions *can* update the data. READ COMMITTED allows nonrepeatable and phantom reads.

4. ***READ UNCOMMITTED*** transactions read uncommitted data. READ UNCOMMITTED processes concurrent transactions efficiently but allows a broad range of isolation violations, including dirty, nonrepeatable, and phantom reads.

Each successive level enables faster processing of concurrent transactions but allows more types of isolation violations. All four levels are supported by most relational databases, but implementation details vary.

Table 18.2.1: Isolation levels.

	Phantom read	Nonrepeatable read	Dirty read
SERIALIZABLE	Not allowed	Not allowed	Not allowed
REPEATABLE READ	Allowed	Not allowed	Not allowed
READ COMMITTED	Allowed	Allowed	Not allowed
READ UNCOMMITTED	Allowed	Allowed	Allowed

PARTICIPATION ACTIVITY 18.2.5: Isolation levels.

1) A SERIALIZABLE transaction can run concurrently with a READ COMMITTED transaction.
   - ○ True
   - ○ False

2) When two READ UNCOMMITTED transactions run concurrently, the result may vary.
   - ○ True

- False

3) Transactions A and B are both SERIALIZABLE, and A always starts before B. The result may vary.
   - True
   - False

Schedules and recovery

Serializable schedules affect the concurrency system, which supports isolated transactions. Three additional schedule types affect the recovery system, which supports atomic and durable transactions:

- In a ***nonrecoverable schedule***, one or more transactions cannot be rolled back.
- In a ***cascading schedule***, rollback of one transaction forces rollback of other transactions.
- In a ***strict schedule***, rollback of one transaction *never* forces rollback of other transactions.

In nonrecoverable and cascading schedules, a transaction accesses data written by another uncommitted transaction. In a strict schedule, a transaction *cannot* access data written by uncommitted transactions.

Since rollback is necessary for atomic and durable transactions, nonrecoverable schedules may violate the ACID properties. Therefore, most relational databases detect and prevent nonrecoverable schedules. Cascading schedules do not violate the ACID properties, but cascading rollbacks affect multiple transactions and degrade database performance. Therefore, many databases do not allow cascading schedules.

Most databases allow strict schedules only, which simplifies the recovery system and improves database efficiency.

PARTICIPATION ACTIVITY 18.2.6: Nonrecoverable, cascading, and strict schedules.

Nonrecoverable Schedule

T_1	T_2
write X	
	read X
	commit

Cascading Schedule

T_1	T_2
write X	
rollback	read X

Strict Schedule

T_1	T_2
write X	
commit	
	read X

~~rollback~~			*must rollback*		rollback

Animation content:

Static figure:
Three transaction schedules appear, with captions nonrecoverable schedule, cascading schedule, and strict schedule. Each schedule shows two concurrent transactions, named T1 and T2. Individual actions of these transactions appear sequentially.

The nonrecoverable schedule shows actions in the following sequence:
T1: write X
T2: read X
T2: commit
T1: rollback (highlighted and struck out)

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

The strict schedule shows actions in the following sequence:
T1: write X
T1: commit
T2: read X
T2: rollback (highlighted)

Step 1: Rolling back T1 would change data that T2 has read. Since T2 has committed, T1 cannot roll back. The nonrecoverable schedule appears.

Step 2: Rolling back T1 changes data read by T2. T2 must also rollback. The cascading schedule appears.

Step 3: T2 reads after T1 commits. Rollback of T2 does not cascade to T1. The strict schedule appears.

Animation captions:

1. Rolling back T_1 would change data that T_2 has read. Since T_2 has committed, T_1 cannot roll back.
2. Rolling back T_1 changes data read by T_2. T_2 must also rollback.
3. T_2 reads after T_1 commits. Rollback of T_2 does not cascade to T_1.

PARTICIPATION ACTIVITY | 18.2.7: Schedules and recovery.

Match the schedule type to the description.

If unable to drag and drop, refresh the page.

Cascading **Strict** **Nonrecoverable**

	Transaction A writes data and rolls back before transaction B reads the data.
	Transaction A writes data. Transaction B reads the data and commits before transaction A commits.
	Transaction A writes data. Transaction B reads the data. Transaction A rolls back before B commits.

Reset

CHALLENGE ACTIVITY | 18.2.1: Schedules.

544874.3500394.qx3zqy7

Start

Enter the value of Z after each schedule executes. Initial values: X = 4, Y = 2, Z = 0.

Schedule A

T_1	T_2
read Y	
X = Y + 4	
write X	
	read X
	Z = X * 2
	write Z
commit	commit

Z = Ex: 5

Schedule B

T_1	T_2
read Y	
	read X
	Z = X * 2
	write Z
X = Y + 4	commit
write X	
commit	

Z =

Schedule C

T_1	T_2
read X	
Z = X * 2	
write Z	
commit	
	read Y
	X = Y + 4
	write X
	commit

Z =

A and B are Pick schedules.

A and C are Pick schedules.

B and C are Pick schedules.

1

Check Try again