7.1 Transactions

This section has been set as optional by your instructor.

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 databases 'kills' the transaction permanently.

PARTICIPATION ACTIVITY 7.1.1: Commit and rollback.

1 2 3 4 5 2x speed

transaction commits

	AccountID	AccountName	BalanceAmount
initial state	A	Maria Rodriguez	11,980
	B	Sam Snead	1,000
	C	Sam Snead	2,900

transaction rolls back

AccountID	AccountName	Balanc
A	Maria Rodriguez	11,980
B	Sam Snead	1,000
C	Sam Snead	2,900

transaction

subtract $100 from account B add $100 to account C commit

subtract $100 from account B rollback ~~add $100 to account C~~

final state

AccountID	AccountName	BalanceAmount
A	Maria Rodriguez	11,980
B	Sam Snead	900
C	Sam Snead	3,000

AccountID	AccountName	Balanc
A	Maria Rodriguez	11,980
B	Sam Snead	~~900~~
C	Sam Snead	2,900

After rollback, the transaction terminates.
The final operation is not executed, so account C remains $2,900.

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.

Feedback?

PARTICIPATION ACTIVITY 7.1.2: Transactions.

1) How many SQL statements must be in one transaction?

- Exactly one
- At least one
- At least two

Correct

Although most transactions contain several SQL statements, some transactions contain only one statement.

2) After a transaction commits, the transaction can be rolled back:

- Always
- Sometimes
- Never

Correct

Rollback is possible only prior to commit. Once committed, transaction results are permanently saved in the database.

3) After a rollback, the

database restarts a transaction:

- ○ Always
- ◉ Sometimes
- ○ Never

Correct

Restarting a transaction depends on the circumstances of the rollback. The database usually restarts a transaction that is rolled back due to a system failure, but does not automatically restart a transaction that was explicitly rolled back by a user command.

Feedback?

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 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 7.1.1: Subsystem support for ACID properties.

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

Feedback?

PARTICIPATION ACTIVITY 7.1.3: ACID properties.

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

Atomic	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. This transaction writes incomplete results to the database. An atomic transaction must write complete results, or none at all.	Correct
	A transaction saves a row with a foreign key. The foreign key is not NULL and does not match	Correct

Consistent	any values of the corresponding primary key. A foreign key that does not match the corresponding primary key violates referential integrity. Referential integrity is a universal rule. The result of a consistent transaction must conform to both universal and business rules.	
Isolated	Two transactions running in parallel reserve the same seat for different passengers. Isolated transactions run as if no other transactions are running at the same time. Once one transaction reserves a seat, the seat should be unavailable to any other transaction.	**Correct**
Durable	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. The result of durable transactions is permanent. The database must ensure changes are never lost after a transaction commits.	**Correct**

Reset

Feedback?

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. The read is invalid If the second transaction either 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 7.1.4: Dirty, nonrepeatable, and phantom reads.

1 2 3 2x speed

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

$\mathbf{T_2}$ reserves a first class seat after $\mathbf{T_1}$ reads the first class seats, so $\mathbf{T_1}$ writes an incorrect seat total.

Captions

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

Feedback?

PARTICIPATION ACTIVITY 7.1.5: Conflicting transactions.

Match the conflict type with the corresponding transactions.

Phantom 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 transfers Maria Rodriguez out of Accounting while T_1 is reading Accounting employees. As a result, the total salary may be invalid. T_1 executes a phantom read.	Correct

Dirty read	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 bonus based on a salary value that is rolled back. Since T_1 reads a value that has not yet been committed, T_1 executes a dirty read.	Correct
Nonrepeatable read	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 T_1 reads salaries once to compute the company total and again to compute total by department. Sam Snead's salary changes between the two reads, so the totals are inconsistent. Since T_1 repeatedly reads changing data, T_1 executes a nonrepeatable read.	Correct

Reset

Feedback?

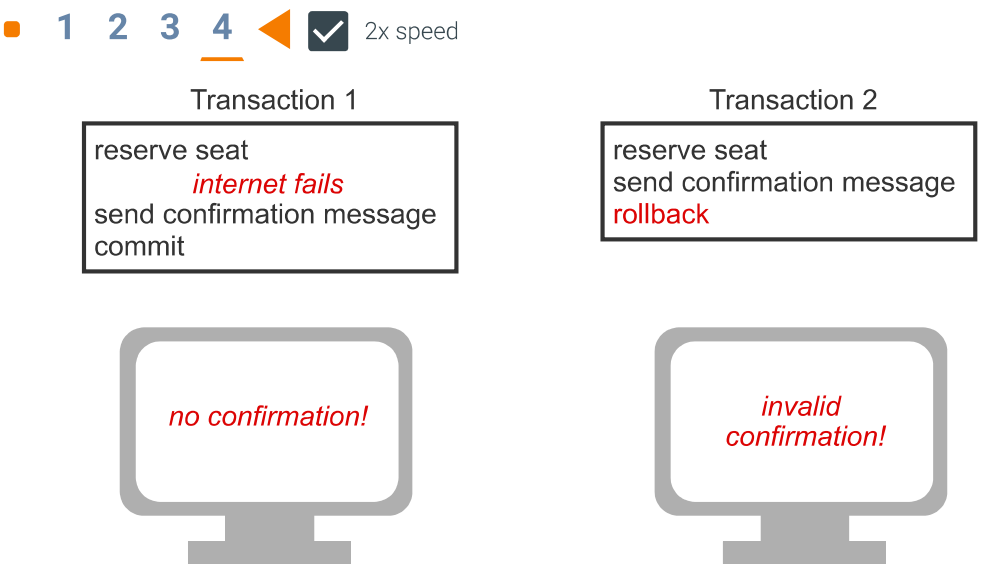
External resources

Database support for transactions extends to internal resources, specifically database reads and writes. Databases have no control over external resources, such as computer networks and laptop computers. As a result, data that is written to external resources during a transaction may violate the ACID properties.

Ex: A customer with a laptop computer connects to an airline reservation system via the internet. The customer reserves a seat on a flight. The database commits the transaction and sends confirmation to the customer, but the internet fails and the customer does not see the confirmation. In this example, the database is inconsistent with the information on the laptop computer.

Transactions involving external resources can be managed with external technology. In the seat reservation example, the application might send confirmation to a message server. The message server repeatedly sends confirmation to the laptop until the internet is available and the laptop responds. The database itself, however, cannot ensure ACID properties extend to external resources.

PARTICIPATION ACTIVITY 7.1.6: Database inconsistencies with external resources.



Transaction 2 is rolled back. The seat is not reserved, so the confirmation is invalid.

Captions

1. Transaction 1 reserves a seat, but the internet fails before a confirmation message is sent.
2. Transaction 1 commits, but the confirmation is not delivered.
3. Transaction 2 reserves a seat, and the confirmation message is delivered.
4. Transaction 2 is rolled back. The seat is not reserved, so the confirmation is invalid.

Feedback?

PARTICIPATION ACTIVITY 7.1.7: External resources.

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

1) The transaction is always atomic.
   - ○ True
   - ◉ False

Correct

If the file system fails during the transaction, the owner may be added to the account but not inserted in the file. Since incomplete results are stored, the transaction is not atomic.

2) The transaction is always consistent.
   - ○ True
   - ◉ False

Correct

If the file system fails during the transaction, the owner may be added to the account but not inserted in the file. As a result, a foreign key in the checking account table refers to an owner that does not exist in the file. The transaction violates referential integrity and thus is not consistent.

3) The transaction is always isolated.
   - ○ True
   - ◉ False

Correct

A second transaction might delete the new owner before the first transaction commits. Since owners are in an external file, the first transaction is unaware of the deletion and adds a non-existent owner to the account. The two concurrent transactions are not isolated.

4) The transaction is always durable.
   - ○ True
   - ◉ False

Correct

If the disk drive containing the file fails, information on the new owner may be lost. The transaction results are not stored permanently, so the transaction is not durable.

Feedback?

Exploring further:

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

CHALLENGE ACTIVITY 7.1.1: Transactions.

379958.2369558.qx3zqy7

Jump to level 1

1

2

Describe each set of transactions.

Set 1
T1 adds $50 to account A
T3 reads account A
T1 rolls back
T2 reads account A
T2 pays CEO 1% of account A
T2 commits
T3 increases account A by 10%
T3 commits

Set 2
T3 computes sum of account
T1 increases account A by $5
T1 commits
T3 if sum > expected, add $10
T3 commits
T2 finds largest account
T2 pays CEO 1% of largest acc
T2 commits

T1 executes without a conflict.
T2 executes without a conflict.
T3 executes a dirty read.

T1 executes without a con
T2 executes without a con
T3 executes a nonrepeatab

1 2

Check Next **Done**. Click any level to practice more. Completion is preserved.

Expected:
Set 1:
T1 executes without a conflict.
T2 executes without a conflict.
T3 executes a dirty read.
Set 2:
T1 executes without a conflict.
T2 executes without a conflict.
T3 executes a nonrepeatable read.

Set 1: T3 increases account A based on a value that is rolled back by T1. Since T3 reads subsequently rolled back, T3 executes a dirty read. T2 reads after T1's rollback.

Set 2: T3 reads account A once to compute sum and again, if sum is greater than expec

accounts. In between the two reads, T1 changes account A. Since T3 repeatedly reads executes a nonrepeatable read. T2 reads after T3 and T1 commit.

Feedback?

How was this section? Provide feedback