purpose and conclusion

# SQL – Transactions

# Cau hoi

A

B

C

D

Table CH

Id, A, B, C, D, DaD, IdBCH

they help you deal with crashes, failures, data consistency, error handling. And the list of benefits will continue to grow with time.

**What Is a Transaction?**

A transaction is a set of operations performed so all operations are guaranteed to succeed or fail as one unit.  
  
**Transaction is all or none**  
  
A common example of a transaction is the process of transferring money from a checking account to a savings account.   
  
This involves two operations:

1. Deducting money from the checking account and
2. Adding it to the savings account.

Both must succeed together and the changes must be committed to the accounts, or both must fail together and rolled back so that the accounts are maintained in a consistent state. You would at least not want money be deducted from the checking account but not added to the savings account.

By using a transaction concept, both the operations, can be guaranteed to succeed or fail together. So both accounts remain in a consistent state all the time.

**When to Use Transactions**  
You should use transactions when several operations must succeed or fail as a unit. Such as :

* In batch processing, where multiple rows must be inserted, updated, or deleted as a single unit
* Whenever a change to one table requires that other tables be kept consistent
* When modifying data in two or more databases concurrently

When you use transactions, you put locks on data that is pending for permanent change to the database. No other operations can take place on locked data until the acquired lock is released. You could lock anything from a single row up to the entire database. This is called concurrency, which means how the database handles multiple updates at one time.  
  
In the bank example above, locks will ensure that two separate transactions don't access the same accounts at the same time. If they do then either deposits or withdrawals could be lost.  
  
  
**Understanding ACID Properties**A transaction is characterized by four properties, often referred to as the ACID properties: atomicity, consistency, isolation, and durability.  
  
**Atomicity**: Either all operations of the transaction are properly reflected in the database or none are.  
  
**Consistency**: Execution of a transaction in isolation preserves the consistency of the database.

**Isolation**: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

**Durability**: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

**Syntax**

BEGIN TRANSACTION  
Marks the starting point of an explicit transaction.

COMMIT [ WORK ]  
Marks the end of an explicit or autocommit transaction. This statement causes the changes in the transaction to be permanently committed to the database.

The statement COMMIT is identical to COMMIT WORK, COMMIT TRAN, and COMMIT TRANSACTION.

ROLLBACK [ WORK ]  
Rolls back a transaction to the beginning of the transaction. No changes for the transaction are committed to the database. The statement ROLLBACK is identical to ROLLBACK WORK, ROLLBACK TRAN, and ROLLBACK TRANSACTION.

**Transaction State**

A transaction must be in one of the following states:

Active : the initial state, the transaction stays in this state while it is executing.

Partially: committed: after the final statement has been executed

Failed: after the discovery that normal execution can no longer proceed

Aborted: after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction

Committed: after successful completion

In the absence of failures, all transactions complete successfully. However, a transaction may not always complete its execution successfully. Such a transaction is termed aborted.

A transaction that completes its execution successfully is said to be committed. Figure 1-1 shows that if a transaction has been partially committed then it will be committed but only if it has not failed and if the transaction has failed, it will be aborted.

