CHAPTER 8

Understanding Transactions

For any business, transactions, which may comprise many individual operations and even other transactions, play a key role. Transactions are essential for maintaining data integrity, both for multiple related operations and when multiple users update the database concurrently.

This chapter will talk about the concepts related to transactions and how transactions can be used in SQL Server 2005 and C#.

In this chapter, we'll cover the following:

- What is a transaction?
- When to use transactions
- Understanding ACID properties
- Transaction design
- Transaction state
- Specifying transaction boundaries
- T-SQL statements allowed in a transaction
- Local transactions in SQL Server 2005
- Distributed transactions in SQL Server 2005
- Guidelines to code efficient transactions
- How to code transactions

What Is a Transaction?

A *transaction* is a set of operations performed so all operations are guaranteed to succeed or fail as one unit.

A common example of a transaction is the process of transferring money from a checking account to a savings account. This involves two operations: deducting money from the checking account and adding it to the savings account. Both must succeed together and be *committed* to the accounts, or both must fail together and be *rolled back* so that the accounts are maintained in a consistent state. Under no circumstances should money be deducted from the checking account but not added to the savings account (or vice versa)—at least you would not want this to happen with the transactions occurring with your bank accounts. By using a transaction, both the operations, namely debit and credit, 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. The following are some frequent scenarios where use of transactions is recommended:

- 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
- In distributed transactions, where data is manipulated in databases on different servers

When you use transactions, you place locks on data pending permanent change to the database. No other operations can take place on locked data until the lock is released. You could lock anything from a single row up to the whole database. This is called *concurrency*, which means how the database handles multiple updates at one time.

In the bank example, locks ensure that two separate transactions don't access the same accounts at the same time. If they did, either deposits or withdrawals could be lost.

■**Note** It's important to keep transactions pending for the shortest period of time. A lock stops others from accessing the locked database resource. Too many locks, or locks on frequently accessed resources, can seriously degrade performance.

Understanding ACID Properties

A transaction is characterized by four properties, often referred to as the *ACID properties*: atomicity, consistency, isolation, and durability.

■ **Note** The term ACID was coined by Andreas Reuter in 1983.

Atomicity: A transaction is atomic if it's regarded as a single action rather than a collection of separate operations. So, only when all the separate operations succeed does a transaction succeed and is committed to the database. On the other hand, if a single operation fails during the transaction, everything is considered to have failed and must be undone (rolled back) if it has already taken place. In the case of the order-entry system of the Northwind database, when you enter an order into the Orders and Order Details tables, data will be saved together in both tables, or it won't be saved at all.

Consistency: The transaction should leave the database in a consistent state—whether or not it completed successfully. The data modified by the transaction must comply with all the constraints placed on the columns in order to maintain data integrity. In the case of Northwind, you can't have rows in the Order Details table without a corresponding row in the Orders table, as this would leave the data in an inconsistent state.

Isolation: Every transaction has a well-defined boundary—that is, it is isolated from another transaction. One transaction shouldn't affect other transactions running at the same time. Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. A transaction sees data in the state it was in before another concurrent transaction modified it, or it sees the data after the second transaction has completed, but it doesn't see an intermediate state.

Durability: Data modifications that occur within a successful transaction are kept permanently within the system regardless of what else occurs. Transaction logs are maintained so that should a failure occur the database can be restored to its original state before the failure. As each transaction is completed, a row is entered in the database transaction log. If you have a major system failure that requires the database to be restored from a backup, you could then use this transaction log to insert (roll forward) any successful transactions that have taken place.

Every database server that offers support for transactions enforces these four ACID properties automatically.

Transaction Design

Transactions represent real-world events such as bank transactions, airline reservations, remittance of funds, and so forth.

The purpose of transaction design is to define and document the high-level characteristics of transactions required on the database system, including the following:

- Data to be used by the transaction
- Functional characteristics of the transaction
- Output of the transaction
- Importance to users
- Expected rate of usage

There are three main types of transactions:

- *Retrieval transactions*: Retrieves data from display on the screen
- *Update transactions*: Inserts new records, deletes old records, or modifies existing records in the database
- *Mixed transactions*: Involves both retrieval and updating of data

Transaction State

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 8-1 shows that if a transaction has been partially committed, it will be committed but only if it has not failed; and if the transaction has failed, it will be aborted.

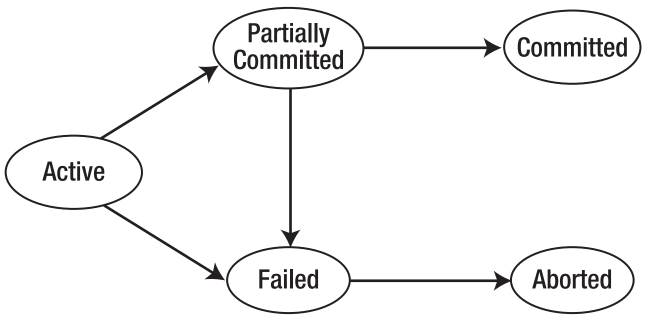

Figure 8-1. *States of a transaction*

Specifying Transaction Boundaries

SQL Server transaction boundaries help you to identify when SQL Server transactions start and end by using API functions and methods:

- *Transact-SQL statements*: Use the `BEGIN TRANSACTION`, `COMMIT TRANSACTION`, `COMMIT WORK`, `ROLLBACK TRANSACTION`, `ROLLBACK WORK`, and `SET IMPLICIT_TRANSACTIONS` statements to delineate transactions. These are primarily used in DB-Library applications and in T-SQL scripts, such as the scripts that are run using the `osql` command-prompt utility.

- *API functions and methods*: Database APIs such as ODBC, OLE DB, ADO, and the .NET Framework `SQLClient` namespace contain functions or methods used to delineate transactions. These are the primary mechanisms used to control transactions in a database engine application.

Each transaction must be managed by only one of these methods. Using both methods on the same transaction can lead to undefined results. For example, you should not start a transaction using the ODBC API functions, and then use the T-SQL `COMMIT` statement to complete the transaction. This would not notify the SQL Server ODBC driver that the transaction was committed. In this case, use the ODBC `SQLEndTran` function to end the transaction.

T-SQL Statements Allowed in a Transaction

You can use all T-SQL statements in a transaction, except for the following statements: `ALTER DATABASE`, `RECONFIGURE`, `BACKUP`, `RESTORE`, `CREATE DATABASE`, `UPDATE STATISTICS`, and `DROP DATABASE`.

Also, you cannot use `sp_dboption` to set database options or use any system procedures that modify the master database inside explicit or implicit transactions.

Local Transactions in SQL Server 2005

All database engines are supposed to provide built-in support for transactions. Transactions that are restricted to only a single resource or database are known as *local transactions*. Local transactions can be in one of the following four transaction modes: