Transactions

A *transaction* is a unit of work that might include multiple activities that query and modify data and that can also change the data definition.

You define the beginning of a transaction explicitly with a *BEGIN TRAN* (or *BEGIN TRANSACTION*) statement. You define the end of a transaction explicitly with a *COMMIT TRAN* statement if you want to commit it

ROLLBACK TRAN (or *ROLLBACK TRANSACTION*) statement if you want to undo its changes. Here's an example of marking the boundaries of a transaction with two *INSERT* statements:

```
BEGIN TRAN;
INSERT INTO dbo.T1(keycol, col1, col2) VALUES(4, 101, 'C');
INSERT INTO dbo.T2(keycol, col1, col2) VALUES(4, 201, 'X');
COMMIT TRAN;
```

If you do not mark the boundaries of a transaction explicitly, by default, SQL Server treats each individual statement as a transaction; in other words, by default, SQL Server automatically commits the transaction at the end of each statement

Transactions have **FOUR** properties—atomicity, consistency, isolation, and durability—abbreviated with the acronym *ACID*:

Atomicity A transaction is an atomic unit of work. Either all changes in the transaction take place or none do. If the system fails before a transaction is completed (before the commit instruction is recorded in the transaction log), upon restart, SQL Server undoes the changes that took place. Also, if errors are encountered during the transaction and the error is considered severe enough, such as the target filegroup being full when you try to insert data, SQL Server automatically rolls back the transaction.

Consistency The term *consistency* refers to the state of the data that the relational database management system (RDBMS) gives you access to as concurrent transactions modify and query it. As you can probably imagine, consistency is a subjective term, which depends on your application's needs. The "Isolation levels" section later in this chapter explains the level of consistency that SQL Server provides by default and how you can control it if the default behavior is not suitable for your application. Consistency also refers to the fact that the data- base must adhere to all integrity rules that have been defined within it by constraints (such as primary keys, unique constraints, and foreign keys). The transaction transitions the database from one consistent state to another.

Isolation *Isolation* ensures that transactions access only consistent data. You control what consistency means to your transactions through a mechanism called *isolation levels*. With disk-based tables, SQL Server supports two different models to handle isolation: one based purely on locking, and another based on a combination of locking and row versioning. For simplicity, I'll refer to the latter as just *row versioning*. The model based on locking is the default in a box product. In this model, readers require shared locks. If the current state of the data is inconsistent, readers are blocked until the state of the data becomes consistent. The model based on row versioning is the default in Azure SQL Database. In this model, readers don't take shared locks and don’t need to wait. If the current state of the data is inconsistent, the reader gets an older consistent state. The “Isolation levels” section later in this chapter provides more details about both ways of handling isolation.

Durability Data changes are always written to the database’s transaction log on disk before they are written to the data portion of the database on disk. After the commit instruction is recorded in the transaction log on disk, the transaction is considered durable even if the change hasn’t yet made it to the data portion on disk. When the system starts, either normally or after a system failure, SQL Server inspects the transaction log of each database and runs a recovery process with two phases: redo and undo. The redo phase involves rolling forward (replaying) all the changes from any transaction whose commit instruction is written to the log but whose changes haven’t yet made it to the data portion. The undo phase involves rolling back (undoing) the changes from any transaction whose commit instruction was not recorded in the log.