Instructor Notes:

Add instructor notes here.

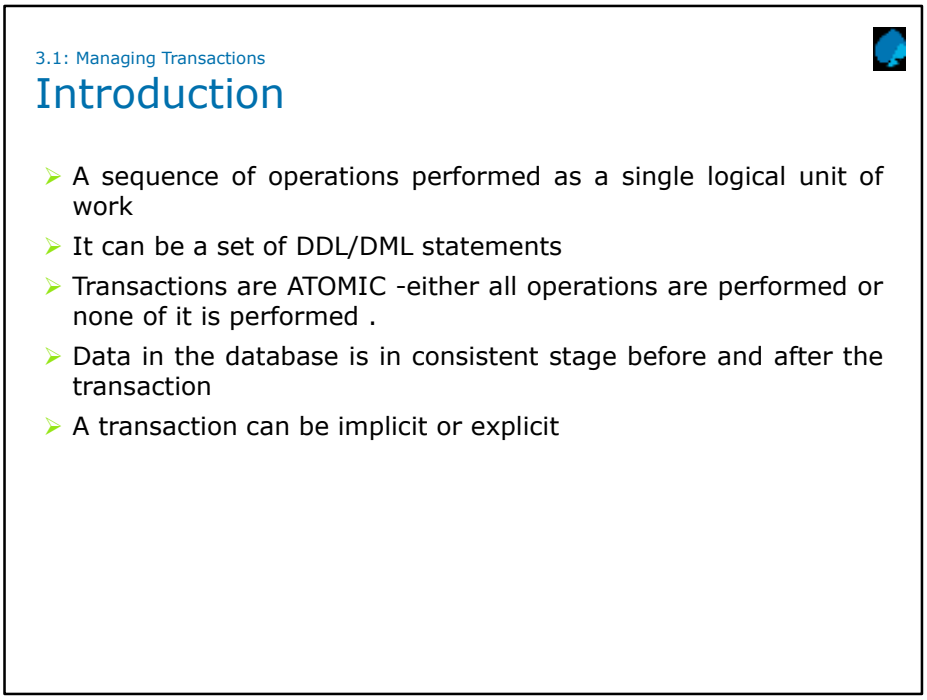
Instructor Notes:

Explain the lesson coverage

Lesson Objectives

- In this lesson, you will learn:
  - Managing Transactions
  - TCL statements
  - @@trancount global variable

Instructor Notes:



What is transaction?
Like a batch a user-declared transaction typically consists of several SQL commands that read and update the database. However, a batch is basically a client-side concept; it controls how many statements are sent to SQL Server for processing at once. A transaction, on the other hand, is a server-side concept. It deals with how much work SQL Server will do before it considers the changes committed. A multistatement transaction doesn't make any permanent changes in a database until a COMMIT TRANSACTION statement is issued. In addition, a multistatement transaction can undo its changes when a ROLLBACK TRANSACTION statement is issued.

Now let's look at a simple transaction in a little more detail. The BEGIN TRAN and COMMIT TRAN statements cause the commands between them to be performed as a unit:

```
BEGIN TRAN
INSERT authors VALUES (etc.)
 SELECT * FROM authors
 UPDATE publishers
    SET pub_id = (etc.)
 COMMIT TRAN
GO
```

Instructor Notes:

3.1: Managing Transactions

Transaction

- A transaction can have the following outcome
  - COMMIT : Changes made on the data are made permananent
  - ROLLBACK : Undo the transaction , data goes back to the original state
- Implicit Transactions automatically starts a new transaction after the current transaction is committed /rolled back .Does not require explicit mention of start /end of transaction
- Explicit transaction requires defining the beginning and end of the transactions

Transaction processing in SQL Server ensures that all commands within a transaction are performed as a unit of work
when we talk about transactions, we're generally talking about data modification statements (INSERT, UPDATE, and DELETE). So any individual data modification statement by itself is an *implicit transaction*.

```
USE PUBS
BEGIN TRAN
        UPDATE authors
        SET state = 'FL'
        WHERE state = 'KS‘
        IF @@ERROR <> 0
        BEGIN
                ROLLBACK TRAN
                GOTO ON_ERROR
        END
        UPDATE jobs
        SET  min_lvl = min_lvl - 10
        IF @@ERROR <> 0
        BEGIN
                ROLLBACK TRAN
                GOTO ON_ERROR
        END
COMMIT TRAN
ON_ERROR:
SELECT * FROM authors
WHERE state = 'FL'
```

Instructor Notes:

Commit or Rollback ends a Transaction

3.1: Managing Transactions

Transaction

- Implicit Transaction

```
Insert into Employees values
(…)
Update employees
Set …..
```

- Explicit Transaction

```
BEGIN TRAN
insert into Employees (..)
Update Employees Set …
COMMIT [WORK ]
```

Instructor Notes:

3.1: Managing Transactions

Setting the Implicit Transactions Option

- Automatically starts a Transaction when you execute certain statements
- Nested Transactions are not allowed
- Transaction must be explicitly completed with COMMIT or ROLLBACK TRANSACTION
- By default, setting is Off

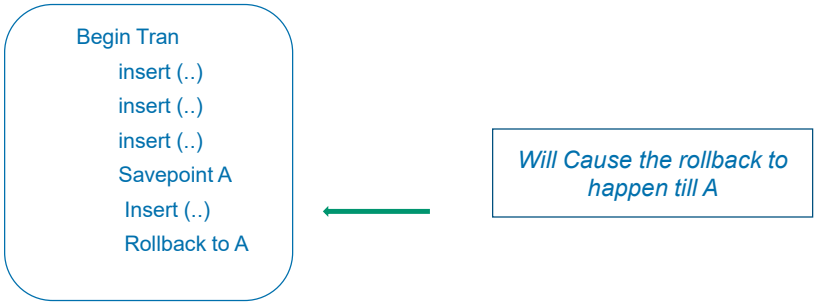
```
SET IMPLICIT_TRANSACTIONS ON
```

Instructor Notes:

3.1: Managing Transactions

Save Point

- A savepoint is a special mark inside a transaction that allows all commands that are executed after it to be rolled back to that point
- For example



If only Rollback is given , the entire transaction will be rolled back

Begin Tran
insert (..)
insert (..)
insert (..)
Savepoint A
Insert (..)
Rollback
Will cause the entire transaction to be rolled back

Instructor Notes:

3.1: Managing Transactions

Restrictions on User-defined /Explicit Transactions

- Certain Statements May Not Be Included
  - ALTER DATABASE
  - BACKUP LOG
  - CREATE DATABASE
  - DROP DATABASE
  - RECONFIGURE
  - RESTORE DATABASE
  - RESTORE LOG
  - UPDATE STATISTICS

Instructor Notes:

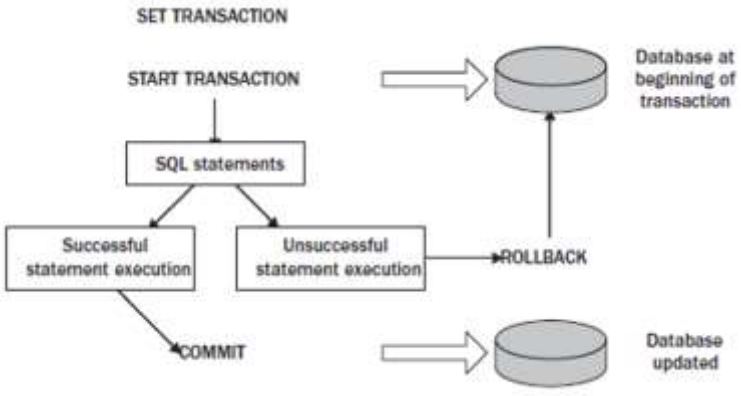
3.1: Managing Transactions

Considerations for Using Transactions

- Transaction Guidelines
  - Keep transactions as short as possible
  - Avoid transactions that require user interaction
- Issues in Nesting Transactions
  - Allowed, but not recommended
  - Use @@trancount to determine nesting level

Instructor Notes:

SQL Server Transaction

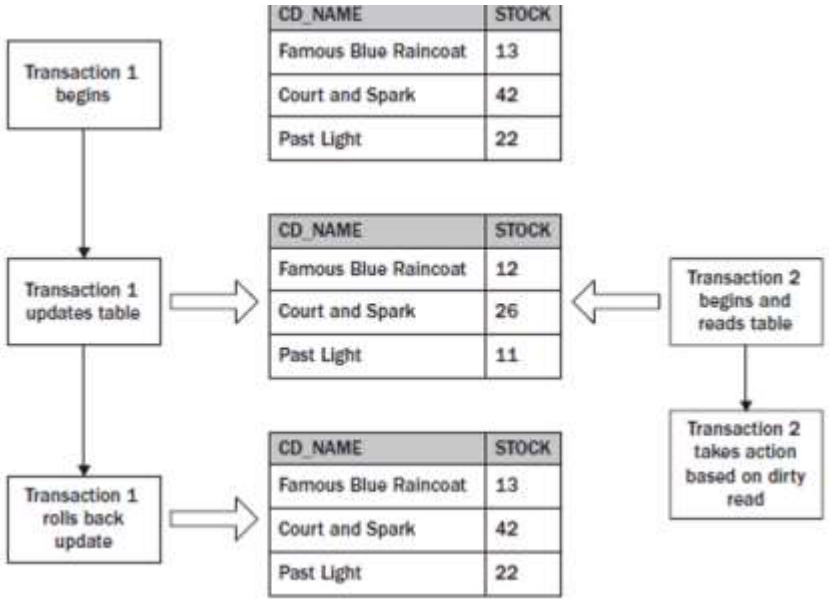


Instructor Notes:

SQL Server Transaction Concurrency Issue

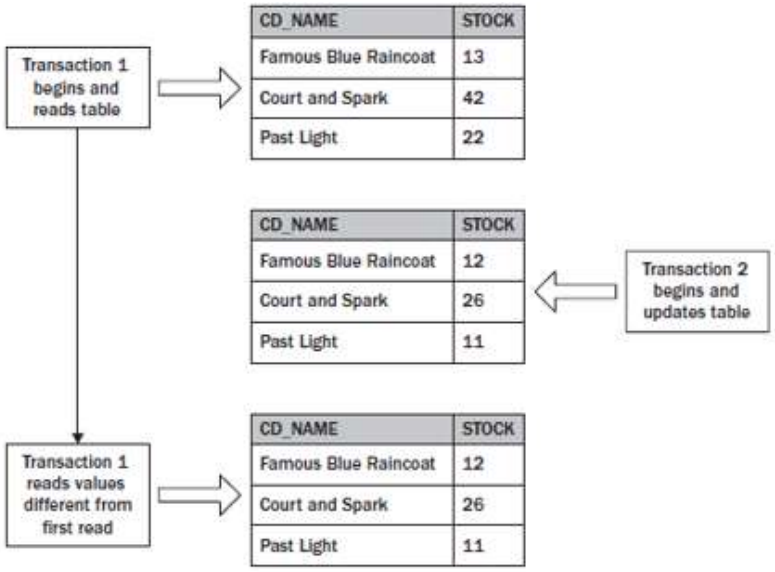
- Concurrency is the capability of the machine to support two or more transactions working with the same data at the same time.
- This usually comes up with data is being modified, as during the retrieval of the data this is not the issue.
- Most of the concurrency problems can be avoided by SQL Locks.
- There are four types of concurrency problems visible in the normal programming.
  - Lost Update
  - Dirty Read
  - NonRepeatable Read
  - Phantom Read

- Lost Update
  A lost update is considered to have taken place when data that has been updated by one transaction is overwritten by another transaction, before the first transaction is either committed or rolled back.
- Dirty Read
  If data that has been changed by an open transaction is accessed by another transaction, a dirty read has taken place. A dirty read can cause problems because it means that a data manipulation language (DML) statement accessed data that logically does not exist yet or will never exist.

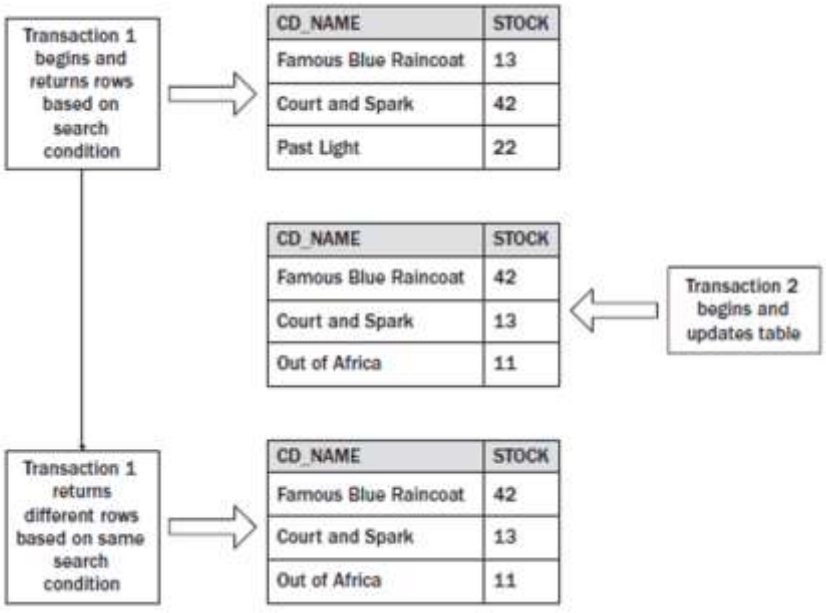


Instructor Notes:

- NonRepeatable Read
  If a specific set of data is accessed more than once in the same transaction (such as when two different queries against the same table use the same WHERE clause) and the rows accessed between these accesses are updated or deleted by another transaction, a non-repeatable read has taken place. That is, if two queries against the same table with the same WHERE clause are executed in the same transaction, they return different results.



- Phantom Read
  Phantom reads are a variation of non-repeatable reads. A phantom read is when two queries in the same transaction, against the same table, use the same WHERE clause, and the query executed last returns more rows than the first query.



Instructor Notes:

SQL Server Transaction Isolation Levels

- Transaction isolation level controls the locking and row versioning behavior of Transact-SQL statements issued by a connection to SQL Server.
- Following are the different types of isolation levels available in SQL Server.
  - READ COMMITTED
  - READ UNCOMMITED
  - REPEATABLE READ
  - SERIALIZABLE
  - SNAPSHOT
- Syntax
  - SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
- Only one of the isolation level options can be set at a time, and it remains set for that connection until it is explicitly changed.
- The transaction isolation levels define the type of locks acquired on read operations

- READ COMMITTED
  Specifies that statements cannot read data that has been modified but not committed by other transactions. This prevents dirty reads. Data can be changed by other transactions between individual statements within the current transaction, resulting in nonrepeatable reads or phantom data. This option is the SQL Server default.
  The behavior of READ COMMITTED depends on the setting of the READ_COMMITTED_SNAPSHOT database option:
  - If READ_COMMITTED_SNAPSHOT is set to OFF (the default), the Database Engine uses shared locks to prevent other transactions from modifying rows while the current transaction is running a read operation.
  - If READ_COMMITTED_SNAPSHOT is set to ON, the Database Engine uses row versioning to present each statement with a transactionally consistent snapshot of the data as it existed at the start of the statement.
- READ UNCOMMITED
  Specifies that statements can read rows that have been modified by other transactions but not yet committed.
  Transactions running at the READ UNCOMMITTED level do not issue shared locks to prevent other transactions from modifying data read by the current transaction. READ UNCOMMITTED transactions are also not blocked by exclusive locks that would prevent the current transaction from reading rows that have been modified but not committed by other transactions. When this option is set, it is possible to read uncommitted modifications, which are called dirty reads. Values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction.

Instructor Notes:

SQL Server Transaction Isolation Levels (Contd…)

Isolation	Dirty Read	Lost Update	NonRepeatable Read	Phantom Read
READ COMMITTED	No	Yes	Yes	Yes
READ UNCOMMITTED	Yes	Yes	Yes	Yes
REPEATABLE READ	No	No	No	Yes
SERIALIZABLE	No	No	No	No
SNAPSHOT	No	No	No	No

- REPEATABLE READ
  Specifies that statements cannot read data that has been modified but not yet committed by other transactions and that no other transactions can modify data that has been read by the current transaction until the current transaction completes.
  Shared locks are placed on all data read by each statement in the transaction and are held until the transaction completes. This prevents other transactions from modifying any rows that have been read by the current transaction. Other transactions can insert new rows that match the search conditions of statements issued by the current transaction. If the current transaction then retries the statement it will retrieve the new rows, which results in phantom reads.
- SERIALIZABLE
  Specifies the following:
  - Statements cannot read data that has been modified but not yet committed by other transactions.
  - No other transactions can modify data that has been read by the current transaction until the current transaction completes.
  - Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes.
- SNAPSHOT
  Specifies that data read by any statement in a transaction will be the transactionally consistent version of the data that existed at the start of the transaction. The transaction can only recognize data modifications that were committed before the start of the transaction. Data modifications

Instru

made by other transactions after the start of the current transaction are not visible to statements executing in the current transaction. The effect is as if the statements in a transaction get a snapshot of the committed data as it existed at the start of the transaction.

Instructor Notes:

Demo

- Managing Transactions

Instructor Notes:

None

Summary

- In this lesson, you have learnt:
  - How to write explicit transactions.
  - TCL statements

Instructor Notes:

Answers for the Review Questions:

Answer 1: True
Answer 2: Transaction
Answer 3: Explicit transaction

Review Question

- Question 1: ---------- is a sequence of operations performed as a single logical unit of work.
- Question 2: ---------- transaction requires defining the beginning and end of the transactions.