MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 15: Managing Transactions and Locks

Contents

Microsoft

Project Lead: Rich Rose
Instructional Designers: Rich Rose, Cheryl Hoople, Marilyn McGill
Instructional Software Design Engineers: Karl Dehmer, Carl Raebler, Rick Byham
Technical Lead: Karl Dehmer
Subject Matter Experts: Karl Dehmer, Carl Raebler, Rick Byham
Graphic Artist: Kirsten Larson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Edward McKillop (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: TestingTesting123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Data Base: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
45 Minutes

Lab:
60 Minutes

This module provides students with information about how transactions and locks are used to ensure transaction integrity while allowing for concurrent use. The module continues with a discussion of how transactions are executed and rolled back. A short animation helps to convey how transaction processing works.

The module next describes how Microsoft® SQL Server™ 2000 locks maintain data consistency and concurrency. Resources that can be locked, the different types of locks, and lock compatibility are introduced. The final section describes some locking options, discusses deadlocks, and explains how to display information on active locks.

In this lab, students define a transaction and observe the impact of BEGIN TRAN, COMMIT TRAN, and ROLLBACK TRAN statements. They then observe the effect of applying different locking options to a transaction.

After completing this module, the students will be able to:

- Describe transaction processing.
- Execute, cancel, or roll back a transaction.
- Identify locking concurrency issues.
- Identify resource items that can be locked and the types of locks.
- Describe lock compatibility.
- Describe how SQL Server uses dynamic locking.
- Set locking options and display locking information.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2073a__15.ppt
- The C:\Moc\2073A\Demo\D15_Ex.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.
- Practice the presentation, including the animated slide.
- Review any relevant white papers located on the Trainer Materials compact disc.

Multimedia Presentation

This section provides multimedia presentation procedures that do not fit in the margin notes and are not appropriate for the student notes.

SQL Server Transactions

► **To prepare for the multimedia presentation**

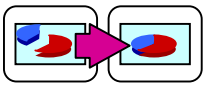
- Click the button in the slide to start the multimedia presentation.

  This multimedia presentation introduces SQL Server transaction processing. It starts with the definition of a transaction and explains the two types of transactions, implicit and explicit. Then it explains how it is possible to cancel or roll back a transaction.

  The presentation continues with a description of the transaction log that maintains database consistency, explains how modifications are recorded in the log on disk before they are written to the database, and describes how checkpoints in the log indicate which transactions have been applied to the database. The presentation concludes with a description of the automatic recovery process.

Other Activities

This section provides procedures for implementing interactive activities to present or review information, such as games or role playing exercises.



Displaying the Animated PowerPoint Slide

The animated slide is identified with an icon of links on the lower left corner of the slide.

► **To display the Transaction Recovery and Checkpoints slide**

This slide shows how SQL Server can easily recover transactions in the event of a system failure.

1. Display the topic slide where the actions required for transaction recovery appear.
2. Advance to the first animation where the Transaction 1 arrow ends before the checkpoint. No action is required because the transaction is reflected in the database.
3. Advance to the next animation where the Transaction 2 arrow begins before the checkpoint and completes after it. The transaction must be rolled forward because part of the transaction occurred after the checkpoint.
4. Advance to the next animation where the Transaction 3 arrow begins before the checkpoint and does not complete before system failure. It must be rolled back.
5. Advance to the next animation where the Transaction 4 arrow commits after the checkpoint. It must be reconstructed from the log (rolled forward).
6. Advance to the final animation where the Transaction 5 arrow starts after the checkpoint but does not complete before system failure. It must be rolled back.

Module Strategy

Use the following strategy to present this module:

- Introduction to Transactions and Locks

  Introduce the terms and concepts that are used in the module. Be sure that students understand the interrelationship of the concepts.

- Managing Transactions

  Play the multimedia presentation. Describe how to define a transaction, restrictions on transactions, how SQL Server processes transactions, transaction recovery, and checkpoints. Make sure that students understand transactions and how a transaction is used in automatic recovery.

- SQL Server Locking

  Describe locking concurrency issues, lockable resources, types of locks, and lock compatibility. Emphasize the problems that can arise due to lack of locking and that SQL Server automatically handles these situations. Caution students that changing locking options in their transactions can negatively impact performance.

- Managing Locks

  Describe session-level and table-level locking options, explain deadlocks, and describe how to display information on locks. Emphasize that students should design transactions to minimize deadlocks. They should also test to determine whether a deadlock is occurring in a transaction.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Lab Setup

There are no lab setup requirements that affect replication or customization.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, we'll cover...

- **Introduction to Transactions and Locks**
- **Managing Transactions**
- **SQL Server Locking**
- **Managing Locks**

Objectives

After completing this module, you will be able to:

- Describe transaction processing.
- Execute, cancel, or roll back a transaction.
- Identify locking concurrency issues.
- Identify resource items that can be locked and the types of locks.
- Describe lock compatibility.
- Describe how Microsoft® SQL Server™ 2000 uses dynamic locking.
- Set locking options and display locking information.

Introduction to Transactions and Locks

Topic Objective
To provide an introduction to this topic.

Lead-in
Transactions and locks ensure transaction integrity.

- **Transactions Ensure That Multiple Data Modifications Are Processed Together**
- **Locks Prevent Update Conflicts**
  - Transactions are serializable
  - Locking is automatic
  - Locks allow concurrent use of data
- **Concurrency Control**

Delivery Tip
This module focuses on online transaction processing rather than query-intensive applications, such as data warehousing and decision support.

Transactions use locking to prevent other users from changing or reading data in a transaction that has not completed. Locking is required in online transaction processing (OLTP) for multiuser systems. SQL Server uses the transaction log to ensure that updates are complete and recoverable.

Transactions

Transactions ensure that multiple data modifications are processed as a unit; this is known as *atomicity*. For example, a banking transaction might credit one account and debit another. Both steps must be completed together. SQL Server supports transaction processing to manage multiple transactions.

Locks

Locks prevent update conflicts. Users cannot read or modify data that other users are in the process of changing. For example, if you want to compute an aggregate and ensure that another transaction does not modify the set of data that is used to compute the aggregate, you can request that the system hold locks on the data. Consider the following facts about locks:

- Locks make possible the serialization of transactions so that only one person at a time can change a data element. For example, locks in an airline reservation system ensure that only one person is assigned a particular seat.
- SQL Server dynamically sets and adjusts the appropriate level of locking during a transaction. It also is possible to manually control how some of the locks are used.
- Locks are necessary for concurrent transactions to allow users to access and update data at the same time. High concurrency means that there are a number of users who are experiencing good response time with little conflict. From the system administrator's perspective, the primary concerns are the number of users, the number of transactions, and the throughput. From the user's perspective, the overriding concern is response time.

Concurrency Control

Concurrency control ensures that modifications that one person makes do not adversely affect modifications that others make. There are two types.

- Pessimistic concurrency control locks data when data is read in preparation for an update. Other users cannot then perform actions that would alter the underlying data until the user who applied the lock is done with the data. Use pessimistic concurrency where high contention for data exists and the cost of protecting the data with locks is less than the cost of rolling back transactions if concurrency conflicts occur.
- Optimistic concurrency control does not lock data when data is initially read. Rather, when an update is performed, SQL Server checks to determine whether the underlying data was changed since it initially read it. If so, the user receives an error, the transaction rolls back, and the user must start over. Use optimistic concurrency when low contention for data exists and the cost of occasionally rolling back a transaction is less than the cost of locking data when it is read.

SQL Server supports a wide range of optimistic and pessimistic concurrency control mechanisms. Users specify the type of concurrency control by specifying the transaction isolation level for a connection.

◆ Managing Transactions

Topic Objective
To provide an overview of this topic.

Lead-in
In this section, we'll cover…

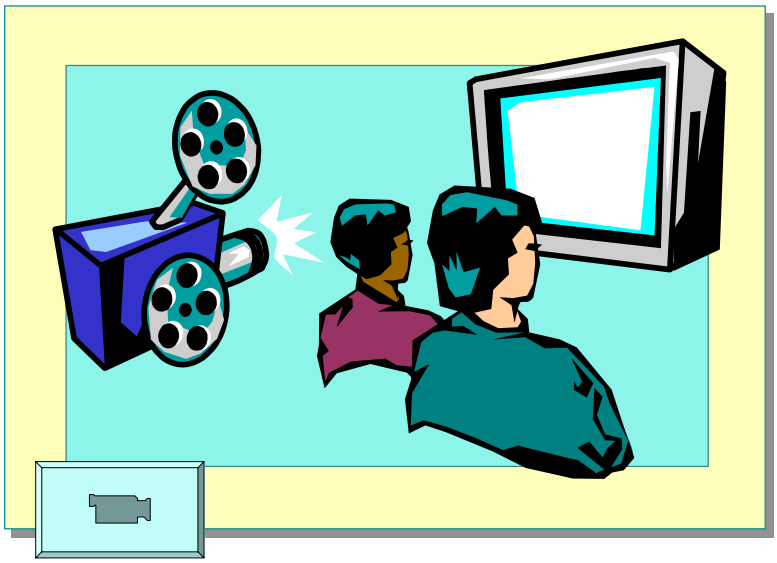
- **Multimedia Presentation: SQL Server Transactions**
- **Transaction Recovery and Checkpoints**
- **Considerations for Using Transactions**
- **Setting the Implicit Transactions Option**
- **Restrictions on User-defined Transactions**

This section describes how to define transactions, what to consider when you use them, how to set an implicit transaction option, and the restrictions on using transactions. It also discusses transaction processing and recovery.

Multimedia Presentation: SQL Server Transactions

Topic Objective
To view an example of transaction processing.

Lead-in
In this presentation, you will see how transactions are processed and how the transaction log works.



Key Point
A committed transaction cannot be undone or rolled back.

There are two kinds of transactions in SQL Server:

- In an implicit transaction, each Transact-SQL statement, such as INSERT, UPDATE, or DELETE, executes as a transaction.
- In an explicit or user-defined transaction, the statements of the transaction are grouped between a BEGIN TRANSACTION and a COMMIT TRANSACTION clause.

A user can set a savepoint, or marker, within a transaction. A savepoint defines a location to which a transaction can return if part of a transaction is conditionally cancelled. The transaction must then proceed to completion or be rolled back entirely.

SQL Server transactions employ the following syntax.

Syntax

```
BEGIN TRAN[SACTION] [transaction_name | @tran_name_variable [WITH
  MARK ['description']]]
```

The *transaction_name* option specifies a user-defined transaction name. The *tran_name_variable* is the name of a user-defined variable containing a valid transaction name. WITH MARK specifies that the transaction is marked in the transaction log. *Description* is a string that describes the mark that WITH MARK allows for restoring a transaction log to a named mark.

Syntax

```
SAVE TRAN[SACTION] {savepoint_name | @savepoint_variable}
```

Syntax

```
BEGIN DISTRIBUTED TRAN[SACTION]
  [transaction_name | @tran_name_variable]
```

Syntax

```
COMMIT [TRAN[SACTION] [transaction_name | @tran_name_variable]]
```

Syntax

ROLLBACK [TRAN[SACTION] [*transaction_name* | *@tran_name_variable* | *savepoint_name* | *@savepoint_variable*]]

Example

This example defines a transaction to transfer funds between the checking and savings accounts of one customer.

This example cannot be executed because the stored procedures do not exist.

```
BEGIN TRAN fund_transfer
   EXEC debit_checking 100, 'account1'
   EXEC credit_savings 100, 'account1'
COMMIT TRAN fund_transfer
```

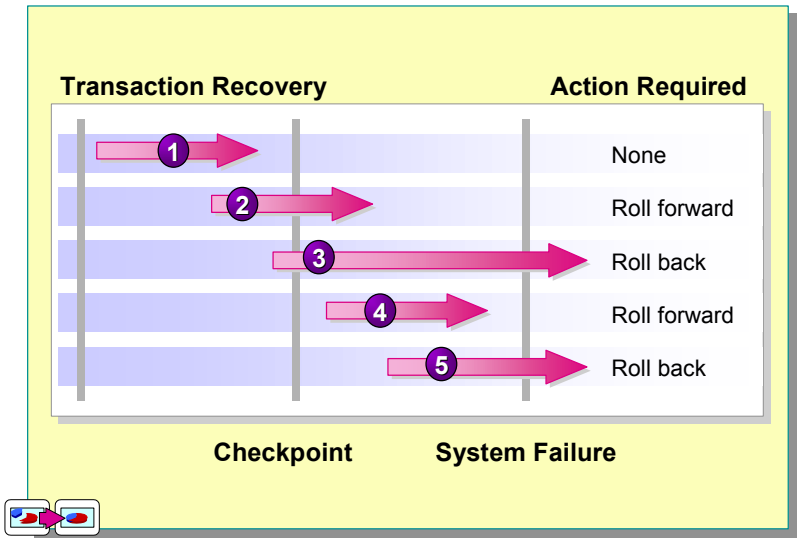
Describing the Transaction Log

Every transaction is recorded in a transaction log to maintain database consistency and to aid in recovery. The log is a storage area that automatically tracks all changes to a database, with the exception of non-logged operations. Modifications are recorded in the log on disk as they are executed, before they are written in the database.

Transaction Recovery and Checkpoints

Topic Objective
To explain the recovery process.

Lead-in
Because the transaction log records all modifications, data can be easily recovered in the event of system failure.



Because the transaction log records all transactions, SQL Server can recover data automatically in the event of a power loss, system software failure, client problem, or a transaction cancellation request.

Delivery Tip
Use the slide as a discussion point. Ask students why each action is required.

SQL Server automatically guarantees that all committed transactions are reflected in the database in the event of a failure. It uses the transaction log to roll forward all committed transactions and to roll back any uncommitted transactions. In the slide example:

- Transaction 1 is committed before the checkpoint, so it is reflected in the database.
- Transactions 2 and 4 were committed after the checkpoint, so they must be reconstructed from the log (rolled forward).
- Transactions 3 and 5 were not committed, so SQL Server rolls them back.

Initially, pages are the same in the data cache and on the disk. The following process then occurs:

- Changes appear in the data cache as transactions are committed.
- As the cache becomes full, the changed pages are written to disk.
- When a checkpoint occurs, the cache is written to disk. The disk is once again the same as the cache.

Important Use a write-caching disk controller with SQL Server only if it was designed for use with a database server. Failure to do so compromises the ability of SQL Server to manage transactions. A write-caching disk controller can make it appear that write-ahead logging is complete, even when it has not.

Considerations for Using Transactions

Topic Objective
To identify issues in using transactions.

Lead-in
In general, keep transactions as short as possible.

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

It is usually a good idea to keep transactions short and to avoid nesting transactions.

Transaction Guidelines

Transactions should be as short as possible. Longer transactions increase the likelihood that users will not be able to access locked data. Some methods to keep transactions short include the following:

- To minimize transaction time, use caution when you use certain Transact-SQL statements, such as a WHILE statement or Data Definition Language (DDL) statements.
- Do not require input from users during a transaction. Address issues that require user interaction before you start the transaction. For example, if you are updating a customer record, obtain the necessary information from the user before you begin the transaction.
- INSERT, UPDATE, and DELETE should be the primary statements in a transaction, and they should be written to affect the fewest number of rows. A transaction should never be smaller than a logical unit of work.
- Do not open a transaction while browsing through data, if at all possible. Transactions should not start until all preliminary data analysis has been completed.
- Access the least amount of data possible while in a transaction. This decreases the number of locked rows and reduces contention.

Issues in Nesting Transactions

Consider the following issues regarding nesting transactions:

- It is possible to nest transactions, but nesting does not affect how SQL Server processes the transaction. You should use nesting carefully, if at all, because the failure to commit or roll back a transaction leaves locks in place indefinitely.

  Only the outermost BEGIN…COMMIT statement pair applies. Usually, transaction nesting occurs when stored procedures with BEGIN...COMMIT statement pairs or triggers invoke one another.

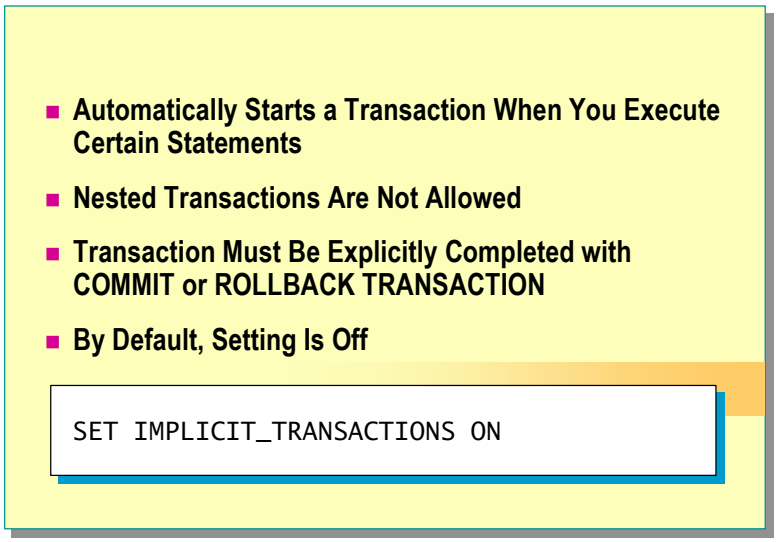
- You can use the **@@trancount** global variable to determine whether any open transactions exist and how deeply they are nested:
  - **@@trancount** equals zero when no open transactions exist.
  - A BEGIN TRAN statement increments **@@trancount** by one, and a ROLLBACK TRAN statement sets **@@trancount** to zero.

Note You also can use the DBCC OPENTRAN statement within your current session to retrieve information on active transactions.

Setting the Implicit Transactions Option

Topic Objective
To discuss how to set implicit transactions.

Lead-in
Setting implicit transactions can be useful when you migrate applications to SQL Server.



In most cases, it is best to define transactions explicitly with the BEGIN TRANSACTION statement. However, for applications that were originally developed on systems other than SQL Server, the SET IMPLICIT_TRANSACTIONS option can be useful. It sets the implicit transaction mode for a connection.

Syntax

SET IMPLICIT_TRANSACTIONS {ON | OFF}

Consider the following facts when you set implicit transactions:

- When the implicit transaction mode for a connection is on, executing any of the following statements triggers the start of a transaction:

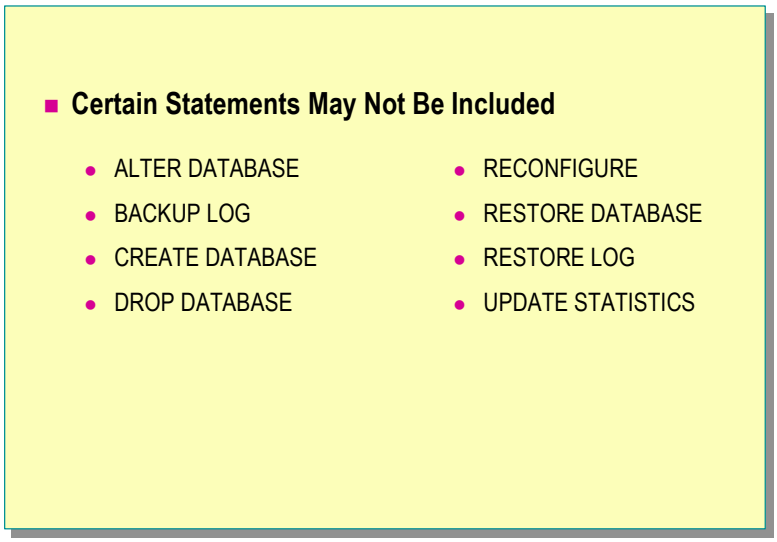
ALTER TABLE	INSERT
CREATE	OPEN
DELETE	REVOKE
DROP	SELECT
FETCH	TRUNCATE TABLE
GRANT	UPDATE

- Nested transactions are not allowed. If the connection is already in an open transaction, the statements do not start a new transaction.
- When the setting is on, the user must commit or roll back the transaction explicitly at the end of the transaction. Otherwise, the transaction and all data changes that it contains are rolled back when the user disconnects.
- The setting is off by default.

Restrictions on User-defined Transactions

Topic Objective
To indicate the items that cannot be used in user-defined transactions.

Lead-in
There are some restrictions on user-defined transactions.



Some restrictions exist on user-defined transactions:

- Certain statements may not be included inside an explicit transaction. For example, some are long-running operations that you are not likely to use within the context of a transaction. Restricted statements include the following statements:
  - ALTER DATABASE
  - BACKUP LOG
  - CREATE DATABASE
  - DROP DATABASE
  - RECONFIGURE
  - RESTORE DATABASE
  - RESTORE LOG
  - UPDATE STATISTICS

SQL Server Locking

Topic Objective
To provide an overview of this topic.

Lead-in
In this section, we'll cover…

- Concurrency Problems Prevented by Locks
- Lockable Resources
- Types of Locks
- Lock Compatibility

This section describes concurrency issues, the resource items that can be locked, the types of locks that can be placed on those resources, and how locks can be combined.

Concurrency Problems Prevented by Locks

Topic Objective
To describe common locking concurrency issues.

Lead-in
Locks are useful to ensure transaction integrity in these situations…

- **Lost Update**
- **Uncommitted Dependency (Dirty Read)**
- **Inconsistent Analysis (Nonrepeatable Read)**
- **Phantoms Reads**

Locks can prevent the following situations that compromise transaction integrity:

Lost Update An update can get lost when a transaction overwrites the changes from another transaction. For example, two users can update the same information, but only the last change saved is reflected in the database.

Uncommitted Dependency (Dirty Read) An uncommitted dependency occurs when a transaction reads uncommitted data from another transaction. The transaction can potentially make changes based on data that is either inaccurate or nonexistent.

Inconsistent Analysis (Nonrepeatable Read) An inconsistent analysis occurs when a transaction reads the same row more than one time and when, between the two (or more) readings, another transaction modifies that row. Because the row was modified between readings within the same transaction, each reading produces different values, which introduces inconsistency.

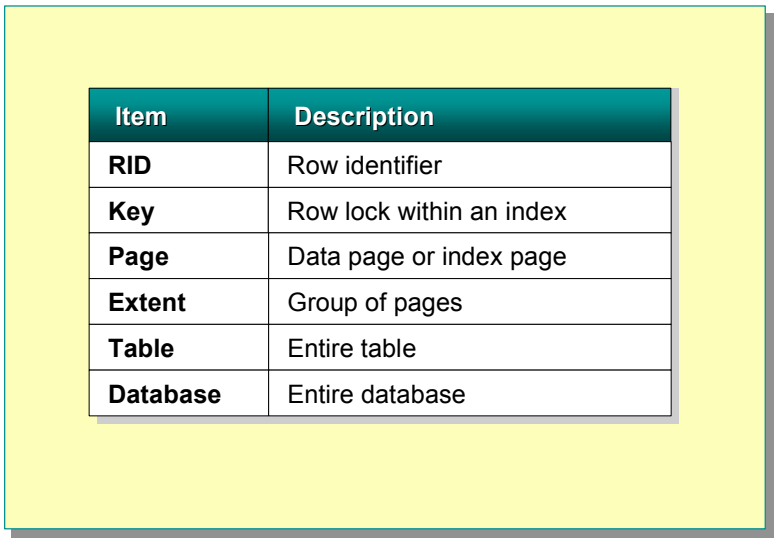
For example, an editor reads the same document twice, but between each reading, the writer rewrites the document. When the editor reads the document for the second time, it has completely changed. The original reading is not repeatable, leading to confusion. It would be better if the editor only starts reading the document after the writer has completely finished writing it.

Phantom Reads Phantom reads can occur when transactions are not isolated from one another. For example, you could perform an update on all records in a region at the same time that another transaction inserts a new record for the region. The next time that the transaction reads the data, an additional record is present.

Lockable Resources

Topic Objective
To list the resource items that SQL Server can lock.

Lead-in
The number of locks must be balanced with the amount of data that each lock holds.



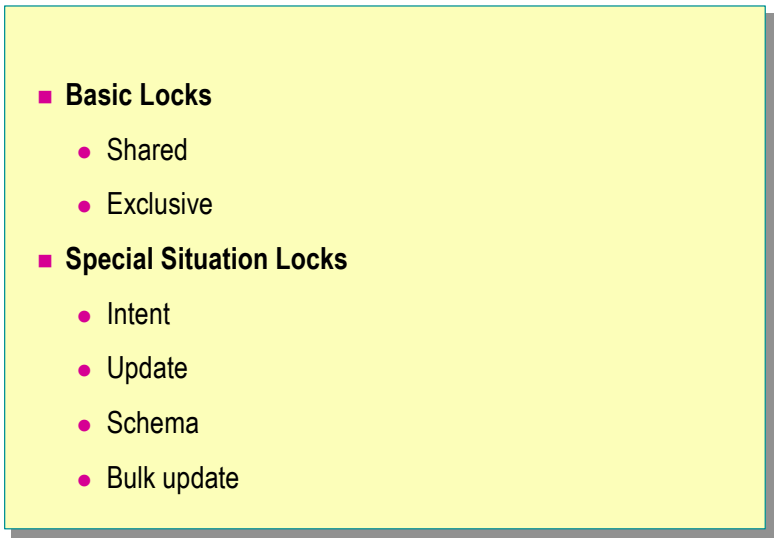
For optimal performance, the number of locks that SQL Server maintains must be balanced with the amount of data that each lock holds. To minimize the cost of locking, SQL Server automatically locks resources at a level that is appropriate to the task. SQL Server can lock the following types of items.

Item	Description
RID	A row identifier—used to lock a single row within a table
Key	A row lock within an index—used to protect key ranges in serializable transactions
Page	An 8-KB data page or index page
Extent	A contiguous group of data pages or index pages—used during space allocation
Table	An entire table, including all data and indexes
Database	An entire database—used during the restoration of a database

Types of Locks

Topic Objective
To list the types of locks.

Lead-in
SQL Server has two main types of locks: basic locks and locks for special situations.



SQL Server has two main types of locks: basic locks and locks for special situations.

Basic Locks

In general, read operations acquire shared locks, and write operations acquire exclusive locks.

Shared Locks

SQL Server typically uses shared (read) locks for operations that neither change nor update data. If SQL Server has applied a shared lock to a resource, a second transaction also can acquire a shared lock, even though the first transaction has not completed.

Consider the following facts about shared locks:

- They are used for read-only operations; data cannot be modified.
- SQL Server releases shared locks on a record as soon as the next record is read.
- A shared lock will exist until all rows that satisfy the query have been returned to the client.

Exclusive Locks

SQL Server uses exclusive (write) locks for the INSERT, UPDATE, and DELETE data modification statements.

Consider the following facts about exclusive locks:

- Only one transaction can acquire an exclusive lock on a resource.
- A transaction cannot acquire a shared lock on a resource that has an exclusive lock.
- A transaction cannot acquire an exclusive lock on a resource until all shared locks are released.

Special Situation Locks

Depending on the situation, SQL Server may use other types of locks:

Intent Locks

SQL Server uses intent locks internally to minimize locking conflicts. Intent locks establish a locking hierarchy so that other transactions cannot acquire locks at more inclusive levels. For example, if a transaction has an exclusive row-level lock on a specific customer record, the intent lock prevents another transaction from acquiring an exclusive lock at the table-level.

Intent locks include intent share (IS), intent exclusive (IX), and shared with intent exclusive (SIX).

Update Locks

SQL Server uses update locks when it will modify a page at a later point. Before it modifies the page, SQL Server promotes the update page lock to an exclusive page lock to prevent locking conflicts.

Consider the following facts about update locks. Update locks are:

- Acquired during the initial portion of an update operation when the pages are first being read.
- Compatible with shared locks.

Schema Locks

Schema locks ensure that a table or index is not dropped, or its schema modified, when it is referenced by another session.

SQL Server provides two types of schema locks:

- Schema stability (Sch-S), which ensures that a resource is not dropped.
- Schema modification (Sch-M), which ensures that other sessions do not reference a resource that is under modification.

Bulk Update Locks

Bulk update locks allow processes to bulk copy data concurrently into the same table while preventing other processes, that are not bulk-copying data, from accessing the table.

SQL Server uses bulk update locks when either of the following options is specified: the TABLOCK hint or the **table lock on bulk load** option, which is set using the **sp_tableoption** system stored procedure.

Lock Compatibility

Topic Objective
To explain which locks are compatible.

Lead-in
Locks may or may not be compatible with other locks.

- **Locks May or May Not Be Compatible with Other Locks**
- **Examples**
  - Shared locks are compatible with all locks except exclusive
  - Exclusive locks are not compatible with any other locks
  - Update locks are compatible only with shared locks

Delivery Tip
Address lock matrix by using the slide examples.

Locks may or may not be compatible with other locks. Locks have a *compatibility matrix* that shows which locks are compatible with other locks that are obtained on the same resource. The locks in the following table are listed in order from the least restrictive (shared) to the most restrictive (exclusive).

	Existing granted lock					
Requested lock	**IS**	**S**	**U**	**IX**	**SIX**	**X**
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Note An IX lock is compatible with other IX locks because IX means the intention to update only some of the rows, rather than all of them.

In addition, compatibility for schema locks is as follows:

- The schema modification lock (Sch-M) is incompatible with all locks.
- The schema stability lock (Sch-S) is compatible with all locks except the schema modification lock (Sch-M).

Managing Locks

Topic Objective
To provide an overview of this topic.

Lead-in
In this section, we'll cover…

- Session-Level Locking Options
- Dynamic Locking Architecture
- Table-Level Locking Options
- Deadlocks
- Displaying Locking Information

This section describes locking options that you can specify at the session and table levels. It also describes how SQL Server handles deadlocks and how you can view information about locks.

Session-Level Locking Options

Topic Objective
To introduce the transaction isolation level.

Lead-in
SQL Server allows you to control locking options at the session level.

- **Transaction Isolation Level**
  - READ COMMITTED (DEFAULT)
  - READ UNCOMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- **Locking Timeout**
  - Limits time waiting for a locked resource
  - Use SET LOCK_TIMEOUT

SQL Server allows you to control locking options at the session level by setting the transaction isolation level.

Transaction Isolation Level

An *isolation level* protects a specified transaction from other transactions. Use the transaction isolation level to set the isolation level for all transactions during a session. When you set the isolation level, you specify the default locking behavior for all statements in your session.

Setting transaction isolation levels allows programmers to accept increased risk of integrity problems in exchange for greater concurrent access to data. The higher the isolation level, the longer locks are held and the more restrictive these locks are.

You can override a session-level isolation level in individual statements by using lock specification. You also can use the DBCC USEROPTIONS statement to specify transaction isolation for a statement.

Syntax

SET TRANSACTION ISOLATION LEVEL {READ COMMITTED | READ UNCOMMITTED | REPEATABLE READ | SERIALIZABLE}

The following table describes the locking isolation level options.

Option	Description
READ COMMITTED	Directs SQL Server to use shared locks while reading. At this level, you cannot experience dirty reads.
	Directs SQL Server to not issue shared locks and does not honor exclusive locks. You can experience dirty reads.
REPEATABLE READ	Indicates that dirty reads and nonrepeatable reads cannot occur. Read locks are held until the end of the transaction.
SERIALIZABLE	Prevents other users from updating or inserting new rows that match the criteria in the WHERE clause of the transaction. Phantoms cannot occur.

Example

The following example sets the isolation level for the current session to READ UNCOMMITTED and then checks DBCC USEROPTIONS to verify that SQL Server has made the change.

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
DBCC USEROPTIONS
```

Result

```
set option                value
textsize                  64512
language                  us_english
dateformat                mdy
datefirst                 7
.
.
.
isolation level           read uncommitted

(13 row(s) affected)
```

Note DBCC always prints the following message when it is executed:

```
DBCC execution completed. If DBCC printed error messages, see
your System Administrator.
```

Delivery Tip
Demonstrate locking timeouts by using multiple query windows.

Locking Timeout

With the SET LOCK_TIMEOUT option, it is possible to set the maximum amount of time that SQL Server allows a transaction to wait for the release of a blocked resource.

Syntax

SET LOCK_TIMEOUT *timeout_period*

timeout_period is the number of milliseconds that pass before SQL Server returns a locking error. A value of -1 (the default) indicates no timeout period. After you change it, the new setting is in effect for the remainder of the session.

Example

This example sets the lock timeout period to 180,000 milliseconds.

```
SET LOCK_TIMEOUT 180000
```

To determine the current session value, query the **@@lock_timeout** global variable.

Example

This example displays the current **@@lock_timeout** setting.

```
SELECT @@lock_timeout
```

Result

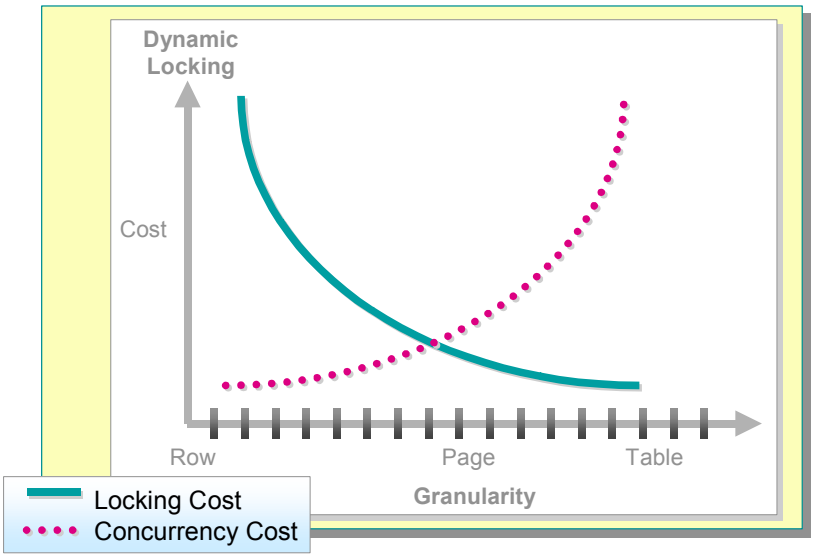
```
180000

(1 row(s) affected)
```

Dynamic Locking Architecture

Topic Objective
To show how the most cost effective locks are found.

Lead-in
Dynamic locking architecture helps determine the most cost effective locks.



SQL Server uses a dynamic locking architecture to determine the most cost-effective locks. It automatically determines what locks are most appropriate when a query is executed, based on the characteristics of the schema and query.

SQL Server dynamically increases and decreases the granularity and types of locks. The query optimizer usually chooses the correct lock granularity at the time that the execution plan is compiled, thus minimizing the need to escalate locks.

For example, if an update acquires a large number of row-level locks and has locked a significant percentage of a table, the row-level locks are escalated to a table lock. Then the transaction holds the row-level locks, thereby reducing lock overhead.

Dynamic locking has the following advantages:

- Simplified database administration, because database administrators no longer have to be concerned with adjusting lock escalation thresholds
- Increased performance, because SQL Server minimizes system overhead by using locks appropriate to the task

Table-Level Locking Options

Topic Objective
To introduce table locking options.

Lead-in
In usual practice, you should not specify a table-level locking option.

- **Use with Caution**
- **Can Specify One or More Locking Options for a Table**
- **Use *optimizer_hints* Portion of FROM Clause in SELECT or UPDATE Statement**
- **Overrides Session-Level Locking Options**

Although SQL Server uses dynamic locking architecture to select the best lock for your client, it is possible to specify table-level locking options. A table hint can specify a locking method for the query optimizer to use with a specific table and for a statement.

Note Use table-level locking options only after you thoroughly understand how your application works and have determined that the lock that you request will continue, over time, to be better than that which SQL Server would use.

The following characteristics apply to table-level locking options:

- You can specify one or more locking options for a table.
- Use the *optimizer_hints* portion of the FROM clause in a SELECT or UPDATE statement.
- These locking options override corresponding session-level (transaction isolation level) options that were previously specified with the SET statement.

The following table describes the table-level locking options.

Option	Description
HOLDLOCK SERIALIZABLE REPEATABLEREAD READCOMMITTED READUNCOMMITTED NOLOCK	Control locking behavior for a table and override the locks that would be used to enforce the isolation level of the current transaction
ROWLOCK PAGLOCK TABLOCK TABLOCKX	Specify the size and type of the locks to be used for a table
READPAST	Skip locked rows
UPDLOCK	Use update locks instead of shared locks

Deadlocks

Topic Objective
To illustrate why and how a deadlock occurs.

Lead-in
A deadlock occurs when two transactions have locks on separate objects and each transaction requests a lock on the other transaction's object.

- How SQL Server Ends A Deadlock
- How to Minimize Deadlocks
- How to Customize the Lock Time-Out Setting

Delivery Tip
Be sure to emphasis the difference between deadlocks and blocking locks.

A deadlock occurs when two transactions have locks on separate objects and each transaction requests a lock on the other transaction's object. Each transaction must wait for the other to release the lock.

A deadlock can occur when several long-running transactions execute concurrently in the same database. A deadlock also can occur as a result of the order in which the optimizer processes a complex query, such as a join, in which you cannot necessarily control the order of processing.

How SQL Server Ends a Deadlock

SQL Server ends a deadlock by automatically terminating one of the transactions. The process SQL Server uses is in the following list.

1. Rolls back the transaction of the deadlock victim.

   In a deadlock, SQL Server gives priority to the transaction that has been processing the longest; that transaction prevails. SQL Server rolls back the transaction with the least amount of time invested.
2. Notifies the deadlock victim's application (with message number 1205).
3. Cancels the deadlock victim's current request.
4. Allows the other transaction to continue.

Important In a multiuser environment, each client should check regularly for message number 1205, which indicates that the transaction was rolled back. If message 1205 is found, the application should attempt the transaction again.

How to Minimize Deadlocks

While it is not always possible to eliminate deadlocks, you can reduce the risk of a deadlock by observing the following guidelines:

- Use resources in the same sequence in all transactions. For example, if possible, reference tables in the same order in all transactions that reference more than one table.
- Shorten transactions by minimizing the number of steps.
- Shorten transaction times by avoiding queries that affect many rows.

How to Customize the Lock Time-Out Setting

If a transaction becomes locked while waiting for a resource and a deadlock results, SQL Server will terminate one of the participating transactions with no time-out.

If no deadlock occurs, SQL Server blocks the transaction requesting the lock until the other transaction releases the lock. By default, there is no mandatory time-out period that SQL Server observes. The only way to test whether a resource that you want to lock is already locked is to attempt to access the data, which could result in getting locked indefinitely.

The LOCK_TIMEOUT setting allows an application to set a maximum time that a statement waits on a blocked resource before the blocked statement is automatically cancelled. The cancellation does not roll back or cancel the transaction. The application must trap the error to handle the time-out situation and take remedial action, such as resubmitting the transaction or rolling it back.

The KILL command terminates a user process based on the server process ID (spid).

Displaying Locking Information

Topic Objective
To show the different information that you can find on active locks.

Lead-in
To display a report of active locks, execute the **sp_lock** system stored procedure.

- Current Activity Window
- sp_lock System Stored Procedure
- SQL Profiler
- Windows 2000 System Monitor
- Additional Information

Delivery Tip
Demonstrate the Current Activity window in SQL Server Enterprise Manager and SQL Profiler.

Typically, you use SQL Server Enterprise Manager or the **sp_lock** system stored procedure to display a report of active locks. You can use SQL Profiler to get information on a specific set of transactions. You can also use Microsoft Windows® 2000 System Monitor to display SQL Server locking histories.

Current Activity Window

Use the Current Activity window in SQL Server Enterprise Manager to display information on current locking activity. You can view server activity by user, detail activity by connection, and locking information by object.

sp_lock System Stored Procedure

The **sp_lock** system stored procedure returns information about active locks in SQL Server.

Syntax

```
EXECUTE sp_lock
```

Result

A typical result set resembles the following.

spid	dbid	ObjId	IndId	Type	Resource	Mode	Status
12	5	0	0	DB		S	GRANT
12	5	0	0	DB		S	GRANT
12	2	0	0	EXT	1:280	X	GRANT
12	5	0	0	PAG	1:528	IX	GRANT
12	5	981578535	0	RID	1:528:0	X	GRANT
12	1	5575058	0	TAB		IS	GRANT
12	5	981578535	0	TAB		IX	GRANT
13	1	0	0	DB		S	GRANT

The first four columns refer to various IDs: server process ID (spid), database ID (dbid), object ID (ObjId), and the index identification number ID (IndId).

The **Type** column shows the type of resource that is currently locked. Resource types can include: DB (database), EXT (extent), TAB (table), KEY (key), PAG (page), or RID (row identifier).

The **Resource** column has information on the resource type that is being locked. A resource description of 1:528:0 indicates that row number 0, on page number 528, on file 1 has a lock applied to it.

The **Mode** column describes the type of lock that is being applied to the resource. Types of locks can include: shared (S), exclusive (X), intent (I), update (U), or schema (Sch).

The **Status** column shows whether the lock has been obtained (GRANT), is blocking on another process (WAIT), or is in the process of being converted (CNVRT).

SQL Profiler

SQL Profiler is a tool that monitors server activities. You can collect information about a variety of events by creating traces, which provide a detailed profile of server events. You can use this profile to analyze and resolve server resource issues, monitor login attempts and connections, and correct deadlock problems.

Windows 2000 System Monitor

You can view SQL Server locking information with System Monitor. Use the **SQL Server: lock manager** and **SQL Server: locks** objects.

Additional Information

To find information about locks and current server activity, you can query the **syslockinfo**, **sysprocesses**, **sysobjects**, **systables**, **and syslogins** system tables or you can execute the **sp_who** system stored procedure.

Recommended Practices

Topic Objective
To list the recommended practices for managing transactions and locks.

Lead-in
Use the following recommended practices when you manage transactions and locks.

You should consider adopting the following recommended practices when you manage transactions and locks:

- Keep transactions as short as possible, as this reduces the possibility of locking conflicts with other transactions. A transaction should never be smaller than a logical unit of work.
- Design transactions to minimize deadlocks so that transactions do not have to be resubmitted.
- Use SQL Server defaults for locking because the optimizer generally uses the best locks based on the particular transaction and other activity in the database.
- Be careful when you use locking options, and test your transactions to ensure that your locking choices are better than the SQL Server defaults.

Lab A: Managing Transactions and Locks

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will execute transactions, set locking options, and view locking results.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Define transactions with the BEGIN TRANSACTION and COMMIT TRANSACTION statements.
- Determine the number of active transactions by querying the **@@trancount** global variable.
- Use the **sp_lock** system stored procedure and SQL Server Enterprise Manager to view locking information.
- Use the SET TRANSACTION ISOLATION LEVEL statement to control session-level locking behavior.
- Use table-level locking options to control locking behavior for specific tables.
- Use the SET LOCK_TIMEOUT statement to control the maximum amount of time that a statement will wait for a lock to be released.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L15.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore15.cmd batch file.

  This command file restores the **ClassNorthwind** database to a state required for this lab.

For More Information

If you require help with executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious **nwtraders.msft** domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.*x*.1
SQLAdmin2	Denver	192.168.*x*.2
SQLAdmin3	Perth	192.168.*x*.3
SQLAdmin4	Brisbane	192.168.*x*.4
SQLAdmin5	Lisbon	192.168.*x*.5
SQLAdmin6	Bonn	192.168.*x*.6
SQLAdmin7	Lima	192.168.*x*.7
SQLAdmin8	Santiago	192.168.*x*.8
SQLAdmin9	Bangalore	192.168.*x*.9
SQLAdmin10	Singapore	192.168.*x*.10
SQLAdmin11	Casablanca	192.168.*x*.11
SQLAdmin12	Tunis	192.168.*x*.12
SQLAdmin13	Acapulco	192.168.*x*.13
SQLAdmin14	Miami	192.168.*x*.14
SQLAdmin15	Auckland	192.168.*x*.15
SQLAdmin16	Suva	192.168.*x*.16
SQLAdmin17	Stockholm	192.168.*x*.17
SQLAdmin18	Moscow	192.168.*x*.18
SQLAdmin19	Caracas	192.168.*x*.19
SQLAdmin20	Montevideo	192.168.*x*.20
SQLAdmin21	Manila	192.168.*x*.21
SQLAdmin22	Tokyo	192.168.*x*.22
SQLAdmin23	Khartoum	192.168.*x*.23
SQLAdmin24	Nairobi	192.168.*x*.24

Estimated time to complete this lab: 60 minutes

Exercise 1
Creating and Executing a Transaction

In this exercise, you will use the BEGIN TRANSACTION and COMMIT TRANSACTION statements to understand the impact of the statements on the way that data is modified. You also will see how SQL Server uses the **@@trancount** global variable to determine whether a transaction is active.

► To create and execute a transaction

In this procedure, you will use the BEGIN TRANSACTION and COMMIT TRANSACTION statements to control how an UPDATE statement is processed on the **Customers** table.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the **nwtraders.msft** classroom domain)
Password	**password**

2. Open SQL Query Analyzer and, if prompted, log in to the (local) server with Microsoft Windows® Authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

3. In the **DB** list, click **ClassNorthwind**.

4. Open C:\Moc\2073A\Labfiles\L15\Tran1.sql and review its contents.

   Notice that the BEGIN TRAN statement is followed by an UPDATE statement, but no corresponding COMMIT TRAN or ROLLBACK TRAN statement is present. The SELECT and PRINT statements and the **@@trancount** global variable are used in the script to show the progress of the transaction.

5. Execute the script and review the results.

   At this point, are the changes that were made with the UPDATE statement committed in this transaction? How can you determine this?

   No, the transaction must be completed with a COMMIT TRAN statement. The transaction is still active, and it still holds any locks that it acquired. The @@trancount global variable value is 1, indicating that one BEGIN TRAN statement has been issued on this session.

   _____

   _____

   Would other transactions be able to query or update the changed data?

   Other transactions would not be able to query or modify the changed data until the transaction has been committed (or rolled back).

   _____

   _____

6. Enter a COMMIT TRANSACTION statement in the query window, and then highlight and execute it to complete the transaction and make the change permanent.
7. Highlight and execute one of the SELECT statements for the **Customers** table to verify that the change has now been completed.

Exercise 2
Rolling Back a Transaction

In this exercise, you will use the ROLLBACK TRANSACTION statement to understand the impact of the way that data is modified within a transaction.

► To use the ROLLBACK TRANSACTION statement

In this procedure, you will use the BEGIN TRANSACTION and ROLLBACK TRANSACTION statements to control how an UPDATE statement is processed on the **member** table.

1. Open C:\Moc\2073A\Labfiles\L15\Tran2.sql and review its contents.

   Notice that this script is similar to Tran1.sql, but the contact name is different, and a ROLLBACK TRAN statement has been added.

2. Execute the script and review the results.

   Is the change made by the UPDATE statement stored permanently in the database?

   No. The transaction was rolled back, so any changes that were made during the transaction are undone.

   Is the transaction complete?

   Yes. The ROLLBACK TRAN statement completes the transaction and releases any locks that the transaction has acquired. Querying the @@trancount global variable returns zero.

Exercise 3
Viewing Locking Information

In this exercise, you will execute multiple transactions simultaneously to determine the impact that such activity has on locking. You will be asked to open multiple connections with SQL Query Analyzer to simulate multiple users sending transactions to SQL Server.

Notice that the scripts used for this exercise do not always include COMMIT TRAN or ROLLBACK TRAN statements. The absence of these statements keeps the transactions open and the associated locks active so that you can view locking information.

► To view locking information

In this procedure, you will use the BEGIN TRANSACTION and ROLLBACK TRANSACTION statements to control how an UPDATE statement is processed on the **Customers** table.

1. Start SQL Query Analyzer (connection 1), click **Clear Query Window**.
2. Execute the **sp_lock** system stored procedure and review the output.
3. Start SQL Server Enterprise Manager.
4. In SQL Server Enterprise Manager, in the console tree, expand your server, expand **Management**, and then expand **Current Activity**. Review the information that is displayed in **Process Info**, **Locks / Process ID** and **Locks / Object**.
5. Open a second connection with SQL Query Analyzer (connection 2) and in the **DB** list, click **ClassNorthwind**.
6. Open C:\Moc\2073A\Labfiles\L15\Lock1.sql by using connection 2 and review its contents.

   Notice that a transaction is started with the BEGIN TRAN statement but a corresponding COMMIT TRAN or ROLLBACK TRAN statement to complete the transaction does not exist.
7. Execute \Lock1.sql by using connection 2 and review the results.
8. Switch to connection 1, execute the **sp_lock** system stored procedure, and then review the lock information.

   Identify the different lock types and resources locked by the transaction. Make a note of this information for subsequent use in Exercise 4.

   _____

   _____

9. Switch to SQL Server Enterprise Manager, right-click **Current Activity**, and then click **Refresh**.
10. Review the information in **Current Activity** that is displayed in **Process Info**, **Locks / Process ID** and **Locks / Object**.
11. Switch to connection 2 and cancel the transaction that you started in step 7 by executing a ROLLBACK TRAN statement.
12. Switch to connection 1 and execute the **sp_lock** system stored procedure.

    You will see that the locks acquired by the transaction in step 6 have now been released.

Exercise 4
Setting Locking Options

In this exercise, you will use some SQL Server locking options to determine how they affect the way that transactions are processed. You will use the connections that you established with SQL Query Analyzer in Exercise 3 to simulate multiple users sending transactions to SQL Server. You also can use the Current Activity window in SQL Server Enterprise Manager to view locking information for this exercise.

► To set the transaction isolation level

In this procedure, you will use the SET TRANSACTION ISOLATION LEVEL statement to control session-level locking behavior. You will use the connections that you established with SQL Query Analyzer in Exercise 3.

1. Switch to SQL Query Analyzer (connection 2).
2. Open the C:\Moc\2073A\Labfiles\L15\Lock2.sql script file by using connection 2, review its contents, and then execute it.
3. Switch to connection 1, execute the **sp_lock** system stored procedure, and then review the lock information.

   Identify and review the different lock types and resources that are locked by the transaction.

   Did SQL Server use different locks than those that were used in step 8 of Exercise 3? Why or why not?

   SQL Server used an exclusive table lock this time because the SET TRANSACTION ISOLATION LEVEL statement instructed it to use the type of lock that is required to achieve the isolation level that is specified.

   _____

   _____

4. Switch to connection 2 and cancel the transaction by executing a ROLLBACK TRAN statement.

► To use locking options for tables

In this procedure, you will use table-level locking options to control locking behavior.

1. Switch to SQL Query Analyzer (connection 1).
2. Execute the **sp_lock** system stored procedure by using connection 1 and review the output.
3. Switch to SQL Query Analyzer (connection 2).
4. Open the C:\Moc\2073A\Labfiles\L15\Lock3.sql script file by using connection 2 and review its contents.

   Notice that a table locking option has been defined in the FROM clause of the SELECT statement. Also notice that there is no COMMIT TRAN or ROLLBACK TRAN statement in the script.

5. Execute the Lock3.sql script file in connection 2.
6. Switch to connection 1 and execute the **sp_lock** system stored procedure.

   Make a note of the types of locks that are in use and the resources that are locked.

7. Open a third connection with SQL Query Analyzer (connection 3) and in the **DB** list, click **ClassNorthwind**.
8. Open the C:\Moc\2073A\Labfiles\L15\Lock1.sql script file by using connection 3 and then execute it.
9. Switch to connection 1 and execute the **sp_lock** system stored procedure.

   Is one transaction unable to execute? If so, why?

   Yes, the second transaction is attempting to update data that is locked by the first transaction, so the second transaction must wait for the first one to complete. You can determine whether a transaction cannot execute due to a locking conflict by using the sp_lock system stored procedure and then looking for the word WAIT in the status column. In this case, the transaction on connection 3 must wait for the transaction on connection 2 to complete.

   How long will a transaction wait on a locked resource?

   A transaction will wait indefinitely on a locked resource as long as it is not deadlocked. You can set a lock timeout period for the session in order to control how long SQL Server waits for locked resources.

10. Switch to connection 3 and on the toolbar, click **Cancel Query Execution**.
11. Switch to connection 1 and execute the **sp_lock** system stored procedure to verify that the waiting transaction has been cancelled.

► To set the lock timeout period for a transaction

In this procedure, you will set the lock timeout period so that the transaction will wait to acquire a lock for a specified time.

1. Switch to connection 3 and edit the C:\Moc\2073A\Labfiles\L15\Lock1.sql script file by adding the following statement immediately before the BEGIN TRAN statement:

   ```
   SET lock_timeout 500
   ```

2. Execute the edited script by using connection 3.

   What happened, and why?

   The transaction on connection 3 was waiting for resources that were locked by the transaction on connection 2. SQL Server cancelled the transaction in connection 3 because the lock timeout period that you specified had expired.

   _____

   _____

3. Close all windows in SQL Query Analyzer.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Introduction to Transactions and Locks
- Managing Transactions
- SQL Server Locking
- Managing Locks

Use these questions to review module topics.

Ask students whether they have any questions before continuing.

1. You are developing a new order entry system for your company. You expect that the system will be very active because 450 operators take orders from customers 24 hours a day. Should operators process all items that a customer orders during a single phone call in a single transaction?

   It probably is best to treat each product that is ordered as a separate transaction.

2. Once a month you are required to perform an update to the **products** table in your order entry system. The **products** table contains millions of items. Each monthly update is expected to affect at least 65 percent of the rows in the table. You can write a single, complex UPDATE statement to perform the update, which typically takes at least 30 minutes to execute. Is this the best way to perform the update?

   No, break the statement into a group of smaller transactions, if possible, to minimize locking contention with other users.

3. You are receiving calls from users. They say that the response time of the order entry system periodically increases to more than 20 seconds. You have promised them a three-second response time. You suspect that locking conflicts may exist within the system. How would you determine the source of the problem?

   Use the sp_lock system stored procedure or the current activity window in SQL Server Enterprise Manager to identify the problem. Then, modify the transactions involved.