SQL Server 2005

Targeted at: Entry Level Trainees

Session 33: Transactions and Locking

About the Author

Created By:	Shashidharamurthy R L (124294)
Credential Information:	Senior Instructor, Cognizant Academy MCP, MCTS, SCJP, SCWCD
Version and Date:	SQLSERVER2005/PPT/0408/1.0

Cognizant Certified Official Curriculum

Icons Used

Questions

Tools

Hands on Exercise

Coding Standards

Test Your Understanding

Reference

Demonstration

A Welcome Break

Contacts

SQL Server 2005 Session 33: Overview

- **Introduction:**

  This session explains about query processing, transactions, locks and provides the overview of using Multiple Servers.

SQL Server 2005 Session 33: Objective

- **Objective:**

After completing this session, you will be able to:

» Work with transactions in SQL Server

» Define locks

» Explain different Isolation levels

» Describe query processing

» Explain the overview of using multiple servers

Transactions in SQL Server

- A transaction is a single unit of work:
  - If a transaction is successful, then all of the data modifications made during the transaction are committed and become a permanent part of the database.
  - If a transaction encounters errors must be canceled or rolled back, then all of the data modifications are erased.
- SQL Server operates in the following transaction modes:
  - **Autocommit:** Each individual statement is a transaction.
  - **Explicit:** Each transaction is explicitly started with the `BEGIN TRANSACTION` statement and explicitly ended with a `COMMIT` or `ROLLBACK` statement.

Transactions in SQL Server (Contd.)

- **Implicit:** A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a `COMMIT` or `ROLLBACK` statement.
- **Batch-scoped:** Applicable only to multiple active result sets (MARS), a Transact-SQL explicit or implicit transaction that starts under a MARS session becomes a batch-scoped transaction. A batch-scoped transaction that is not committed or rolled back when a batch completes is automatically rolled back by SQL Server.

Explicit Transaction

- EXPLICIT TRANSACTION:
  - Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
  - BEGIN TRANSACTION:
    - Marks the starting point of an explicit transaction for a connection.

Explicit Transaction (Contd.)

» `COMMIT TRANSACTION` or `COMMIT WORK`:

- Used to end a transaction successfully if no errors are encountered. All data modifications made in the transaction become a permanent part of the database. Resources held by the transaction are freed.

» `ROLLBACK TRANSACTION` or `ROLLBACK WORK`:

- Used to erase a transaction in which errors are encountered. All data modified by the transaction is returned to the state as it was in at the start of the transaction. Resources held by the transaction are freed.

Explicit Transaction (Contd.)

- Example to commit transaction:

```
/*This example deletes a job candidate. */
USE AdventureWorks;
GO
BEGIN TRANSACTION;
GO
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
GO
COMMIT TRANSACTION;
GO
```

Explicit Transaction (Contd.)

- Example to name a transaction:

```
/*This example demonstrates how to name a transaction*/

DECLARE @TranName VARCHAR(20);
SELECT @TranName = ‘MyTransaction’;

BEGIN TRANSACTION @TranName;
GO
USE Adventure Works;
GO
DELETE FROM AdventureWorks.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
GO

COMMIT TRANSACTION MyTransaction;
GO
```

Nesting Transactions

- Explicit transactions can be nested. This is primarily intended to support transactions in stored procedures that can be called either from a process already in a transaction or from processes that have no active transaction.
- The transaction is either committed or rolled back based on the action taken at the end of the outermost transaction. If the outer transaction is committed, then the inner nested transactions are also committed. If the outer transaction is rolled back, then all inner transactions are also rolled back, regardless of whether or not the inner transactions were individually committed.

Nesting Transactions: Example

```
USE AdventureWorks;
GO
CREATE TABLE TestTrans(Cola INT PRIMARY KEY,
               Colb CHAR(3) NOT NULL);
GO
CREATE PROCEDURE TransProc @PriKey INT, @CharCol CHAR(3) AS
BEGIN TRANSACTION InProc
INSERT INTO TestTrans VALUES (@PriKey, @CharCol)
INSERT INTO TestTrans VALUES (@PriKey + 1, @CharCol)
COMMIT TRANSACTION InProc;
GO
/* Start a transaction and execute TransProc. */
BEGIN TRANSACTION OutOfProc;
GO
```

Nesting Transactions: Example (Contd.)

```
EXEC TransProc 1, 'aaa';
GO
/* Roll back the outer transaction, this will
   roll back TransProc's nested transaction. */
ROLLBACK TRANSACTION OutOfProc;
GO
EXECUTE TransProc 3,'bbb';
GO
/* The following SELECT statement shows only rows 3 and 4 are
   still in the table. This indicates that the commit
   of the inner transaction from the first EXECUTE statement of
   TransProc was overridden by the subsequent rollback. */
SELECT * FROM TestTrans;
GO
```

Error Handling in transactions

- It is the responsibility of the programmer to code the application to specify the correct action (`COMMIT` or `ROLLBACK`) if a run-time or compile error occurs.
- Effective tool for handling errors in transactions is the Transact-SQL TRY…CATCH construct.

Distributed Transactions

- A transaction within a single instance of the Database Engine that spans two or more databases is actually a distributed transaction.
- Distributed transactions span two or more servers known as resource managers. The management of the transaction must be coordinated between the resource managers by a server component called a transaction manager.

Distributed Transactions (Contd.)

- Instance of the SQL Server Database Engine can operate as a resource manager in distributed transactions coordinated by transaction managers, such as Microsoft Distributed Transaction Coordinator (MS DTC).
- Achieved by managing the commit process in two phases known as a two-phase commit (2PC), which are as follows:
  - » Prepare phase
  - » Commit phase

Locks

- Locking is a mechanism used by the Microsoft SQL Server Database Engine to synchronize access by multiple users to the same piece of data at the same time.
- Ensures transactional integrity:
  - » Prevent “reads” on data being modified by other users
  - » Prevent multiple users from changing same data at simultaneously
- Locks are managed internally by a part of the Database Engine called the lock manager.

Lock: Resources

- SQL server can lock the following resources:
  - **RID:** Row identifier used to lock a single row within a table
  - **Key:** Row lock within an index used to protect key changes in serializable transactions
  - **Page:** Data page or index page (8 KB)
  - **Extent:** Contiguous group of eight data pages
  - **Table:** Entire table including all table and indexes
  - **DB:** Database
  - **FILE:** A database file.

Lock Modes

- The Microsoft SQL Server Database Engine locks resources using different lock modes that determine how the resources can be accessed by concurrent transactions
- Lock modes that uses Database Engine :
  - » **Shared (S):** Used for read operations that do not change or update data, such as a SELECT statement.
  - » **Exclusive (X):** Ensures that multiple updates cannot be made to the same resource at the same time.

Lock Modes (Contd.)

- Lock modes that uses Database Engine:
  - **Update (U):** Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and potentially updating resources.
  - Other lock modes like Intent, Schema, Bulk Update (BU), and Key-range.

Isolation Levels

- Isolation levels allow you to control the consistency level while manipulating data when multiple processes might be running concurrently.
- SQL Server supports following Isolation levels:
  - » Read committed:
    - Specifies the shared locks held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction resulting in phantom data.
    - Prevents "dirty reads", only committed data can be read.

Isolation Levels (Contd.)

» Read uncommitted

- No locks specified, “dirty reads” are possible.

» REPEATABLE READ:

- Locks set on all data read through query
- Prevents update of the data used in a query, but new rows can be added by other users

» SERIALIZABLE:

- Places ‘range’ lock on all data being read, not released until transaction ends
- Most restrictive Isolation level

Isolation Levels (Contd.)

- SQL Server 2005 also supports two transaction isolation levels that use row versioning.
  - » Snapshot:
    - The snapshot isolation level uses row versioning to provide transaction-level read consistency. Read operations acquire no page or row locks; only SCH-S table locks are acquired. When reading rows modified by another transaction, they retrieve the version of the row that existed when the transaction started.
    - Snapshot isolation is enabled when the ALLOW_SNAPSHOT_ISOLATION database option is set ON. By default, this option is set OFF for user databases

Isolation Levels (Contd.)

- SQL Server 2005 also supports two transaction isolation levels that apply row versioning:
  - Read Committed Snapshot:
    - `READ_COMMITED_SNAPSHOT` database option is set **ON**, read committed isolation applies row versioning to provide statement-level read consistency. Read operations require only SCH-S table level locks and no page or row locks. When the `READ_COMMITED_SNAPSHOT` database option is set **OFF**, which is the default setting.

Query Processing

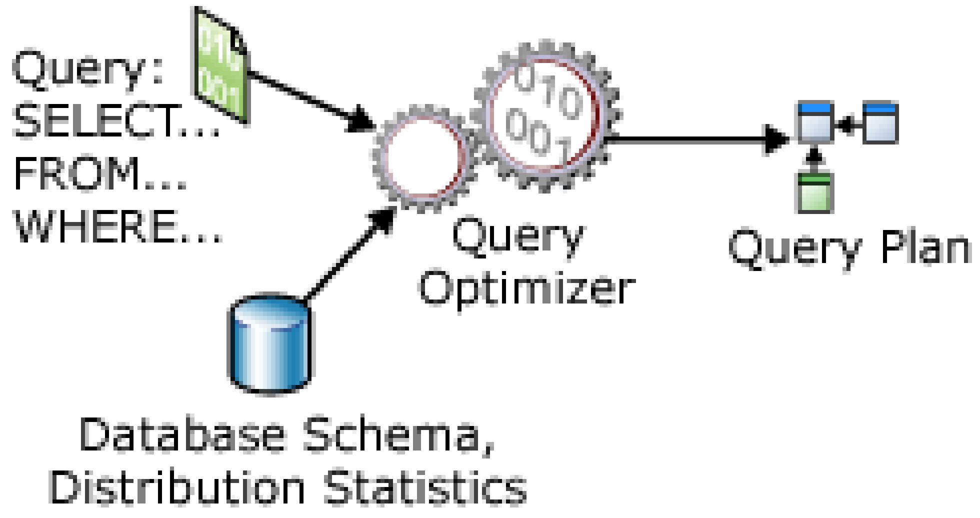
- Steps that SQL Server applies to process a SELECT statement:
  1. The parser scans the SELECT statement and breaks it into logical units such as keywords, expressions, operators, and identifiers.
  2. A query tree, sometimes referred to as a sequence tree, is built describing the logical steps needed to transform the source data into the format required by the result set.
  3. The query optimizer analyzes different ways the source tables can be accessed. It then selects the series of steps that returns the results fastest while using fewer resources. The query tree is updated to record this exact series of steps. The final, optimized version of the query tree is called the execution plan.

Query Processing (Contd.)

4. The relational engine starts executing the execution plan. As the steps that require data from the base tables are processed, the relational engine requests that the storage engine pass up data from the row sets requested from the relational engine.
5. The relational engine processes the data returned from the storage engine into the format defined for the result set and returns the result set to the client.

Query Processing (Contd.)

Query Optimization of a single SELECT statement

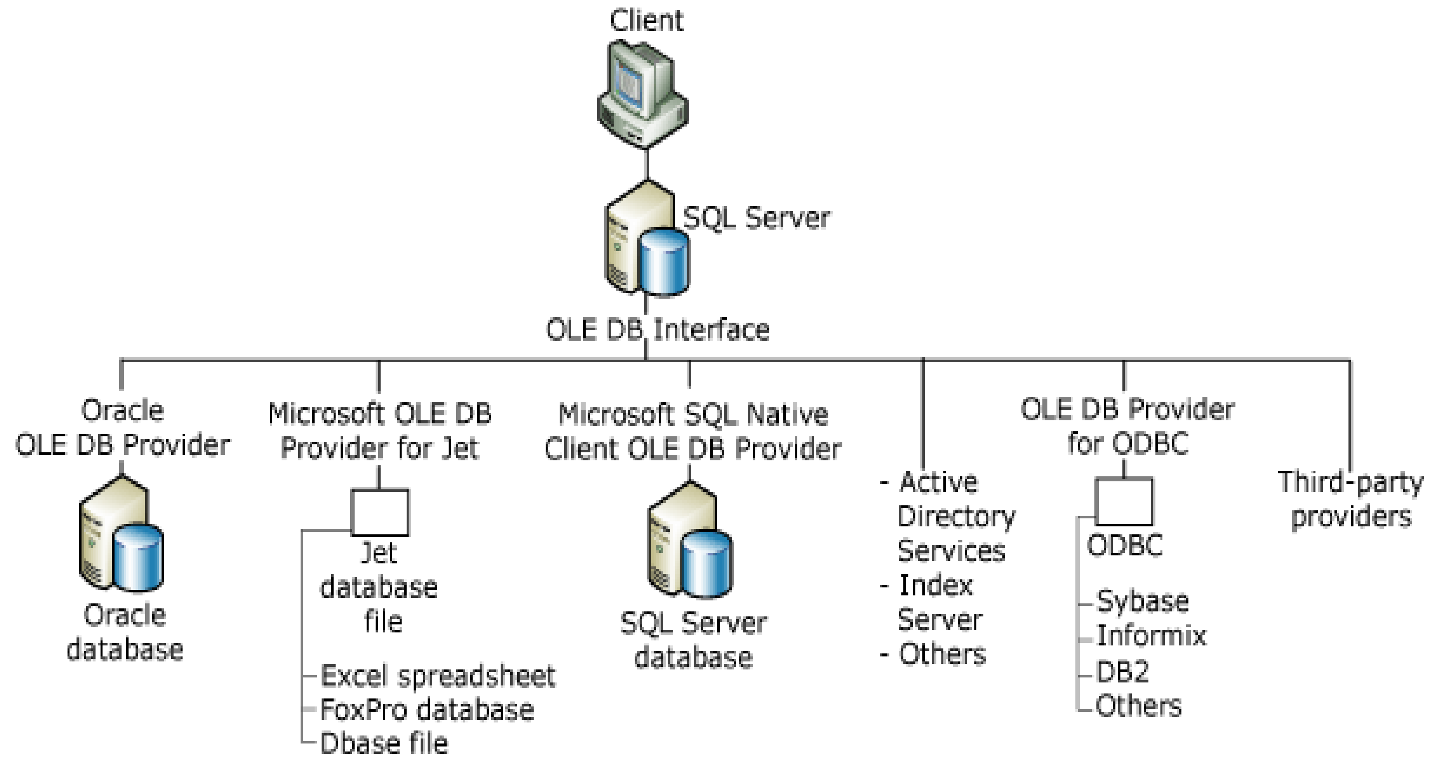


Distributed Queries

- Access data stored in multiple SQL servers
- Access data from heterogeneous data sources
- Implemented using OLE DB
  - OLE DB to communicate between Relational and storage engines
  - OLE DB exposes data in tabular objects (rowsets)
  - Allows rowsets to be accessed in T-SQL statements
  - Any data source with OLE DB provider allows heterogeneous access
- Two types of access are as follows:
  - AD HOC queries
  - Using linked servers

Distributed Queries (Contd.)

Architecture shows the connections between a client computer, an instance of SQL Server and an OLE DB provider.



Distributed Queries: AD HOC queries

- OpenRowSet:
  - A one-time, ad hoc method of connecting and accessing remote data.
  - Specify connection information and a Table, view or Select query.

```
SELECT a.*
FROM OPENROWSET('SQLNCLI', 'Server=Seattle1;
Trusted_Connection=yes;','SELECT GroupName, Name,
 DepartmentID FROM AdventureWorks.HumanResources.Department
      ORDER BY GroupName, Name') AS a;
```

Distributed Queries: AD HOC queries (Contd.)

- OPENDATASOURCE function:
  - Provides ad hoc connection information as part of a four-part object name.
  - Syntax :

```
OPENDATASOURCE ( provider_name, init_string )
```

  - AD HOC naming only for rarely used quires
  - **Frequent usage:** Apply *Linked servers*

Distributed Queries: Linked Servers

- A linked server configuration enables SQL Server to execute commands against OLE DB data sources on remote servers.
- Linked servers are applied to handle distributed queries.
- Linked servers offers the following advantages:
  - Remote server access.
  - The ability to issue distributed queries, updates, commands, and transactions on heterogeneous data sources across the enterprise.
  - The ability to address diverse data sources similarly.

Linked Servers

- Apply stored procedures and catalog views to manage linked server definitions:
  - » Create a linked server definition by running `sp_addlinkedserver` procedure.
  - » View information about the linked servers defined in a specific instance of SQL Server by running a query against the sys.servers system catalog views.
  - » Delete a linked server definition by running `sp_dropserver` procedure.

Linked Servers (Contd.)

- Syntax:

```
sp_addlinkedserver [ @server= ] 'server' [ , [ @srvproduct= ] 'product_name' ]
     [ , [ @provider= ] 'provider_name' ]
     [ , [ @datasrc= ] 'data_source' ]
     [ , [ @location= ] 'location' ]
     [ , [ @provstr= ] 'provider_string' ]
     [ , [ @catalog= ] 'catalog' ]
```

- Example:

```
sp_addlinkedserver @server = N'LinkServer',
    @srvproduct = N' ',
    @provider = N'SQLNCLI',
    @datasrc = N'ServerNetName',
    @catalog = N'AdventureWorks'
GO

SELECT *
FROM LinkServer.AdventureWorks.dbo.Vendor
```

Q & A

- Allow time for questions from participants

Try it Out

Problem Statement:

- Create a table by name `tblTest` and set isolation level isolation levels in SQL Server 2005 as `READ UNCOMMITTED` and observe the behavior.

Cognizant | Academy
Passion for making a difference

Try it Out (Contd.)

- Explore setting isolation level by executing following steps:
  1. Create a table called `Test` with the following data in it.

     ```
     TestID TestColumn
     -------------------
     1         100
     ```
  2. Open two instances of SQL Server Management Studio. In each instance, use the database that contains the `Test` table. These two instances will be used to simulate two users running two concurrent transactions.

Try it Out (Contd.)

3. In instance 1, execute an UPDATE on the row of data by executing the following code block:

```
BEGIN TRANSACTION UPDATE Test SET TestColumn
= 200 WHERE TestId =1
```

4. In instance 2 of Management Studio, execute the following query:

```
SELECT TestColumn FROM Test WHERE TestId = 1
```

Cognizant | Academy
Passion for making a difference

Try it Out (Contd.)

- You will notice that your SELECT query is blocked. This makes sense because you are trying to read the same data that instance 1 is busy modifying. Unless instance 1 issues a COMMIT or a ROLLBACK, your query will remain blocked or will simply time out.

5. Cancel your blocked SELECT query by pressing Alt+Break or clicking the **Cancel** button on the toolbar. Execute the following T-SQL command to set the isolation level of your SELECT query to read uncommitted on the connection held by instance 2.

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Try it Out (Contd.)

6. Execute the SELECT query again.

```
SELECT TestColumn FROM Test WHERE TestId = 1
```

- You will find that the query is not blocked and it produces 200 as a result.

7. Go back to instance 1 and issue a ROLLBACK.
8. Back in instance 2, execute the same SELECT query again. You should get 100 as the result.

Cognizant | Academy
Passion for making a difference

Try it Out (Contd.)

How it Works:

- Observe the following when the steps are executed:
  - » Instance two returns different results for the same query at different times. As a matter of fact, the value 200 was never committed to the database.
  - » Observe dirty read, because of `READ UNCOMMITTED` isolation level.

Q & A

- Allow time for questions from participants

Cognizant | Academy
Passion for making a difference

Test Your Understanding

1. Which is the default isolation level for SQL Server 2005?
   a. READ UNCOMMITTED
   b. READ COMMITTED
   c. REPEATABLE READ
   d. SNAPSHOT
   e. SERIALIZABLE

SQL Server 2005 Session 33: Summary

- A transaction is a single unit of work.
- Locking is a mechanism used by the Microsoft SQL Server Database Engine to synchronize access by multiple users to the same piece of data at the same time.
- Distributed queries access data from multiple heterogeneous data sources.
- A linked server configuration enables SQL Server to execute commands against OLE DB data sources on remote servers.

SQL Server 2005 Session 33: Source

- SQL Server Books Online
- Microsoft Official Curriculum:
  - 2779A Implementing a Microsoft SQL Server 2005 Database

You have completed the Session 33 of SQL Server 2005.